


## REVIEW

# An evaluation of data stream clustering algorithms

Stratos Mansalis<sup>1</sup>  | Eirini Ntoutsi<sup>2</sup> | Nikos Pelekis<sup>3</sup> | Yannis Theodoridis<sup>1</sup>

<sup>1</sup>Department of Informatics, University of Piraeus, Piraeus, Greece

<sup>2</sup>L3S Research Center, Leibniz Universität Hannover, Hannover, Germany

<sup>3</sup>Department of Statistics and Insurance Science, University of Piraeus, Piraeus, Greece

## Correspondence

Stratos Mansalis, Department of Informatics, University of Piraeus, Piraeus, Greece.  
Email: efmansalis@gmail.com

Data stream clustering is a hot research area due to the abundance of data streams collected nowadays and the need for understanding and acting upon such sort of data. Unsupervised learning (clustering) comprises one of the most popular data mining tasks for gaining insights into the data. Clustering is a challenging task, while clustering over data streams involves additional challenges such as the single pass constraint over the raw data and the need for fast response. Moreover, dealing with an infinite and fast changing data stream implies that the clustering model extracted upon such sort of data is also subject to evolution over time. Several stream clustering surveys exist already in the literature; however, they focus on a theoretical presentation of the surveyed algorithms. On the contrary, in this paper, we survey the state-of-the-art stream clustering algorithms and we evaluate their performance in different data sets and for different parameter settings.

## KEYWORDS

data stream clustering, data streams, evaluation, experimental, survey

## 1 | INTRODUCTION

In recent years, with the enormous growth of World Wide Web and the advances in hardware and software technologies, we have the ability to track in real time any kind of transactions such as customer click data, patient health data, TCP/IP traffic, GPS data etc, in order to support real-time decision-making. Typically, to extract useful information out of large amounts of data, data mining techniques are employed, like clustering, classification and frequent item sets mining.

However, conventional data mining techniques used for static data sets are not suitable for data stream mining. First, in a data stream environment, we have a continuous inflow of data objects which is potentially infinite, in contrast to conventional data mining where the whole data set is known in advance and is given as input to the mining algorithm. As a result, there is a need for fast processing for each incoming data object from the stream. Moreover, due to the never ending nature of the streams, random accesses to the data as in conventional data mining are not allowed, rather there is a limitation of having only a single look at the data, upon their arrival. Except for the challenges imposed due to the size and arrival rate of the streams, another challenge is triggered by

the volatility of the data. Data streams are evolving in their nature in contrast to static data sets which are assumed to be generated by a static distribution. As a result, the corresponding patterns (clustering models) extracted upon such data are also subject to change over time.

Due to the aforementioned challenges, the stream clustering field has attracted many researchers in the recent years and as a result, a variety of algorithms has been proposed, such as Stream [25], CluStream [4], DenStream [17], and DStream [19]. Some of the algorithms “transfer” the conventional clustering algorithms to the stream scenario, for example, Stream [25] comprises the *k*-Means version for data streams. Other algorithms though, such as CluStream [4], provide solutions that are tailored to data streams.

Due to the interest in the field, several surveys exist in the related literature. An overview of the field and its challenges is presented in an early work by Guha et al. [26]. More recent surveys also exist like the ones by Aggarwal [3] and Silva et al. [47]. Another survey by Amini et al. [8] focuses on density- and grid-based stream clustering algorithms. Finally, a more general data mining data streams survey recently published by Nguyen et al. [39]. However, none of the aforementioned surveys provides an experimental evaluation on the performance of the different algorithms.

In this paper, we overview the state-of-the-art data stream clustering algorithms and we evaluate their performance in different data sets and for different parameter settings in the open-source framework MOA [13,14].

We make the code, data, and experimental results publicly available.\* At a glance, our contributions are as follows:

1. We provide a review of the state-of-the-art stream clustering algorithms and their follow-ups.
2. We experimentally evaluate their performance in a variety of data sets and for different parameter settings and evaluation measures.

The remainder of the paper is organized as follows: Section 2 describes the preliminaries on data streams and data stream clustering. Section 3 surveys the existing algorithms, focusing mainly on state-of-the-art methods but their follow-ups are also discussed. The experimental setup, data sets, and evaluation measures are presented in Section 4. The evaluation analysis is presented in Section 5. Conclusions are discussed in Section 6, open issues and future work are discussed in Section 7.

## 2 | BASIC CONCEPTS AND CHALLENGES

A data stream [2] is a massive sequence of objects  $o_1, o_2, \dots$  which arrive continuously over time and at a rapid rate. Each object  $o_i$  from the stream is a multidimensional vector,  $o_i = \langle o_i^1, o_i^2, \dots, o_i^d \rangle$  where  $d$  is the dimensionality of the feature space.

Due to the infinite nature of data streams, it is impossible to store all these data in memory or even in disk, so we have the constraint of a single pass over the data, typically upon their arrival. That is, once an object has been processed, it cannot be processed again (ie, random access is not allowed). Finally, the system has no control over the order in which data objects arrive. Typically, the objects are assumed to arrive independently of each other.

We already mentioned that conventional clustering algorithms [29] are not adequate for dealing with data streams, due to the nature of the streams [22]. We overview below the challenges for data stream clustering, which lead to the development of new algorithms, cf, Section 2.4. The first 3 challenges are closely related to clustering whereas the last 3 hold in general for any data processing in streams.

1. Evolving nature of data streams: Data in a data stream are not stationary, rather they evolve over time. The clusters extracted upon this data, therefore, should also evolve over time. This means that a stream clustering algorithm should update the extracted clusters continuously in order to capture these changes in the underlying data.

2. Number of clusters: Determining the correct number of clusters is a challenging problem even for a static data set. On top of that, in a stream setting, one has to configure the number of clusters over time. Assuming that a fixed number of clusters will be able to capture a stream population in the long run is quite restrictive.
3. Outliers: In real world, many noisy and outlier objects may appear due to, for example, failure of sensors, bad connections, etc. Clustering algorithms must be resilient to outliers. In a stream environment, this is very challenging as both clusters and outliers are developing over time. In other words, it is difficult to know upon the arrival of an object whether it is an outlier or the first member of a new cluster.
4. Single-pass constraint: Due to the massive volume of data, it is impossible to store the stream in memory, even on disk. Therefore, the data need to be processed in a single-pass way, upon their arrival and no random access is allowed.
5. Limited processing time: Since data stream objects arrive continuously and at a high speed, it is necessary for the algorithms to respond fast once they receive an object from the stream.
6. Limited memory: A common way to deal with this restriction is to maintain summaries over the data instead of the original raw data and use these summaries for clustering or other mining tasks latter on.

To illustrate the previously discussed challenges, we provide an example of a real world problem: Suppose a weather station, which receives and processes information about weather conditions in 1 forest from thousands of sensors along the forest. The weather station receives continuously and at a high-speed information from all these sensors regarding temperature, wind speed and direction, humidity, location of sensor, etc. Obviously, it is impossible to do a batch processing by applying some conventional clustering algorithm, as the stream is unbounded and continuously developing. Moreover, it is impossible to store all this information in memory, rather incremental and quick processing of the data should be carried. Due to the exposure of the sensors in all different weather conditions, it is possible for a sensor to fail, for example, low battery, no network or “strange measurements” due to, for example, some fire nearby. A clustering algorithm should be able to provide valid clusters over time and also “highlight” the outliers as they might call for action, for example, sensor replacement, fire extinguishing, etc.

### 2.1 | Window models

Theoretically, a data stream is infinite. To control which part of the stream contributes to the data mining patterns, window models are employed. Several window models have been proposed in the literature, the most popular are landmark window, sliding window, damped window,

\*<http://smansalis.me/EDSCpaper/>

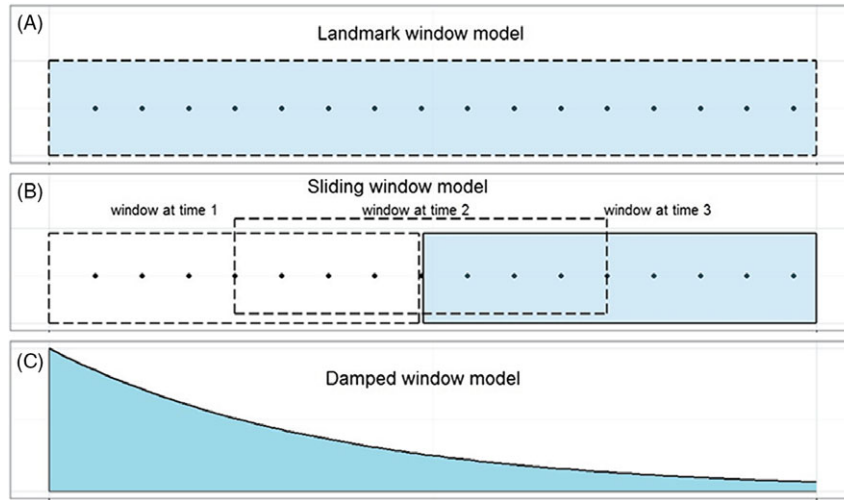


FIGURE 1 Window models illustrations: (A) landmark, (B) sliding, (C) damped

and tilted window models, described in more details hereafter.

- **Landmark window model:** In this model, clustering is applied from a starting time point, called landmark, to the current time point. The landmark window can be defined in terms of the number of objects observed since a new landmark is set (eg, every 1000 objects) or in terms of time (eg, weekly, monthly). When a new window period starts, all objects kept in from the previous landmark are removed. A special case of the landmark window is when  $startpoint = 1$ . In this case, we are interested over the entire history of the stream. One of the limitations of this window model is that it is difficult to define the proper landmarks. Also, all points in the window are treated as equally important.
- **Sliding window model:** In this model, there is a window of fixed size  $w$  from the current time point  $t$ . As time goes by, starting from the current time  $t$ , the window keeps its size  $w$  and slides. Thus, each window consists only of the objects that lie in the interval  $[t - w + 1, t]$ , while older objects are discarded. The window can be defined in terms of time points (eg, the last 100 time points) or in terms of objects (eg, the last 1000 objects); in the latter case, the data are equidistant. This model is suitable for applications where we are interested only in the most recent objects. Obviously, depending on the window size, one can consider more or less data from the past. Moreover, all instances within the active sliding window are considered of equal importance.
- **Damped window model:** In this model, each object is associated with a weight which depends on its arrival time. When a new object arrives, it is assigned the highest possible weight; this weight decreases exponentially over time according to some aging function. A typically used aging function for the damped window model is the exponential fading function, cf, Equation (1).

$$f(t) = 2^{-\lambda(t_c - t_o)} \quad (1)$$

The weight decrease rate is controlled by the fading factor  $\lambda$ ;  $t_c$  is the current timepoint and  $t_o$  is the creation time. In contrast to the previous models, the damped window model does not discard objects completely, rather older objects contribute less as they are assigned lower weights. The interpretation of this model is more difficult as (potentially all) objects are “active” but with a weight that depends on their age.

- **Tilted window model:** Time is registered at different levels of granularity using bounded space in a particular kind of windowing system that keep summaries over the whole window period. There are several kinds of tilted window techniques, one of them is the progressive logarithmic tilted time window model [4] which allows an efficient temporal dimension organization taking different snapshots that describe the system where each snapshot is represented by its time-stamp and the snapshots are stored at different levels of granularity in a pyramid structure. As a result, the most recent objects are kept at the finest granularity, whereas the old ones are registered at a coarser granularity. The intuition is that the recent objects are the most interesting and should be fully modeled, whereas for older objects a coarser representation is enough. As in the damped window model, the tilted window model focuses more on the recent data, without completely discarding old data as, for example, the sliding window model does.

Figure 1 shows an illustration of how landmark, sliding, and damped window models control the part of the stream that contributes to the data mining model over time. As illustrated, the landmark window model controls the whole stream history (or, starting from some landmark), in contrast to the sliding window model which steadily slides its window focusing solely on the most recent, within the window, observations, and the damped window model that exponentially decreases the significance of the past objects.

## 2.2 | Data processing models

Processing of data for clustering purposes follows typically one of the following directions:

1. **Online clustering:** The initial proposed methods [25,41] view the stream clustering problem as a single-pass clustering challenge and adopt a general adaptive strategy to maintain the clusters. A single clustering model is maintained over the stream and it is updated as new data objects arrive from the stream. Such an approach though does not allow for investigation of the cluster structure at different time intervals.
2. **Online-offline clustering:** To overcome the limitations of the online approach, Aggarwal et al. [4] introduced the online-offline clustering approach. The 2-phase clustering consists of an online phase that maintains statistics over the stream in an online fashion and of an offline phase that performs the actual clustering upon these statistics based on user-defined temporal predicates. Such an approach provides more flexibility in data stream exploration, as multiple clusterings can be extracted from the stream by selecting different time windows from the summaries.

## 2.3 | Data summaries

Clusters already offer an abstraction over the original raw data, and even in conventional batch clustering, clusters are usually described by some summary description, for example, the cluster centroid. In the case of streams, summarization is a necessity since, as we already mentioned, we have the requirements for a single pass over the data and fast response. We overview the most commonly used summaries for stream clustering hereafter.

1. **Cluster feature (CF):** The CF feature concept was first introduced in the BIRCH algorithm [49]. A CF for a set of  $N$   $d$ -dimensional data objects is defined as the triple:

$$CF = (\overrightarrow{CF1}^d, \overrightarrow{CF2}^d, N) \quad (2)$$

where  $N$  is the number of data objects,  $\overrightarrow{CF1}^d$  is the linear sum of the data objects, and  $\overrightarrow{CF2}^d$  is the squared sum of the data objects—both  $\overrightarrow{CF1}^d$  and  $\overrightarrow{CF2}^d$  are  $d$ -dimensional arrays. CF allows for computing basic cluster measures like cluster centroid (cf, Equation (3)), radius (cf, Equation (4)) and diameter (cf, Equation (5)).

$$\text{centroid} = \frac{\overrightarrow{CF1}^d}{N} \quad (3)$$

$$\text{radius} = \sqrt{\frac{\overrightarrow{CF2}^d}{N} - \left(\frac{\overrightarrow{CF1}^d}{N}\right)^2} \quad (4)$$

$$\text{diameter} = \sqrt{\frac{2N * \overrightarrow{CF2}^d - 2 * (\overrightarrow{CF1}^d)^2}{N(N-1)}} \quad (5)$$

The CF summary is a very appealing summary structure for streams as it can be easily maintained online, thanks to the incrementality property (cf, Equation (6)). According to this property, a new object  $x$  can be easily incorporated into the summary.

$$\begin{aligned} \overrightarrow{CF1}^d &\leftarrow \overrightarrow{CF1}^d + x, \\ \overrightarrow{CF2}^d &\leftarrow \overrightarrow{CF2}^d + (x)^2, \\ N &\leftarrow N + 1 \end{aligned} \quad (6)$$

Similarly, 2 summaries can be easily merged, thanks to the additivity property (cf, Equation (7)); the merged summary is just a summation of the corresponding summaries.

$$\begin{aligned} \overrightarrow{CF1}_l^d &\leftarrow \overrightarrow{CF1}_k^d + \overrightarrow{CF1}_m^d, \\ \overrightarrow{CF2}_l^d &\leftarrow \overrightarrow{CF2}_k^d + \overrightarrow{CF2}_m^d, \\ N_l &\leftarrow N_k + N_m \end{aligned} \quad (7)$$

Many CF-based summaries for stream clustering have been proposed, the most known one is the *microcluster* summary structure. We present it below along with other summaries.

2. **Micro-clusters:** As already mentioned, CFs were initially introduced for static data in BIRCH [49]. Their adaptation to data streams was done by Aggarwal et al. [4], where they extend the summaries with temporal information reflecting their recency.

The new summaries are called micro-clusters and are defined as follows:

$$MC = (\overrightarrow{CF1}^d, \overrightarrow{CF2}^d, CF1^t, CF2^t, N) \quad (8)$$

The new definition keeps the basic CF components and extends the summary by adding 2 more components, the sum of the timestamps  $CF1^t$  and the sum of the squares of the timestamps  $CF2^t$ ; these additions allow for the computation of the temporal cluster measures. Micro-clusters are employed by CLuStream [4] and FlockStream [21].

3. **Core-micro-clusters:** In Cao et al. [17] the micro-cluster definition is extended for density-based clustering. In particular, 3 micro-cluster variations have been proposed, core-micro-clusters, potential-core-micro-clusters, and outlier-micro-clusters

A core-micro-cluster at time  $t$ , for a group of data objects  $o_1, \dots, o_n$  is defined as a triple:

$$CMC = (w, c, r) \quad (9)$$

where  $w$  denotes the weight of this core micro-cluster (CMC),  $r$  is the radius, and  $c$  is the center of the micro-cluster. The weight of the core-micro-cluster  $w$  at time  $t$  is computed as:

$$w_t = \sum_{j=1}^n f(t - T_j) \quad (10)$$

where  $T_1, \dots, T_n$  are the arrival times of the corresponding objects  $o_1, \dots, o_n$ .

Note that the importance of each data object  $o_i$  decreases exponentially with time via the function  $f(t) = 2^{-\lambda \delta t}$ , where



$\lambda > 0$  is a user-specified parameter, so the importance of each summary decreases similarly and  $\delta t$  is the interval from the current timepoint to the creation timepoint of the object.

The center of the micro-cluster is computed as follows:

$$c = \frac{\sum_{j=1}^n f(t - T_{ij}) p_{ij}}{w} \quad (11)$$

and the radius as:

$$r = \frac{\sum_{j=1}^n f(t - T_{ij}) \text{dist}(p_{ij}, c)}{w} \quad (12)$$

where  $\text{dist}(p_{ij})$  denotes the Euclidean distance between point  $p_{ij}$  and the center  $c$ .

The weight of a core-micro-cluster must be above or equal to a threshold  $\mu$ , where  $\mu$  is a user-defined parameter and the radius must be below or equal to a user defined boundary  $\varepsilon$ . The core-micro-cluster summary has been employed by DenStream [17], rDenStream [37], C-DenStream [46], HDDStream [40], MuDi-Stream [7], HDenStream [36], and PreDeConStream [28].

4. Temporal CF: A temporal CF for a set of  $d$ -dimensional records  $o_1, o_2, \dots, o_n$  with timestamps  $t_1, t_2, \dots, t_n$  is defined as

$$CFT = (\overrightarrow{CF1}^x, \overrightarrow{CF2}^x, N, T) \quad (13)$$

It is a temporal extension of CF which keeps its basic components (cf, Equation (2)) but also adds the timestamp  $T$  of its most recent objects. Such a summary has been employed by SWClustering [50] and SDStream [43].

5. Prototype array: It is an array of the prototypes derived from  $k$ -median clustering. Because the first algorithms which were proposed simply view the stream clustering as a single-pass problem, the goal of these algorithms is to handle the infinite size of streams summarizing the stream history by dividing the stream into batches of predefined size  $m$ , in which each batch returns the  $k$ -representatives in the array. When the size of the representative array reaches the maximum boundary  $m$  these algorithms perform clustering in these  $k$ -representatives making the next level of representatives. Such a summary has been employed by Stream [25] and Stream LSearch [41] methods.

6. Grids: A grid also constitutes a summary as from the raw data one abstracts to the grid cells. The summary depends on the grid parameters, like the cell size. Another parameter regarding cell density is usually employed, which distinguish between enough populated cells and empty or low-populated cells. Each grid cell is represented by a summary, which can be seen as a virtual point, that summarizes the data points within the cell. Such a summary has been employed by D-Stream [19], DDStream [30], MR-Stream [34], DENGRIS [6], and PKS-Stream [44].

7. Coreset Tree: Coreset tree  $T$  is a binary tree in which for a point set  $P$ , starting from the root that contains the whole point set  $P$  as a single cluster, divides the current cluster into 2 subclusters until the number of clusters corresponds to the predefined  $k$  number of clusters. Each node of  $T$  contains: (1) a point set  $P_i$ , (2) a representative point  $q_i$ , (3) the total number

of objects of this node  $N_i$ , and (4) the sum of the squared distances of the objects of all points in  $P_i$  to  $q_i$   $\text{cost}(i)$ . Such a summary has been employed by StreamKM++ [1].

## 2.4 | Clustering approaches

Following the conventional clustering algorithms taxonomy, stream clustering algorithms can be categorized into partitioning methods, density-based, grid-, and model-based methods.

1. *Partitioning algorithms* partition the data into  $k$  clusters, where  $k$  is specified by the user. The partitioning is based on some criterion optimization like Sum of Square Errors. In general, these algorithms produce spherical clusters and do not handle outliers. Related data stream clustering algorithms are Stream Framework algorithms [25], Clustream [4], SWClustering [50] and StreamKM++ [1].
2. *Density-based algorithms* consider clusters as high-density regions which are well separated by low-density regions. They can discover clusters of arbitrary shapes and identify outliers. The number of clusters is not required as input, the input parameters concern the definition of the object's neighborhood and its density. Related data stream clustering algorithms are DenStream [17], rDenStream [37], C-DenStream [46], SDStream [43], HDDStream [40], MuDi-Stream [7], and HDenStream [36].
3. *Grid-based algorithms* are a special category of density-based algorithms, where the regions consist of the grid cells. In particular, the data space is partitioned into a finite number of cells that form a grid structure on which clustering is performed. Related data stream clustering algorithms are D-Stream [19], DDStream [30], MR-Stream [34], DENGRIS [6] and PKS-Stream [44].
4. *Model-based algorithms* try to fit a model to the data, assuming that data are generated from  $k$  probability distributions (typically Gaussian). In this category belongs SWEM [20].

## 3 | STREAM CLUSTERING ALGORITHMS

In this section, we review the state-of-the-art stream clustering algorithms and their follow-ups. The main characteristics of these algorithms are summarized in Table 1. A detailed presentation is provided below, following the traditional classification of stream clustering algorithms into partitioning-based, density-based, grid-based and model-based approaches.

### 3.1 | Partitioning-based stream clustering

For static data, partitioning-based clustering aims at partitioning the data into  $k$  clusters by optimizing some optimization criterion. For data streams, the challenge is how to update

TABLE 1 Stream clustering algorithms reviewed in this paper

Algorithm	Year	Approach	Window	Summary	Underlying method	Processing
Stream [25]	2000	Partitioning	Landmark	Representatives	k-median	Single-Pass
Stream LSearch [41]	2002	Partitioning	Landmark	Representatives	$k$ -median	Single-pass
CluStream [4]	2003	Partitioning	Tilted	Micro-cluster	$k$ -means	Online-offline
DUCStream [23]	2005	Grid-based	Landmark	Grid/graph	Dense-units	Single-pass
DenStream [17]	2006	Density-based	Damped	Core-micro-cluster	DBSCAN	Online-offline
D-Stream [19]	2007	Grid-based	Damped	Grid/hash-table	Dense regions	Online-offline
SWClustering [50]	2008	Partitioning	Sliding	EHCF	k-means	Online-offline
DDStream [30]	2008	Grid-based	Damped	Grid/hash-table	DCQ-means	Online-offline
SDStream [43]	2009	Density-based	Sliding	EHCF	DBSCAN	Online-offline
HDenStream [36]	2009	Density-based	Damped	Core-micro-cluster	DBSCAN	Online-offline
FlockStream [21]	2009	Density-based	Damped	Micro-cluster	Swarms	Single-pass
rDenStream [37]	2009	Density-based	Damped	Core-micro-cluster	DBSCAN	Online-offline
C-DenStream [46]	2009	Density-based	Damped	Core-micro-cluster	C-DBSCAN	Online-offline
SWEM [20]	2009	Model-based	Sliding	Micro-components	EM algorithm	Online-offline
MR-Stream [34]	2009	Grid-based	Damped	Grid/tree	Dense regions	Online-offline
PKS-Stream [44]	2011	Grid-based	Damped	Grid/PKS-tree	Dense regions	Online-offline
ClusTree [32]	2011	Any time	Damped	Micro-clusters/tree	k-means	Self-adaptive
StreamKM++ [1]	2012	Partitioning	Landmark	Coreset tree	k-means++	Merge-reduce
PreDeConStream [28]	2012	Density-based	Damped	Core-micro-cluster	DBSCAN	Online-offline
DENGRIS [6]	2012	Grid-based	Sliding	Grid	Dense regions	Single-pass
HDDStream [40]	2012	Density-based	Damped	Core-micro-cluster	DBSCAN	Online-offline
MuDi-Stream [7]	2014	Density-based	Damped	Core-micro-cluster	DBSCAN	Online-offline
TS-Stream [42]	2015	Hierarchical	Sliding	tree	Decision tree	Single-pass
pcStream [38]	2015	Model-based	Damped	PCA	SIMCA	Single-pass
StreamXM [9]	2015	Partitioning	Landmark	Coreset	X-means	Merge-reduce
SNCStream [11]	2015	sn-model	Damped	Micro-cluster	Network	Online
SNCStream <sup>+</sup> [12]	2016	sn-model	Damped	Micro-cluster	Network	Online
EDDS [5]	2017	Density-based	Damped	Micro-cluster	DBSCAN	Online-Offline
WCDS [18]	2017	Neural network	Sliding	Micro-cluster	Agglomerative	Online-offline

such a partitioning as the stream progresses. In this category, we discuss 3 methods: Stream [25], CluStream [4], and CluStree [32].

The Stream framework by Guha et al. [25] is one of the earliest methods for stream clustering. The method is based on  $k$ -median clustering and the core idea is to break the stream into batches  $B_1, B_2, \dots, B_i, \dots$  of fixed size  $m$ . At each batch  $B_i$ , the  $k$ -median clustering algorithm is applied optimizing the sum of squared error  $SSE$ . After processing  $i$  batches, a total of  $i \cdot k$  medians will have been stored in the prototype array,  $k$  per batch. Whenever the number of stored medians exceeds a threshold  $m$ , the  $k$ -median algorithm is applied over these stored medians, generating the new set of medians.

Although this algorithm manages to handle the memory limitation and the single-pass constraint of unbounded streams, it does not deal with noise and data aging. Rather, old data are considered equally important to the new data. Moreover, a single model is maintained over the stream, which is not adequate to describe the whole evolution of the stream.

In O'Callaghan et al. [41], the framework was extended in order to get a more improved solution using an effective subroutine for  $k$ -median (LSearch) based on facility location.

Aggarwal et al. propose CluStream [4], a 2-phase clustering algorithm that allows for clustering at different time horizons. CluStream consists of: (1) a fast online phase which summarizes the stream into summaries (microclusters) and (2) an offline phase which runs on user-demand and performs a  $k$ -means clustering over the summaries located in a user-specified time horizon. The clusters over the micro-cluster summaries are called macroclusters. CluStream also introduced the use of the tilted window model in stream clustering; the microclusters are stored as snapshots in time.

In the initialization phase, the first  $q$  microclusters are built after a certain number of objects arrives from the stream. In the online phase, CluStream maintains, through the microclusters, statistical summary information about the incoming objects from the stream. There are 2 options for incorporating a new object into a microcluster: if the new object  $o$  is close enough to the centroid of an existing micro-cluster  $\mu_i$  and falls within its maximum boundary as captured by its radius,  $o$  is assigned into  $\mu_i$ . Otherwise, a new microcluster is created with  $o$  as its first object-member. Due to the fact that the numbers of microclusters are fixed and in order to accommodate the newly created microcluster, the old microclusters

have to be reduced by 1; this is achieved either by merging 2 old microclusters or by deleting the oldest microcluster. In the offline phase, the algorithm uses the summaries lying within the user specified time-horizon  $h$  and performs a modification of  $k$ -means over the summaries to derive the final  $k$  macroclusters;  $k$  is a user-defined parameter. The assumption behind the selection of  $q$ ,  $k$  is that the number of microclusters ( $q$ ) should be considerably smaller than the number of raw objects seen thus far and much larger than the number of final macroclusters ( $k$ ).

Although CluStream is designed to cluster streams over different time horizons in order to capture changes and patterns in evolving data streams which is crucial for stream clustering, there are still limitations in the method. Firstly, the number of microclusters is fixed over the course of the stream, which is nonrealistic in an evolving data stream. Also, CluStream produces convex-shaped clusters, as it adopts the  $k$ -means clustering paradigm, but real clusters are arbitrarily shaped. Finally, CluStream does not deal with outliers and noise, which is “risky” in a stream environment and might lead to the destruction of existing valid microclusters in order to accommodate noisy and/or outlier objects. ClusTree [32] is a parameter free self-adaptive *anytime stream clustering* algorithm that automatically adapts to the speed of the data stream without any assumption of the model and relies on a compact hierarchical index structure from the R-tree family to efficiently organize the data for maintaining stream summaries. The algorithm makes best use of the time available under the current constraints to provide a clustering of the objects which have arrived and it incorporates the age of the objects in order to give more importance to recent arriving objects.

ClusTree uses micro-clusters as a compact representation of the data. The basic idea is to build a hierarchy of micro-clusters at different levels of granularity in a balanced multidimensional tree-based indexing structure, where each inner node contains between  $m$  and  $M$  entries and stores a CF of the objects it summarizes, a CF of the objects in the buffer and a pointer to its child node. Each leaf node contains between  $l$  and  $L$  entries and stores a CF of the object it represents, the path from the root to any leaf node has always the same length.

The algorithm is based on 2 operations: (1) handling the incoming objects and (2) maintaining the tree. The maintenance is executed only when there are no incoming objects from the stream. It is possible that an object does not have enough time to reach its closest leaf node in the tree because of the interruptions for arriving new objects. In such a case, ClusTree has an important property that reflects its anytime capability, it uses aggregates and saves these objects temporarily in a local aggregate in the local subtree and whenever the object insertion process in a leaf node starts again the object insertion continues.

Another characteristic of ClusTree algorithm is its ability to adapt to data streams with different speeds. In fast streams, it is possible that insertion stops at the top levels and thus a lot of

objects might remain at buffers will become difficult to reach the leaf nodes. In such a case, the algorithm creates several aggregates for dissimilar objects in order to summarize similar objects in the same aggregate using a non user-specified parameter  $max_{radius}$ . This is the maximum distance of objects in an aggregate to the mean of the closest aggregate, so for very fast streams the algorithm stores interrupted objects in their closest aggregate, if  $max_{radius}$  is exceeded a new aggregate is created. In slow streams, when the insertion process reaches the leaf level and additional time is available, the leaf is split so automatically adapted to the stream speed and the algorithm tries to optimize the insertion method and decreases the memory consumed.

The user has to define only 2 parameters that control the size of the tree, namely the capacity (number of entries) of internal nodes and the capacity of leaf nodes. The clustering result is a set of CFs which are stored at the leaf level. Based on this set of CFs any known clustering method such as  $k$ -means, DBSCAN, etc. can be applied taking the means of the CFs as representatives.

*StreamKM++* [1] is an extension for clustering data streams of the  $k$ -means++ [10] algorithm, which is a seeding procedure of  $k$ -means algorithm that guarantees a solution with certain quality and gives good experimental results. A coresets  $C_i$  is a subset of an input set  $P_i$  where we can get a good approximation with small error solving the optimization problem directly on  $C_i$  instead of working on  $P_i$  and without even reading the original input. Coresets construction is a nonuniform sampling process in which a small weighted sample is computed from the whole input, coresets are easy to implement and the running time has low dependency on the dimensionality of the data. *StreamKM++* uses coresets tree as data structure in order to speed up the time for the nonuniform sampling during coresets construction. The algorithm maintains a fixed number of  $L$  buffers,  $B_0, \dots, B_{L-1}$ , with maximal capacity  $m$  points. Each buffer  $B_i$  can be empty or contain  $m$  points except  $B_0$  which can store any number between 0 and  $m$ . When new points arrive, initially they are stored in  $B_0$ . If  $B_0$  is full, all points are moved to  $B_1$ . If  $B_1$  already contains  $m$  points,  $B_0$  and  $B_1$  are merged and out of their  $2 * m$  points in total a new coresets of size  $m$  is created in reduce step, of course if  $B_2$  is full, this process is repeated until a buffer  $B_i$  is empty. Similarly, *StreamXM* [9] is a novel stream clustering technique that does not require an arbitrary selection of number of clusters and it uses the X-stream algorithm to find the clusters.

*SWClustering* [50] is capable, except for the clustering, of also analyzing the evolution of the individual clusters. Zhou et al. [50] introduced a new data structure called Exponential Histogram of Cluster Feature (EHCF) which is a combination of Exponential Histogram used to handle in cluster evolution with Temporal Cluster Feature (TCF) that represents the changes of the cluster distribution in order to record the evolution of each cluster and to capture the distribution of recent records. *SWClustering* consists of 2 phases. In the

online phase, the algorithm maintains the incoming objects and stores them as synopses in the EHCF structure, which consists of TCFs, each denoted by  $h_i$ . In the offline phase, the algorithm clusters these collections of synopses using a variation of  $k$ -means in which the center of each  $h_i$  is treated as a pseudopoint with a weight  $m_i$ , where  $m_i$  is the number of records contained in  $h_i$ .

### 3.2 | Density-based stream clustering

Similarly to CluStream [4], *DenStream* [17] follows the online-offline rationale but it adopts a density-based cluster model based on the DBSCAN paradigm. For the online summarization, the micro-cluster summaries are adopted; however, the authors propose 3 different types of micro-clusters: CMC, potential micro-clusters (PMC), and outliers micro-clusters (OMC). The reason for this differentiation is to distinguish between actual micro-clusters and noise. All micro-clusters should have the same extent around their center  $c$  which is modeled by the radius parameter  $r$  and should be above a certain limit, that is,  $r \leq \epsilon$ . The different types of micro-clusters differ with respect to how many points they summarize, which is modeled by the weight parameter  $w$ . In particular, a CMC should contain more than  $\mu$  points, that is,  $w \geq \mu$ , a potential CMC should contain more than a user-defined fraction  $\beta$  of the data, that is,  $w \geq \beta \cdot \mu$  and an OMC should contain  $w < \beta \cdot \mu$  points. OMCs are annotated with a timestamp  $t_0$  denoting their creation time. The creation time is used to determine their life span, based on which nonpromising outliers are pruned. For discarding of old points, *DenStream* adopts the damped window model with an exponential aging function.

In the initialization phase, DBSCAN is applied upon the first objects from the stream and the initial *PMCs* are created. In the online phase, 2 lists of micro-clusters are maintained: *PMCs* and the *OMCs*. When a new object  $o$  arrives from the stream, the algorithm first tries to accommodate it in some existing *PMC*. To this end, the closest micro-cluster  $pmc \in PMC$  is located. The assignment is possible only if  $p$  falls in the extend of  $pmc$ , which is defined by its center and radius. If this is the case,  $p$  is assigned to  $pmc$  and its statistics are updated. If there is no *PMC* where  $p$  fits, a similar attempt is made for the *OMCs*. In particular the closest  $omc \in OMC$  is found and is tested whether it can accommodate  $p$ . If this is possible the statistics of  $omc$  are updated. Then it is checked whether  $omc$  has turned into a *PMC* due to the addition of  $p$ . If the assignment is not possible, a new *OMC* is created with  $p$ . Moreover, as part of the online phase a periodical bookkeeping of the micro-clusters takes place to ensure that aging is applied also to micro-clusters that do not receive any new points. As a result of this procedure some *PMCs* might turn into *OMCs*.

The offline phase runs on demand and takes place over the micro-cluster summaries instead of the raw data. In particular, each micro-cluster is treated as a virtual point located at its

center and a variant of DBSCAN is applied upon these virtual points. The resulting density-based macro-clusters, consist of such virtual points, that is, micro-clusters.

*DenStream* seems ideal for streams as it does not require a constant number of clusters over time, neither spherical-shaped clusters. Moreover, the differentiation between the different summary types, allows the algorithm to adapt to the underlying data distribution, that is, in times of drifts in the stream many new *OMC* might be created, whereas in times of stability the existing *PMC* might still be suitable to accommodate new data. On the other side, the density parameters  $\mu$  and  $\epsilon$  remain constant over the stream although in reality the stream density might change over time or there might be clusters of different densities in the stream.

Ren and Ma [43] have developed a density-based stream clustering algorithm named *SDStream* which is a variant of *DenStream* for sliding windows. Like *SWClustering* [50], *SDStream* stores the CMCs in the form of Exponential Histogram. Although it satisfies many of the stream clustering challenges, by using sliding window model, the algorithm ignores parts of stream history entirely.

Ruiz et al. [46] developed a density-based stream clustering algorithm named *C-DenStream* which is a combination of *DenStream* algorithm and C-DBSCAN [45] for clustering data streams. The algorithm uses background knowledge about instances that must belong to the same cluster and instances that must belong to different clusters; in the offline phase it uses C-DBSCAN to perform clustering.

Li-xiong et al. [37] proposed a variant of *DenStream* algorithm named *rDenStream* which extends the 2-phase framework of *DenStream* adding an extra phase called “retrospect”. In the third retrospect step of the algorithm, the discarded objects are placed in an outlier buffer and have a new chance to be added in the clustering in order to improve the clustering accuracy.

A significantly different method called *FlockStream* proposed in the study of Forestiero et al. [21]. *FlockStream* is a biologically inspired model [31] for simulating the animation of a flock in the stream scenario. Each data object is assigned to an agent which moves for a fixed time independently from the others onto a 2 dimensional space, in predefined range. When a new object arrives it compared only with the agents lie into its range. Although *Flockstream* needs fewer calculations in contrast with *DenStream* it is not clear how it handles outliers and fast processing.

Because some of the data streams produce high-dimensional data, many high-dimensional data streams clustering algorithms have been proposed in the literature. One of them is *HDDStream* [36] which is an extension of *DenStream* for clustering high-dimensional data. The algorithm in the online phase keeps information about the dimensions of the data stream and in the offline phase generates the final clusters using a projected clustering algorithm called *PreDeCon* [16]. Similarly, *PreDeConStream* [28] handles high-dimensional data. Amini et al. [7] proposed



another high-dimensional stream clustering algorithm named MuDi-Stream. The algorithm has 2 phases, an online which consists of 3 components for keeping and maintaining the incoming objects in core mini-clusters and an offline for generating the final clusters. Gong et al. [24] developed an effective and efficient method for clustering data stream algorithms called EDMStream. This method can track the evolution of data stream data by monitoring the density mountain as well as to capture the response cluster update in real time by using a structure called DP-Tree and a number of filtering schemes. Similarly, enhanced density-based data stream algorithm [5] is a new incremental algorithm known as an Enhanced Density-Based Method for Clustering Data Streams (EDDS) that developed to overcome limitations with the existing solutions. The algorithm detects clusters and outliers in an incoming data chunk, merges new clusters from the chunk with the existing clusters, and filters out new outliers for the next round. It modified the traditional DBSCAN algorithm to summarize each cluster in terms of a set of surface-core points.

### 3.3 | Grid-based stream clustering

D-stream [19] follows the online-offline rationale of CluStream but it adopts a grid-based clustering model. The data space is partitioned through a grid and the incoming objects are mapped to grid cells. Intuitively, a grid cluster is a connected group of grid cells which has higher density than the surrounding grid cells. The density of a cell is defined as the number of points falling into the cell. Data objects are subject to aging, based on the exponential aging function. Grid cells act as summaries and their statistics are maintained online, whereas the final clustering over the grid cells takes place offline. The grid partitioning is fixed through the whole stream lifespan; however, the number of points resulting in each grid cell, that is, its density, might vary as the stream evolves. As cells are gradually populated from the stream and in order to distinguish between outliers and potential dense cells, the authors propose a categorization/labeling of grid cells into dense, transitional, or sparse depending on their density.

Each grid cell  $g$  stores information about its objects in the form of a characteristic vector:  $tuple = \langle t_g, t_m, D, label, status \rangle$ , where  $t_g$  is the last update time for  $g$ ,  $t_m$  is the last time when  $g$  was categorized as sporadic,  $D$  is the last updated density of the grid and  $label$  is the class label of  $g$ , that is, one of dense, transitional, or sparse, depending on grid density. When an object arrives, it is mapped into its corresponding grid cell and the grid cell statistics are updated accordingly. Due to the coming of new data from the stream and the data aging, the statistics of the grid cells change over time, dense cells might not only degrade into transitional or sparse cells but also sparse or transitional cells might upgrade to transitional and dense cells, respectively.

The final clusters are dynamically adjusted in the offline phase every  $gap$  steps. The parameter  $gap$  is the minimum value among 2 alternatives: (1) the minimum time interval needed for a dense grid to degenerate to a sparse grid and (2) the minimum time needed for a sparse grid to become a dense grid. This periodical check (instead of checking at each timepoint) results in efficiency gain. Not all grid cells are considered in the offline phase, rather those with a low density, the so-called sporadic cells, are ignored. Sporadic cells include sparse cells with a low overall density in their lifespan. Note that sparse cells due to the natural aging of the data are kept. The removal of the sporadic cells also increases the efficiency as the number of grid cells against which new data are compared for assignment is decreasing.

As a density-based approach, D-stream is flexible in detecting nonspherical clusters. However, the grid partitioning remains constant over time, although the underlying stream might evolve in different ways. The online approach allows for a fast summarization of the data in the grid; the assignment of a point is really trivial due to the known cell coordinates. The offline component runs periodically but also exploits the latest clustering. This is in contrast to CluStream and DenStream that perform the offline clustering from scratch over the valid summaries.

DENGRIS [6] is similar to D-Stream but adopts a sliding window model, thus focusing exclusively on the most recent (within the active window) data. Similarly, DD-Stream [30] follows the D-stream rationale, however, in the offline phase uses an algorithm called DCQ-means to detect the border points of the grids which usually considered as sporadic grids and add these points in the clustering analysis in order to improve clustering quality. Gao et al. [23] proposed *DUCstream* a grid-based method for stream clustering. DUCstream is an incremental single-pass clustering algorithm which breaks the stream history in chunks each of which fit in memory, it partitions the data space in units and map the incoming objects in these units. The basic idea of the algorithm is initially to find the local dense units. Each unit has a local dense which is the number of mapped objects on it and a local relative density which is the proportion of the objects which mapped to the unit to the total number of objects in this chunk, in order to use these dense units later and perform clustering. PKS-Stream [44] is grid-based algorithm for clustering high-dimensional data streams, it uses a **PKS-tree for storing the nonempty grids** in the online phase and based on the PKS-tree, PKS-Stream performs clustering in the offline phase.

### 3.4 | Model-based algorithms

SWEM [20] is a model-based stream clustering algorithm which is based on the EM technique using the sliding window model. In each window period, the algorithm tries to find the  $k$  clusters each of which corresponds to a model that follows a multivariate normal distribution and it is characterized by a

parameter  $\Phi_h = \{\alpha_h, \mu_h, \Sigma_h\}$ , where  $\alpha_h$  is the cluster weight,  $\mu_h$  is its vector mean and  $\Sigma_h$  is its covariance matrix. The goal of the algorithm is to find the set  $\Phi_G = \{\Phi_1, \dots, \Phi_k\}$  that optimal fit the data objects for each window period in the  $k$  clusters.

In order to find the clusters the algorithm has 2 stages. In the first stage, SWEM computes  $m$  distributions which are also called micro components. Each data object belongs to all these components with different probability  $P(\Phi_l|x)$  and the log-likelihood measure is used to evaluate the selection of micro-components, so the algorithm applies the EM technique to maximize the log-likelihood form  $Q(\Phi)$ . In the second stage, SWEM decides when to split or merge the micro-components based on the variance between 2 consecutive time periods. Finally, the algorithm uses a fading function to discard the oldest data objects from the current window period.

SNCStream [11] is an online clustering algorithm capable of finding non-hyper-spherical clusters. SNCStream, in contrast to other data stream clustering algorithms, uses only 1-step processing to find clusters by using a social network generation and evolution model, which is based on homophily, it uses a scale-free-like homophily procedure to track the evolution of clusters during data streams. SNCStream<sup>+</sup> [12] is another high-quality real-time data stream clustering algorithm and an extension of SNCStream [11] algorithm. SNCStream<sup>+</sup> adapts the characteristics of the SNCStream algorithm and this method is more efficient as it executes in decreased complexity in the average case. pcStream [38] is a novel data stream clustering algorithm for dynamically detecting and managing sequential temporal contexts, it takes into account the properties of sensor-fused data streams in order to accurately infer the present concept, and dynamically detect new contexts as they occur. The algorithm is capable of detecting point anomalies and can operate with high-velocity data streams, it is applicable to any data stream with sequential temporal clusters that have unique correlated distributions. Another novel stream clustering method is the two-phase weightless neural system for data stream clustering (WCDS) [18]; this algorithm presents novel features where a mechanism based on the WiSARD artificial neural network model is applied.

#### 4 | COMPLEXITY ANALYSIS

In this section, we review the computational time complexity of the stream clustering algorithms which have been proposed in the literature and summarize the results in Table 2. Stream [25] is a single-pass algorithm which breaks the stream history into  $i$  batches of predefined size  $m$  in order to find the  $k$  medians of each batch combining a heuristic version of k-median algorithm and the Facility location algorithm. The algorithm starts by clustering each batch and then at a second level, the algorithm clusters the above points, this process is

repeated till the algorithm find the final  $k$  medians which takes  $O(i \cdot n \cdot k)$  to execute, where  $n$  is the number of the iterations of the k-median algorithm for the  $i$  batches. Similarly, Stream LSearch [41] breaks the stream history into  $i$  batches of predefined size  $m$ ; however, the algorithm uses the LSearch, a more complicated subroutine for the k-median algorithm in order to get an improved solution which starts with an initial solution with  $k'$  clusters, where  $k' < k$  that takes  $O(i \cdot n \cdot k')$  and overall the algorithm takes  $O(i \cdot n \cdot k \cdot \log k)$ .

CluStream [4] algorithm consists of a fast online phase which summarizes the stream and an offline phase which performs the clustering analysis. In the initialization phase, the first  $q$  microclusters are built after a certain number of  $N$  objects arrives from the stream, this process takes  $O(q \cdot N_{init} \cdot n \cdot i)$ , where  $q$  is the number of microclusters,  $N_{init}$  is the initial number of objects used for the creation of the first microclusters and  $i$  is the number of k-means iterations. In the online phase, through the microclusters, the algorithm maintains statistical summary information about the incoming objects, if the incoming object  $o$  is close enough to the centroid of an existing microcluster  $\mu_i$  and falls within its maximum boundary,  $o$  it is assigned into  $\mu_i$ , this process takes  $O(q \cdot n)$ . Because the numbers of microclusters is fixed and in order to accommodate the newly created microcluster, the old microclusters have to be reduced by 1; the process of merging 2 old microclusters takes  $O(q^2 \cdot n)$  and the process of discarding old microclusters takes  $O(q)$ . In the offline phase, the algorithm uses the summaries and performs a modification of k-means algorithm, this process takes  $O(q \cdot n \cdot k \cdot i)$ . Similarly, SWClustering [50] algorithm consists of 2 phases and uses a novel data structure called EHCF which captures the cluster evolution. The cost of inserting a new object into the nearest EHCF is  $O(h_n \cdot n)$ , where  $h_n$  is the total number of EHCFs. In the offline phase, the algorithm uses the EHCFs and performs a modification of k-means algorithm, this process takes  $O(h_n \cdot n \cdot k \cdot i)$ , where  $i$  is the number of the iterations of the k-means algorithm.

StreamKM++ [1] is a partitioning-based stream clustering algorithm in which the data objects are stored in buffers, each with a maximum capacity of  $m$  points. The algorithm uses a coresets tree that stores the data in a binary tree and using a merge-and-reduce technique maintains the  $L$  buffers in the tree. The cost of merging is  $O(m^2 \cdot n)$ , where  $m$  is the maximum number of data points each buffer can contain. In order to construct a coresets tree for the union of all  $L$  buffers of size  $m$  the merge-and-reduce technique of all buffers is executed in  $O(m^2 \cdot n)$  and in order to find the final  $k$  clusters, k-means++ algorithm is used which takes  $O(m \cdot k \cdot n \cdot i)$ , where  $i$  is the number of the iterations of the k-means++ algorithm. ClusTree [32] algorithm uses micro-clusters as a compact representation of the data and upon them builds a hierarchy in a balanced multidimensional tree indexing structure. The process of inserting an incoming object in the tree takes  $O(\log q)$  where  $q$  is the number of CFs, the total number of entries in a leaf node. The clustering result is a set

**TABLE 2** Time complexity of data stream clustering algorithms

Algorithm	Initialization	Insert/update	Merge	Discard	Clustering result
Stream [25]	—	—	—	—	$O(i \cdot n \cdot k)$
Stream LSearch [41]	$O(i \cdot n \cdot k')$	—	—	—	$O(i \cdot n \cdot k \cdot \log k)$
CluStream [4]	$O(q \cdot N_{init} \cdot n \cdot i)$	$O(q \cdot n)$	$O(q^2 \cdot n)$	$O(q)$	$O(q \cdot n \cdot k \cdot i)$
DUCStream [23]	—	—	—	—	$O(b)$
DenStream [17]	$O(N_{init} \cdot \log N_{init})$	$O(q \cdot n)$	$O(q \cdot n)$	$O(q \cdot n)$	$O(q \cdot \log q)$
D-Stream [19]	—	—	—	—	$O(p \cdot n)$
SWClustering [50]	—	$O(h_n \cdot n)$	$O(h_n^2 \cdot n)$	$O(h_n)$	$O(h_n \cdot n \cdot k \cdot v)$
DDStream [30]	—	—	—	—	$O(p^2)$
SDStream [43]	—	—	—	—	$O(d \cdot p)$
HDenStream [36]	—	—	—	—	$O(d \cdot p)$
FlockStream [21]	—	—	—	—	$O(d) + O(p)$
rDenStream [37]	$O(N_{init} \cdot \log N_{init})$	$O(q \cdot n)$	$O(q \cdot n)$	$O(q \cdot n)$	$O(q \cdot \log q + q_b)$
C-DenStream [46]	$O(N_{init} \cdot \log N_{init})$	$O(q \cdot n)$	$O(q \cdot n)$	$O(q \cdot n)$	$O(q \cdot \log q \cdot q_c)$
SWEM [20]	—	—	—	—	—
MR-Stream [34]	—	—	—	—	$O(2^g \cdot \lambda) + O(g \cdot \log N)$
PKS-Stream [44]	—	—	—	—	$O(d \cdot p)$
ClusTree [32]	—	$O(\log q)$	$O(m^2)$	—	—
StreamKM++ [1]	—	$O(m^2 \cdot n)$	$(m^2 \cdot n \cdot N)$	—	$O(m \cdot k \cdot n \cdot i)$
PreDeConStream [28]	—	—	—	—	$O(d \cdot p)$
DENGRIS [6]	—	—	—	—	$O(p \cdot n)$
HDDStream [40]	—	—	—	—	$O(d \cdot p)$
MuDi-Stream [7]	—	$O(r_{mc}) O(\log \log N)$	—	$O(r_{mc}) O(\log N)$	$O(r_{mc}) + O(mc) + O(\log \log N) + O(\log N)$
TS-Stream [42]	—	—	—	—	—
pcStream [38]	—	—	$O(m)$	—	$O(n)$
StreamXM [9]	—	—	—	—	$O(d \cdot n)$
SNCStream [11]	—	—	—	—	—
SNCStream <sup>+</sup> [12]	—	—	—	—	—
EDDS [5]	—	—	—	—	$O(q \cdot \log q)$
WCDS [18]	—	—	—	—	$O(\beta \cdot \delta \cdot d)$

of CFs which are stored at the leaf level. Based on this set of CFs any known clustering method such as  $k$ -means, DBSCAN, etc. can be applied taking the means of the CFs as representatives.

DenStream [17] algorithm follows the online-offline rationale, initial DBSCAN algorithm is applied to the first  $N$  objects from the stream in order to create the first  $q$  core-micro-clusters, this process takes  $O(N_{init} \cdot \log N_{init})$ . In the online phase of the algorithm, when a new object  $o_i$  arrives from the stream, the algorithm tries to accommodate it in a existing PMC which takes  $O(q \cdot n)$ . Also, periodically for each p-micro-cluster the algorithm checks if the p-micro-cluster is a potential o-micro-cluster which takes  $O(q \cdot n)$  or if an o-micro-cluster is a potential p-micro-cluster which also takes  $O(q \cdot n)$ . In the offline phase each micro-cluster is treated as a virtual point and a variant of DBSCAN algorithm is applied upon these virtual points, this process takes  $O(q \cdot \log q)$ .

C-DenStream [46] is an extension of DenStream algorithm for clustering with constraints for data streams. In C-DenStream, instance-level constraints are used also as PMCs and the clusters are generated based on these PMCs using the C-DBSCAN algorithm, the time complexity

is similar to DenStream algorithm; however, for the final clusters in contrast with DenStream, C-DenStream takes  $O(q \cdot \log q \cdot q_c)$ , where  $q_c$  are the instance-level constraints as PMCs. Another variant of DenStream algorithm is the rDenStream [37] algorithm, which extends DenStream adding an extra phase called retrospect where the discarded objects are placed in a outlier buffer and have a new chance to be added in the clustering result.

D-Stream [19] is a grid-based stream clustering algorithm that uses a 2-phase framework which consists of an online phase that processes data objects from the stream and maps each input data object into a grid and an offline phase which computes the grid density and clusters the grids based on the density. The time-complexity of D-Stream [19] is  $O(p \cdot n)$  and is based exclusively on the number of partitions of the grid, at each iteration of the algorithm there are  $p^n$  grid cells where  $p$  is the number of partitions in each dimension. Similarly, DD-Stream [30] follows the D-stream rationale; however, in the offline phase uses an algorithm called DCQ-means to detect the border points of the grids and add them in the clustering analysis in order to improve clustering quality, the time complexity of the algorithm is  $O(p^2)$ , where  $p$  is the number

of partitions of the grid. DENGRIS [6] is another grid-based stream clustering algorithm which is similar to D-Stream but adopts a sliding window model for focusing on the most recent data objects, the time complexity of DENGRIS is  $O(p \cdot n)$ , where  $p$  is the number of partitions.

DUCStream [23] is an incremental single-pass clustering algorithm which breaks the stream history in chunks each with its data points. The algorithm partitions the data space into units each with its density, the density of each unit is the number of points on it. If the density of a unit is higher than a user-specified threshold, it is considered as a dense unit and based on these dense units the algorithm performs clustering. The algorithm keeps the clustering results in clustering bits which are strings that keep the number of dense units, due to the bit clustering model the time complexity of DUCStream is  $O(b)$ , where  $b$  is the clustering bits.

MuDi-Stream [7] is an online-offline framework with 4 main components that combines CMCs with a hybrid method which uses a density-based method to generate the final clusters and a grid-based method to handle outliers. In the online phase, the algorithm keeps summary information about the evolving multidensity data stream in the form of CMCs and based on these CMCs in the offline phase the algorithm generates the final clusters, also the algorithm maintains a grid list in the form of tree with the grids that are under consideration for clustering. When a new data object arrives it can be merged to an existing core-micro-cluster, which takes  $O(r_{mc})$ , where  $mc$  is the number of core-micro-clusters or it can be mapped to the grid list which takes  $O(\log \log N)$ , where  $N$  is the height of the tree. The cost of discarding a CMC is  $O(mc)$  and the cost of discarding a grid from the grid list is  $O(\log N)$ . The overall time complexity of MuDi-Stream is  $O(r_{mc}) + O(mc) + O(\log \log N) + O(\log N)$ .

HDDStream [36] is a density-based stream clustering algorithm for high-dimensional data. The algorithm in the online phase keeps information about the dimensions of the stream in the form of CMCs and in the offline phase generates the final clusters using a projected clustering algorithm called PreDeCon [16], the time complexity of HDDStream [36] is  $O(d \cdot p)$ . Similarly, PreDeConStream [28] algorithm, is based on the 2-phase model of stream clustering algorithms and generates the final clusters using the PreDeCon [16], the time complexity of PreDeConStream [28] is  $O(d \cdot p)$ . HDenStream [36] is another density-based stream clustering algorithm for high-dimensional data, the time complexity of the algorithm is  $O(d \cdot p)$ .

## 5 | EXPERIMENTAL SETUP

### 5.1 | Evaluation measures

In this section, we describe the measures used in our evaluation of the stream clustering algorithms.

#### 5.1.1 | Sum of squared distance

The most commonly used criterion to evaluate the similarity within a cluster in partitioning based algorithms is the sum of squared (SSQ):

$$SSQ = \sum_{k=1}^k \sum_{x_i \in D_k} \|x_i - \mu_k\|^2$$

$D_k$  is the set of objects in cluster  $k$  and  $\mu_k$  is its centroid. It measures how far the objects of each cluster are from the center of its cluster, small SSQ means better compactness of each cluster.

#### 5.1.2 | Purity

Purity is the most commonly used external criterion for clustering evaluation.

$$\text{Purity}(\Omega, C) = \frac{1}{N} \sum_{j=1}^k \max_j |\omega_k \cap c_j|$$

To compute purity initially for all  $k$  clusters, each cluster  $c_j$  is assigned to the class  $\omega_k$  that has the max count in this cluster and measure the correctly assigned objects of this class in each cluster, then dividing this total by  $N$  which denotes the total number of objects.

#### 5.1.3 | Clustering mapping measure

Data Streams are evolving in their nature, thus, the corresponding patterns extracted upon such data evolve over time. This characteristic leads to new kind of errors and thus to new kind of faults which may occur in stream clustering evaluation in contrast to the evaluation of conventional static data sets such as (1) misplaced objects, (2) missed objects, and (3) noisy objects. The well-known evaluation measures developed for evaluation static data sets, either interior or exterior, cannot handle these kinds of faults. In order to evaluate the stream clustering algorithms properly, new evaluation measures which must take into account the above faults must be developed. A novel evaluation measure for stream clustering called Cluster Mapping Measure (CMM) [33] deals with these faults. CMM is a normalized sum of the penalties that occur because of the above faults and it is defined as:

$$\text{CMM}(C, CL) = 1 - \frac{\sum_{o \in F} w(o) \cdot \text{pen}(o, C)}{\sum_{o \in F} w(o) \cdot \text{con}(o, CL(o))}$$

If no fault occurs CMM is equal to 1. In order to compute CMM 2 important properties have to be computed initially, the *connectivity* “how well the point is connected to the cluster” and the *cluster mapping* “how well the point is assigned to the clusters given truth classes”. The connectivity of a point  $p$  to a cluster  $C_i$  is defined as:

$$\text{con}(p, C_i) = \begin{cases} 1, & \text{Knhdist}(p, C_i) < \text{Knhdist}(C_i) \\ 0, & C_i = \emptyset \\ \frac{\text{Knhdist}(C_i)}{\text{Knhdist}(p, C_i)}, & \text{else} \end{cases}$$



**TABLE 3** Relation of density-based algorithms

Algorithm	Added characteristic
DenStream [17]	—
SDStream [43]	DenStream + sliding window
HDenStream [36]	DenStream + categorical data
FlockStream [21]	DenStream + bioinspired model
rDenStream [37]	DenStream + retrospect phase
C-DenStream [46]	DenStream + constraints
PreDenConS [28]	DenStream + high-dim. Data
HDDStream [40]	DenStream + high-dim. Data
MuDiStream [7]	DenStream + hybrid grid model

**TABLE 4** Relation of grid-based algorithms

Algorithm	Added characteristic
DUCstream [23]	Single-pass chunk-based model
D-Stream [19]	—
DDStream [30]	D-Stream + boundary detection
MR-Stream [34]	D-Stream + improved offline
PKS-Stream [44]	D-Stream + high-dim. Data
DENGRIS [6]	D-Stream + sliding window

where 0 means no connectivity and 1 indicates a strong connectivity. To decide whether a data object is misplaced a mapping from the clusters returned by a stream algorithm to ground truth classes is used to penalized the faults and to compute an overall penalty.

## 5.2 | Algorithms

In our experimental evaluation we use CluStream [4], DenStream [17], and ClusTree [32] algorithms for 2 reasons. First, all algorithms which have been proposed in the literature afterward are variations or extensions of these algorithms. Also, we wanted to focus on general purpose algorithms that satisfy the aforementioned challenges and to show the advantages and disadvantages of each category. For example, all density-based algorithms, except DUCStream [23] which is a single-pass density-based algorithm with many limitations, are based on DenStream [17] algorithm and extend it for a particular application. For instance, C-DenStream [46] is an extension of DenStream [17] algorithm in clustering with constraints, and it is a proper algorithm in clustering geographic information system (GIS) data streams because it takes the constraints a map can have into account (Tables 3 and 4 show the relation between density- and grid-based algorithms, respectively). Similarly, Clustream [4], which introduced the online-offline rationale and the micro-clusters, technique was selected because it is a partitioning-based general purpose algorithm in clustering data streams and satisfies many of the challenges in contrast with Stream framework [25,41] which is a single-pass partitioning-based algorithm with many limitations. (Table 5 shows the relation between

**TABLE 5** Relation of partitioning-based algorithms

Algorithm	Added characteristic
Stream [25]	Single-pass batch algorithm
Stream LSearch [41]	Single-pass batch algorithm
Clustream [4]	Micro-clusters
SWclustering [50]	Exponential Histogram
StreamKM++ [1]	Coresets

**TABLE 6** Overview of the data sets

Dataset	Instances	Attrib.	Num.	Classes
Covertime	581.012	54	10	7
Poker-Hand	829.201	10	5	10
Electricity	45.312	8	7	2
Adult-Census	32.541	14	6	2

partitioning-based algorithms). Finally, we added the ClusTree [32] algorithm in the experimental evaluation because it is the only anytime stream clustering algorithm and we wanted to show where this method outperforms the others.

## 5.3 | Data sets

We experimented with 4 real-world data sets namely Forecast Cover Type, Poker Hand, Electricity, and Adult data set (Table 6) from different domains in order to avoid domain bias. We chose real-world multidimensional data sets which depict temporal evolution and therefore call for stream clustering solutions. We selected data sets from different domains to evaluate the performance of the algorithms on different problems that also contain the true class labels which are needed for the evaluation of the algorithms.

### 5.3.1 | Forecast

The study of Blackard and Dean [15] contains 581.012 observations and 54 variables out of which 10 are numeric, from 4 areas located in Roosevelt National Forest of northern Colorado. The challenge of this data set is to predict the cover type using only cartographic variables, these areas are a result of ecological process and not a from human-caused. The actual type was determined from US Forest Service and the others variables form a US Geological survey. We used a normalized version of this data set.

### 5.3.2 | Poker-Hand

The study of Lichman [35] consists of over of 800.000 instances and 10 attributes out of which 5 are numeric. Each record of the Poker-Hand data set is an example of a hand consisting of 5 playing cards drawn from a standard deck of 52. Each card is described using 2 attributes (suit and rank), for a total of 10 predictive attributes. There is 1 class attribute that describes the “Poker Hand”.

### 5.3.3 | Electricity

The study of Lichman [35] contains 45,312 observations from Australian New South Wales Electricity Market. The electricity price in this market is not fixed and is affected by demand and supply of the market, the class label identifies the change of the price relative to a moving average of the last 24 h, which can be moved up or down. We used a normalized version of this data set.

### 5.3.4 | Adult

This data set is a part of the well-known census data set [35]. It consists of over of 32,000 instances out of which 6 is numeric, it has 2 classes which determine whether a person makes over 50 K a year or not.

## 5.4 | More experimental settings

We implemented all the evaluated algorithms in MOA [13,14], an open-source benchmarking software for evolving data streams that is built on the work of WEKA. It is a java-based software package that contains state-of-the-art algorithms and evaluations measures for running experiments. Recently, an R package called Steam [27] published that allows to perform clustering experiments, the main advantage of stream is that it seamlessly integrates with the large existing infrastructure provided by R.

Also, in order to clear any ambiguity between the terms window size  $w$  and horizon  $H$ , window size  $w$  defines the number of data objects which arrive in each time period and horizon means the number of windows in which the clustering analysis is evaluated; we have set for the entire evaluation analysis that horizon.  $H = 1$ .

## 6 | EXPERIMENTAL RESULTS

We selected the 3 most representative algorithms to experimentally evaluate, these algorithms are the general stream clustering algorithms and all algorithms which have proposed afterward are variations or extensions of these algorithms. All the data sets and the experimental results of our evaluation are available at <http://smansalis.me/EDSCpaper/>

### 6.1 | Sensitivity analysis

The goal of sensitivity analysis is to find the values or the range of values for the parameters of the algorithms where they achieve the best quality of clustering and have good performance.

#### 6.1.1 | Clustream

One important decision about setting Clustream algorithm parameters is the selection of the proper number of

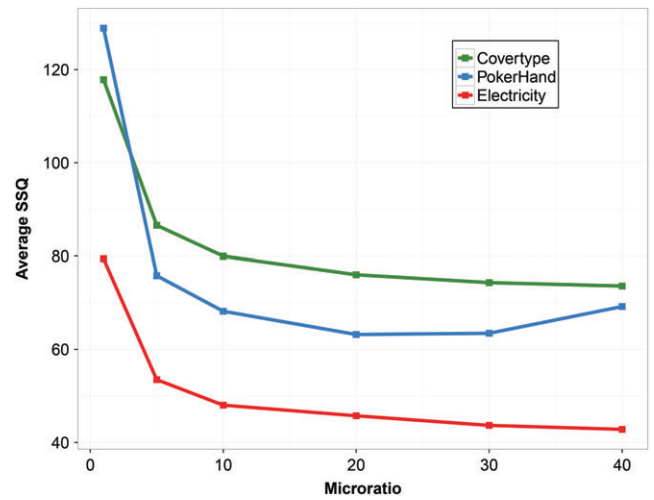


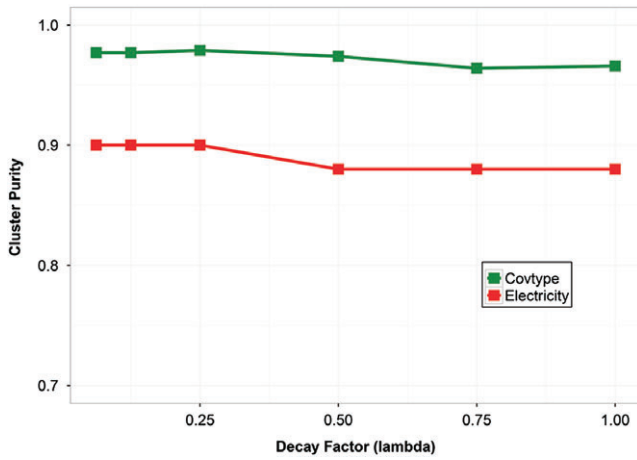
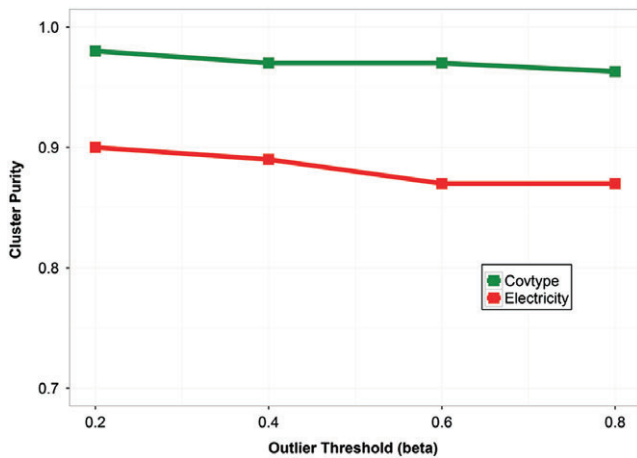
FIGURE 2 Clustream: Effect of the number of micro-clusters in accuracy

micro-clusters. In Section 2.3 where we present Clustream, we mention that the number of micro-clusters should be larger than the number of macro-clusters (actual clusters); however, a very large number of micro-clusters is time and memory consuming. In order to make a right decision about the number of micro-clusters as in Aggarwal et al. [4] we also set micro-ratio, as the number of micro-clusters divided by the number of macro-clusters (natural clusters).

Then, using 3 real-world data sets examine the clustering quality with sum of square distance (SSQ) as criterion. We set, window size  $w = 1000$  for the coverttype data set, for the pokerhand data set  $w = 1000$  and for electricity data set  $w = 500$ ; we also set micro-ratios values to 1, 5, 10, 20, 30, and 40; this means that when the micro-ratio is about 10 for the coverttype data set the actual number of micro-clusters is 70, for pokerhand data set 100 and 20 for the electricity data set, respectively. Figure 2 shows our experimental results; we can see that if micro-ratio is equal to 1, which means that the number of micro-clusters is equal to the number of natural clusters, the clustering quality is poor, also using very small number of micro-clusters we do not utilize the real purpose of micro-clusters to keep synopsis information. When the micro-ratio increases the average SSQ reduces until it becomes stable, the average SSQ for each of the 3 real-world data sets becomes stable when the micro-ratio is about 15; notice in pokerhand data set if we keep increasing the micro-ratio the average SSQ start increasing which is also time- and memory consuming. This means to achieve high-quality clustering and keep the amount of memory small in order to avoid memory consuming the micro-ratio must not be too small neither too large, a good choice is about 15 to 20 micro-cluster per actual cluster.

#### 6.1.2 | Denstream

Two important user-specified parameters of DenStream algorithm are the outlier threshold  $\beta$  and the decay factor  $\lambda$  which controls the importance of historical objects. We test

FIGURE 3 Denstream: Clustering quality vs decay factor  $\lambda$ FIGURE 4 Denstream: Clustering quality vs outlier threshold  $\beta$ 

the clustering quality for both parameters by varying them using covtype and electricity datasets; for both of the experiments we set  $w = 1000$  for the Covtype data set and  $w = 500$  for Electricity data set. Also, we set Initial Points = 1000 and 500 for Covtype and Electricity data set, respectively. Finally, we set  $\mu = 1$ ,  $\epsilon = 0.02$  and the processing speed at 100 for DenStream execution.

Figure 3 shows the average clustering quality of DenStream when  $\lambda$  varies from 0.0625 to 1. The higher the value of  $\lambda$  the lower the importance of the historical objects. When  $\lambda$  is relative small ( $\lambda = 0.25$ ) the algorithm for both data sets reaches the best average clustering purity, although it discards the old objects sooner and only a small amount of the total objects is included in the clustering result than when  $\lambda = 1$  where the amount of the total objects which is included in the result is bigger. Similarly, Figure 4 shows the average clustering quality of DenStream when  $\beta$  varies from 0.2 to 0.8. When  $\beta$  is about 0.2 for both data sets the average purity is very good, as  $\beta$  increases the average purity slightly decreases; also note that very high value of  $\beta$  (ie,  $\beta = 1$ ) means faster pruning of the PMC.

In summary, among the 3 state-of-the-art algorithms that were evaluated, CluStream achieves good clustering quality

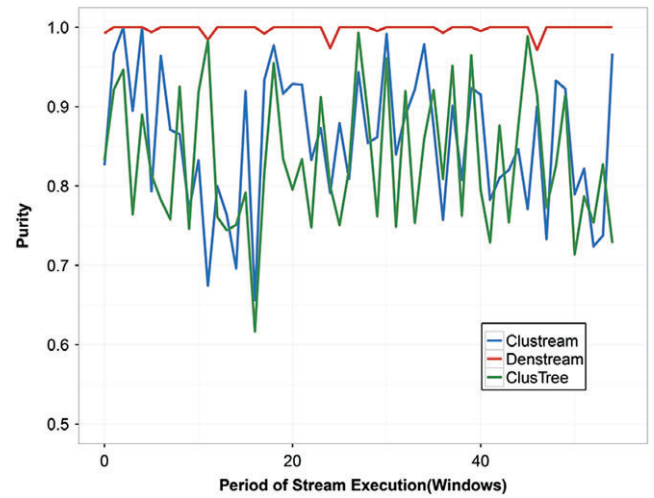


FIGURE 5 Clustering quality using Pokerhand data set

for a number of micro-clusters about 20 times the number of final clusters, for DenStream algorithm as the decay factor  $\lambda$  and the outlier threshold  $\beta$  increases, the clustering quality decreases.

For the rest of the entire experimental evaluation we set the parameters of DenStream as follows: decay factor  $\lambda = 0.25$ ,  $\mu = 1$ ,  $\epsilon = 0.02$ , processing speed equal to 1000 and  $\beta = 0.2$ ;

## 6.2 | Cluster quality

### 6.2.1 | Purity

In this section, we evaluate the accuracy of the 3 state-of-the-art stream clustering algorithms using all data sets in order to answer the crucial question: which is the algorithm that has the best clustering results.

We begin by evaluating the algorithms using Pokerhand data set; Figure 5 shows the comparison of the algorithms for the whole period of stream execution. It can be seen that DenStream clearly outperforms both Clustream and ClusTree and the purity of DenStream is always above 0.9 and reaches 1.0 (100%) while that of ClusTree and Clustream ranges in the interval [0.7, 0.9].

In order to better understand what this means in conjunction with Pokerhand data set, in Figure 6 we illustrate the purity that each algorithm has in contrast with the number of classes in each clustering result. It is clear that Clustream and ClusTree algorithm are affected by the number of classes; the more classes are included in clustering result: the less the purity for these algorithm while DenStream is hardly affected.

Next, we compare the algorithms on Covtype data set. Figure 7 shows the comparison of the algorithms for the whole period of stream execution; like with Pokerhand data set DenStream algorithm reaches better clustering purity in contrast with Clustream and ClusTree. However, this time there are clustering results where Clustream and ClusTree reach DenStream purity. Similarly, in Figure 8 we display what this means in conjunction with Covtype data set, in this data set ClusTree is slightly affected by the number of

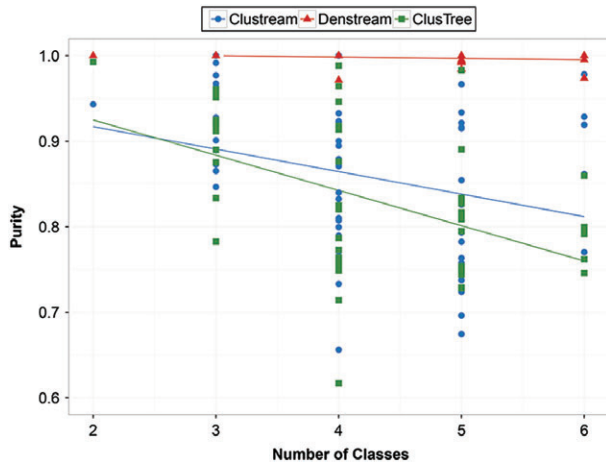


FIGURE 6 Clustering quality vs number of classes using Pokerhand data set

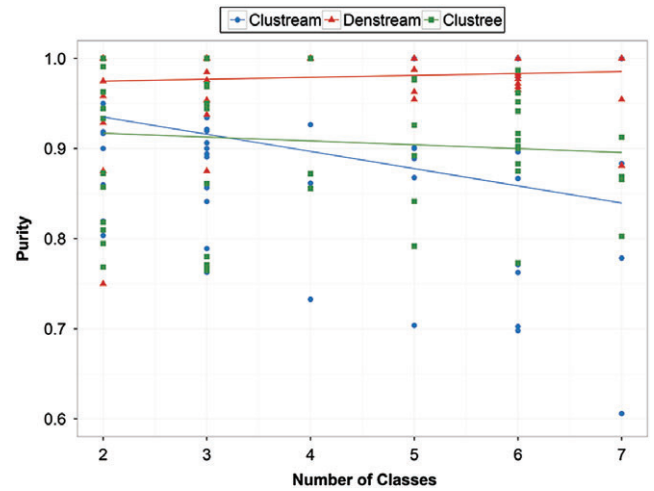


FIGURE 8 Clustering quality vs number of classes using covertype data set

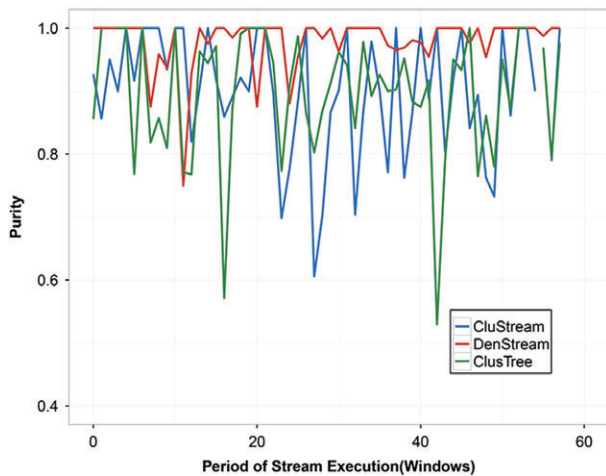


FIGURE 7 Clustering quality using covertype data set

classes while Clustream is much less affected and DenStream is hardly affected.

We continue by evaluating the algorithms using Electricity data set. Figure 9 shows the comparison of the algorithms for the whole period of stream execution. Clearly DenStream has better purity than Clustream and ClusTree, which have similar purity. In order to better understand what this means in conjunction with Electricity data set, we show in Figure 10 the instances from stream execution aligned with clustering results, (as we have already described Electricity data set consist of 2 classes). Figure 10 shows from which of the 2 classes the instances in each window come from. In the whole execution the majority of instances come from “Down” class; however, notice when the number of 2 class becomes similar (about in 20th window) Clustream and ClusTree reach their worst purity.

Lastly, we evaluate the algorithms using Adult data set. Figure 11 shows the comparison of the algorithms for the whole period of stream execution. As in previous experiments, DenStream clearly outperforms Clustream and ClusTree, the purity of DenStream is always above Clustream and ClusTree and sometimes reaches 1.0 (100%).

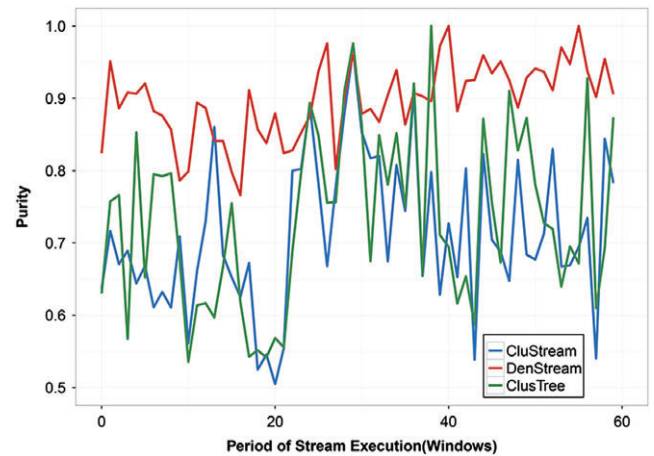


FIGURE 9 Clustering quality using electricity data set

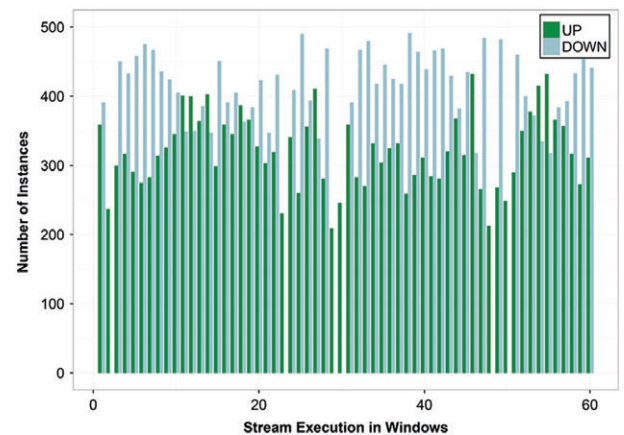


FIGURE 10 Instances from stream execution using electricity data set

Like before, in order to understand what this means in conjunction with Adult data set in Figure 12 we display the instances from stream execution aligned with clustering results. Adult data set consists of 2 classes, however, for the entire execution the number of instances came from each class for each window is about similar, so we cannot make assumptions about how much and when the purity is affected by the stream execution.



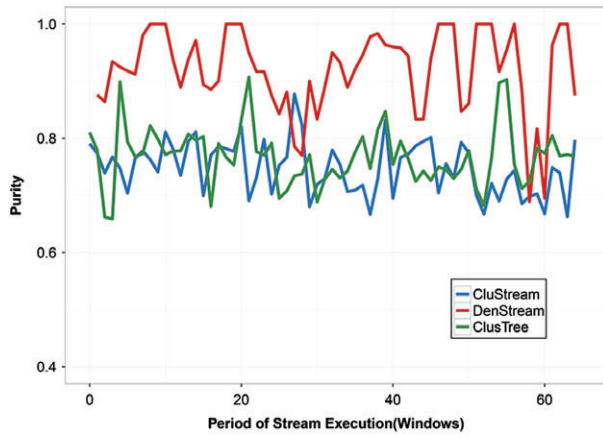


FIGURE 11 Clustering quality using adult data set

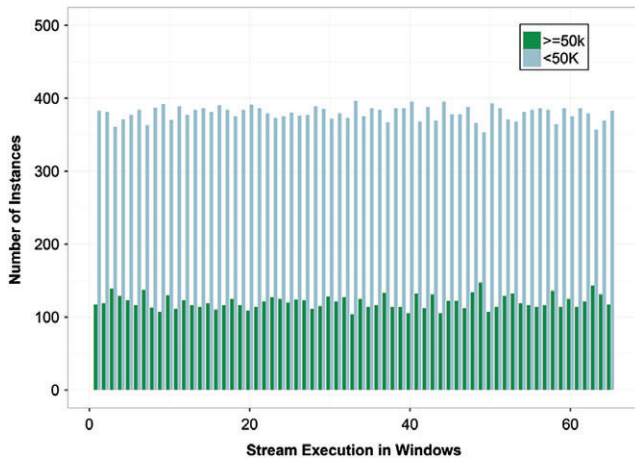


FIGURE 12 Instances from stream execution using adult data set

Another important factor from stream clustering execution is the selection of the proper size of window, which defines the number of data objects which arrive in each time period. A large size of window may be memory consuming but provides the flexibility to analyze bigger parts of the stream history in contrast with a small window, also with a large size of window we need fewer data mining models.

We examine in Figures 13 and 14, using Covertypes and Pokerhand data sets, respectively, how much the average quality is affected by varying the window size from  $w = 200$  to  $w = 3000$  for horizon  $H = 1$ . Figure 13 shows that for Clustream and ClusTree algorithm the clustering quality is affected at all slightly while DenStream algorithm is not affected. Similarly, Figure 14 shows that for Clustream and ClusTree algorithm the clustering quality is affected slightly, in particular for Clustream algorithm the bigger the size of window it is, the better the clustering purity while for ClusTree the smaller the size of window it is, the better the clustering purity, as in previous data set DenStream algorithm is not affected by varying window size.

### 6.2.2 | CMM

We have also evaluated the algorithms using CMM [33] evaluation measure. In Figure 15, we display the result for

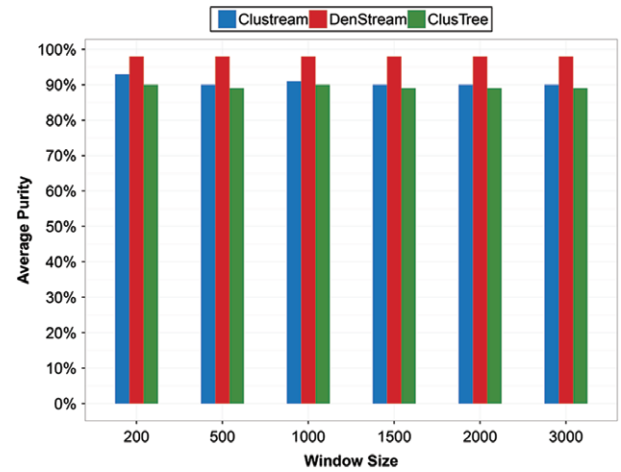


FIGURE 13 Clustering quality for different window sizes using covertype data set

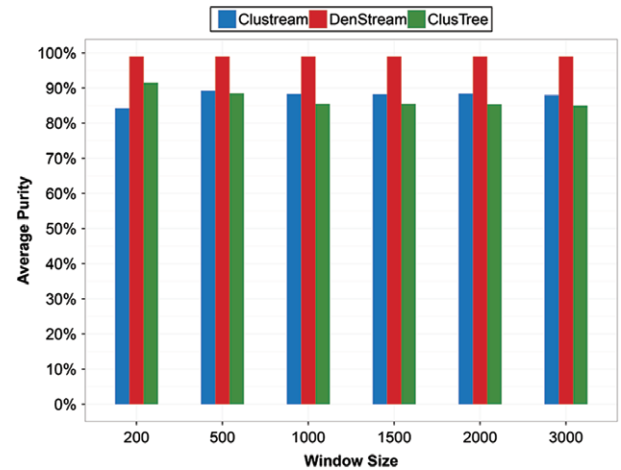


FIGURE 14 Clustering quality for different window sizes using Pokerhand data set

Pokerhand data set where the average CMM score for both Clustream and ClusTree is better than the average CMM score for DenStream algorithm, we can see that the total amount of error caused by missed points, as  $CMM_{missed}$  shows. Similarly, for Covertypes data set in Figure 16, for Electricity data set in Figure 17 and for Adult data set in Figure 18 the average CMM for both Clustream and ClusTree is better than the average CMM for DenStream and the primary effect of decreasing the CMM is from  $CMM_{missed}$ .

### 6.3 | Summary of findings

In this paper, we conducted an extensive survey of the the stream clustering algorithms which have been proposed in the literature and an extensive experimental evaluation of the state-of-the-art stream clustering algorithms. Our experiments are conducted on 4 real-world data sets, the findings can be summarized as follows:

1. Regarding the window model, damped, and tilted windows models turned out to be better for stream clustering algorithms due to the fact that they provide the flexibility to

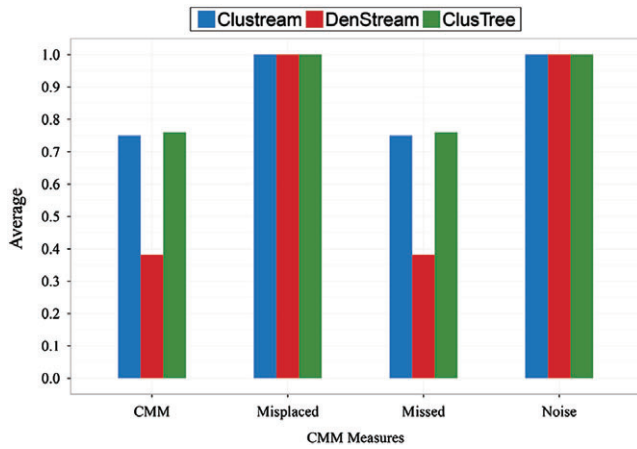


FIGURE 15 CMM evaluation using Pokerhand data set

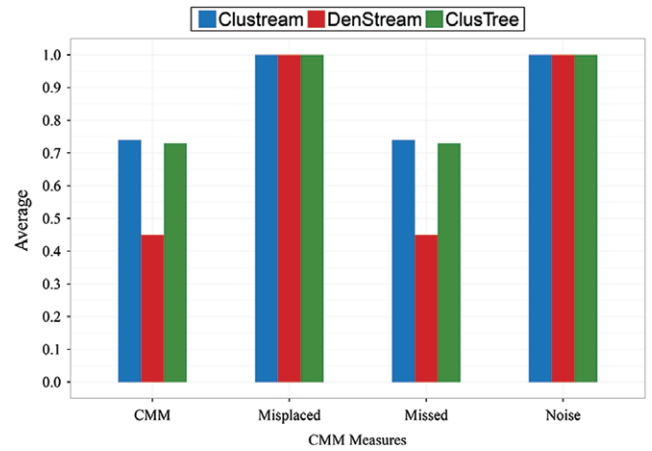


FIGURE 18 CMM evaluation using adult data set

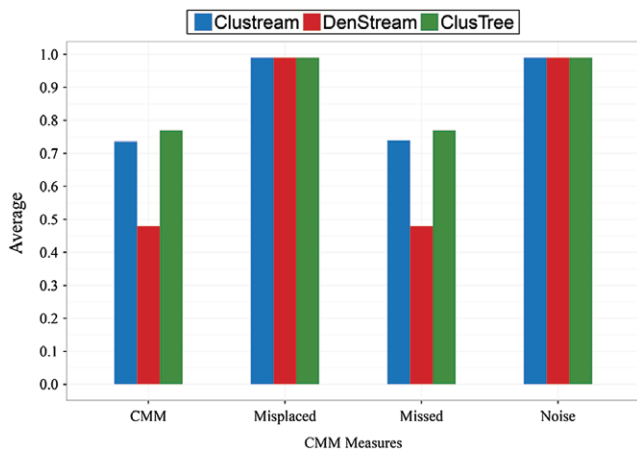


FIGURE 16 CMM evaluation using covtype data set

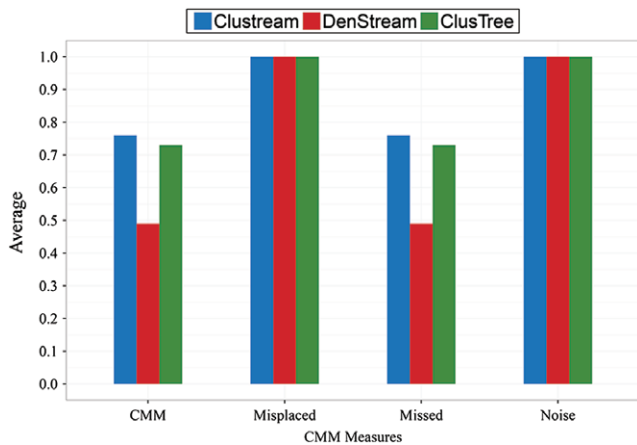


FIGURE 17 CMM evaluation using electricity data set

give more importance to recent objects and do not discard the old ones entirely. Also, tiled window model ensures that the total amount of data which is saved is relative small.

- Regarding data processing, the 2-phase algorithms are by far better than single-pass algorithms because they can capture the evolution of streams which is crucial for clustering data streams.

- Regarding the clustering paradigm, density- and grid-based algorithms turn out to be the best choices for stream clustering, because they do not need require the number of clusters as input and also because they can handle the outliers.
- Regarding the summary structure, CF-based summaries for stream clustering like microclusters and core-micro-clusters are the most efficient choice since they provide some important properties such as the additive property in which 2 CMCs can be combined together, the incremental property in which once a new object arrived simple can be added to an existing CMCs, and the ability to handle outliers.

The characteristics of each algorithm are summarized in Table 7. Among the state-of-the-art algorithms that were evaluated, CluStream achieves good clustering quality for a number of micro-ratio of 20. For DenStream as the decay factor  $\lambda$  and the outlier threshold  $\beta$  increase, the clustering quality decreases. DenStream achieves better clustering quality comparing to Clustream and ClusTree for every window size and during the whole stream execution. The number of classes of the data set (which indirectly reflects the number of clusters) affects Clustream and ClusTree—the more diverse in terms of classes a data set is, the worse the achieved clustering quality, whereas DenStream is more robust. Clustream and ClusTree appear to have better CMM compared to DenStream and the total amount of error caused by missed points.

## 7 | CONCLUSIONS, OPEN ISSUES, AND RESEARCH DIRECTIONS

Clustering is one of the most important tasks in data mining, used both as a stand-alone tool to get insights into the data distribution and as a preprocessing step for other algorithms. Clustering is a challenging task even in the batch scenario where all instances are available and multiple passes over the data are allowed. Clustering over streams is much more challenging as the underlying data distribution might evolve over

**TABLE 7** The challenges that satisfy each surveyed algorithm

<b>Clustering Algorithm</b>	<b>Handling Evolving data</b>	<b>No assumption Number of clusters</b>	<b>Limited Memory</b>	<b>Limited Time</b>	<b>Handling Outliers</b>	<b>Handling High dim.</b>
Stream [25]	—	—	✓	✓	—	—
Stream LSearch [41]	—	—	✓	✓	—	—
CluStream [4]	✓	—	✓	✓	—	—
DUCStream [23]	—	✓	✓	✓	—	—
DenStream [17]	✓	✓	✓	✓	✓	—
DStream [19]	✓	✓	✓	—	✓	—
SWClustering [50]	✓	—	✓	✓	—	—
DDStream [30]	✓	✓	—	—	✓	—
SDStream [43]	✓	✓	✓	✓	✓	—
HDenStream [36]	✓	✓	✓	✓	✓	✓
FlockStream [21]	—	✓	✓	✓	✓	—
rDenStream [37]	✓	✓	—	—	✓	—
C-DenStream [46]	✓	✓	—	—	✓	—
SWEM [20]	✓	—	✓	✓	✓	—
MR-Stream [34]	✓	✓	—	—	✓	—
PKS-Stream [44]	✓	✓	✓	✓	✓	✓
ClusTree [32]	✓	—	✓	✓	—	—
StreamKM++ [1]	—	—	✓	✓	—	—
PreDeConStream [28]	✓	✓	✓	✓	✓	✓
DENGRIS [6]	✓	✓	—	—	✓	—
HDDStream [40]	✓	✓	✓	✓	✓	✓
MuDi-Stream [7]	✓	✓	✓	✓	✓	✓
TS-Stream [42]	✓	✓	✓	✓	✓	—
pcStream [38]	✓	✓	✓	✓	✓	—
StreamXM [9]	✓	✓	✓	✓	✓	—
SNCStream [11]	✓	✓	✓	✓	—	—
SNCStream <sup>+</sup> [12]	✓	✓	✓	✓	—	—
EDDS [5]	✓	✓	✓	✓	✓	—
WCDS [18]	✓	✓	✓	✓	✓	—

time and therefore the number of clusters in the stream might vary and their generative distributions might be subject to drifts and shifts over time.

Comparing to batch clustering, the amount of work in stream clustering is much less. This is because of the difficulty of the task and also because of the recent attention on data streams. Although the first clustering algorithms showed up in the beginning of 2000, only recently with the big data discussion the community (both academia and research) realized the need for dealing with data streams.

There are a lot of directions for future work in the stream clustering area. First, exploiting the recent developments with MapReduce and distributed computing: this would allow to speed up the algorithms, but also to deal with the bounded memory issue. In particular, since storage is cheap nowadays we could allow for the storage of representative objects (like centroids or summaries) over time. Storing these intermediate results will allow the end user to focus on specific time periods and further analyze the summarized data through, for example, macroclustering or by applying other data mining tasks.

Another important direction is that of effective summarization over the stream. Thus far, microclusters and their variations are the most common summaries for stream clustering. However, as already mentioned, microclusters create convex-shaped clusters and in reality the clusters might come at any shape. For batch clustering, there are more elaborated structures like data-bubbles, but their “adaptation” to the data stream scenario is not that straightforward. Finally, mining data series has a tremendous growth of interest in today’s world, thus, clustering data streams series is becoming a hot research area due to the huge volume of data series that are produced daily from diverse domains (finance, scientific, biology, etc.). Several data series streams clustering algorithms like Ts-stream [42] have been proposed in the literature.

Evaluation is another important direction. Typically clustering algorithms are evaluated through external evaluation measures like class-labeled data; however, clustering is different from classification and a mapping between classes and clusters is not always feasible. Internal evaluation measures like SSQ are also available, however, they are not appropriate

for all cluster types. For example, SSQ is appropriate for spherical clusters like those produced by  $k$ -means but not for arbitrary-shaped clusters like those produced by DBSCAN.

Dealing with noise and outliers is another direction for future research. The models thus far follow a blind-adaptation approach by incorporating each new instance to the existing model without verifying whether it is an outlier or noise. This results in destroying valid cluster structures, which might be recreated later on; however, an always changing clustering structure is not the best for the end user as it requires a lot of effort for understanding and reacting/taking actions.

In a similar direction, understandability of clustering results is an important aspect, especially in a stream environment where data flow in and out of the system at a high rate and clustering structures are frequently updated. Toward this goal, one could exploit the evolution of the individual clusters and relations between clusters (eg, merge, absorption etc) [48].

## ORCID

**Stratos Mansalis**  <http://orcid.org/0000-0002-0184-207X>

## REFERENCES

- Marcel R. Ackermann et al., Streamkmm+: A clustering algorithm for data streams, *ACM J. Exp. Algorithmics* 17 (2012), no. 4, 2.4:2.1–2.4:2.30.
- Charu C. Aggarwal, *Data Streams: Models and Algorithms (Advances in Database Systems)*, Springer-Verlag, New York, 2006.
- Charu C. Aggarwal, A survey of stream clustering algorithms, in *Data Clustering: Algorithms and Applications*, C. Aggarwal and C. Reddy, Eds., CRC Press, Siena, Italy, 2013, 361–366.
- C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, A framework for clustering evolving data streams, *Proc. of the 29th Int. Conf. on Very Large Data Bases, VLDB '03*, vol. 29, 2003.
- A. Al Abd Alazeez, S. Jassim, and H. Du, *Edds: An enhanced density-based method for clustering data streams*, 2017 46th Int. Conf. on Parallel Process., Bristol, UK, 2017.
- A. Amini and T. Y. Wah, A density-grid based clustering algorithm for evolving data streams over sliding window, *Int. Conf. Data Mining Comput. Eng.*, Las Vegas, Nevada, USA, 2012.
- A. Amini, H. Saboohi, T. Herawan, and T. Y. Wah, *Mudi-stream: A multi density clustering algorithm for evolving data stream*, 2013 IEEE 13th Int. Conf. on Data Mining Workshops (ICDMW), Shenzhen, China, 2014a.
- Amineh Amini, Ying Wah Teh, and Hadi Saboohi, On density-based data streams clustering algorithms: A survey, *J. Comput. Sci. Technol.* 29 (2014b), no. 1, 116–141.
- R. Anderson and Y. S. Koh, *StreamXM: An adaptive partitioned clustering solution for evolving data streams*, Big Data Analytics and Knowledge Discovery, Valencia, Spain, 2015, pp. 270–282.
- D. Arthur and S. Vassilvitskii, *K-means++: The advantages of careful seeding*, *Proc. Eighteenth Annu. ACM-SIAM Symp. Discrete Algorithms*, New Orleans, Louisiana, 2007.
- J. P. Barddal, H. M. Gomes, and F. Enembreck, *Sncstream: A social network-based data stream clustering algorithm*, *Proc. 30th Annu. ACM Symp. Appl. Comput., SAC '15*, New York, NY, USA, 2015, 935–940.
- Jean Paul Barddal et al., Sncstream+, *Inf. Syst.* 62 (2016), no. C, 60–73.
- Albert Bifet et al., Moa: Massive online analysis, *J. Mach. Learn. Res.* 11 (2010), no. 8, 617–620.
- A. Bifet, G. Holmes, B. Pfahringer, J. Read, P. Kranen, Hardy Kremer, T. Jansen, and T. Seidl, *Moa: A real-time analytics open source framework*, *Proc. 2011 Eur. Conf. Mach. Learn. Knowl. Discov. Databases, ECML PKDD '11*, Athens, Greece, vol. Part III, 2011.
- J. A. Blackard and D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, *Comput. Electron. Agric.* 24 (1999), 131–151.
- C. Bohm, K. Kailing, H.-P. Kriegel, and P. Kroger, *Density connected clustering with local subspace preferences*, *Proc. Fourth IEEE Int. Conf. Data Mining*, Washington, DC, 2004, 27–34.
- F. Cao, M. Ester, W. Qian, and A. Zhou, *Density-based clustering over an evolving data stream with noise*, *Proc. Sixth SIAM Conf. Data Mining*, 2006.
- Douglas O. Cardoso, Felipe M. G. França, and João Gama, Wcds: A two-phase weightless neural system for data stream clustering, *New Gener. Comput.* 35, San Diego, CA, USA, (2017), no. 4, 391–416.
- Y. Chen and Li Tu, *Density-based clustering for real-time stream data*, *Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, San Jose, CA, 2007, 133–142.
- X. H. Dang, V. Lee, W. K. Ng, A. Ciptadi, and K. L. Ong, *An EM-based algorithm for clustering data streams in sliding windows*, *Proc. 14th Int. Conf. Database Syst. Adv. Appl.*, 2012.
- A. Forestiero, C. Pizzuti, and G. Spezzano, *Flockstream: A bio-inspired algorithm for clustering evolving data streams*, *Proc. 21st IEEE Int. Conf. Tools Artif. Intell.*, Newark, New Jersey, USA, 2009.
- J. Gama and M. Gaber, Eds., *Learning from Data Streams*, Springer, 2007.
- J. Gao, J. Li, Z. Zhang, and P.-N. Tan, *An incremental data stream clustering algorithm based on dense units detection*, *Proc. 9th Pacific-Asia Conf. Adv. Knowl. Discov. Data Mining*, Hanoi, Vietnam, 2005.
- S. Gong, Y. Zhang, and G. Yu, *Clustering stream data by exploring the evolution of density mountain*, *ArXiv*, 2017.
- S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, *Clustering data streams*, *Proc. 41st Annu. Symp. Foundations Comput. Sci., IEEE Computer Society*, CA, USA, 2000.
- Sudipto Guha et al., Clustering data streams: Theory and practice, *IEEE Trans. Knowl. Data Eng.* 15 (2009), no. 3, 515–528.
- Michael Hahsler, Matthew Bolaos, and John Forrest, Introduction to stream : An extensible framework for data stream clustering research with R, *J. Stat. Softw.* 76 (2017), 1–28.
- M. Hassani, P. Spaus, M. M. Gaber, and T. Seidl, *Density-based projected clustering of data streams*, *Proc. 6th Int. Conf. Scalable Uncertainty Manage.*, Marburg, Germany, 2012.
- A. K. Jain, M. N. Murty, and P. J. Flynn, Data clustering: A review, *ACM Comput. Surv.* 31 (1999), no. 3, 264–323.
- C. Jia, C. Tan, and A. Yong, *A grid and density-based clustering algorithm for processing data stream*, *Conf. Genet. Evol. Comput.*, 2008.
- James Kennedy and Russell C. Eberhart, *Swarm Intelligence*, Morgan Kaufmann Publishers Inc, 2001.
- Philipp Kranen et al., The clustree: Indexing micro-clusters for anytime stream mining, *Knowl. Inf. Syst.* 29 (2011), no. 2, 249–272.
- H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer, *An effective evaluation measure for clustering on evolving data streams*, *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, San Diego, CA, USA, 2011.
- Wan Li et al., Density-based clustering of data streams at multiple resolutions, *ACM Trans. Knowl. Discov. Data* 3 (2009), no. 3, 1–28.
- M. Lichman, *UCI Machine Learning Repository*, Univ. of California, Irvine, CA, 2013.
- J. Lin and H. Lin, *A density-based clustering over evolving heterogeneous data stream*, *ISECS Int. Colloquium on Comput. Commun. Control Manage.* 2009, CCCM 2009, 2009.
- L. Li-xiong, H. Hai, G. Yun-fei, and C. Fu-Cai, *rDenstream, a clustering algorithm over an evolving data stream*, *Proc. Int. Conf. Inf. Eng. Comput. Sci., ICIECS*, Wuhan, China, 2009.
- Y. Mirsky, B. Shapira, L. Rokach, and Y. Elovici, *pcstream: A stream clustering algorithm for dynamically detecting and managing temporal contexts*, *Adv. Knowl. Discov. Data Mining*, Auckland, New Zealand, 2015.
- Hai-Long Nguyen, Yew-Kwong Woon, and Wee Keong Ng, A survey on data stream clustering and classification, *Knowl. Inf. Sys.* 45 (2014), 535–569.
- I. Ntoutos, A. Zimek, T. Palpanas, P. Kröger, and H.-P. Kriegel, *Density-based projected clustering over high dimensional data streams*, *Proc. 12th SIAM Int. Conf. Data Mining*, Anaheim, CA, USA, 2012.
- L. O'Callaghan, N. Mishra, A. Meyreson, S. Guha, and R. Motwani, *Streaming-data algorithms for high-quality clustering*, *Proc. 18th Int. Conf. Data Eng.*, 2002.
- C. M. M. Pereira and R. F. de Mello, Ts-stream: Clustering time series on data streams, *J. Intell. Inf. Syst.* 42 (2014), no. 3, 531–566.



43. J. Ren and R. Ma, *Density-based data streams clustering over sliding windows*, Proc. Sixth Int. Conf. Fuzzy Syst. Knowl. Discov., Tianjin, China, 2009.
44. Jiadong Ren, Binlei Cai, and Changzhen Hu, Clustering over data streams based on grid density and index tree, *J. Conver. Inf. Technol.* 6 (2011), no. 1, 1–8.
45. C. Ruiz, M. Spiliopoulou, and E. Menasalvas, *C-dbscan: Density-based clustering with constraints*, Proc. 11th Int. Conf. Rough Sets, Fuzzy Sets, Data Mining and Granular Computing, Toronto, Canada, 2007.
46. C. Ruiz, E. Menasalvas, and M. Spiliopoulou, *C-denstream: Using domain knowledge on a data stream*, Proc. Int. Conf. Inf. Eng. Comput. Sci., ICIECS, Wuhan, China, 2009.
47. Jonathan A. Silva et al., Data stream clustering: A survey, *ACM Comput. Surv.* 46 (2013), no. 1, 1–31.
48. M. Spiliopoulou, I. Ntoutsis, Y. Theodoridis, and R. Schult, *Monic: modeling and monitoring cluster transitions*, Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, ACM, 2006, 706–711.
49. T. Zhang, R. Ramakrishnan, and M. Livny, *Birch: An efficient data clustering method for very large databases*, Proc. ACM SIGMOD Int. Conf. Manage. Data, 1996.
50. Aoying Zhou et al., Tracking clusters in evolving data streams over sliding windows, *Knowl. Inf. Sys.* 15 (2008), no. 2, 181–214.

**How to cite this article:** Mansalis S, Ntoutsis E, Pelekis N, Theodoridis Y. An evaluation of data stream clustering algorithms. *Stat Anal Data Min: The ASA Data Sci Journal*. 2018;11:167–187. <https://doi.org/10.1002/sam.11380>