Big Data Analytics

REVIEW

Open Access

State-of-the-art on clustering data streams

Mohammed Ghesmoune*, Mustapha Lebbah and Hanene Azzag

*Correspondence:
ghesmoune@lipn.univ-paris13.fr
University of Paris 13, Sorbonne Paris City, LIPN-UMR 7030 - CNRS, 99, av. J-B Clément, F-93430 Villetaneuse, France

Abstract

Clustering is a key data mining task. This is the problem of partitioning a set of observations into clusters such that the intra-cluster observations are similar and the inter-cluster observations are dissimilar. The traditional set-up where a static dataset is available in its entirety for random access is not applicable as we do not have the entire dataset at the launch of the learning, the data continue to arrive at a rapid rate, we can not access the data randomly, and we can make only one or at most a small number of passes on the data in order to generate the clustering results. These types of data are referred to as data streams. The data stream clustering problem requires a process capable of partitioning observations continuously while taking into account restrictions of memory and time. In the literature of data stream clustering methods, a large number of algorithms use a two-phase scheme which consists of an online component that processes data stream points and produces summary statistics, and an offline component that uses the summary data to generate the clusters. An alternative class is capable of generating the final clusters without the need of an offline phase. This paper presents a comprehensive survey of the data stream clustering methods and an overview of the most well-known streaming platforms which implement clustering.

Keywords: Data stream clustering, Streaming platforms, State-of-the-art

Background

In today's applications, evolving data streams are ubiquitous. Indeed, examples of applications relevant to streaming data are becoming more numerous and more important, including network intrusion detection, transaction streams, phone records, web click-streams, social streams, weather monitoring, etc. There is active research on how to store, query, analyze, extract and predict relevant information from data streams. Clustering is a key data mining task. This is the problem of partitioning a set of observations into clusters such that the intra-cluster observations are similar (or close) and the inter-cluster observations are dissimilar (or distant). The other objective of clustering is to reduce the complexity of the data by replacing a group of observations (cluster) with a representative observation (prototype).

In this paper, we consider the problem of clustering data in the form of a stream, i.e. a sequence of potentially infinite, non-stationary data (the probability distribution of the unknown data generation process may change over time) arriving continuously (which requires a single pass through the data) where random access to the data is not feasible and storing all the arriving data is impractical. When applying data mining techniques, and specifically clustering algorithms, to data streams, restrictions in execution time and memory have to be considered carefully. To deal with time and memory restrictions, many of the existing data stream clustering algorithms modify traditional non-streaming methods to use the two-phase framework proposed in [1] to deal with streaming data, e.g., DenStream [2] is an extension of DBSCAN algorithm, StreamKM++ [3] of *k*-means++, StrAP [4] of AP, etc.

Real-time processing means that the ongoing data processing requires a very low response delay. The *velocity*, which refers to that Big Data are generated at high speed (speed of data in and out), is an important concept in the Big Data domain [5]. With the increasing importance of data stream mining applications, many streaming platforms have been proposed. These can be classified in two categories: traditional or non-distributed such as MOA [6] and distributed streaming platforms such as Spark Streaming [7] and Flink [8]. The latter two, may be considered as the most widely used streaming platforms. These distributed streaming systems are based on two processing models, *record-at-a-time* and *micro-batching*. On a *record-at-a-time* processing model, long-running stateful operators process records as they arrive, update the internal state, and send out new records. On the other hand, the *micro-batching* processing model runs each streaming computation as a series of deterministic batch computations on small time intervals, which is implemented in Spark Streaming [7].

General surveys have been recently published in the literature for mining data streams [9–13]. The authors of [14] introduced a taxonomy to classify data stream clustering algorithms. The work presented in [15] is a thorough survey of state-of-the-art density-based clustering algorithms over data streams. This paper presents a thorough survey of the state-of-the-art for a wide range of data stream clustering algorithms and an overview of the most well-known streaming platforms. The remainder of this paper is organized as follows. Section "Data stream clustering methods" presents in a comprehensive manner the most relevant works on data stream clustering algorithms. These algorithms are categorized according to the nature of their underlying clustering approach. Section "Streaming platforms" overviews the most well-known streaming platforms with a focus on the streaming clustering task. Section "Conclusion" concludes this paper.

Data stream clustering methods

This section discusses previous works on data stream clustering problems, and highlights the most relevant algorithms proposed in the literature to deal with this problem. Most of the existing algorithms (e.g. *CluStream* [1], *DenStream* [2], *StreamKM++* [3], or *ClusTree* [16]) divide the clustering process in two phases: (*a*) *Online*, the data will be summarized; (*b*) *Offline*, the final clusters will be generated. Figure 1 is a flowchart of the data stream clustering algorithms presented in this paper. These algorithms are categorized according to the nature of their underlying clustering approach.

GNG based algorithms

Growing Neural Gas

Growing Neural Gas (GNG) [17] is an incremental self-organizing approach which belongs to the family of topological maps such as Self-Organizing Maps (SOM) [18] or Neural Gas (NG) [19]. It is an unsupervised clustering algorithm capable of representing a high dimensional input space in a low dimensional feature map. Typically, it is used for

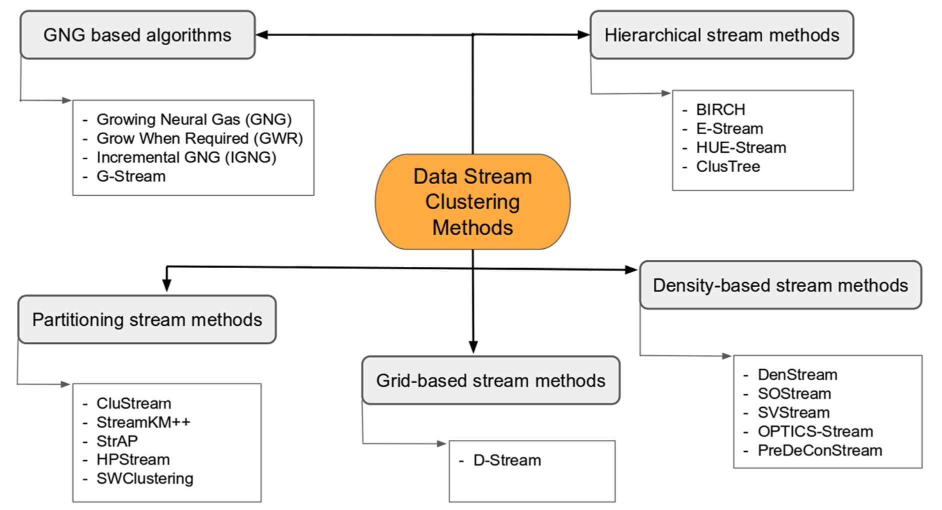

Fig. 1 Data stream clustering methods. The presented algorithms categorized according to the nature of their underlying clustering approach

finding topological structures that closely reflect the structure of the input distribution. Therefore, it is used for *visualization tasks* in a number of domains [19, 20] as neurons (nodes), which represent prototypes, are easy to understand and interpret.

The GNG algorithm constructs a graph of nodes in which each node has its associated prototype. Prototypes can be regarded as positions in the input space of their corresponding nodes. Pairs of nodes are connected by edges (links), which are not weighted. The purpose of these links is to define the topological structure. These links are temporal in the sense that they are subject to aging during the iteration steps of the algorithm and are removed when they become "too old" [20].

Starting with two nodes, and as a new data point is available, the nearest and the second-nearest nodes are identified, linked by an edge, and the nearest node and its topological neighbors are moved toward the data point. Each node has an accumulated error variable. Periodically, a node is inserted into the graph between the nodes with the largest error values. Nodes can also be removed if they are identified as being superfluous. This is an advantage compared to SOM and NG, as there is no need to fix the graph size in advance. Algorithm 1 outlines an online version of the GNG approach. In this version, unlike the standard approach of GNG, the data is seen only once.

A number of authors have proposed variations on the Growing Neural Gas (GNG) approach. The GNG algorithm creates a new node every λ iterations (λ is fixed by the user as an input parameter). Hence, it is not adapted for data streams, or non-stationary datasets, or to novelty detection. In order to deal with non-stationary datasets, the author of [21] has investigated modifying the network by proposing an on-line criterion for identifying "useless" nodes. The algorithm proposed is known as the Growing Neural Gas with Utility (GNGU). Slow changes of the distribution are handled by adaptation of existing nodes, whereas rapid changes are handled by removal of "useless" neurons and subsequent insertions of new nodes in other places.

The authors of [22] modified GNG to detect incrementally emerging cluster structures. The proposed GNGC algorithm is able to match the temporal distribution of the original

```
Algorithm 1: GNG online
  Data: $\mathcal{DS} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$
  Result: set of nodes $\mathcal{C} = \{c_1, c_2, \ldots\}$ and their prototypes $\mathbf{W} = \{\mathbf{w}_{c_1}, \mathbf{w}_{c_2}, \ldots\}$
1 Initialize node set $\mathcal{C}$ to contain two nodes, $c_1$ and $c_2$: $\mathcal{C} = \{c_1, c_2\}$;
2 while there is a data point to proceed do
3     Get the next data point in the data stream, $\mathbf{x}_i$;
4     Find the nearest node $bmu_1$ and the second nearest node $bmu_2$;
5     Update edges as described in Algorithm 2;
6     if the number of data points passed is an integer multiple of a parameter $\beta$ then
7         Insert a new node as described in Algorithm 3;
8     Delete each isolated node;
9     Finally, decrease the error of all units;
```

```
Algorithm 2: Edge Management
1 Increment the age of all edges emanating from $bmu_1$ and weight them;
2 if $bmu_1$ and $bmu_2$ are connected by an edge then
3     set the age of this edge to zero
4 else
5     create an edge between $bmu_1$ and $bmu_2$, and mark its time stamp;
6 Remove edges whose age is greater than $age_{max}$;
```

```
Algorithm 3: Node Insertion
1 Find node $q$ with the maximum accumulated error;
2 Find the neighbor $f$ of $q$ with the largest accumulated error;
3 Add the new node, $r$, half-way between nodes $q$ and $f$: $\mathbf{w}_r = 0.5(\mathbf{w}_q + \mathbf{w}_f)$;
4 Insert edges connecting the new node $r$ with nodes $q$ and $f$, and remove the original
  edge between $q$ and $f$;
```

dataset by creating a new node whenever the received new data point is too far from its nearest node. It is noted that the algorithm is computationally demanding.

The clustering method proposed in [23] consists of two steps. In the first step, the data are prepared by generating the Voronoi partition using a modified GNG algorithm (which does not exceed linear complexity). The result is that the number of intermediate clusters is much smaller than the number of original objects. In the second step the intermediate clusters are clustered using conventional algorithms that have a much higher computational complexity (for this reason they should not be used for clustering the full volume of initial data). The approach examines hierarchical clustering of GNG units using single linkage and Ward's method as linkage criteria. Although the clustering results look promising, the approach has the drawback that they have to manually identify the "right" level in the cluster hierarchy to obtain an adequate clustering of the input space.

GWR

The 'Grow When Required' (GWR) network [24] may add a new node at any time, whose position is dependent on the input and the current winning node. The GWR deals with the problem of novelty detection by adding new nodes into the network structure whenever the activity of the current best-matching node is below some threshold, which implies that the best-matching node is not trained to deal with that particular input. This means that the network grows very quickly when new data is presented, but stops growing once the network has matched the data to a given accuracy. This has benefits in that there is no need to decide in advance how large the network should be, as nodes will be added until the network is saturated. This means that for small datasets the complexity of the network is significantly reduced. In addition, if the dataset changes at some time in the future, further nodes can be added to represent the new data without disturbing the network that has already been created [24, 25]. Considering one iteration of the GWR algorithm, GWR has approximatively the same time complexity as one iteration of GNG. Hence, the complexity of GWR is $O(knm)$ where k is the number of iterations, n is the number of data points of the data stream m is the number of nodes in the graph. For more details on the complexity of GNG the reader is referred to [26].

IGNG

Still in the same idea of relaxing the constraint of periodical evolution of the network, the IGNG [27] algorithm has been proposed. In this algorithm a new neuron is created each time the distance of the current input data to the existing neuron is greater than a predefined fixed threshold σ, which is dependent on the global datasets. However, one disadvantage of this algorithm is the global character of the parameter σ and also that it must be computed prior to the learning. In order to resolve this weakness, I2GNG [28] associates a threshold variable σ to each neuron. However, its major drawback is the initialization of the σ values at the creation of each node. The authors of [29] address the problem of choosing the final winner neuron among the many input equidistant neurons. They proposed some adaptations of the IGNG and I2GNG algorithms. Notably, the use of a labeling maximization approach as a clustering similarity measure (IGNG-F) to replace the distance in the winner selection process.

The ability of self-organizing neural network models to manage real-time applications, using a modified learning algorithm for a growing neural gas network is addressed in [30]. The proposed modification aims to satisfy real-time temporal constraints in the adaptation of the network. The proposed learning algorithm can add a dynamic number of neurons per iteration. Indeed, a detailed study has been conducted to estimate the optimal parameters that keep a good quality of representation in the available time. The authors concluded that the use of a large number of neurons made it difficult to obtain a representation of the distribution of training data with good accuracy in real-time [30, 31].

AING [32] is an incremental GNG that learns automatically the distance thresholds of nodes based on its neighbors and data points assigned to the node of interest. It merges nodes when their number reaches a given *upper-bound*.

G-Stream

More recently, *G-Stream* [33, 34] was proposed as a data stream clustering approach based on the Growing Neural Gas algorithm. G-Stream uses a stochastic approach to update the prototypes, and it was implemented on a "centralized" platform, which can be summarized as follows: starting with two nodes, and as a new data point is reached, the nearest and the second-nearest nodes are identified, linked by an edge, and the nearest node with its topological neighbors are moved toward the data point. Each node has an accumulated error variable and a weight, which varies over time using a fading function. Using an edge management procedure, one, two or three nodes are inserted into the graph between the nodes with the largest error values. Nodes can also be removed if they are identified as being superfluous.

G-Stream can discover clusters of arbitrary shape in an evolving data stream, whose main features and advantages are:

(i) the topological structure is represented by a graph wherein each node represents a cluster, which is a set of "close" data points, and neighboring nodes (clusters) are connected by edges. The graph size is not fixed but may evolve;

(ii) to reduce the impact of old data whose relevance diminishes over time, G-Stream uses an exponential fading function

$$f(t) = 2^{-\lambda_1(t-t_0)}$$

where $\lambda_1 > 0$, defines the rate of decay of the weight over time, t denotes the current time and t_0 is the timestamp of the data point. The weight of a node is based on data points associated with it:

$$weight(c) = \sum_{i=1}^{m} 2^{-\lambda_1(t-t_{i_0})}$$

where m is the number of points assigned to the node c at the current time t. If the weight of a node is less than a threshold value then this node is considered as outdated and then deleted (with its links). For the same reason, links between nodes are also weighted by an exponential function;

(iii) unlike many other data stream algorithms that start by taking a significant number of data points for initializing the model (these data points can be seen several times), G-Stream starts with only two nodes. Several nodes (clusters) are created in each iteration, unlike the traditional Growing Neural Gas (GNG) [17] algorithm;

(iv) all aspects of G-Stream (including creation, deletion and fading of nodes, edges management, and reservoir management) are performed online;

(v) a reservoir is used to hold, temporarily, the very distant data points, compared to the current prototypes.

However, the design of a "distributed" version of G-Stream would raise difficulties, which are resolved by *MBG-Stream* [35]. This later operates with parameters to control the decay (or "forgetfulness") of the estimates. The MBG-Stream algorithm is implemented on a distributed streaming platform based on the *micro-batching* processing model, i.e., the Spark Streaming API[1]. In the proposed algorithm, the topological structure is represented by a graph wherein each node represents a cluster, which is a set of "close" data points and neighboring nodes (clusters) are connected by edges. Starting with only two nodes, the graph size is not fixed but may also evolve as several nodes (clusters) are created in each iteration. We use an exponential fading function to reduce the impact of old data whose relevance diminishes over time. For the same reason, links between nodes are also weighted by an exponential function. The data received in each interval is stored reliably across the cluster to form an input dataset for that interval. Once the time interval is completed, this dataset is processed via deterministic parallel operations, such as *Map* and *Reduce* to produce new datasets representing either program outputs or intermediate states [7]. The input data is split and the master assigns the splits to the Map workers. Each worker processes the corresponding input split, generates key/value pairs and writes them to intermediate files (on disk or in memory). The Reduce function is responsible for aggregating information received from the Map functions. The algorithm uses a generalization of the mini-batch GNG update rule, where the nearest node and all of its neighbors are moved in the direction of the data point. However, in MBG-Stream, for each batch of data $\mathbf{X}_p$, we assign all points $\mathbf{x}_i$ to their best match unit, compute new cluster centers, then update each cluster. The update rule (the adaptation step) in a mini-batch version without taking into account the neighbors of the referent is described in Eq. 1 as:

$$\mathbf{w}_c^{(t+1)} = \frac{\mathbf{w}_c^{(t)} n_c^{(t)} \alpha + \mathbf{z}_c^{(t)} m_c^{(t)}}{n_c^{(t)} \alpha + m_c^{(t)}} \tag{1}$$

whereas Eq. 2 updates the number of points assigned to the cluster,

$$n_c^{(t+1)} = n_c^{(t)} + m_c^{(t)} \tag{2}$$

where $\mathbf{w}_c^{(t)}$ is the previous center for the cluster, $n_c^{(t)}$ is the number of points assigned to the cluster thus far, $\mathbf{z}_c^{(t)}$ is the new cluster center from the current batch, and $m_c^{(t)}$ is the number of points added to the cluster c in the current batch.

Hierarchical stream methods

A hierarchical clustering method groups the given data into a tree of clusters which is useful for data summarization and visualization. This is a binary-tree based data structure called the *dendrogram*. Once the dendrogram is constructed, one can automatically choose the right number of clusters by splitting the tree at different levels to obtain different clustering solutions for the same dataset without rerunning the clustering algorithm again. Hierarchical clustering can be achieved in two different ways, namely, bottom-up and top-down clustering. Though both of these approaches utilize the concept of dendrogram while clustering the data, they might yield entirely different sets of results depending on the criterion used during the clustering process [36]. In hierarchical clustering once a step (merge or split) is done, it can never be undone. Methods for improving the quality of hierarchical clustering have been proposed such as integrating hierarchical clustering with other clustering techniques, resulting in multiple-phase clustering such as BIRCH [37].

BIRCH

Balanced Iterative Reducing and Clustering using Hierarchies (*BIRCH*) incrementally and dynamically clusters multi-dimensional data points to try to produce the best quality clustering with the available resources (i. e., memory and time constraints) by making a single scan of the data, and to improve the quality further with a few additional scans. It should be noted that the BIRCH method is not designed for clustering data streams and cannot address the concept drift problem. The key characteristic of the BIRCH is to introduce a new data structure called a *clustering feature* (CF) as well as a CF-tree. The CF can be regarded as a concise summary of each cluster. This is motivated by the fact that not every data point is equally important for clustering and we cannot afford to keep every data point in the main memory given that the overall memory is limited. On the other hand, for the purpose of clustering, it is often enough to keep up to the second order of data moment. In other words, CF is not only efficient, but also sufficient to cluster the entire data set [36, 37].

More precisely, a CF structure is a triple (N, LS, SS), where N is the number of data points in the cluster, LS is the linear sum of the N data points, and SS is the squared sum of the N data points. The CF vector has two main properties giving the incremental aspect, in an intuitive way, to any algorithm that uses this structure:

- **Incrementality** If a point x is added to the cluster, the sufficient statistics are updated as follows:

$$N_i \leftarrow N_i + 1;$$
$$LS_i \leftarrow LS_i + x;$$
$$SS_i \leftarrow SS_i + x^2;$$

- **Additivity** If $CF_1 = (N_1, LS_1, SS_1)$ and $CF_2 = (N_2, LS_2, SS_2)$ are the CF vectors of two disjoint clusters, merging them is equal to the sum of their parts. The additive property allows us to merge sub-clusters incrementally without accessing the original data set.

$$CF_1 + CF_2 = (N_1 + N_2, LS_1 + LS_2, SS_1 + SS_2).$$

Figure 2 presents the CF-tree structure in BIRCH. The CF-tree is a height-balanced tree which keeps track of the hierarchical clustering structure for the entire data set.

BIRCH requires two user defined parameters: B the branch factor or the maximum number of entries in each non-leaf node; and T the maximum diameter (or radius) of any CF in a leaf node. The maximum diameter T defines the examples that can be absorbed by a CF. Increasing T, more examples can be absorbed by a CF node and smaller CF-Trees are generated. Each node in the CF-tree represents a cluster which is in turn made up of at most B sub-clusters. All the leaf nodes are chained together for the purpose of efficient scanning.

When a data point is available, it traverses down the current tree from the root, until it finds the appropriate leaf following the *closest*-CF path, with respect to the L_1 or L_2 norms. The insertion on the CF-tree can be performed in a similar way as the insertion in the classic B-tree. If the closest-CF in the leaf cannot absorb the data point, a new CF entry is created. If there is no room for new leaf, the parent node is split. A leaf node might be expanded due to the constraints imposed by B and T. The process consists of taking the two farthest CFs and creates two new leaf nodes. BIRCH operates in two main steps: the first step builds an initial CF-tree in memory using the given amount of memory and recycling space on disk; the second step tries to cluster all the sub-clusters in the leaf nodes, called also the "global clustering". There are two optional steps: the"tree condensing" step which aims to refine the initial CF-tree by re-inserting its leaf entries; and the "clustering refinement" step which re-assigns all the data points based on the cluster centroid produced by the global clustering step.

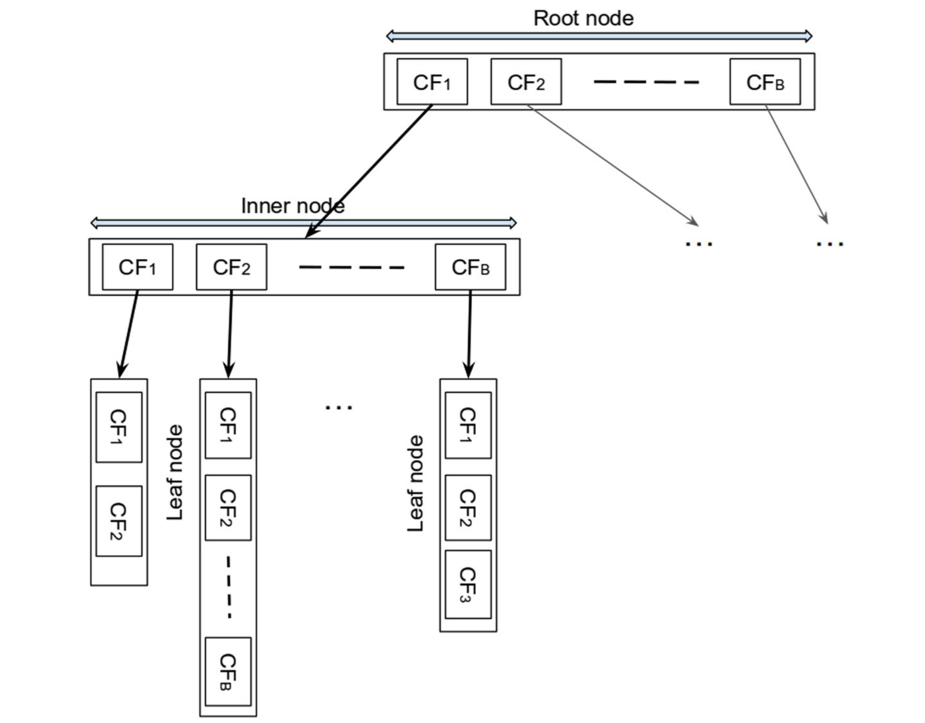

Fig. 2 The clustering feature tree in BIRCH. *B* is the maximum number of CFs in a level of the tree

E-Stream

E-Stream [38] classifies the evolution of data into five categories: appearance, disappearance, self evolution, merge, and split. This algorithm is an *evolution-based stream clustering* method, i.e., a stream clustering method that supports the monitoring and the change detection of clustering structures. It uses another data structure for saving summary statistics, named the α-bin histogram. Indeed, each cluster is represented as a *Fading Cluster Structure* (FCS) utilizing an α-bin histogram for each feature of the dataset. A histogram of the cluster data values is utilized to identify cluster splits. The range of each bin is calculated as the difference between the maximum and minimum feature values divided by α. When the maximum or minimum value changes, a new range is calculated and the values in each range are updated from the intersection between the new and old ranges. Each cluster has a histogram of feature values, but the histogram is utilized only for the split of active clusters. Only an active cluster can assemble an incoming data point. If a statistically significant valley is found between two peaks in any of the marginal histograms, the cluster is split. Figure 3 illustrates the histogram management in a split.

E-Stream starts empty, and every new point either is mapped onto one of the existing clusters (based on a radius threshold) or a new cluster is created around it. Any cluster not meeting a predefined density level is considered inactive and remains isolated until achieving a desired weight. The weight of a cluster is the number of data elements assigned to this cluster. The algorithm employs an exponential decay function to weigh down the influence of older data, thus focuses on keeping an up-to-date view of the data distribution. Clusters which are not active for a certain time period may be deleted from the data space.

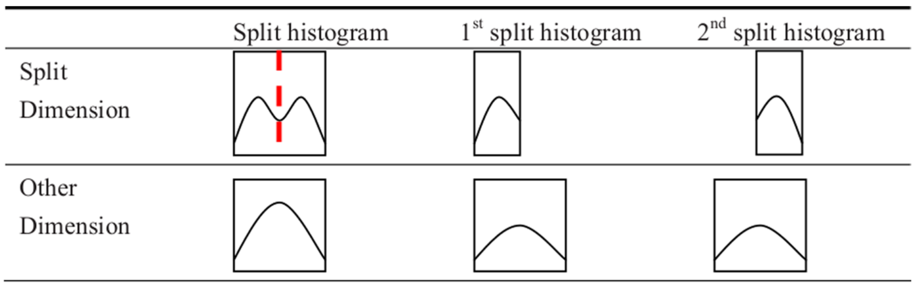

Fig. 3 Histogram management in a split dimensionop and other dimension [38]

HUE-Stream

HUE-Stream [39] which extends E-Stream in order to support uncertainty in heterogeneous data, i.e., including numerical and categorical attributes simultaneously. Uncertain data streams pose a special challenge because of the dual complexity of high volume and data uncertainty. This uncertainty is due to errors in the reading of sensors or other hardware collection technology. In many of these cases, the data errors can be approximated either in terms of statistical parameters, such as the standard deviation, or the probability density functions [9]. The Uncertain MICROclustering (*UMicro*) algorithm is proposed as a method for clustering uncertain data streams, which enhances the micro-clusters with additional information about the uncertainty of the data points in the clusters [40]. This information is used to improve the quality of the distance functions for the cluster assignments. HCluStream [41] extends the definition of the cluster feature vector to include categorical features, replaces the modified *k*-means clustering with the corresponding *k*-prototypes clustering which is able to handle heterogeneous attributes. The centroid of continuous attributes and the histogram of the discrete attributes are used to represent the micro-clusters, and the *k*-prototype clustering algorithm is used to create the micro-clusters and macro-clusters.

The distance function, cluster representation and histogram management are introduced for the different types of clustering structure evolution. A *distance function* between the probability distributions of two objects is introduced to support uncertainty in categorical attributes. To detect changes in the clustering structure, the proposed distance function is used to merge clusters and find the closest cluster of a given incoming data and the proposed histogram management to split clusters for categorical data. To decrease the weight of old data over time, a fading function is used. Experimental results show that HUE-Stream gives better cluster quality, in terms of *purity* and the *F-measure*, compared to UMicro for the KDD-CUP'99 dataset [39].

ClusTree

ClusTree [16] is a parameter-free stream clustering algorithm that is capable of processing the stream in a single pass, with limited memory usage. It always maintains an up-to-date cluster model and reports concept drift, novelty, and outliers. This is ensured by weighing data points with an exponential time-dependent decay function. Moreover, this approach makes no apriori assumptions on the size of the clustering model, but dynamically self-adapts. *ClusTree* is an anytime algorithm that organizes micro-clusters in a tree structure for faster access and automatically adapts micro-cluster sizes based on the variance of the assigned data points. The tree used in ClusTree is a balanced multi-dimensional indexing structure with the following properties:

- an inner node contains between m and M entries. Leaf nodes contain between l and L entries. The root has at least one entry (m, M, l and L are input parameters).
- an entry in an inner node stores: (i) a cluster feature of the objects that it represents. (ii) a cluster feature of the objects in the buffer. (iii) a pointer to its child node.
- an entry in a leaf stores a cluster feature of the data point(s) it represents.
- a path from the root to any leaf node has always the same length (balanced).

So, it uses also the micro-cluster structure as a compact representation of the data distribution. Anytime algorithms denote approaches that are capable of delivering a result at any given point in time, and of using more time if it is available to refine the result. The basic idea is to maintain measures for incremental computation of the mean and variance of micro-clusters so that the infeasible access to all past stream objects is no longer necessary. We recall that a micro-cluster is a cluster feature tuple (or a variant of it) $CF = (n, LS, SS)$ of the number n of represented data points, their linear sum LS, and their squared sum SS. In the proposed method, CFs are created and updated by extending index structures from the R-tree family [42]. Such hierarchical indexing structures provide the means for efficiently locating the right place to insert any object from the stream into a micro-cluster. The idea is to build a hierarchy of micro-clusters at different levels of granularity. Given enough time, the algorithm descends the hierarchy in the index to reach the leaf entry that contains the micro-cluster that is most similar to the current object. If this micro-cluster is similar enough, it is updated incrementally by this object's values. Otherwise, a new micro-cluster may be formed [16]. However, in anytime clustering of streaming data, there might not always be enough time to reach leaf level to insert the object. To deal with this, the authors provide some strategies for anytime inserts. By incorporating local aggregates, i.e., temporary buffers for "hitchhikers", a solution is provided for the easy interruption of the insertion process so that it can be simply resumed at any later point in time. For very fast streams, aggregates of similar objects allow insertion of groups instead of single objects for even faster processing. For slower stream settings, alternative insertion strategies that exploit possible idle times of the algorithm to improve the quality of the resulting clustering are proposed [16].

Taking the means of the CFs as representatives, we can apply a k-center clustering or density based clustering (e.g. k-means or DBSCAN) to detect clusters of arbitrary shape.

Partitioning stream methods

A partitioning-based clustering algorithm organizes the objects into some number of partitions, where each partition represents a cluster. The clusters are formed based on a distance function like the k-means algorithm which leads to finding only spherical clusters and the clustering results are usually influenced by noise.

CluStream

The idea behind the *CluStream* [1] method is to divide the clustering process into an online component which periodically stores detailed summary statistics and an offline component which uses only this summary statistics. The offline component is utilized by the analyst who can use a wide variety of inputs (such as time horizon or number of clusters) in order to provide a quick understanding of the broad clusters in the data stream. The summary information is defined by the following structures:

- **Micro-clusters:** Statistical information about the data locality in terms micro-clusters are maintained. The *micro-cluster* structure is a temporal extension of the *cluster feature vector* [37]. The additivity property of the micro-clusters makes them a natural choice for the data stream problem. More precisely, a micro-cluster is tuple (N, LS, SS, LST, SST) where (N, LS, SS) are the three components of the CF vector (namely, the number of data points in the cluster, N; the linear sum of the N data points, LS; and the squared sum of the N data points, SS). The two other components are LST and SST (the sum and the sum of the squares of the time stamps of the N data points).
- **Pyramidal time frame:** The micro-clusters are stored at time snapshots which follow a pyramidal pattern. This pattern provides an effective trade-off between the storage requirements and the ability to recall summary statistics from different time horizons.

The data stream clustering algorithm proposed in [1] can generate approximate clusters in any user-specified length of history from the current moment. The online phase stores q micro-clusters in (secondary) memory, where q is an input parameter. Each micro-cluster has a maximum boundary, which is computed as the standard deviation of the mean distance of the data points to their centroids multiplied by a factor f. Each new point is assigned to its closest micro-cluster (according to the Euclidean distance) if the distance between the new point and the closest centroid falls within the maximum boundary. If so, the point is absorbed by the cluster and its summary statistics are updated. If none of the micro-clusters can absorb the point, a new micro-cluster is created. This is accomplished by either deleting the oldest micro-cluster or by merging two micro-clusters. The oldest micro-cluster is deleted if its time-stamp is below a given threshold δ (input parameter). The q micro-clusters are stored in a secondary storage device in particular time intervals that decrease exponentially, which are referred to as *snapshots*. These snapshots allow the user to search for clusters in different time horizons through a *pyramidal time window* concept. This summary information in the micro-clusters is used by an offline component which is dependent upon a wide variety of user inputs such as the time horizon or the granularity of clustering. When the user specifies a particular time horizon of length h over which to find the clusters, then we need to find micro-clusters which are specific to that time-horizon. For this purpose, we find the additive property of the cluster feature vector very useful. The final clusters are determined by using a modification of a k-means algorithm. In this technique, the micro-clusters are treated as *pseudo-points* which are re-clustered in order to determine higher level clusters.

StreamKM++

StreamKM++ [3] is a two-phase (online-offline) algorithm which maintains a small outline of the input data using the *merge-and-reduce* technique. The merge step is performed by via a data structure, named the *bucket set*, which is a set of L buckets (also named buffers), where L is an input parameter. The reduce step is performed by a significantly different summary data structure that is suitable for high-dimensional data, the *coreset tree*, when we consider that it reduces $2m$ data points to m data points (m is an input parameter). The advantage of such a coreset is that we can apply any fast approximation algorithm (for the weighted problem) on the usually much smaller coreset to compute an approximate solution for the original dataset more efficiently.

The *coreset tree* is constructed as follow. First, the tree has only the root node v, which contains all the $2m$ data points in the set of data points E_v. The prototype of the root node w_v is chosen randomly from A_v and $N_v = |E_v| = 2m$. The computation of the sum of squared distances of the data points in E_v to w_v (SSE_v) follows from the definition of w_v. Afterwards, two child nodes for v are created: v_1 and v_2. To create these nodes, it is necessary to choose a data point from E_v with probability proportional to $\frac{Dist(x^j, w_v)^2}{SSE_v}, \forall x^j \in E_v$, i.e., the data points that are farthest away from w_v has the highest probability of being selected. We call the selected data point $w_{v'}$. The next step is to allocate the data points in E_v to E_{v1} and E_{v2}, such that:

$$E_{v_1} = \left\{ x^i \in E_v \mid Dist(x^i, w_v) < Dist(x^i, w_{v'}) \right\}, E_{v_2} = E_v \setminus E_{v_1}.$$

Consequently, the summary statistics of the child nodes v_1 and v_2 are updated. This is the *expansion* step of the tree, which creates two child nodes for a given inner node. When the tree has many leaf nodes, we have to decide which one should be expanded first. In this case, we start from the root node of the coreset tree and descend it by iteratively selecting a child node with probability proportional to $\frac{SSE_{child}}{SSE_{parent}}$, until a leaf node is reached for the expansion step to be re-started. The coreset tree expansion stops when the number of leaf nodes is m.

When a new data point arrives, it is stored in the first bucket. If the first bucket is full, all of its data are moved to the second bucket. If the second bucket is full, the two buckets are merged resulting in $2m$ data points, which are then reduced to m data points, by the construction of a *coreset tree*, as previously detailed. The resulting m data points are stored in the third bucket, unless it is also full, and then again a new merge-and-reduce step is needed [3, 14]. In its offline phase, the k-means++ [43], which is executed on an input set of size m, is used for finding the final clusters. The k-means++ method is a seeding procedure for the k-means algorithm that guarantees a solution with a certain quality and gives good practical results.

StrAP

StrAP [4] is an extension of the Affinity Propagation (AP) [44] algorithm for data streams, which uses a reservoir for saving potential outliers. The Affinity Propagation approach proposes an equivalent formulation of the k-medoids problem in the sense that a prototype is an effective data point, with the difference that the number of clusters to be found is not fixed. Formulating the clustering problem in terms of energy minimization, AP outputs a set of clusters, each of which is characterized by an actual data point, referred to as an *exemplar* or a *prototype*; the penalty value parameter controls the cost of adding another prototype. AP provides some asymptotic guarantees of the optimality of the solution. The trade-off for these properties is the AP's quadratic computational complexity, excluding its use on large scale datasets. The StrAP algorithm, as an online version of AP, proceeds by incrementally updating the current model if the current data point fits the model, and putting it in a reservoir otherwise. A change point detection test enables StrAP to catch drifting exemplars that significantly deviate away. StrAP involves four main steps as illustrated in Algorithm 4 with a diagram in Fig. 4 [4]:

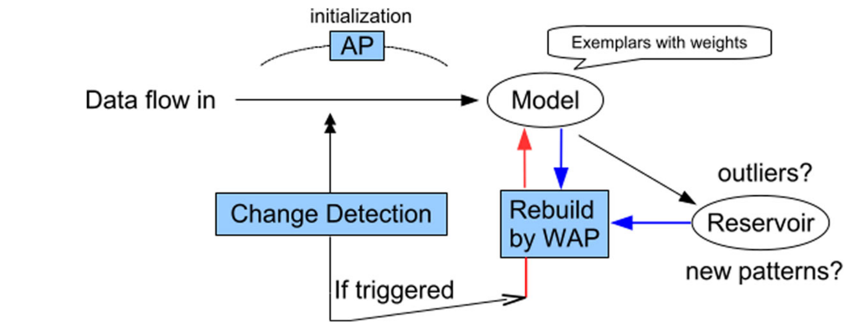

Fig. 4 Diagram of StrAP algorithm [4]

- The first batch of data is used by AP to identify the first clusters and initialize the stream model.
- As the stream flows in, each data point x_t is compared to the prototypes; if too far from the nearest exemplar, x_t is put in the reservoir, otherwise the stream model is updated accordingly.
- The data distribution is checked for change point detection, using the Page-Hinkley significance test.
- Upon triggering the change detection test, or if the number of outliers exceeds the reservoir size, the stream model is rebuilt based on the current model and reservoir, using a weighted version of AP (WAP).

Algorithm 4: StrAP

```
   Data: $\mathcal{DS} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, fit threshold $\epsilon$
1  Init: StrAP Model $\leftarrow$ AP($\mathbf{x}_1, \ldots, \mathbf{x}_T$) ;
2  Reservoir = {};
3  for $t > T$ do
4      Compute $c_i$ = nearest cluster to $\mathbf{x}_t$;
5      if $dist(c_i, \mathbf{x}_t) < \epsilon$ then
6          Update StrAP model;
7      else
8          Reservoir $\leftarrow \mathbf{x}_t$;
9      if Restart criterion then
10         Rebuild StrAP model;
11         Reservoir = {};
```

The model of the data stream used in StrAP is inspired by DenStream [2]. It consists of a set of 4-tuples $(c_i, n_i, \Sigma_i, t_i)$, where c_i ranges over the clusters, n_i is the number of items associated to cluster c_i, Σ_i is the distortion of c_i (sum of $d(x, c_i)^2$, where x ranges over all data points associated to c_i), and t_i is the last time stamp when a data point was associated to c_i.

At time t, the data point x_t is considered and its nearest cluster c_i (w.r.t. distance d) in the current model is selected; if $d(x_t, c_i)$ is less than some threshold δ, heuristically set to the average distance between points and clusters in the initial model, x_t is assigned to the i-th cluster and the model is updated accordingly; otherwise, x_t is considered to be an outlier, and put into the reservoir [4].

Approximations of the k-means algorithm in the one-pass streaming setting have been proposed in [45–47]. The streaming k-means algorithm proposed in [45] is based on a divide and conquer approach. It uses the result of [43] as a subroutine, finding $3k \log k$ centers of each block. Their experiment showed that this algorithm is an improvement over an online version of k-means algorithm and was comparable to the batch version of k-means.

The High-dimensional Projected Stream clustering method (*HPStream*) [48] introduces the concept of *projected clustering* to data streams. This algorithm is a projected clustering for high-dimensional streaming data with higher clustering quality compared to CluStream [1].

SWClustering uses an EHCF (Exponential Histogram of Cluster Features) structure by combining Exponential Histogram with Cluster Feature to record the evolution of each cluster and to capture the distribution of recent data points [49]. It tracks the clusters in evolving data streams over sliding windows.

Density-based stream methods

Density-based algorithms are based on the connection between regions and density functions. In these types of algorithms, dense areas of objects in the data space are considered as clusters, which are segregated by low density area (noise). These algorithms find clusters of arbitrary shapes and generally they require two parameters: the radius and the minimum number of data points within a cluster.

The main challenge in the streaming scenario is to construct density-based algorithms which can be efficiently executed in a single pass of the data, since the process of density estimation may be computationally intensive [9]. Amini [15] gives a survey on recent density-based data streams clustering algorithms.

DenStream

DenStream [2] is a density-based data stream clustering algorithm that also uses a feature vector based on the *CF* vector. By creating two kinds of micro-clusters (*potential* and *outlier micro-clusters*), in its online phase, *DenStream* overcomes one of the drawbacks of *CluStream*, its sensitivity to noise. Potential and outlier micro-clusters are kept in separate memories since they require different processing. Each *potential-micro-cluster* structure has an associated weight w that indicates its importance based on temporality. The weight of each data point decreases exponentially with time t via a fading function $f(t) = 2^{-\lambda t}$, where $\lambda > 0$. If the weight $w = \sum_{j=1}^{n} f(t - T_{ij})$ is above a threshold input parameter μ then the corresponding cluster is considered as a *core-micro-cluster*, where $T_{i1}, \ldots, T_{in}$ are timestamps of data points $p_{i1}, \ldots, p_{in}$. At the time t, if $w \geq \beta\mu$ then the micro-cluster is considered as *potential-micro-cluster*, else it is an *outlier-micro-cluster*, where β is the threshold of the outlier relative to core-micro-clusters ($0 < \beta < 1$). Micro-clusters with no recent points tend to lose importance, i.e. their respective weights continuously decrease over time in *outdated-micro-clusters*. However, the latter could grow into a potential micro-cluster when, by adding new points, its weight exceeds the threshold. Weights of micro-clusters are periodically calculated and decision about removing or keeping them is made based on the weight threshold.

When a new data point arrives, the algorithm tries to insert it into its nearest *potential-micro-cluster* based on its updated radius. If the insertion is not successful, the algorithm tries to insert the data point into its closest *outlier micro-cluster*. If the insertion is successful, the cluster summary statistics will be updated accordingly. Otherwise, a new *outlier micro-cluster* is created to absorb this point. The Euclidean distance between the new data point and the center of the nearest potential or outlier micro-cluster is measured. A micro-cluster is chosen with the distance less than or equal to the radius threshold. *DenStream* has a pruning method in which it frequently checks the weights of the outlier-micro-clusters in the outlier buffer to guarantee the recognition of the real outliers. However, the non-release of the allocated memory when either deleting a micro-cluster or merging two old micro-clusters is considered as a limitation of the *DenStream* algorithm as well as the time-consuming pruning phase for removing outliers [15]. In the offline phase, the potential-micro-clusters found during the online phase are considered as *pseudo-points* and will be passed to a variant of the DBSCAN algorithm in order to determine the final clusters.

SOStream

SOStream [50] is a density-based clustering algorithm inspired by both the principle of the DBSCAN algorithm and self-organizing maps (SOM) [18], in the sense that a winner influences its immediate neighborhood. Generally speaking, density-based clustering algorithms need setting a threshold manually (similarity threshold, grid size, etc.) for which is difficult to choose the most suitable value and if it is set to an unsuitable value, then the algorithm will suffer from over-fitting, or from unstable clustering. SOStream addresses this problem by using a dynamically learned threshold value for each cluster based on the idea of building neighborhoods with a minimum number of points.

SOStream is also represented by a set of micro-clusters where for each cluster a cluster feature (CF) vector is stored, which is a tuple with three elements $N_i = (n_i, r_i, C_i)$, n_i is the number of data points assigned to N_i, r_i is the cluster's radius and C_i is the centroid.

When a new point arrives, the nearest cluster is selected, based on the Euclidean distance to existing micro-clusters, and then absorbs this point if the calculated distance is less than a dynamically defined threshold. It also assigns the micro-clusters' neighbors to the nearest cluster, i.e., the centroids of clusters sufficiently close to the winning cluster have their centroids modified to be closer to the winning cluster's centroid. This approach is used to assist in merging similar clusters and increasing separation between different clusters. The neighborhood of the winner is defined based on the idea of a *MinPts* distance given by a minimum number of neighboring objects [2]. This distance is found by computing the Euclidean distance from any existing clusters to the winning cluster. If the new point is not absorbed by any micro-cluster, a new micro-cluster is created for it. In the *SOStream* algorithm, merging, updating and adapting dynamically the threshold value for each cluster are performed in an online manner. Clusters are merged if they overlap with a distance that is less than the merge-threshold, i.e., the spheres in d-dimensional space defined by the radius of each cluster overlap. Hence, the threshold value is a determining factor for the number of clusters. When two clusters are merged, the largest radius of these two clusters is chosen to be the radius of the cluster to avoid losing any data points within the clusters. However, no split feature is proposed in the algorithm. *SOStream* also uses an exponential fading function to reduce the impact of old data whose relevance diminishes over time.

SVStream

SVStream (Support Vector based Stream clustering) [51] is a data stream clustering algorithm based on support vector clustering (SVC) and support vector domain description (SVDD).

In the Support Vector Clustering (SVC) [52] algorithm data points are mapped from the data space to a high dimensional feature space using a Gaussian kernel. In the feature space we look for the smallest sphere that encloses the image of the data. This sphere is mapped back to data space, where it forms a set of contours which enclose the data points. These contours are interpreted as cluster boundaries. Points enclosed by each separate contour are associated with the same cluster. Support vectors are used to construct cluster boundaries of arbitrary shape in SVC.

Support vector domain description (SVDD) [53] is a one-class classifier inspired by the support vector classifier. The idea is to use kernels to project data into a feature space and then to find the sphere enclosing almost all data, namely not including outliers. SVDD has the possibility to reject a fraction of the training data points, when this sufficiently decreases the volume of the hypersphere. One inherent drawback of SVDD, which affects not only its outlier detection performance but also its general properties significantly, is that the resulting description is highly sensitive to the selection of the trade-off parameter, which is difficult to estimate in practice.

Given a set of M data elements, the Gaussian kernel parameter q and the trade-off parameter C, the sphere structure S is defined as

$$S = \{SV, BSV, \|\mu\|^2, R_{SV}, R_{BSV}\}.$$

where,

- SV is a support vector set.
- BSV is a bounded support vector set.
- $\|\mu\|^2$ is the squared length of the sphere center μ.
- R_{SV} is the radius of the sphere.
- R_{BSV} is the maximum Euclidean distance of the bounded support vectors from the sphere center μ.

The multi-sphere set SS is defined as a set consisting of multiple spheres, that is, $SS = \{S^1, \ldots, S^{|SS|}\}$, where the superscript denotes the index of a sphere. In SVStream, the elements of a data stream are mapped to a kernel space, and the support vectors are used as the summary information of the historical elements to construct the cluster boundaries of arbitrary shape. To adapt both dramatic and gradual changes, multiple spheres are dynamically maintained, each describing the corresponding data domain presented in the data stream. When a new data batch arrives, if a dramatic change occurs, a new sphere is created; otherwise, only the existing spheres are updated to take into account the new batch. The data elements of this new batch are assigned with cluster labels according to the cluster boundaries constructed by the sphere set. Bounded support vector (BSVs) and a newly designed BSV decaying mechanism are introduced so as to respectively identify overlapping clusters and automatically detect outliers (noise) [51]. In the clustering process, if two spheres are too close to each other, they should be merged. In addition, eliminating old BSVs by the BSV decaying mechanism would help detect the tendency of a cluster to shrink or split.

OPTICS-Stream is an extension of the OPTICS algorithm [54] to the streaming data model. OPTICS uses a density identification method to create a one-dimensional cluster-ordering of the data. *OPTICS-Stream* is an online visualization algorithm producing a map representing the clustering structure where each valley represents a cluster [55].

PreDeConStream [56] is based on the two phase process of mining data streams, which builds a micro-cluster-based structure to store an online summary of the streaming data. The technique is based on subspace clustering, targeting applications with high data dimensionality.

Grid-based stream methods

Grid-based clustering is another group of the clustering methods for data streams where the data space is quantized into finite number of cells which form the grid structure and perform clustering on the grids. Grid-based clustering maps the infinite number of data records in data streams to a finite number of grids. Then, the grids are clustered based on their density.

D-Stream

D-Stream [57] is also a two-phase scheme which consists of an online component that processes input data stream and produces summary statistics and an offline component that uses the summary data to generate clusters. In the online component, the algorithm maps each input data point into a grid whereas in the offline component, it computes the grid density and clusters the grids based on the density. The algorithm adopts a density decaying technique to capture the dynamic changes of a data stream and it can find clusters of arbitrary shapes. Unlike other algorithms such as CluStream [1], D-Stream automatically and dynamically adjusts the clusters without requiring user specification of target time horizon and number of clusters. Algorithm 5 outlines the overall architecture of D-Stream.

For a data stream, at each time step, the online component of D-Stream continuously reads a new data point, places the multi-dimensional data into a corresponding discretized density grid in the multi-dimensional space, and updates the characteristic vector of the density grid (Lines 4-6 of Algorithm 5). The density for a grid g, at a given time t, $D(g,t)$ is defined as the sum of the density coefficients of all data records that are mapped to g. That is the density of g at t is:

$$D(g,t) = \sum_{\mathbf{x} \in E(g,t)} D(\mathbf{x},t),$$

where $E(g,t)$ is the set of data points that are mapped to g at or before time t. The density of any grid is constantly changing. However, the updating operation is executed only when a new data record is mapped to that grid.

D-Steam uses the *characteristic vector* concept associated to each grid. This is a tuple $(t_g, t_m, D, label, status)$, where t_g is the last time when g is updated, t_m is the last time when g is removed from grid list as a sporadic grid (if ever), D is the grid density at the last update, *label* is the class label of the grid, and $status = \{SPORADIC, NORMAL\}$ is a label used for removing sporadic grids.

The procedures *initial_clustering* (used in Line 8 of Algorithm 5) and *adjust_clustering* (used in Line 11 of 5) update the density of all active grids to the current time, first. Once the density of grids are determined at the given time, the clustering procedure is similar to the standard method used by density-based clustering.

The offline component dynamically adjusts the clusters every *gap* time steps, where *gap* is an integer parameter. After the first *gap*, the algorithm generates the initial cluster (Lines 7-8). Then, the algorithm periodically removes sporadic grids and adjusts the clusters (Lines 9-11) [57].

Algorithm 5: D-Stream

```
1  $time_c = 0$;
2  initialize an empty hash table grid_list;
3  while there is a data point to proceed do
4      Get the next data point in the data stream, $\mathbf{x} = (x_1, x_2, \ldots, x_d)$;
5      Determine the density grid $g$ that contains $\mathbf{x}$;
6      if $g$ not in grid_list then Insert $g$ to grid_list Update the characteristic vector of
       $g$;
7      if $time_c = gap$ then
8          Call initial_clustering(grid_list);
9      if $time_c \mod gap == 0$ then
10         Detect and remove sporadic grids from grid_list;
11         Call adjust_clustering(grid_list);
12     $time_c = time_c + 1$;
```

One weakness of the approach is that a significant number of non-empty grid cells need to be discarded in order to keep the memory requirements in check. In many cases, such grid-cells occur at the borders of the clusters. The discarding of such cells may lead to a degradation in cluster quality [9].

Analogously to *D-Stream*, *MR-Stream* [58] facilitates the discovery of clusters at multiple resolutions by using a grid of cells that can dynamically be sub-divided into more cells using a tree data structure. In the online phase, it assigns new incoming data to the appropriate cell and updates the summary information. The offline component obtains a portion of the tree at a fixed hight h and performs clustering at the resolution level determined by h.

Summary

Table 1 summarizes the main features offered by each algorithm considered in terms of: the basic clustering algorithm, whether the algorithm identifies a topological structure, whether the links (if they exist) between clusters (nodes) are weighted, how many phases it adopts (online and offline), the types of operations for updating clusters (remove, merge, and split cluster), and whether a *fading* function is used.

Table 1 Comparison between algorithms (WL: weighted links, 2 phases : online+offline)

Algorithms	Based on	Topology	WL	Phases	Remove	Merge	Split	Fade
SVStream	SVC, SVDD	✗	✗	online	✓	✓	✓	✓
StreamKM++	*k*-means++	✗	✗	2 phases	✓	✓	✓	✓
StrAP	AP	✗	✗	2 phases	✓	✗	✗	✓
SOStream	DBSCAN, SOM	✗	✗	online	✓	✓	✗	✓
OPTICS-Stream	OPTICS	✗	✗	2 phases	✓	✓	✗	✓
IGNG	NG	✓	✗	online	✗	✗	✗	✗
HCluStream	*k*-prototypes	✗	✗	2 phases	✓	✓	✓	✓
GWR	NG	✓	✗	online	✗	✗	✗	✗
G-Stream	NG	✓	✓	online	✓	✗	✗	✓
E-Stream	*k*-means	✗	✗	2 phases	✓	✓	✓	✓
D-Stream	-	✗	✗	2 phases	✓	✓	✓	✓
DenStream	DBSCAN	✗	✗	2 phases	✓	offline	✗	✓
ClusTree	*k*-means or DBSCAN	✗	✗	2 phases	✓	offline	✓	✓
CluStream	*k*-means	✗	✗	2 phases	✓	offline	✗	✗
AING	NG	✓	✗	online	✗	✓	✗	✗

Streaming platforms

In today's applications, evolving data streams are ubiquitous. As the need by industry for real time analysis has emerged, an increasing number of systems to support real-time data integration and analytics in the recent years. Generally speaking, there exists two types of streaming processing systems. Traditional streaming platforms, on which we can implement a streaming algorithm using a *traditional* programming language in a *sequential* manner. Distributed streaming platforms, where the data is distributed across a cluster of machine and the processing model is implemented using the MapReduce framework. This section gives a survey on the most well-known streaming platforms with a focus on the streaming clustering task. Liu [59] gives a general survey on real-time processing systems for big data.

Spark streaming

Spark Streaming [7] is an extension of the Apache Spark [60] project by adding the ability to perform online processing through a similar functional interface to Spark, such as map, filter, reduce, etc. Spark is a cluster computing system originally developed by UC Berkeley AMPLab. Now it is an umbrellaed project of Apache foundation. The execution model of Spark is based on an abstraction called Resilient Distributed Dataset (RDD), which is a distributed memory abstraction of data. Spark performs in-memory computations on large clusters in a fault-tolerant manner through RDDs [61]. Spark Streaming runs streaming computations as a series of short batch jobs on RDDs withing a programming model called *discretized streams* (D-Streams). The key idea behind D-Streams is to treat a streaming computation as a series of deterministic batch computations on small time intervals. For example, we might place the data received each second into a new interval, and run a MapReduce operation on each interval to compute a count. Similarly, we can perform a running count over several intervals by adding the new counts from each interval to the old result. Spark Streaming can automatically parallelize the jobs across the nodes in a cluster. It also supports fault recovery for a wide array of operations.

Spark Streaming comes with a new approach for fault recovery, while classical streaming systems update the mutable state on a per-record basis and use either replication or upstream backup for fault recovery. The replication approach creates two or more copies of each process in the data flow graph [62]. This can double the hardware cost, and if two nodes in the same replica fail, the system is not recoverable. In upstream backup [63], upstream nodes act as backups for their downstream neighbors by preserving tuples in their output queues while their downstream neighbors process them. If a server fails, its upstream nodes replay the logged tuples on a recovery node. The disadvantage of this approach is long recovery times, as the system must wait for the standby node to catch up.

To address these issues, D-Streams employ another approach: *parallel recovery*. The system periodically checkpoints some of the state RDDs, by asynchronously replicating them to other nodes. For example, in a view count program computing hourly windows, the system could checkpoint results every minute. When a node fails, the system detects the missing RDD partitions and launches tasks to recover them from the latest checkpoint [7].

In the streaming clustering point of view, Spartakus[2] is an open-source project on top of Spark-notebook[3] which provides front-end packages for some clustering algorithms implemented using the MapReduce framework. This includes the MBG-Stream[4] algorithm [35] (detailed in "Background" section) with an integrated interface for execution and visualization checks. MLlib [64] gives implementations of some clustering algorithms, especially a Streaming k-means[5] open-source code. streamDM[6] is another open source software for mining big data streams using Spark Streaming, developed at Huawei Noah's Ark Lab. For streaming clustering, it includes Clustream [1] and StreamKM++ [3].

Flink

Flink[7] is an open source platform for distributed stream and batch data processing. The core of Flink is a streaming iterative data flow engine. On top of the engine, Flink exposes two language-embedded fluent APIs, the DataSet API for consuming and processing batch data sources and the DataStream API for consuming and processing event streams. The key idea behind Flink is the optimistic recovery mechanism that does not checkpoint every state [8]. Therefore, it provides optimal failure-free performance and simultaneously uses less resources in the cluster than traditional approaches. Instead of restoring such a state from a previously written checkpoint and restarting the execution, a user-defined, algorithm-specific *compensation* function is applied. In case of a failure, this function restores a consistent algorithm state and allows the system to continue the execution.

MOA

MOA[8] (Massive On-line Analysis) is a framework for data stream mining [6]. It includes tools for evaluation and a collection of machine learning algorithms. Related to the WEKA project[9] (Waikato Environment for Knowledge Analysis), it is also written in Java, while scaling to more demanding problems. The goal of MOA is a benchmark framework for running experiments in the data stream mining context by proving storable settings for data streams (real and synthetic) for repeatable experiments, a set of existing algorithms and measures from the literature for comparison, and an easily extendable framework for new streams, algorithms and evaluation methods. MOA currently supports stream classification, stream clustering, outlier detection, change detection and concept drift and recommender systems. Currently MOA contains several stream clustering methods including: StreamKM++ [3], CluStream [1], ClusTree [16], DenStream [2], D-Stream [57].

SAMOA

SAMOA[10] (Scalable Advanced Massive Online Analysis) is distributed streaming machine learning (ML) framework that contains a programing abstraction for distributed streaming ML algorithms. It is a project started at Yahoo Labs Barcelona. SAMOA is both a framework and a library [65]. As a framework, it allows algorithm developers to abstract from the underlying execution engine, and therefore reuse their code on different engines. It features a pluggable architecture that allows it to run on several distributed stream processing engines such as Storm[11], S4[12], and Samza[13]. As a library, SAMOA contains implementations of state-of-the-art algorithms for distributed machine learning on streams. For streaming clustering, it includes an algorithm based on CluStream [1].

Open challenges in data stream clustering

In today's applications, evolving data streams are ubiquitous. Mining, knowledge discovery, or more specifically clustering streaming data is a recent domain compared to the offline (or batch) model. Thus, many of the challenges, issues and problems remain to be addressed in the streaming model. This section is devoted to discuss some challenging issues and further directions from the viewpoints of both academic research and industrial applications [11, 14, 66–68].

Protecting privacy and confidentiality. Data streams present new challenges and opportunities with respect to protecting privacy and confidentiality in data mining. The main objective is to develop such data mining techniques that would not uncover information or patterns which compromise confidentiality and privacy obligations. Privacy-by-design seems to be an interesting paradigm to use.

Handling incomplete information. The problem of missing values, which corresponds to incompleteness of features, has been discussed extensively for the offline, static settings.

Uncertain data. In most applications we don't have sufficient data for statistical operations so new methods are needed to manage uncertain data stream in an accurate and fast manner.

Variety of data. Data type diversity in a given stream (text, video, audio, static image, etc.) as well as differences in data processability (structured, semi-structured, unstructured data). Clustering these diverse types of data together, coming in a streaming form, is very challenging. Another interesting future application of data stream clustering is social network analysis. The activities of social network members can be regarded as a data stream, and a clustering algorithm can be used to show similarities among members, and how these similar profiles (clusters) evolve over time.

Synopsis, sketches and summaries. A synopsis is compact data structures that summarize data for further querying. Samples, Histograms, Wavelets, Sketches describe basic principles and recent developments in building approximate synopses (that is, lossy, compressed representations) of massive data [69]. Data sketching via random projections is a tool for dimensionality reduction. Although this technique is extremely efficient, its main drawback is that it may ignore relevant features.

Distributed streams. Data streams are distributed in nature. For learning from distributed data, we need efficient methods in minimizing the communication overheads between nodes. Most importantly, in applications like monitoring, centralized solutions introduce delays in event detection and reaction, that can make mining systems inefficient. Many data clustering techniques are not trivial to parallelize. To develop distributed versions of some methods, a lot of research is needed with practical and theoretical analysis to provide new methods.

Evaluation of data stream algorithms. Although in the field of static classification such tools exist, they are insufficient in data stream environments due to such problems as: concept drift, limited processing time, verification latency, multiple stream structures, evolving class skew, censored data, and changing misclassification costs. Indeed, in the streaming context, we are more interested in how the evaluation metric evolves over time [66].

Autonomous and self-diagnosis. Knowledge discovery from data streams requires the ability for predictive self-diagnosis. A significant and useful intelligence characteristic is diagnostics, not only after failure has occurred, but also predictive (before failure) and advisory (providing maintenance instructions). The development of such self-configuring, self-optimizing, and self-repairing systems is a major scientific and engineering challenge. All these aspects require monitoring the evolution of the learning process, taking into account the available resources, and the ability to reason and learn about it [67, 68].

Combining offline and online models. Online (or real-time) and offline (or batch) learning are mostly considered as mutually exclusive, but it is their combination that might enhance the value of data the most. Lambda Architecture [70] is a useful framework for designing big data applications where we can combine these two models in a same plateform. Figure 5 is a diagram of the Lambda Architecture.

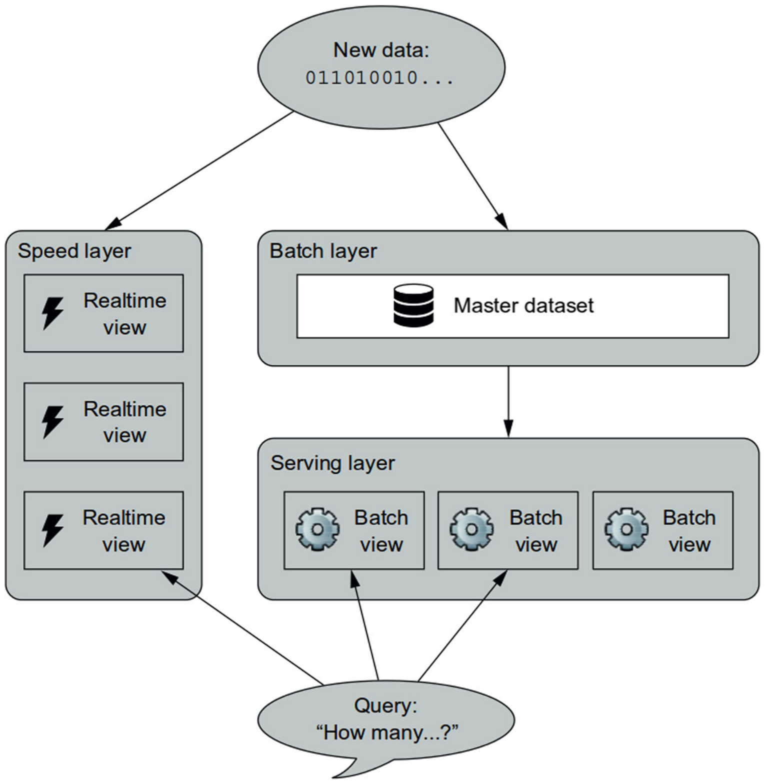

Fig. 5 Lambda architecture diagram [70]

Essentially, the Lambda Architecture comprises the following components, processes, and responsibilities:

- New Data: All data entering the system is dispatched to both the batch layer and the speed layer for processing.
- Batch layer: This layer has two functions: (i) managing the master dataset, an immutable, append-only set of raw data, and (ii) to pre-compute arbitrary query functions from scratch, called batch views.
- Serving layer: This layer indexes the batch views so that they can be queried in ad hoc with low latency.
- Speed layer: This layer compensates for the high latency of updates to the serving layer, due to the batch layer. Using fast and incremental algorithms, the speed layer deals with recent data only.
- Queries: Any incoming query can be answered by merging results from both batch views and real-time views.

Designing data stream clustering methods in a Lambda Architecture where we can benefit from the high accuracy of the batch model is very interesting and challenging.

Conclusion

Recently, examples of applications relevant to streaming data have become more numerous and more important, including network intrusion detection, transaction streams, phone records, web click-streams, social streams, weather monitoring, etc. Indeed, the data stream clustering problem has become an active research in recent years. This problem requires a process capable of partitioning observations continuously while taking into account restrictions of memory and time.

In this paper, we surveyed, in a detailed and comprehensive manner, a number of the representative state-of-the-art algorithms for the clustering over data streams. These algorithms are categorized according to the nature of their underlying clustering approach, including GNG, hierarchical, partitioning, density, and grid-based stream methods. Motivated by the need by industry for real time analysis, an increasing number of systems to support real-time data integration and analytics has emerged in recent years. We have made an overview of the most well-known open-source streaming systems, including Spark Streaming, Flink, MOA, and SAMOA, with a focus on the streaming clustering task.

Endnotes

[1] See http://spark.apache.org/streaming/

[2] See https://hub.docker.com/r/spartakus/coliseum/

[3] See http://spark-notebook.io/

[4] See https://github.com/mghesmoune/spark-streaming-clustering

[5] See http://spark.apache.org/docs/latest/mllib-clustering.html#streaming-k-means

[6] See http://huawei-noah.github.io/streamDM/

[7] See https://flink.apache.org/

[8] See http://moa.cms.waikato.ac.nz/

[9] See http://weka.wikispaces.com/

[10] See http://samoa-project.net/

[11] See http://storm.apache.org

[12] See http://incubator.apache.org/s4

[13] See http://samza.incubator.apache.org

Acknowledgements
This work has been supported by the French Foundation FSN, PIA Grant Big data-Investissements d'Avenir. The project is titled "Square Predict" (http://ns209168.ovh.net/squarepredict/). We thank anonymous reviewers for their insightful remarks.

Authors' contributions
All authors contributed equally. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Received: 20 April 2016 Accepted: 14 September 2016
Published online: 01 December 2016

References

1. Aggarwal CC, Watson TJ, Ctr R, Han J, Wang J, Yu PS. A framework for clustering evolving data streams. In: In VLDB. Berlin: VLDB Endowment; 2003. p. 81–92.
2. Cao F, Ester M, Qian W, Zhou A. Density-based clustering over an evolving data stream with noise. In: SDM. SIAM; 2006. p. 328–39.
3. Ackermann MR, Märtens M, Raupach C, Swierkot K, Lammersen C, Sohler C. StreamKM++: A clustering algorithm for data streams. ACM J Exp Algorithmics. 2012;17(1):173–187.
4. Zhang X, Furtlehner C, Sebag M. Data streaming with affinity propagation. In: ECML/PKDD (2). Berlin: Springer Berlin Heidelberg; 2008. p. 628–43.
5. Demchenko Y, Grosso P, De Laat C, Membrey P. Addressing big data issues in scientific data infrastructure. In: Collaboration Technologies and Systems (CTS), 2013 International Conference On. IEEE; 2013. p. 48–55.
6. Bifet A, Holmes G, Pfahringer B, Kranen P, Kremer H, Jansen T, Seidl T. MOA: massive online analysis, a framework for stream classification and clustering. In: Proceedings of the First Workshop on Applications of Pattern Analysis, WAPA 2010, Cumberland Lodge, Windsor, UK September 1–3, 2010; 2010. p. 44–50.
7. Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I. Discretized streams: fault-tolerant streaming computation at scale. In: ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3–6, 2013; 2013. p. 423–38.
8. Schelter S, Ewen S, Tzoumas K, Markl V. "all roads lead to rome": optimistic recovery for distributed iterative data processing. In: 22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013; 2013. p. 1919–28.
9. Aggarwal CC. A survey of stream clustering algorithms. In: Data Clustering: Algorithms and Applications. Chapman and Hall/CRC; 2013. p. 231–58.
10. Nguyen H, Woon Y, Ng WK. A survey on data stream clustering and classification. Knowl Inf Syst. 2015;45(3):535–69.
11. Khalilian M, Mustapha N. Data stream clustering: Challenges and issues. CoRR. 2010;abs/1006.5261.
12. Yogita, Toshniwal D. Clustering techniques for streaming data-a survey. In: Advance Computing Conference (IACC), 2013 IEEE 3rd International. IEEE; 2013. p. 951–6.
13. Mousavi M, Bakar AA, Vakilian M. Data stream clustering algorithms: A review. Int J Adv Soft Comput Appl. 2015;7(3):13:1–13:31.
14. de Andrade Silva J, Faria ER, Barros RC, Hruschka ER, de Carvalho ACPLF, Gama J. Data stream clustering: A survey. ACM Comput Surv. 2013;46(1):13.
15. Amini A, Teh YW, Saboohi H. On density-based data streams clustering algorithms: A survey. J Comput Sci Technol. 2014;29(1):116–41.
16. Kranen P, Assent I, Baldauf C, Seidl T. The ClusTree: indexing micro-clusters for anytime stream mining. Knowl Inf Syst. 2011;29(2):249–72.
17. Fritzke B. A growing neural gas network learns topologies. In: NIPS. MIT Press; 1994. p. 625–32.
18. Kohonen T, Schroeder MR, Huang TS, (eds). Self-Organizing Maps, 3rd edn. Secaucus, NJ, USA: Springer; 2001.
19. Martinetz T, Schulten K. A "Neural-Gas" Network Learns Topologies. Artif Neural Netw. 1991;I:397–402.
20. Beyer O, Cimiano P. Online semi-supervised growing neural gas. Int J Neural Syst. 2012;22(5):21–23.
21. Fritzke B. A self-organizing network that can follow non-stationary distributions. In: Artificial Neural Networks - ICANN '97, 7th International Conference, Lausanne, Switzerland, October 8–10, 1997, Proceedings. Berlin: Springer Berlin Heidelberg; 1997. p. 613–8.
22. Sledge IJ, Keller JM. Growing neural gas for temporal clustering. In: 19th International Conference on Pattern Recognition (ICPR 2008), December 8–11, 2008. Tampa. IEEE; 2008. p. 1–4.
23. Mitsyn S, Ososkov G. The growing neural gas and clustering of large amounts of data. Opt Mem Neural Netw. 2011;20(4):260–70.
24. Marsland S, Shapiro J, Nehmzow U. A self-organising network that grows when required. Neural Netw. 2002;15(8–9):1041–58.
25. Marsland S, Nehmzow U, Shapiro J. On-line novelty detection for autonomous mobile robots. Robot Auton Syst. 2005;51(2):191–206.
26. Mendes CAT, Gattass M, Lopes H. Fgng: A fast multi-dimensional growing neural gas implementation. Neurocomputing. 2014;128:328–40.
27. Prudent Y, Ennaji A. An incremental growing neural gas learns topologies. In: Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference On. MIT Press; 2005. p. 1211–1216.

28. Hamza H, Belaïd Y, Belaïd A, Chaudhuri BB. Incremental classification of invoice documents. In: 19th International Conference on Pattern Recognition (ICPR 2008), December 8–11, 2008, Tampa, Florida, USA. IEEE Computer Society; 2008. p. 1–4.
29. Lamirel JC, Boulila Z, Ghribi M, Cuxac P. A new incremental growing neural gas algorithm based on clusters labeling maximization: application to clustering of heterogeneous textual data. In: Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems - Volume Part III. Berlin: Springer-Verlag; 2010. p. 139–48.
30. GarcíA-RodríGuez J, Angelopoulou A, García-Chamizo JM, Psarrou A, Escolano SO, GiméNez VM. Autonomous growing neural gas for applications with time constraint: optimal parameter estimation. Neural Netw. 2012;32: 196–208.
31. Pimentel MA, Clifton DA, Clifton L, Tarassenko L. A review of novelty detection. Signal Process. 2014;99:215–49.
32. Bouguelia MR, Belaïd Y, Belaïd A. An adaptive incremental clustering method based on the growing neural gas algorithm. In: ICPRAM. SciTePress; 2013. p. 42–9.
33. Ghesmoune M, Azzag H, Lebbah M. G-stream: Growing neural gas over data stream. In: Neural Information Processing - 21st International Conference, ICONIP 2014, Kuching, Malaysia, November 3–6, 2014. Proceedings, Part I; 2014. p. 207–14.
34. Ghesmoune M, Lebbah M, Azzag H. Clustering over data streams based on growing neural gas. In: Advances in Knowledge Discovery and Data Mining - 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19–22, 2015, Proceedings, Part II; 2015. p. 134–45.
35. Ghesmoune M, Lebbah M, Azzag H. Micro-batching growing neural gas for clustering data streams using spark streaming. In: INNS Conference on Big Data 2015, San Francisco, CA, USA, 8–10 August 2015. Elsevier; 2015. p. 158–66.
36. Aggarwal CC, Reddy CK. Data Clustering: Algorithms and Applications, 1st: Chapman & Hall/CRC; 2013.
37. Zhang T, Ramakrishnan R, Livny M. Birch: An efficient data clustering method for very large databases. In: SIGMOD Conference. New York: ACM; 1996. p. 103–14.
38. Udommanetanakit K, Rakthanmanon T, Waiyamai K. E-stream: Evolution-based technique for stream clustering. In: ADMA; 2007. p. 605–15.
39. Meesuksabai W, Kangkachit T, Waiyamai K. Hue-stream: Evolution-based clustering technique for heterogeneous data streams with uncertainty. In: Advanced Data Mining and Applications - 7th International Conference, ADMA 2011, Beijing, China, December 17–19, 2011, Proceedings, Part II; 2011. p. 27–40.
40. Aggarwal CC, Yu PS. A framework for clustering uncertain data streams. In: Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7–12, 2008, Cancún, México; 2008. p. 150–9.
41. Yang C, Zhou J. Hclustream: A novel approach for clustering evolving heterogeneous data stream. In: Workshops Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18–22 December 2006, Hong Kong, China; 2006. p. 682–8.
42. Guttman A. R-trees: A dynamic index structure for spatial searching. In: SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18–21, 1984; 1984. p. 47–57.
43. Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding. In: SODA. Philadelphia: Society for Industrial and Applied Mathematics; 2007. p. 1027–1035.
44. Frey BJ, Dueck D. Clustering by passing messages between data points. Science. 2007;315:2007.
45. Ailon N, Jaiswal R, Monteleoni C. Streaming k-means approximation. In: NIPS. USA: Curran Associates Inc.; 2009. p. 10–18.
46. Braverman V, Meyerson A, Ostrovsky R, Roytman A, Shindler M, Tagiku B. Streaming K-means on Well-clusterable Data. In: Proceedings of the Twenty-second Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics; 2011. p. 26–40.
47. Shindler M, Wong A, Meyerson A. Fast and accurate k-means for large datasets. In: NIPS. USA: Curran Associates Inc.; 2011. p. 2375–383.
48. Aggarwal CC, Han J, Wang J, Yu PS. A framework for projected clustering of high dimensional data streams. In: (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004. VLDB Endowment; 2004. p. 852–63.
49. Zhou A, Cao F, Qian W, Jin C. Tracking clusters in evolving data streams over sliding windows. Knowl Inf Syst. 2008;15(2):181–214.
50. Isaksson C, Dunham MH, Hahsler M. SOStream: Self organizing density-based clustering over data stream. In: MLDM. Berlin: Springer Berlin Heidelberg; 2012. p. 264–78.
51. Wang C, Lai J, Huang D, Zheng W. SVStream: A support vector-based algorithm for clustering data streams. IEEE Trans Knowl Data Eng. 2013;25(6):1410–24.
52. Ben-Hur A, Horn D, Siegelmann HT, Vapnik V. Support vector clustering. J Mach Learn Res. 2001;2:125–37.
53. Tax DMJ, Duin RPW. Support vector domain description. Pattern Recogn Lett. 1999;20(11–13):1191–9.
54. Ankerst M, Breunig MM, Kriegel H, Sander J. OPTICS: ordering points to identify the clustering structure. In: SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1–3, 1999, Philadelphia, Pennsylvania, USA. New York: ACM; 1999. p. 49–60.
55. Tasoulis DK, Ross GJ, Adams NM. Visualising the cluster structure of data streams. In: Advances in Intelligent Data Analysis VII, 7th International Symposium on Intelligent Data Analysis, IDA 2007, Ljubljana, Slovenia, September 6–8, 2007, Proceedings; 2007. p. 81–92.
56. Hassani M, Spaus P, Gaber MM, Seidl T. Density-based projected clustering of data streams. In: Scalable Uncertainty Management - 6th International Conference, SUM 2012, Marburg, Germany, September 17–19, 2012. Proceedings; 2012. p. 311–24.
57. Chen Y, Tu L. Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12–15, 2007. New York: ACM; 2007. p. 133–42.

58. Wan L, Ng WK, Dang XH, Yu PS, Zhang K. Density-based clustering of data streams at multiple resolutions. TKDD. 2009;3(3):14:1–14:28.
59. Liu X, Iftikhar N, Xie X. Survey of real-time processing systems for big data. In: 18th International Database Engineering & Applications Symposium, IDEAS 2014, Porto, Portugal, July 7–9, 2014. New York: ACM; 2014. p. 356–61.
60. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. In: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing. HotCloud'10. Berkeley, CA, USA: USENIX Association; 2010. p. 10–10.
61. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25–27, 2012. Berkeley: USENIX Association; 2012. p. 15–28.
62. Balazinska M, Balakrishnan H, Madden S, Stonebraker M. Fault-c distributed stream processing system. ACM Trans Database Syst. 2008;33(1):3:1–3:44.
63. Hwang J, Balazinska M, Rasin A, Çetintemel U, Stonebraker M, Zdonik SB. High-availability algorithms for distributed stream processing. In: Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5–8 April 2005, Tokyo, Japan. Washington: IEEE Computer Society; 2005. p. 779–90.
64. Meng X, Bradley JK, Yavuz B, Sparks ER, Venkataraman S, Liu D, Freeman J, Tsai DB, Amde M, Owen S, Xin D, Xin R, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A. Mllib: Machine learning in apache spark. CoRR. 2015;abs/1505.06807.
65. Morales GDF, Bifet A. SAMOA: scalable advanced massive online analysis. J Mach Learn Res. 2015;16:149–53.
66. Krempl G, Žliobaite I, Brzeziński D, Hüllermeier E, Last M, Lemaire V, Noack T, Shaker A, Sievi S, Spiliopoulou M, et al. Open challenges for data stream mining research. ACM SIGKDD explorations newsletter. 2014;16(1):1–10.
67. Gama J. A survey on learning from data streams: current and future trends. Prog AI. 2012;1(1):45–55.
68. Gama J. Knowledge Discovery from Data Streams, 1st: Chapman & Hall/CRC; 2010.
69. Cormode G, Garofalakis MN, Haas PJ, Jermaine C. Synopses for massive data: Samples, histograms, wavelets, sketches. Found Trends Databases. 2012;4(1–3):1–294.
70. Marz N, Warren J. Big Data: Principles and Best Practices of Scalable Realtime Data Systems: Manning Publications Co.; 2015.