Artificial Intelligence Review (2021) 54:1201–1236
https://doi.org/10.1007/s10462-020-09874-x

Data stream clustering: a review

Alaettin Zubaroğlu[1] · **Volkan Atalay**[1]

Published online: 21 July 2020

Abstract
Number of connected devices is steadily increasing and these devices continuously generate data streams. Real-time processing of data streams is arousing interest despite many challenges. Clustering is one of the most suitable methods for real-time data stream processing, because it can be applied with less prior information about the data and it does not need labeled instances. However, data stream clustering differs from traditional clustering in many aspects and it has several challenging issues. Here, we provide information regarding the concepts and common characteristics of data streams, such as concept drift, data structures for data streams, time window models and outlier detection. We comprehensively review recent data stream clustering algorithms and analyze them in terms of the base clustering technique, computational complexity and clustering accuracy. A comparison of these algorithms is given along with still open problems. We indicate popular data stream repositories and datasets, stream processing tools and platforms. Open problems about data stream clustering are also discussed.

Keywords Data streams · Data stream clustering · Real-time clustering

1 Introduction

More devices including sensors are becoming interconnected and interconnected devices continuously generate streams of data at high speed. Offline processing of such huge amount of data requires growing storage capacity and may cause delayed analyses. Hence, real-time processing of the data generated by the connected devices has become an active research area.

A data stream is a potentially unbounded, ordered sequence of instances. A data stream S may be shown as $S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_N\}$ where $\mathbf{x}_i$ is ith data instance, which is a d-dimensional feature vector and N goes to infinity. Data stream differs from the traditional, stored data in many aspects. In most cases, true class labels are not available for

✉ Alaettin Zubaroğlu
alaettin.zubaroglu@metu.edu.tr

Volkan Atalay
vatalay@metu.edu.tr

[1] Department of Computer Engineering, Middle East Technical University, Dumlupınar Bulvarı No:1 06800, Çankaya, Ankara, Turkey

stream instances and there is no prior knowledge about the number of classes. Therefore, clustering, being unsupervised is one of the most suitable data mining and data analysis methods for data streams. Border security using sensors, auto monitoring of surveillance cameras, internet of things (IoT) device tracking, real time patient tracking, stock market analysis, network intrusion detection, and earthquake forecasting systems are among the applications of data stream clustering.

Data clustering, is the task of grouping instances such that the instances in the same group are similar to each other and the instances in different groups are dissimilar according to the properties of the instances. Hence, the objective of clustering is to minimize intra-cluster distance and maximize inter-cluster distance. However, data stream clustering differs from traditional clustering in many aspects and it has several challenging issues. Data stream instances can be read only once, in a certain order, and must be processed in a short time interval, before the next instance is received. Data streams can not be stored, only a synopsis of the stream is stored, if required. Table 1 gives the comparison of stream clustering with traditional clustering (Mousavi et al. 2015; Gaber et al. 2009).

Data stream instances may evolve over time, and this is called *concept drift*. Concept drift is the unforeseen change in the properties of the input data instances. For the case of the traditional data, the whole dataset is available and properties of the instances do not change during the processing of the data. This makes the concept drift a data stream specific challenge. A data stream clustering algorithm should detect and adopt concept drift for more accurate results. According to the occurrence style, different types of concept drift exist.

Similar to traditional data, data streams may include outliers. To achieve better performance, outliers in the data streams should be detected, interpreted and possibly removed. In data streams, it is not easy to mark an instance as outlier, because a dissimilar instance might be the first sample of a new, previously unseen cluster, i.e. it might be a precursor of a concept drift. Moreover, a dissimilar instance might be marker of an anomaly, which is very valuable for anomaly detection systems.

Data stream clustering algorithms use special data structures to keep synopsis of the input data, since it is not possible to store the whole data. Storing agglomerative sum or storing only representative samples of the data are two popular alternative structures. Moreover, users are often interested in the most recent data instances rather than the previous ones. This situation creates a requirement of obsolescence for previous data instances. In data stream clustering, it is solved by time window models.

Most of the data stream clustering algorithms use a two phase approach (Silva et al. 2013). In *online phase* which is also called as data abstraction phase, a synopsis of the data stream is generated and stored in specialized data structures. Synopsis of the data stream is updated when a new instance is received. Therefore the synopsis always remains

Table 1 Comparison of stream clustering with traditional clustering

Stream clustering	Traditional clustering
Real time processing	Offline processing
Data arrives on the fly	All data are ready
Only single pass on data is possible	Multiple passes are possible
Data are not feasible to be stored	Data are suitable to be stored
Only synopsis of the data is stored	All raw data are stored
Approximate results are accepted	Accurate results are expected

up-to-date. *Offline phase*, called also as clustering phase, runs periodically or whenever the user requests. In this phase, the final clustering is performed over the generated data synopsis. There also exist several fully online data stream clustering algorithms, which re-cluster the data for every new instance and keep an up-to-date clustering result. Among fully online stream clustering algorithms are DPClust (Xu et al. 2017), CEDAS (Hyde et al. 2017), DBIECM (Zhang et al. 2017), FEAC-Stream (Andrade Silva et al. 2017) and Adaptive Stream *k*-means (Puschmann et al. 2017).

For the evaluation of data stream clustering, traditional techniques are still valid and they are commonly used. A relatively new concept *edge computing* (Shi et al. 2016; Shi and Dustdar 2016; Satyanarayanan 2017) is the technique to process the produced data on several edge nodes that are close to the connected devices, instead of a single central system. It is also an interest arousing novel concept, however it is out of scope of this study. We examine central data stream clustering concept that runs on a single center for the whole system.

In this manuscript, Sect. 2 is devoted to issues in data stream clustering. We give information about some mechanisms of stream clustering, which are data structures, time window models, concept drift and outlier detection methods. In Sect. 3, we give brief information about the categories of stream clustering algorithms. Moreover, we examine seven most recent data stream clustering algorithms that are not mentioned in the previous surveys in more detail and explain them one by one. We make a comparative review of the examined algorithms and highlight their advantages and disadvantages against each other in Sect. 4. We summarize the open problems about data stream clustering in Sect. 5. We indicate popular stream data repositories and datasets, stream processing tools and stream processing platforms in Sects. 6, 7 and 8 respectively, before concluding the study in Sect. 9.

2 Concepts in data stream clustering

The information given here that are the basic concepts used in data stream clustering facilitates explaining the recent data clustering algorithms analyzed in Sect. 3.

2.1 Concept drift

Concept drift is the unforeseen change in statistical properties of data stream instances over time. There are four types of concept drift: sudden, gradual, incremental and recurring (Ramirez-Gallego et al. 2017).

- *Sudden concept drift:* Between two consecutive instances, the change occurs at once, and after this time only instances of the new class are received. An instance that has properties of the previous class never arrives again. A data stream containing sudden concept drift might look like as follows, where different colors indicate different classes.

  $S = \{\ldots, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}, \mathbf{x_{11}}, \mathbf{x_{12}}, \ldots\}$
- *Gradual concept drift:* The number of instances belonging to the previous class decreases gradually while the number of instances belonging to the new class increases over time. During a gradual concept drift, instances of both previous and new classes

are visible. A data stream containing gradual concept drift might look like as follows, where different colors indicate different classes.

$$S = \{\ldots, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}, \mathbf{x_{11}}, \mathbf{x_{12}}, \ldots\}$$

- *Incremental concept drift:* Data instances belonging to the previous class evolves to a new class step by step. After the concept drift is completed, the previous class disappears. The instances that arrive during the concept drift are of transitional forms and they do not have to belong to either of the classes. A data stream containing incremental concept drift might look like as follows, where different colors indicate different classes.

$$S = \{\ldots, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}, \mathbf{x_{11}}, \mathbf{x_{12}}, \ldots\}$$

- *Recurring concept drift :* The data instances change between two or more statistical characteristics several times. Neither of the classes disappears permanently but both of them arrive in turns. A data stream containing recurring concept drift might look like as follows, where different colors indicate different classes.

$$S = \{\ldots, \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}, \mathbf{x_6}, \mathbf{x_7}, \mathbf{x_8}, \mathbf{x_9}, \mathbf{x_{10}}, \mathbf{x_{11}}, \mathbf{x_{12}}, \ldots\}$$

Creation of new clusters, disappearance or evolution of existing clusters are all examples of concept drift. Concept drift may also affect the cluster boundaries. If the cluster boundaries are modified, it is called *real concept drift* while in the other case, it is called *virtual concept drift*. There exist several studies in the literature for concept drift detection. Gama et al. (2014) have a comprehensive survey on concept drift detection.

2.2 Data structures for data streams

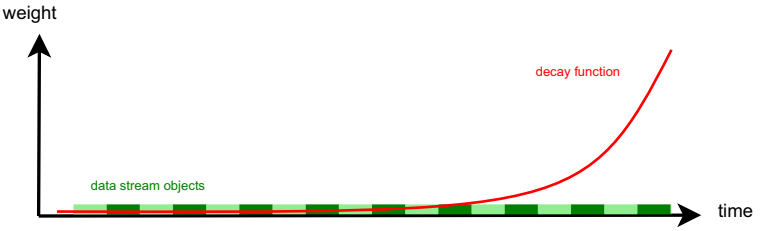
In data stream processing, it is not possible to store the whole input data because data streams are infinite and all existing processing systems have main memory constraint. Therefore, only a synopsis of the input stream is stored and this situation makes it essential to develop special data structures that enables to incrementally summarize the input stream. Four most commonly used data structures are *feature vectors, prototype arrays, coreset trees* and *grids*. Feature vectors keep the summary of the data instances, prototype arrays keep only a number of representative instances that exemplify the data, coreset trees keep the summary in a tree structure and grids keep the data density in the feature space (Silva et al. 2013; Ghesmoune et al. 2016; Mansalis et al. 2018).

2.3 Time window models

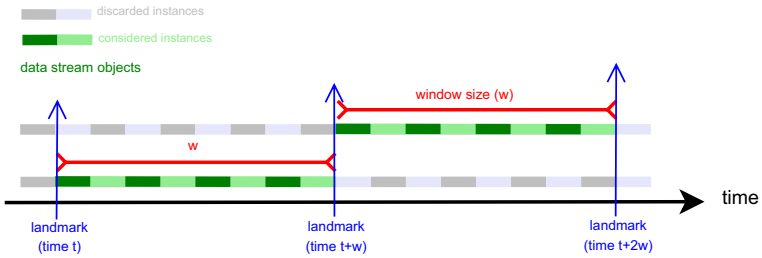
In data stream processing, it is more efficient to process recent data instead of the whole data. Different window models are developed for this purpose. There are three different window models, which are damped window, landmark window and sliding window models. These window models are presented in Fig. 1.

2.3.1 Damped window

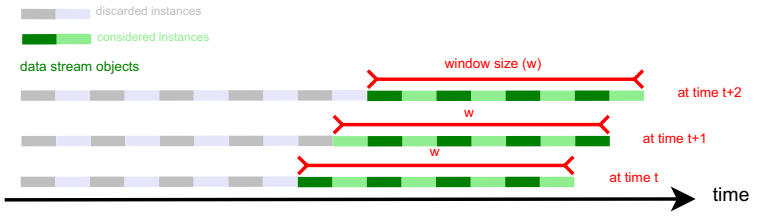
In damped window model, recent data have more weight than the older data. The most recent instance has the most weight and the importance of the instances decreases by time. This method is usually implemented using decay functions which scale down the weight of the instances, depending on the time passed since the instance is received. One of such functions is $f(t) = 2^{-\lambda t}$, where t is the time passed and λ is the decay rate. Higher decay



(a) Damped window model.



(b) Landmark window model.



(c) Sliding window model.

Fig. 1 Time window models

rate in the function means a more rapid decrease in the value. Fig. 1a demonstrates the damped window model.

2.3.2 Landmark window

In landmark window model, the whole data between two landmarks are included in the processing and all of the instances have equal weight. Amount of data that belong to a single window is called *window length* and usually indicated by w. Window length can be defined as instance count or elapsed time. In landmark window method, consecutive windows do not intersect and the new window just begins from the point the previous window ends. According to this definition, data instances belong to a window are calculated using Eq. 1 and window number of a data instance is calculated using Eq. 2 where w is window length, x_i is ith instance and W_m is mth window. Indexes i and m start with zero. Figure 1b shows the landmark window model.

$$W_m = [x_{m*w}, \ldots, x_{(m+1)*w-1}] \tag{1}$$

$$m = \left\lfloor \frac{i}{w} \right\rfloor \tag{2}$$

2.3.3 Sliding window

In sliding window model, the window swaps one instance at each step. The older instance moves out of the window, and the most recent instance moves in to the window by FIFO style. All instances in the window have equal weight and consecutive windows mostly overlap. Window length is a user defined parameter and should be decided according to the input data. Figure 1c describes this window model and data instances belong to a window are calculated using Eq. 3 where w is window length, x_i is ith instance and W_m is m *th* window. Indexes i and m start with zero.

$$W_m = [x_m, \ldots, x_{(m+w-1)}] \tag{3}$$

2.4 Outlier detection

Outlier is a data instance that seems to be different from the remaining of the data. Either it does not belong to any of the clusters, or it belongs to a cluster whose cardinality is far less than other clusters. Let C_i be i *th* cluster, $|C_i|$ be cardinality of C_i and k be cluster count, if $|C_j| << \frac{1}{k}\sum_{i=1}^{k} |C_i|$, then C_j is treated as outlier. There exist several definitions for the outliers in the literature (Modi and Oza 2017). An outlier can occur because of malicious activities, instrumental errors, transmission problems, data collection problems or similar (Merino 2015). In data mining, outliers negatively affect the processing accuracy, because of that, outlier detection has a crucial importance in data mining. It is possible to benefit from several existing surveys (Modi and Oza 2017; Chauhan and Shukla 2015; Souiden et al. 2016; Thakkar et al. 2016; Bhosale 2014; Sadik and Gruenwald 2014) about outlier detection in data streams. Thakkar et al. (2016) have classified outlier detection techniques in four main groups in their survey.

- *Statistical outlier detection* methods make an assumption about the data distribution. Taking the distribution into account, data instances that have a low probability to be generated, are marked as outliers. Statistical outlier detection methods are divided into two categories: parametric methods and non-parametric methods. In parametric methods, a distribution model of the data is assumed before starting, according to the parameters. This method is not suitable for data streams, since the entire dataset is not available in streams and the distribution model may change over time. In non-parametric methods, no distribution model is assumed a priori; instead, the distribution model is learned from the original data instances. This property makes non-parametric statistical outlier detection methods adoptable for data streams.
- *Distance based outlier detection* methods (Chauhan and Shukla 2015) use neighbor count of the instance to decide whether it is an outlier or not. Two parameters R and k play the main role. If the data instance has less than k neighbors in a distance of R, then it is marked as an outlier. A distance measure (or a similarity measure) must

be defined. No domain knowledge is required and no distribution model assumption is done. Therefore, distance based outlier detection methods are suitable for data streams. However, they are not effective for high dimensional data streams.

- *Density based outlier detection* methods compare the density around the data instance to the density around its neighbors. If the instance has a density around it similar to its neighbors, then it is not an outlier. Otherwise it is considered as an outlier. This group of methods, are more effective than distance based methods, however they have a higher computational complexity.
- In *Clustering based outlier detection* methods (Bhosale 2014) data instances that do not belong to any clusters, or far away from their cluster centroids, are potential outliers. Moreover, outliers may belong to a sparse or small cluster that is not close to other clusters. Ordinary data instances are expected to belong to large, dense clusters and they are relatively close to cluster centroids.

Although real-time analysis of data streams is a more recent research subject, it has many similarities with the analysis of time series data which has been studied for longer time and more abundant in the literature. Especially outlier detection in data streams is very similar to anomaly detection in time series data analysis (Kong et al. 2019; Christodoulou et al. 2018; Keogh et al. 2005).

3 Stream clustering algorithms

There exist several data stream clustering algorithms in the literature (Silva et al. 2013; Mousavi et al. 2015; Alam et al. 2016; Kumar 2016; Ding et al. 2015; Ghesmoune et al. 2016; Ramirez-Gallego et al. 2017; Carnein et al. 2017; Yasumoto et al. 2016; Nguyen et al. 2015; Fahy et al. 2018; Mahdiraji 2009; Aggarwal 2013). Data stream clustering algorithms can be categorized following the classification that is used for traditional (batch) clustering algorithms. This categorization is given in Fig. 2 and it consists of five main classes: hierarchical based, partitioning based, density based, grid based and model based clustering. We first give brief information about these categories and related algorithms and we then examine seven most recent data stream clustering algorithms in more detail.

- *Hierarchical algorithms* use the dendrogram data structure. Dendrogram is binary tree based, and it is useful to summarize and visualize the data. Hierarchical algorithms are divided in two: agglomerative and divisive. Agglomerative algorithms start with the assumption every instance is a cluster itself, and merge the instances to create clusters step by step. On the other hand, divisive algorithms start assuming a single cluster contains whole data, and divide the clusters into smaller clusters in each step. Hierarchical algorithms have an informative output, which is the dendrogram. However, they have high complexity and they are sensitive to the outliers. Among hierarchical algorithms are BIRCH (Zhang et al. 1996), CHAMELEON (Karypis et al. 1999), ODAC (Rodrigues et al. 2006), E-Stream (Udommanetanakit et al. 2007) and HUE-Stream (Meesuksabai et al. 2011) (Mousavi et al. 2015; Kumar 2016).
- *Partitioning based algorithms* split the data instances into a predefined number of clusters, based on similarity (or distance) to the cluster centroids. Number of clusters should be predefined in these algorithms, and only hyper-spherical clusters can be determined. Partitioning based algorithms have an easy implementation in general.

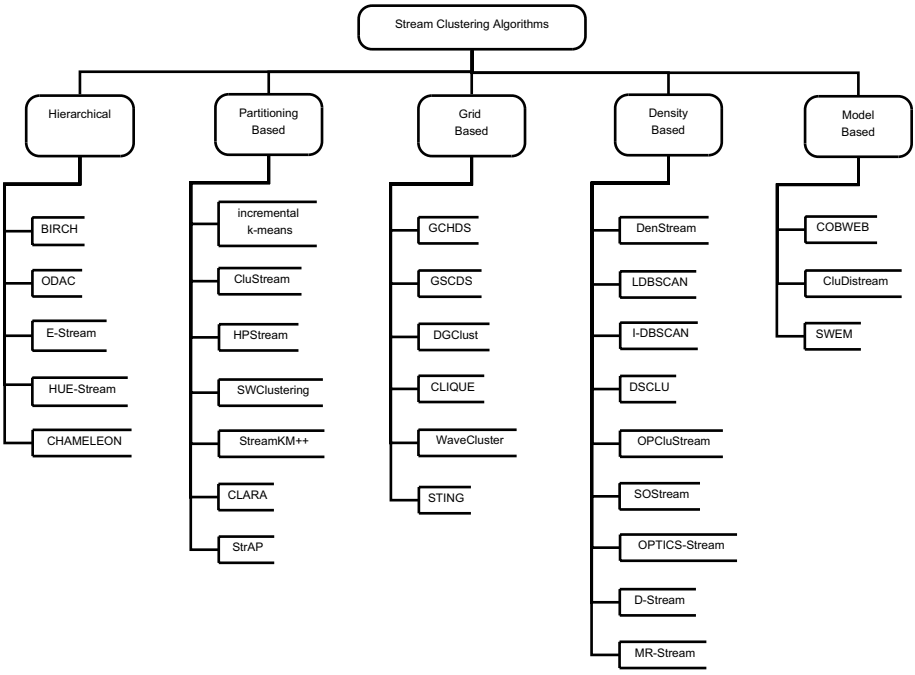


Fig. 2 Classification of data stream clustering algorithms

StreamLSearch (O'Callaghan et al. 2002), incremental *k*-means (Ordonez 2003), CluStream (Aggarwal et al. 2003), HPStream (Aggarwal et al. 2004), SWClustering (Zhou et al. 2008), StreamKM++ (Ackermann et al. 2012), strAP (Zhang et al. 2014) and CLARA (Kaufman and Rousseeuw 1990) are partitioning based algorithms (Mousavi et al. 2015; Kumar 2016; Ghesmoune et al. 2016).

- *Grid based algorithms* use grid data structure. The workspace is divided into a number of cells, in a grid structure, and each instance is assigned to a cell. Then, the grid cells are clustered, according to their density. In grid based algorithms, the run time does not depend on input data count. Therefore, grid based algorithms are fast algorithms. Moreover, they are robust to noise and are able to find arbitrary shaped clusters. However, since their complexity depends on the number of the dimensions of the data, grid based algorithms are more suitable for low dimensional data. Furthermore, they need a predefined grid size. GCHDS (Lu et al. 2005), GSCDS (Sun and Lu 2006), DGClust (Gama et al. 2011), CLIQUE (Agrawal et al. 1998), WaveCluster (Sheikholeslami et al. 2000) and STING (Wang et al. 1997) are all grid based algorithms (Mousavi et al. 2015). D-Stream (Chen and Tu 2007) and MR-Stream (Wan et al. 2009) are classified as grid based by Ghesmoune et al. (2016), despite being classified as density based by Mousavi et al. (2015).
- *Density based algorithms* keep summary of input data in large number of micro-clusters. Micro-cluster is a set of data instances that are very close to each other. Synopsis of a micro-cluster is kept with a feature vector. Then these micro-clusters are merged and formed final clusters according to density reachability and density connectivity concepts. These terms are defined as follows. If the distance between two micro-clusters is less than or equal to the sum of their radii, then they are directly density

reachable. If any adjacent two clusters in a set of micro-clusters are directly density reachable, then the set of micro-clusters is density reachable. All micro-clusters that are density reachable to each other, are density connected (Yin et al. 2017). Density based algorithms are able to find arbitrary shaped clusters and detect number of clusters. They are robust to noise as well. However, several parameters have to be selected and there are problems in finding multi-density clusters. Incremental-DBSCAN (Ester et al. 1998), LDBSCAN (Duan et al. 2006), DenStream (Cao et al. 2006), rDenStream (Liu et al. 2009), DSCLU (Namadchian and Esfandani 2012), OPCluStream (Wang et al. 2012), SOStream (Isaksson et al. 2012), OPTICS-Stream (Tasoulis et al. 2007), D-Stream (Chen and Tu 2007) and MR-Stream (Wan et al. 2009) are classified as density based (Mousavi et al. 2015; Ghesmoune et al. 2016).

- *Model based algorithms* find the data distribution model that fit best to the input data. One of the important advantages of model based algorithms is their property of noise robustness. However, their performance strongly depends on the selected model. COBWEB (Fisher 1996), CluDistream (Zhou et al. 2007) and SWEM (Dang et al. 2009) are examples of model based algorithms (Mousavi et al. 2015).

Advantages and disadvantages of clustering algorithms are summarized in Table 2 (Mousavi et al. (2015); Mansalis et al. (2018); Ghesmoune et al. (2016)).

The aforementioned data stream clustering algorithms have already been reviewed in the previous surveys. On the other hand, the algorithms given below have not been analyzed elsewhere, to the best of our knowledge. We give the main flow of the algorithms, show their evaluation results and present the complexity analysis. During the complexity analysis, we ignore the Euclidean distance calculation complexity, which is $O(d)$, because this is the common practice in the literature. Moreover, this calculation is done in every data stream clustering algorithm, thus ignoring it does not change the comparison. However, any other data dimension related complexity is included in the analysis. Not surprisingly, complexity of partitioning based algorithms is a function of k and complexity of density based algorithms is a function of *micro cluster count*.

We start with Adaptive Streaming k-Means and FEAC-Stream both of which are partitioning based online algorithms. We then examine MuDi-Stream, which is a density based, online–offline algorithm. CEDAS is a density based online algorithm. Improved Data Stream Clustering Algorithm is a density based, online–offline algorithm.

Table 2 Advantages and disadvantages of clustering algorithms based on traditional categorization

Algorithm	Advantages	Disadvantages
Hierarchical	Informative output (dendrogram)	High complexity Outlier sensitivity
Partitioning	Easy implementation	Predefined number of clusters Only hyper-spherical clusters
Grid-based	Arbitrary shaped clusters Fast execution time Noise robustness	Predefined grid size Only low dimensional data
Density-based	Arbitrary shaped clusters Noise robustness	Multidensity cluster difficulties Many predefined parameters
Model-based	Noise robustness	Strong dependency on the model

DBIECM, the only distance based algorithm is fully online. Note that the previous, classical classification does not include the distance based algorithms, probably, because there are not many examples of distance based algorithms. Finally, we examine I-HASTREAM, a density based hierarchical, online–offline algorithm. Figure 3 shows the main characteristics of the examined algorithms. Although, *ant colony optimization* methods are also being used by a number of data stream clustering algorithms (Fahy et al. 2018), they are not evaluated at this time. Moreover, being an active research area, there also exist several recent data stream clustering algorithms that are not evaluated in this manuscript (Din et al. 2020; Bezerra et al. 2020; Kim and Park 2020).

3.1 Adaptive streaming *k*-means (2017)

Adaptive streaming *k*-means is an online, partitioning based data stream clustering algorithm proposed by Puschmann et al. (2017). In general, partitioning based clustering algorithms need k as an input parameter, and these algorithms have difficulties to adapt concept drift in the input data. In this algorithm, Puschmann et al. claim to overcome these two main problems.

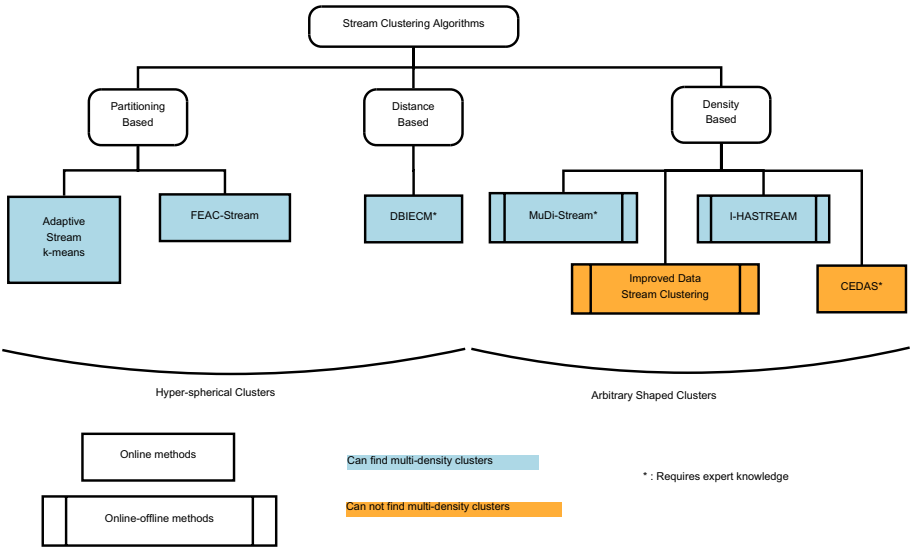


Fig. 3 Recent data stream clustering algorithms

Algorithm 1 streamingKMeans (S, l)

Input: S : the input data stream
Input: l : length of data sequence used for initialization
1: % Initialization phase
2: **for** *candidateCentroids* **in** *determineCentroids*(l number of data instances) **do**
3: run *kmeans* with *candidateCentroids*
4: calculate *silhouette coefficient* of the *kmeans* result
5: **end for**
6: keep *centroids* of the best clustering
7: % Continuous clustering phase
8: **loop**
9: **if** *changeDetected* on the input stream **then**
10: *re-initialize* the algorithm by running again the initialization phase
11: **end if**
12: run *kmeans* with last found, best centroids
13: **end loop**

Algorithm 1 shows the main flow of adaptive streaming k-means algorithm. k-means algorithm and silhouette coefficient calculation function are assumed to be already implemented. The algorithm is composed of two main phases, which are *initialization phase* and *continuous clustering phase*. In the initialization phase, l number of data instances are accumulated. Then groups of candidate centroids are determined at line 2. In function *determineCentroids*, in order to find k and determine candidate centroids, probability density function (PDF) of the data is calculated using kernel density estimation (KDE) (Parzen 1962; Rosenblatt 1956). All directional changes in the shape of PDF curve, are accepted as signs of beginning of a new region. Here the region can be defined as the area between two consecutive directional changes of the PDF curve. Number of regions is considered as a candidate k and centers of these regions are considered as candidate initial centroids. This process is pursued for each feature of the data separately. Because different features generally show different distributions, more than one k values, and different candidate centroids are found.

After finding candidate k values, clustering is performed for a set of k values where $k \in [k_{min}, k_{min} + k_{max}]$. The *for loop* at lines 2–5 is executed for these values of k and candidate centroids. Clustering results of different k values are compared according to silhouette coefficient, and best k is selected with its corresponding centroids.

The *loop* at lines 8–13 runs for continuous clustering phase. Checking for a concept drift (see Sect. 2.1) is performed at line 9. If no concept drift occurs, clustering of the input data proceeds, at line 12. However if a concept drift exists, k and centroids are recalculated (the algorithm is re-initialized) at line 10, and then clustering continues at line 12 with new k and centroids.

For concept drift detection, standard deviation and mean of the input data are stored during the execution. The algorithm tracks how these two values change over time and predicts a concept drift according to the change. When a concept drift is predicted, current cluster centroids are no longer valid. In such a case the concept drift is realized at line 9 and a reinitialization is triggered at line 10. Using this mechanism, the algorithm captures the concept drift and adapts itself to the input stream.

A limitation of this algorithm, being k-means based, only hyper-spherical clusters can be detected. Indeed, the authors indicate that k-means is used as the underlying clustering technique to clarify the approach, and the concepts of the approach can be applied to different clustering techniques.

Evaluation: Adaptive streaming k-means algorithm is evaluated against CluStream and DenStream algorithms, according to silhouette coefficient. Artificial datasets with three to five dimensions, that include concept drift, are used as input data streams. Clustering quality improvement of the adaptive streaming k-means algorithm is 13–40% with respect to CluStream. DenStream gives a better clustering quality for one of the datasets, for short time intervals during the execution. However, for the other datasets, clustering quality improvement of the adaptive streaming k-means algorithm is up to 280% with respect to DenStream. Furthermore, the algorithm is evaluated with real traffic data, against the non-adaptive technique, in which, the centroids are never recalculated. Adaptive streaming k-means algorithm achieves an improvement up to 31% in clustering quality when they are compared over the course of one day. When they are compared over the course of one week, clustering quality improvement of the adaptive streaming k-means is 12% on average.

Complexity Analysis: Let l be the length of the initial data sequence, and d be the data dimension. Complexity of estimating k for a single dimension is $O(l)$, because this part goes along the PDF and it has a length equal to the data length. Since this estimation is performed for all dimensions, total k estimation complexity becomes $O(d \cdot l)$. After determining initial centroids running k-means takes $O(d \cdot k \cdot cs)$ since no iterations of the algorithm are needed, where cs is the number of different centroid sets. Assigning a newly received data instance to the nearest cluster during the online phase is $O(k)$. As a result, total worst case complexity of the algorithm is $O(k) + O(d \cdot l) + O(d \cdot k \cdot cs)$, which equals to $O(d \cdot l) + O(d \cdot k \cdot cs)$

3.2 FEAC-Stream (2017)

Fast evolutionary algorithm for clustering data streams (FEAC-Stream) is an evolutionary algorithm for clustering data streams with a variable number of clusters, proposed by Andrade Silva et al. (2017). FEAC-Stream is a k-means based algorithm, which estimates k automatically using an evolutionary algorithm. Being fully online, FEAC-Stream does not store synopsis of the data, instead maintains the final clustering result. During the execution, clustering quality is tracked using the Page-Hinkley (PH) (Mouss et al. 2004) test and if the quality falls down, the algorithm adjusts itself.

Algorithm 2 FEAC-Stream (S, l, λ, α, *iter*)

Input: S : the input data stream
Input: l : length of data sequence used for initialization
Input: λ : decay rate
Input: α : weight threshold
Input: *iter* : k-means iteration count
1: % Initialization phase
2: Estimate k with l number of data instances, using *evolutionary algorithm*
3: state = *normal*
4: % Continuous clustering phase
5: **loop**
6: Read next data instance x from data stream S
7: Add x to the nearest cluster
8: Calculate weight of all clusters
9: Delete *low weighted* clusters
10: PH_{val} = Calculate PH test.
11: **if** $PH_{val} >$ *warning threshold* **then**
12: state = *warning*
13: **end if**
14: **if** state **is** *warning* **then**
15: Add x to buffer B
16: **end if**
17: **if** $PH_{val} >$ *alarm threshold* **then**
18: Estimate k with data instances in buffer B, using *evolutionary algorithm*
19: state = *normal*
20: **end if**
21: **end loop**

Algorithm 2 shows the main flow of FEAC-Stream algorithm. PH test function and the evolutionary algorithm are assumed to be already implemented. The algorithm is composed of two main phases, which are *initialization phase* and *continuous clustering phase*. In the initialization phase, l number of data instances are accumulated. Then k and initial clustering is calculated using an evolutionary algorithm, at line 2 and state is set to normal, at line 3. In this evolutionary algorithm, clustering is performed using k-means with a maximum of *iter* iterations. Simplified silhouette coefficient is used as the fitness function, k is selected randomly such that $k \in [2, \sqrt{l}]$ and initial centroids are also selected randomly from the input data instances.

After clustering the initial l data instances in the initialization phase, the loop at lines 5-21 is executed for continuous clustering phase. When a new data instance is received, it is added to the nearest cluster at line 7. Weight of all clusters are calculated and low weighted clusters are deleted at line 8 and line 9, respectively. After that, PH test is calculated at line 10 and it is compared to warning and alarm threshold values. When PH test value exceeds the warning threshold, the algorithm enters to warning state. In warning state, clustering process continues and received data instances are stored in a buffer, at line 15. If PH test value exceeds alarm threshold, this means a concept drift (see Sect. 2.1) occurs and current clusters are not valid anymore. When PH test signals an alarm state, it also automatically selects samples from the input data instances that reflects a new partitioning. In such a case, FEAC-Stream clusters the data instances stored in the buffer with the evolutionary algorithm, at line 18 and sets the state back to normal, at line 19. In the evolutionary algorithm, clustering is performed using k-means with a maximum of *iter* iterations. Simplified silhouette coefficient is used as the fitness function, k and initial centroids are specified by the PH test. FEAC-Stream uses damped window model, which is described in Sect. 2.3.1.

Being k-means based, only hyper-spherical clusters can be detected by FEAC-Stream. Moreover, clustering quality of FEAC-Stream strongly depends on the user defined parameters. FEAC-Stream requires three parameters which are length of data sequence used for

initialization (l), decay rate (λ) for damped window model and minimum weight threshold (α). These parameters strongly affect the clustering quality and they are directly dependent to the input data. Because of that, FEAC-Stream requires an expert knowledge about the input data. Iteration count of k-means (*iter*) and generation count of evolutionary algorithm are used as hard coded. Moreover, warning and alarm threshold values of PH test are calculated automatically by the PH test.

Evaluation: FEAC-Stream is evaluated against CluStream-OMRk, CluStream-BkM, StreamKM++-OMRk and StreamKM++-BkM, where CluStream and StreamKM++ are the stream clustering algorithms with fixed k, while BkM and OMRk are k estimating algorithms. Both real and artificial datasets are used for the evaluation. Real datasets are network intrusion detection dataset, forest cover type dataset and localization data for person activity dataset. Adjusted Rand Index (ARI) is used as clustering quality metric in artificial datasets. While all mean ARI results are very close to each other (0.97–0.99), FEAC-Stream has the lowest execution time. Its execution time is less by; 25% than StreamKM++-BkM, 58% than StreamKM++-OMRk, 91% than CluStream-BkM and nearly 93% than CluStream++-OMRk. Furthermore, FEAC-Stream successfully reacts to concept drifts and accordingly estimates k. For network intrusion detection dataset, simplified silhouette (SS) coefficient is used to compare the clustering quality. Again all algorithms give very good and very close (0.90 - 0.92) SS values and still FEAC-Stream gives the best execution time. Its execution time is less by; 65% than StreamKM++-BkM, 87% than StreamKM++-OMRk, 97% than CluStream-BkM and 98% than CluStream++-OMRk. For the other real datasets as well, algorithms have the same running time ordering. These results also show that, StreamKM++ is faster than CluStream and BkM is faster than OMRk.

Complexity Analysis: Let l be the length of the initial data sequence, *gen* is generation count of evolutionary algorithm and *iter* is the iteration count of k-means. In the initialization phase, k is randomly selected as $k \in [2, \sqrt{l}]$. Thus, complexity of initialization phase is $O(gen \cdot iter \cdot \sqrt{l})$. Online maintenance of the algorithm requires a complexity of $O(k)$. When a concept drift occurs, the algorithm is reinitialized by running evolutionary algorithm again. However k and centroids are decided by PH test. Therefore, reinitialization requires a complexity of $O(gen \cdot iter \cdot k)$. As a result, total worst case time complexity of FEAC-Stream is $O(k) + O(gen \cdot iter \cdot k)$, which equals to $O(gen \cdot iter \cdot k)$.

3.3 MuDi-Stream (2016)

Multi density data stream clustering algorithm (MuDi-Stream) is a two phase data stream clustering algorithm proposed by Amini et al. (2016). Main objective of MuDi-Stream is to improve the clustering quality on data streams with multi density clusters. Note that density based algorithms usually have problems with clusters of different densities because of the static density threshold they use. MuDi-Stream customizes the density threshold for each cluster and overcomes the problem of multi density clusters. MuDi-Stream is a hybrid algorithm based on both density based and grid based approaches. Input data instances are clustered in a density based approach and outliers are detected using grids. For data synopsis core mini-clusters are used. Core mini-clusters are specialized feature vectors (see Sect. 2.2), they keep weight, center, radius and the maximum distance from an instance to the mean. In the online phase core mini-clusters are created and kept up to date for each new data instance. In the offline phase final clustering is executed over the core mini-clusters.

Algorithm 3 MuDi-Stream online phase (S, α, λ, $gridGranularity$, G)

Input: S : the input data stream
Input: α : density threshold
Input: λ : decay rate
Input: $gridGranularity$
Input: G : total density grids for all dimensions
1: Initialize the grid structure
2: **loop**
3: Read next data instance x from data stream S
4: cmc_s = Find the nearest cmc to x
5: **if** cmc_s ***involve*** x **then**
6: Add x to cmc_s
7: **else**
8: Map x to the gird
9: **if** Updated grid is *dense enough* **then**
10: Create a cmc from updated grid
11: **end if**
12: **end if**
13: **if** It is pruning period **then**
14: Remove *low weighted* grids
15: Remove *low weighted* cmcs
16: **end if**
17: **end loop**

Algorithm 3 shows the main flow of online phase of MuDi-Stream. When a new data instance is received, it is tried to be added to an existing core mini-cluster. For this purpose, the nearest core mini-cluster is found at line 4 and it is checked whether nearest core mini-cluster can involve this data instance or not, at line 5. If the nearest core mini-cluster is *large enough*, the data instance is added to the nearest core mini-cluster at line 6. Otherwise, the data instance is mapped into the gird in the outlier buffer, at line 8. When a data instance is mapped to a grid, density of this grid is checked and if it is *dense enough* (more than the density threshold), a new core mini-cluster is created from this grid i.e. the grid is converted to a core mini-cluster, at line 10. MuDi-Stream prunes both the grids in the outlier buffer and the core mini-clusters periodically. It is checked at line 13 whether it is pruning time or not. If it is pruning time, weight of grids and core mini-clusters are calculated according to current time, and then low weighted grids and core mini-clusters are pruned at line 14 and line 15 respectively. This pruning mechanism is an implementation of damped window model, which is described in Sect. 2.3.1.

Algorithm 4 MuDi-Stream offline phase (core mini-clusters)

```
Input: core mini-clusters
 1: Mark all cmcs as unvisited
 2: repeat
 3:     Randomly choose an unvisited cmc, called cmc_p
 4:     Mark cmc_p as visited
 5:     if cmc_p has neighbors then
 6:         Create new final cluster C
 7:         Add cmc_p to C
 8:         Add neighbors of cmc_p to C
 9:         for each cmc in C do
10:             if cmc is unvisited then
11:                 Mark cmc as visited
12:                 Add neighbors of cmc to C
13:             end if
14:         end for
15:     else
16:         Mark cmc_p as noise
17:     end if
18: until All cmcs are visited
```

Algorithm 4 shows the main flow of offline phase of MuDi-Stream. Initially all core mini-clusters are marked as unvisited, at line 1. After that, inside a loop, an unvisited core mini-cluster is randomly chosen at line 3 and marked as visited at line 4. If this core mini-cluster has no neighbors, it is marked as noise at line 16. If it has neighbors, a new final cluster is created with this core mini-cluster and its neighbors, at lines 6-8. After that, each unvisited core mini-cluster in the new created final cluster is marked as visited and its neighbors are added to the same final cluster, at lines 9-14. This loop continues until all core mini-clusters are marked as visited.

Damped window model is used, and arbitrary shaped, multi density clusters can be detected by MuDi-Stream. Moreover, MuDi-Stream is able to handle concept drift (see Sect. 2.1), noise and outliers. However it is not suitable for high dimensional data, which makes the processing time longer, because of the grid structure. Furthermore, clustering quality of MuDi-Stream strongly depends on input parameters density threshold (α), decay rate (λ) for damped window model and *grid granularity*. These parameters require an expert knowledge about the data.

Evaluation: MuDi-Stream is tested with two real (network intrusion detection and Landsat satellite) and six artificial datasets. It is compared to DenStream on a data stream with concept drifts, a multi density dataset and a multi density data stream with concept drifts. MuDi-Stream outperforms DenStream on all three types of input data, according to clustering quality (Purity, Normalized Mutual Information (NMI), Rand Index (RI), Adjusted Rand Index (ARI), Folkes and Mallow index (FM), Jaccard Index and F-Measure). Clustering quality improvement of MuDi-Stream is 10–100% with respect to DenStream, on different datasets.

Complexity Analysis: MuDi-Stream performs a linear search on core mini-clusters for each new data instance. Complexity of this linear search is $O(c)$ where c is the number of core mini-clusters. If the new data instance cannot be merged into existing core mini-clusters, it is mapped to the grid. Let G be total density grids for all dimensions, which is exponential to the number of dimensions. Space complexity of the grid is $O(log\ G)$ because the scattered grid are pruned during the execution. Moreover, time complexity of mapping a data instance to the grid is $O(log\ log\ G)$ because the list of the grids is maintained as a

tree. During the pruning, all core mini-clusters and grids are examined. This makes time complexity of pruning $O(c)$ for core mini-clusters and $O(log\ G)$ for grids. As a result, the overall time complexity of MuDi-Stream is $O(c) + O(log\ log\ G) + O(c) + O(log\ G)$, which equals to $O(c) + O(log\ G)$.

3.4 CEDAS (2016)

Clustering of evolving data streams into arbitrarily shaped clusters (CEDAS) is a fully online data stream clustering algorithm proposed by Hyde et al. (2017) CEDAS is a density based algorithm designed for clustering data streams with concept drifts (see Sect. 2.1), into arbitrary shaped clusters. Damped window model (see Sect. 2.3.1) is employed with a linear decay function instead of an exponential one. CEDAS keeps synopsis of the data in micro-clusters and creates a graph structure with the micro-clusters that surpass a user defined threshold. Graph structure, where nodes are the micro-clusters and edges are the connectivity between micro-clusters, keeps the up to date final clustering results.

Algorithm 5 CEDAS (S, α, λ, r_0)

Input: S : the input data stream
Input: α : density threshold
Input: λ : decay rate
Input: r_0 : micro-cluster radius
1: Initialize the micro-cluster structure
2: **loop**
3: Read next data instance x from data stream S
4: dis_{min} = Find the distance from x to the nearest micro-cluster center
5: **if** $dis_{min} < r_0$ **then**
6: Add x to the nearest micro-cluster
7: Energy of the updated micro-cluster = 1
8: **else**
9: Create new micro-cluster with x
10: Energy of the new micro-cluster = 1
11: **end if**
12: Reduce energy of all micro-clusters by λ
13: Remove *negative energy* micro-clusters
14: **if** micro-clusters are *changed* **then**
15: Update graph structure with micro-clusters that surpass α
16: **end if**
17: **end loop**

Algorithm 5 shows the main flow of CEDAS. When a new data instance is received, it is tried to be added to an existing micro-cluster. For that purpose, the distance from new data instance to the nearest micro-cluster is found at line 4 and it is checked whether this distance is less than the micro-cluster radius (r_0) or not, at line 5. Micro-cluster radius is a user defined, static parameter. If the distance is less than the radius, the data instance is added to the nearest micro-cluster, at line 6, and energy of this micro-cluster is set to 1 at line 7. Otherwise, a new micro cluster is created with this data instance, at line 9, and energy of the new micro-cluster is set to 1, at line 10. Energy of micro-clusters linearly fades on every cycle, with an amount of decay rate (λ), at line 12. The micro-clusters whose energy drop below zero are removed at line 13. Lastly, the graph structure is updated with the micro-clusters that surpass the density threshold (α), at line 15. Removed micro-clusters are removed from the graph structure also, and micro-clusters

reached the density threshold (α) added to the graph structure. Therefore, CEDAS creates final clustering results as fully online.

CEDAS is suitable for high dimensional data under favor of maintaining a graph structure where nodes are the micro-clusters and edges are the connectivity between micro-clusters. However, clustering quality of CEDAS strongly depends on the user defined parameters. CEDAS requires three parameters which are decay rate (λ), micro-cluster radius (r_0) and minimum density threshold (α). These parameters strongly affect the clustering quality and they are directly dependent to the input data. Because of that, CEDAS requires an expert knowledge about the input data.

Evaluation: CEDAS is tested with a data stream consisting of two Mackey-Glass time series, to see how it deals with concept drift, cluster separation, cluster merging and noise over time. Moreover, it is compared to CluStream and DenStream according to complexity, processing speed, cluster quality and memory efficiency. CEDAS, CluStream and DenStream are also compared with high dimensional data according to speed and accuracy. CEDAS successfully deals with concept drift. Noise negatively affects the clustering quality, however results are claimed to be still acceptable. Time measurements show that CEDAS is quite suitable for high dimensional data. Firstly CEDAS is compared against only online phases of DenStream and CluStream. For data with less than 10 dimensions, CEDAS is the slowest one. However, processing time of CEDAS stays nearly constant up to 10,000 dimensions. CluStream becomes slower than CEDAS after 10 dimensions and it consumes nearly 300 times more than CEDAS for 6000 dimensions. DenStream is faster than CEDAS up to 200 dimensions. For more than 200 dimension, DenStream becomes slower than CEDAS and consumes nearly 2 times more than CEDAS for 6000 dimensions. After that, CluStream and DenStream are run with a frequent offline phase, to generate near real time final clustering. In this situation CEDAS is the fastest algorithm for both low and high dimensional data. For 5 dimensional data, DenStream consumes 40 times and CluStream consumes 75 times more than CEDAS. For very high dimensional data, time consumption of DenStream grows faster than the others. When the data dimension is 3,000 CluStream consumes nearly 100 times and DenStream consumes nearly 650 times more than CEDAS. The other main advantage of CEDAS is memory efficiency. During the execution, DenStream reaches up to 800 micro-clusters at certain times, while CEDAS reaches up to 100 micro-clusters.

Complexity Analysis: For each new data instance, CEDAS performs a linear search on the micro-clusters. Complexity of this linear search is $O(c)$ where c is the number of micro-clusters. After that, energy of each micro-cluster is reduced, which also requires an $O(c)$ complexity. The last step, which updates the graph structure, is executed only when a new micro-cluster is created or removed. In worst case, all micro-clusters are visited, so worst case time complexity of this step is again $O(c)$. Therefore, the overall time complexity of CEDAS is $O(c)$.

3.5 Improved data stream clustering algorithm (2017)

Improved data stream clustering algorithm is a two phase, density based algorithm that is suitable for arbitrary shaped clusters, proposed by Yin et al. (2017). Main characteristic of this algorithm is adjusting threshold values automatically, according to the input data. This feature gets rid of the requirement of expert knowledge about the input data.

Algorithm 6 Improved data stream clustering online phase (S, l, λ)

Input: S : the input data stream
Input: l : length of data sequence used for initialization
Input: λ : decay rate
1: % Initialization phase
2: Run DBSCAN on l number of data instances
3: % Continuous clustering phase
4: **loop**
5: Read next data instance x from data stream S
6: Add x to the nearest *major micro-cluster* **OR**
7: Add x to the nearest *critical micro-cluster* **OR**
8: Create a new *micro-cluster* with x
9: **if** It is pruning period **then**
10: Remove *low weighted* major micro-clusters
11: Remove *low weighted* critical micro-clusters
12: **end if**
13: **end loop**

Algorithm 6 shows the main flow of online phase of improved data stream clustering algorithm. DBSCAN algorithm is assumed to be already implemented. The algorithm is composed of two main phases, which are *initialization phase* and *continuous clustering phase*. In the initialization phase, l number of data instances are accumulated and clustered using DBSCAN, at line 2. Major micro-clusters and critical micro-clusters are created as output of DBSCAN algorithm. Major micro-clusters have high densities and will be included in the final clustering process. Critical micro-clusters have low densities and treated as potential outliers. In the continuous clustering phase, when a new data instance is received, it is tried to be added to the nearest major micro-cluster, at line 6. If nearest major micro-cluster is not suitable, this time the new data instance is tried to be added to the nearest critical micro-cluster, at line 7. If neither of them is suitable, a new micro-cluster is created with the new data instance, at line 8. Damped window model (see Sect. 2.3.1) is used and low weighted major and critical micro-clusters are removed periodically, at line 10 and line 11 respectively. Threshold values of major and critical micro-clusters are global parameters in the algorithm, instead of being specific to each micro-cluster. However they are dynamic parameters and continuously updated during the execution.

Algorithm 7 Improved data stream clustering offline phase (micro-clusters)

Input: micro-clusters
1: Mark all mcs as unvisited
2: **repeat**
3: Randomly choose an unvisited mc, called mc_p
4: **if** mc_p **is** *major micro-cluster* **then**
5: Find all micro-clusters *density reachable* to mc_p
6: Create a final cluster by them.
7: **else if** mc_p **is** *critical micro-cluster* **then**
8: Continue the next cycle
9: **end if**
10: **until** All mcs are visited

Algorithm 7 shows the main flow of offline phase of improved data stream clustering algorithm. Initially all micro-clusters are marked as unvisited, at line 1. After that, inside a loop, an unvisited micro-cluster is chosen randomly at line 3. If the selected micro-cluster is a major micro-cluster, all micro-clusters that are density reachable to this micro-cluster are found and a new final cluster is created by them, at line 5 and line 6. If the selected

micro-cluster is a critical micro-cluster, then the execution continues with the next cycle, at line 8. When all micro-clusters are visited, the offline phase completes. The term density reachable is defined as follows. If the distance between a micro-cluster and another major micro-cluster is less than or equal to the sum of their radii, then they are directly density reachable. If any adjacent two clusters in a set of micro-clusters are directly density reachable, then the set of micro-clusters is density reachable (Yin et al. 2017).

Evaluation: Improved data stream clustering algorithm is evaluated against DenStream algorithm, using the network intrusion detection dataset. Clustering quality improvement of the improved data stream clustering algorithm is 2–7% with respect to DenStream. Moreover, Yin et al. (2017) claims that this algorithm has a better time and spatial complexity, compared with traditional clustering algorithms, however no measurement results are shared.

Complexity Analysis: Let l be the length of the initial data sequence. Complexity of the initialization equals to complexity of DBSCAN, which is $O(l \cdot log\ l)$ in average and $O(l^2)$ in worst case. In the continuous clustering phase, a linear search is performed on micro-clusters for each new data instance. Complexity of this linear search is $O(c)$ where c is the number of micro-clusters. When it is pruning period, pruning task is executed for each micro-cluster one by one and this also requires a complexity of $O(c)$. Therefore, the total worst case complexity is $O(c) + O(c)$, which equals to $O(c)$.

3.6 DBIECM (2017)

DBIECM is an online, distance based, evolving data stream clustering algorithm proposed by Zhang et al. (2017). DBIECM is the only example of distance based clustering algorithms in this survey. DBIECM is an improved version of Evolving Clustering Method (ECM) (Song and Kasabov 2001). Davies Bouldin Index (DBI) is used as the evaluation criteria, instead of shortest distance.

Algorithm 8 DBIECM (S, r_0)

Input: S : the input data stream
Input: r_0 : max cluster radius
1: Initialize the cluster structure
2: **loop**
3: Read next data instance x from data stream S
4: dis_i = Find the distance from x to all cluster centers C_i, $i \in [1, k]$
5: **if** $dis_i <$ radius of C_i **then**
6: Add x to C_i
7: **else if** $dis_i > r_0$ for all $i \in [1, k]$ **then**
8: Create new micro-cluster with x
9: **else** % There exist clusters such that radius of $C_i < dis_i < r_0$
10: Find all clusters such that radius of $C_i < dis_i$
11: Add x to the best cluster, according to DBI
12: **end if**
13: **end loop**

Algorithm 8 shows the main flow of DBIECM. When a new data instance x is received, an attempt is made to add the new data instance to an existing cluster. For this purpose, the distances between x and all clusters are calculated. If radius of any cluster is greater than or equal to its distance to x, then x is added to this cluster, as indicated at line 6. If the distance from x to any cluster is greater than maximum cluster radius r_0,

which is a user defined, static parameter, then a new cluster is created with x, at line 8. Otherwise, if there exist any clusters such that their radii are less than their distance to x, then x is added to all of these clusters one by one and DBI of the results are calculated separately. x is added to the cluster that gives the least DBI, which means the best clustering.

DBIECM requires the maximum cluster radius as a parameter. This parameter directly affects the final cluster count and consequently the clustering quality. Maximum cluster radius strongly depends on the input data and requires an expert knowledge about the data. Being distance based, DBIECM can detect only hyper-spherical clusters. DBIECM does not employ any time window model, thus no input data instance out dates, all input data exist in the final clustering. Moreover, no outlier detection mechanism is implemented. However, it is possible to specify an outlier threshold value and mark the clusters with low cardinality as outliers.

Evaluation: DBIECM is evaluated against ECM, with Iris, Wine, Seeds, Glass and Breast Cancer datasets, from UCI machine learning database. Both of the algorithms are run with the same maximum cluster radius parameter. Firstly, three different radius values are tried, and their direct impact on the resultant cluster number is observed. This shows the importance of the expert knowledge for radius selection. Moreover, clustering quality is compared according to objective function value, DBI, accuracy and purity. For these tests, radius value is selected according to the correct cluster number. DBIECM achieve up to 43% better DBI, up to 33% better accuracy and up to 11% better purity values than ECM.

Complexity Analysis: When a new data instance is received, a linear search is performed on clusters. Complexity of this linear search is $O(k)$. Pairwise distances between all clusters are used for DBI calculation, thus DBI calculation requires a complexity proportional to $O(k^2)$. When there exist more than one candidate clusters for the new data instance, the instance is added to all of them one by one and DBI is calculated accordingly. This requires a complexity proportional to $O(k^3)$. Therefore, although the average complexity of DBIECM depends on the input data, the total worst case complexity is $O(k) + O(k^3)$ which equals to $O(k^3)$.

3.7 I-HASTREAM (2015)

I-HASTREAM is a two phase, adaptive, density based hierarchical, data stream clustering algorithm proposed by Hassani et al. (Hassani et al. (2015), Hassani et al. (2016)). I-HASTREAM is an improved version of HASTREAM (Hassani et al. 2014). In the online phase, synopsis of the data is created as micro-clusters. In the offline phase, micro-clusters are maintained in a graph structure as a minimum spanning tree and hierarchical clustering is employed for the final clustering. Main contributions of I-HASTREAM are to perform the final clustering on a minimum spanning tree and to incrementally update the minimum spanning tree according to the changes in the micro-clusters, instead of generating it from scratch. Both of these contributions are related to the offline phase. For I-HASTREAM and its ancestor HASTREAM (Hassani et al. 2014) no algorithmic details are specified about the online phase, instead, it is stated that any micro-cluster model can be employed. For evaluation purpose, HASTREAM employs online phases of *DenStream* and *ClusTree* algorithms and these results are presented by Hassani et al. (2014).

Algorithm 9 I-HASTREAM offline phase (micro-clusters, α)

Input: micro-clusters
Input: α : weight threshold
1: MST = Update minimum spanning tree(MST, micro-clusters)
2: HC = Employ hierarchical clustering(MST, α)
3: Extract final clustering(HC)

Algorithm 9 shows main flow of offline phase of I-HASTREAM. The minimum spanning tree is updated according to the changes in the micro-clusters at line 1, and a hierarchical clustering on the minimum spanning tree is employed at line 2. As result of hierarchical clustering, a dendrogram is created. Final clustering is performed according to this dendrogram, at line 3.

Evaluation: Four variants of I-HASTREAM (with different parameters) are evaluated against HASTREAM, MR-Stream and DenStream, using network intrusion detection dataset and the physiological dataset. Purity and Cluster Mapping Measure (CMM) (Kremer et al. 2011) are used as evaluation criteria. One of the I-HASTREAM variants gives up to 25% better purity values than DenStream in network intrusion detection dataset. Its result is also up to 10% better than other versions of I-HASTREAM and HASTREAM. In the physiological dataset, the same variant of I-HASTREAM gives the best CMM and purity values in general. HASTREAM and I-HASTREAM have very close CMM values and both of them outperforms DenStream with up to 30% better CMM values. For purity, again I-HASTREAM has the best values in general and it outperforms both DenStream and MR-Stream with up to 15% better purity values. When we look at the execution time comparison of the algorithms, I-HASTREAM is more than five times faster than DenStream.

Complexity Analysis: Because no algorithmic details are specified about the online phase, we could not analyze complexity of I-HASTREAM.

4 Comparison of the algorithms

As common characteristics of seven data stream clustering algorithms given in Sect. 3, all of them predict number of clusters themselves and they are all able to adopt concept drift in the data streams. All but MuDi-Stream are suitable for high dimensional data. The reason MuDi-Stream is not suitable for high dimensional data is that, it uses a grid based approach for outlier detection. When the data are high dimensional, the number of empty grids increases and the execution time gets higher.

Adaptive Streaming *k*-means and FEAC-Stream are both *k*-means based (partitioning based) algorithms. DBIECM is distance based and the others are density based algorithms. Distance based approaches are similar to density based approaches, however they do not have a density threshold, instead they have maximum cluster radius threshold.

In general, density based algorithms have problem about finding clusters with different densities, because of the static density threshold. However, MuDi-Stream and I-HASTREAM have improvements for this problem and they successfully adopt the density threshold to each cluster separately. This makes them able to find multi-density clusters. Adaptive Streaming *k*-means and FEAC-Stream, being partition based algorithms, are also able to find clusters with different densities. DBIECM is successful for multi density

clusters, but not for multi size clusters. It has a static maximum cluster radius threshold and this is a problem for clusters with different sizes. As a result, CEDAS and Improved Data Stream Clustering algorithm are not able to find multi density clusters, but the others are. Furthermore, all density based algorithms are able to find arbitrary shaped clusters, while partitioning and distance based algorithms are limited with hyper-spherical clusters.

For Adaptive Streaming *k*-means and DBIECM, no outlier detection mechanism is mentioned. However, it is possible to define an outlier threshold and to mark the clusters have less cardinality than the threshold as outliers, for both algorithms. The other algorithms already have outlier detection mechanisms.

Up to the recent years, most of data stream clustering algorithms were online–offline algorithms. A synopsis of the data is employed in the online phase and the final clusters are generated in the offline phase. In this type of algorithms, offline phase is executed periodically or upon user request. Therefore, final clustering results are obtained with a latency and they are not up to date most of the times. However, there exist several recent fully online algorithms in the literature. Fully online algorithms maintain the final clustering results up to date. Therefore, users get the results with no latency. CEDAS, Adaptive Streaming *k*-means, FEAC-Stream and DBIECM are online algorithms, while MuDi-Stream, Improved Data Stream Clustering and I-HASTREAM are online–offline algorithms.

Damped window model is the most popular time window model among data stream clustering algorithms. On the other hand, DBIECM does not use any time window model. Moreover, Adaptive Streaming *k*-means uses sliding window model. All other mentioned algorithms use damped window model.

Finally, clustering quality of MuDi-Stream, CEDAS and DBIECM is strongly sensitive to the input parameter *threshold* value. It directly affects the number of clusters and accordingly the clustering quality. Selecting a proper threshold value requires an expert knowledge about the input data. Therefore, for successful results of MuDi-Stream, CEDAS and DBIECM, it is necessary to have prior information about characteristics of the input data. Tables 3 and 4 show the comparison summary of examined data stream clustering algorithms and Fig. 3 shows their main characteristics.

In conclusion, Adaptive Streaming *k*-means, FEAC-Stream and DBIECM have limitations about the cluster shape; they are able to find only hyper-spherical clusters. MuDi-Stream is not suitable for high dimensional data because of its grid based outlier detection mechanism. CEDAS and Improved Data Stream Clustering algorithm can not be used for clusters with different densities and DBIECM can not be used for clusters with different radii. Finally, an expert knowledge about the input data and the clusters is required for MuDi-Stream, CEDAS and DBIECM. I-HASTREAM claims to have no limitations, however no algorithmic details are specified for online phase of it. It is stated that online phases of *DenStream* and *ClusTree* are employed instead.

5 Open problems

There exist several open problems about data stream clustering. Here, we indicate the most notable open problems and describe them briefly.

- *Finding k:* Finding *k* is still an open problem, especially for partitioning based algorithms. There exist some recent methods for this purpose, however none of them is widely accepted and well matured yet. For density based algorithms, determining *k*

Table 3 Comparison of recent data stream clustering algorithms

Algorithm	Year	Base algorithm	Phases	Window model	Cluster count	Cluster shape
Adaptive Streaming *k*-Means	2017	Partitioning based	Online	Sliding	Auto	Hyper-spherical
FEAC-Stream	2017	Partitioning based	Online	Damped	Auto	Hyper-spherical
MuDi-Stream	2016	Density based	Online–offline	Damped	Auto	Arbitrary
CEDAS	2016	Density based	Online	Damped	Auto	Arbitrary
Improved Data Stream Clustering	2017	Density based	Online–offline	Damped	Auto	Arbitrary
DBIECM	2017	Distance based	Online	None	Auto	Hyper-spherical
I-HASTREAM	2015	Density based	Online–offline	Damped	Auto	Arbitrary

Table 4 Comparison of recent data stream clustering algorithms (continued from Table 3)

Algorithm	Multi density clusters	High dimensional data	Outlier detection	Drift adaption	Expert knowledge
Adaptive Streaming *k*-Means	Yes	Suitable	No	Yes	No
FEAC-Stream	Yes	Suitable	Yes	Yes	No
MuDi-Stream	Yes	Not suitable	Yes	Yes	Required
CEDAS	No	Suitable	Yes	Yes	Required
Improved Data Stream Clustering	No	Suitable	Yes	Yes	No
DBIECM	Yes (not multi size)	Suitable	No	Yes	Required
I-HASTREAM	Yes	Suitable	Yes	Yes	No

is easier, however parameters that depend on domain knowledge are necessary. If cluster characteristics such as density and minimum allowable gap between clusters are known *a priori*, current algorithms are then able to detect k; however, in most cases, knowledge about input data is not available before the execution and it may not be possible to specify parameters that are valid for all clusters. For example, multi-density clusters require different density thresholds and multi-size clusters require different distance thresholds. Determining such parameters is another open problem by itself. Moreover, concept drift, which may invalidate data specific parameters, is very common in data streams. Therefore, finding a k estimation method that adopts to changes in both k and cluster characteristics is a challenge. Such a method should react to concept drift fast, adopt the new data distribution with minimum quality loss and estimate k.

- *Parameter Requirements:* Current data stream clustering algorithms require parameters such as k, density threshold, distance threshold, decay rate and window length. Such parameters are very sensitive to the input data and they directly affect the clustering quality. It is a challenge to automatically specify these parameters without domain knowledge, manage them for each cluster separately, and update them according to the data characteristics.
- *Evaluation Criteria:* There is no *de facto* evaluation criteria for data stream clustering. Traditional evaluation methods are used for stream clustering results. Defining a new evaluation metric that is suitable for data streams might contribute to this field and inspire interest.
- *Benchmark Data:* There is a lack of high quality benchmark data to use in data stream clustering algorithms. One of the most popular datasets for stream clustering is the forest cover type dataset and it is not even a stream data. Artificial and real datasets that include concept drift, outliers and class labels, are necessary for benchmarking purposes in data stream clustering field. Generating and collecting such artificial and real stream datasets and popularizing them is a challenge.
- *Experimental Comparison Environment:* There is not a system that runs more than one data stream clustering algorithms at the same time, feeds them in the same way, and compares their execution performance and clustering quality.
- *Different Data Types:* Handling different data types is another challenging task in data stream clustering. Most of the stream clustering algorithms work with quantitative features and define the similarity based on euclidean distance. Current data structures that keep the data synopsis are also specialized for quantitative features. There exists a lack of clustering algorithms that work with categorical data. It is common to convert categorical data to quantitative data and use existing algorithms.
- *Performance Improvements:* Any performance improvements is always welcome, since the number of connected devices is increasing and the data generated by them are scaling up and accelerating every day. This situation requires a continuous performance improvement in data stream clustering algorithms. It is possible to improve the performance by using *parallel programming* and *edge computing*. However in this study, we focus on processing where the whole data is gathered and processed directly on a single processor.

Concept drift is a data stream specific and it generates several challenges. The number of clusters, cluster densities, sizes and shapes may change over time due to concept drift. The problems of traditional clustering become continuous problems for stream clustering.

6 Popular data repositories and datasets

6.1 Data repositories

There exist several stream data resources on the internet. Moreover, it is common to use traditional datasets as streams or to generate artificial data streams. Traditional datasets are generally read by order and treated as streams for testing and benchmarking purposes. We mention the stream data sources in this section. Data streams in Stream Data Mining Repository (see Sect. 6.1.4) and MOA (see Sect. 6.1.5) already have true class labels. However, Citi Bike System Data (see Sect. 6.1.1) does not possess explicitly a class label. One should decide how to employ the data and then assign accordingly the class labels. Moreover, National Weather Service Public Alerts (see Sect. 6.1.3) and Meetup RSVP Stream (see Sect. 6.1.2) have several features that can be used as class labels.

6.1.1 Citi Bike system data

Citi Bike NYC (2013) is a public bicycle sharing system. It is composed of 750 stations and 12,000 bikes. Citi Bike publicly publishes real time system data in Citi Bike System Data which includes system information, station information, free bike status etc. in a json structure. Moreover, Citi Bike also publishes trip histories, daily ridership and membership data, and monthly operating reports stored as data streams.

6.1.2 Meetup RSVP stream

Meetup (2002) is a website providing membership software, allowing its users to schedule events using a common platform. Meetup has an invitation response mechanism in which the invitees click to RSVP button and enter their responses. Meetup publicly publishes these RSVP responses as a stream (Meetup Stream 2002), which is suitable for data stream clustering.

6.1.3 National weather service public alerts

National Weather Service (NWS) (1870) creates public alerts, watches, warnings, advisories, and other similar products in the Common Alerting Protocol (CAP) and Atom Syndication Format (ATOM) (NWS Public Alerts, n.d.). These are data streams and they can be used for data stream clustering studies.

6.1.4 Stream data mining repository

Stream Data Mining Repository is a public repository by Zhu (2010) holding four different stream datasets, which are Sensor Stream (2,219,803 instances, 5 features, and 54 clusters), Power Supply Stream (29,928 instances, 2 features, and 24 clusters), Network

Intrusion Detection 10% Subset (494,021 instances, 41 features, and 23 clusters) and Hyper Plane Stream (100,000 instances, 10 features, and 5 clusters).

6.1.5 MOA

Massive Online Analysis (MOA) (Bifet et al. 2010) is a popular open source framework for data stream mining. MOA includes 4 different datasets which are suitable for data stream processing. Moreover, it also includes a number of classes to generate artificial data streams. There exist several studies in the literature that use MOA as a data source. More information about MOA is available in Sect. 7.1 and artificial data stream generation classes of MOA are listed in Sect. 6.2.1.

6.1.6 Other repositories

Some other data repositories are listed here.

- Real World Data in Real Time API : https://www.hooksdata.io/
- New York City Open Data : https://opendata.cityofnewyork.us/
- Registry of Open Data on AWS : https://registry.opendata.aws/
- Twitter Data : https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data
- AirNow Air Quality Observations : https://docs.airnowapi.org/
- National Wind Technology Center (NWTC) : https://data.nrel.gov/submissions/33
- Solar Radiation Research Laboratory (SRRL) : https://data.nrel.gov/submissions/7
- Awesome Public Datasets : https://github.com/awesomedata/awesome-public-datasets

6.2 Popular datasets

It is very common to use artificial datasets in data stream clustering for both testing and benchmark purposes. Artificial datasets give the user opportunity to specify the stream properties such as noise ratio, concept drift, cluster shapes and densities. Artificial data stream generation by MOA and details of popular datasets are given. All datasets

Table 5 Properties of popular datasets

Dataset name	Number of instances	Number of features	Number of clusters
Forest cover type	581,012	54	7
Network intrusion detection	4,898,431	41	23
Network intrusion detection subset	494,021	41	23
Charitable donation	191,779	481	Not specified
Sensor stream	2,219,803	5	54
Power supply stream	29,928	2	24
Hyper plane stream	100,000	10	5

mentioned in this section, except Charitable Donation Dataset, have true class labels. Table 5 summarizes properties of popular datasets.

6.2.1 Artificial data streams

Massive Online Analysis (MOA) (described in Sect. 7.1) has a number of classes (Moa Stream Generators 2014) to generate artificial data streams in different shapes and with or without concept drift.

6.2.2 Forest cover type dataset

Forest Cover Type Dataset is publicly available on Machine Learning Repository of UCI. It has totally 581,012 instances and each of them belongs to one of 7 cover types. The instances are described by 54 features, 10 of which are quantitative and 44 of which are binary. Each instance is giving information of an area of 30x30 meters. This dataset is not actually a data stream, but a stationary dataset. It does not have a time stamp or an exclusive order information. However, it is converted into a data stream by taking the data input order as the streaming order.

6.2.3 Network intrusion detection dataset

Network Intrusion Detection Dataset is used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was a session of KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining. It is publicly available on KDD archive of UCI. This set has 4,898,431 records of network traffic data and each of them belongs to one of 23 types of connection (22 attack types and normal connection). The instances are described by 41 features, some of which are discrete and the others are continuous. There exists also a 10% subset of this dataset which is more concentrated than the original dataset. The subset itself is yet another most used dataset.

6.2.4 Charitable donation dataset

Charitable Donation Dataset is used for The Second International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-98, The Fourth International Conference on Knowledge Discovery and Data Mining. This dataset has 191,779 instances and each instance has 481 features. These instances, are information about people who have made charitable donations in response to direct mailing requests. This dataset is publicly available on KDD archive of UCI.

6.2.5 Various spam mail datasets

There exist several spam mail datasets publicly available in different online data repositories. Spam mail datasets are suitable for stream clustering because mails inherently are data streams. They have a date-time information which makes them easily interpreted as data streams.

6.2.6 Various sensor network datasets

There exist several sensor network datasets publicly available on the Internet. One of sensor network data repositories is A Community Resource for Archiving Wireless Data At Dartmouth (CRAWDAD). It is very common to use sensor network datasets in data stream clustering, since they inherently are data streams.

7 Data stream processing tools

We provide brief information about popular tools that are used for data stream mining.

7.1 MOA

Massive Online Analysis (MOA) (Bifet et al. 2010) is a popular open source framework for data stream mining. It is implemented in Java and released under the GNU General Public License. MOA is specialized for data streams. It includes algorithms for regression, clustering, classification, outlier detection, concept drift detection and recommender systems, and it also includes tools for evaluation. Data stream generators are provided. It can be used as both a stream processing tool and an environment to develop stream processing algorithms. Furthermore, MOA has the ability to interact with Waikato Environment for Knowledge Analysis (1993), which is a data mining software.

7.2 RapidMiner

RapidMiner (2001), formerly known as Yet Another Learning Environment (YALE), is another data mining tool but it is developed by a private company. It has an integrated development environment, which is called RapidMiner Studio. It supports all data preparation, result visualization, model validation and optimization steps of the machine learning process. It has a *Streams* plugin (Bockermann 2018) which integrates the stream oriented processing into the RapidMiner suite. This plugin allows developing data stream processing tools using utilities of RapidMiner.

7.3 R

R (1993) is a free software environment and programming language for statistical computing. R is an open source project and it is released under the GNU General Public License. R, a rich in packages software environment, has special packages for clustering, data streams, stream mining etc. These packages are as follows.

- stream: A framework for data stream modeling and associated data mining tasks such as clustering and classification.
- rstream: Unified object oriented interface for multiple independent streams of random numbers from different sources.

- streamMOA: Interface for data stream clustering algorithms implemented in the MOA framework.
- RMOA: Connects R with MOA framework to build classification and regression models on streaming data.

8 Data stream processing platforms

Currently there exist several data stream mining platforms (Janardan and Mehta 2017; Prasad and Agarwal 2016) developed by different organizations.

- Apache Storm (2011) is a distributed, real time stream processing computation framework. It is free and open source. Moreover, Apache Storm is scalable and fault tolerant. It is designed to be used with any programming language.
- Apache Spark (2012) is a well known, open source, fast and general engine for large-scale data processing. Apache Spark has an extension, called Spark Streaming (2012), that enables scalable, high-throughput, fault tolerant stream processing of live data streams. Spark Streaming can be seen as a layer between data streams and Apache Spark. Spark Streaming gets a data stream, creates data batches from the stream and feeds Apache Spark with these batches. In this way, results of the data stream processing are produced by Apache Spark batch by batch. Spark Stream accepts input from many different sources such as Kafka, Flume, Twitter, ZeroMQ, Kinesis, or TCP sockets.
- Apache Samza (2013) (Ramesh 2013) is another open source, distributed stream processing framework. It is near real time and asynchronous. It provides fault tolerance, processor isolation, security, and resource management using Apache Hadoop Yarn. It uses Apache Kafka for messaging. Apache Samza, together with Apache Kafka, is developed by LinkedIn engineers, and commonly known as LinkedIn's framework for stream processing.
- Apache Kafka (2011) is an open source stream processing software platform. The objective of the project is to provide a unified, high throughput, low latency platform for real time data streams. It is scalable and fault tolerant. It has a publish-subscribe messaging system. Apache Kafka is the other platform developed by LinkedIn, similar to Apache Samza.
- AmazonKinesis (2013) is one of the Amazon web services. It is a cloud based, real time data processing service that is developed for large and distributed data streams. In functionality, Amazon Kinesis has similarities to Apache Kafka. It is scalable and able to pull any amount of data, from any number of sources. It is designed to make it easier to develop real time applications and it has a fully managed infrastructure.
- IBM Infosphere (1996) (Gedik and Andrade 2012) is a commercial, enterprise-grade stream processing platform, that is designed to retrieve meaningful information from data in motion, working on time window models with windows of minutes to hours. It provides low latency for time critical applications such as fraud detection and network management. It also has the ability to fuse streams. IBM Inforsphere adapts rapidly to changing data forms and types and it manages high availability itself.
- Google Cloud Stream (2012) is Google's solution for data stream processing. It has a fully managed infrastructure and it provides ingesting, processing and analyzing event streams in real time. It is an integrated, scalable and open stream analytics solution.

Google Cloud Stream works with a full harmony with other solutions of Google Cloud, like Cloud Pub/Sub, Cloud Dataflow, BigQuery, Cloud Machine Learning etc.

- Microsoft Azure Stream Analytics (2012) is Microsoft's solution for data stream processing. It is a serverless, scalable, on demand real time, complex event processing engine. It is able to run on multiple streams from different sources. Azure Stream Analytics has a declarative SQL like language. It can be used as integrated with other Azure solutions such as Azure Machine Learning, Azure IoT Hub, Power BI etc.

9 Conclusions

With the technological improvements, number of interconnected devices is increasing. Connected devices continuously generate large scale data with high speed, which are called data streams. Therefore, processing data streams in real time is arousing more interest and clustering seems to be the most suitable data processing method for data streams.

We present a survey of recent progress in data stream clustering algorithms. There are essential differences between traditional data clustering algorithms and data stream clustering algorithms. We emphasize the most important data stream clustering concepts such as concept drift, window models, outlier detection methods and data structures. Seven most recent data stream clustering algorithms are analyzed in detail. For each algorithm, a comprehensive analysis is presented including algorithmic detail, evaluation of the results and complexity. Global comparison of these algorithms highlighting their advantages and disadvantages is also presented. An overview of the most popular stream processing tools and platforms is given along with stream datasets.

Several open challenges exist regarding data stream clustering. Finding number of clusters and adopting to changes in the number of clusters in data streams are the most crucial challenges. Furthermore, existing algorithms need critical parameters that directly affect clustering quality and require prior knowledge about input data. Moreover, concept drift may change data characteristics and invalidate these parameters. Developing generic and self-adapting algorithms is another popular data stream clustering challenge. Additionally, there is a lack of algorithms that handle different data types. Most of existing algorithms are able to deal with only quantitative data. Last but not least, data stream clustering algorithms should execute with high performance in despite of memory restrictions.

It may be ideal to compare the efficiency and the effectiveness of the data stream clustering algorithms on a benchmarking framework under controlled conditions of artificial datasets that contain concept drift, outliers and class labels and of real datasets. Data stream clustering using deep neural network models and within edge computing are the two emerging topics to be explored further.

References

A community resource for archiving wireless data at Dartmouth (CRAWDAD) (n.d.) https://crawdad.org/keyword-sensor-network.html. Accessed 25 August 2018

Ackermann MR, Märtens M, Raupach C, Swierkot K, Lammersen C, Sohler C (2012) Streamkm++: a clustering algorithm for data streams. J Exp Algorithm 17:2.4:2.1–2.4:2.30

Aggarwal CC (2013) A survey of stream clustering algorithms. In: Reddy CK, Aggarwal CC (eds) Data clustering: algorithms and applications. CRC Press, Boca Raton, pp 231–258

Aggarwal CC, Han J, Wang J, Yu PS (2003) A framework for clustering evolving data streams. In: Proceedings of the 29th international conference on very large data bases, VLDB '03, vol 9, pp 81–92

Aggarwal C, Han J, Wang J, Yu P (2004) A framework for projected clustering of high dimensional data streams, pp 852–863. https://doi.org/10.1016/B978-012088469-8/50075-9
Agrawal R, Gehrke J, Gunopulos D, Raghavan P (1998) Automatic subspace clustering of high dimensional data for data mining applications. In: Proceedings of the 1998 ACM SIGMOD international conference on management of data, association for computing machinery, SIGMOD '98, New York, NY, USA, pp 94–105. https://doi.org/10.1145/276304.276314
Alam F, Mehmood R, Katib I, Albeshri A (2016) Analysis of eight data mining algorithms for smarter internet of things (IoT). Procedia Comput Sci 98:437–442
AmazonKinesis (2013) Amazon Kinesis. https://aws.amazon.com/kinesis/. Accessed 25 Mar 2018
Amini A, Saboohi H, Herawan T, Wah TY (2016) Mudi-stream: a multi density clustering algorithm for evolving data stream. J Netw Comput Appl 59(C):370–385
Andrade Silva J, Hruschka ER, Gama J (2017) An evolutionary algorithm for clustering data streams with a variable number of clusters. Expert Syst Appl 67:228–238
Apache Kafka (2011) https://kafka.apache.org/. Accessed 25 Mar 2018
Apache Samza (2013) Samza. https://samza.apache.org/. Accessed 25 Mar 2018
Apache Spark (2012) Apache Spark lightning-fast cluster computing. https://spark.apache.org/. Accessed 25 Mar 2018
Apache Storm (2011) http://storm.apache.org/. Accessed 25 Mar 2018
Bezerra CG, Costa BSJ, Guedes LA, Angelov PP (2020) An evolving approach to data streams clustering based on typicality and eccentricity data analytics. Inf Sci 518:13–28. https://doi.org/10.1016/j.ins.2019.12.022
Bhosale SV (2014) A survey: outlier detection in streaming data using clustering approached. Int J Comput Sci Inf Technol 5:6050–6053
Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. J Mach Learn Res 11:1601–1604
Bockermann C (2018) RapidMiner streams plugin. https://sfb876.de/streams/doc/rapidminer.html. Accessed 25 Mar 2018
Cao F, Ester M, Qian W, Zhou A (2006) Density-based clustering over an evolving data stream with noise. vol 2006. https://doi.org/10.1137/1.9781611972764.29
Carnein M, Assenmacher D, Trautmann H (2017) An empirical comparison of stream clustering algorithms. In: Proceedings of the computing frontiers conference, CF'17, pp 361–366
Chauhan P, Shukla M (2015) A review on outlier detection techniques on data stream by using different approaches of K-Means algorithm. In: 2015 international conference on advances in computer engineering and applications
Chen Y, Tu L (2007) Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '07, pp 133–142
Christodoulou V, Bi Y, Wilkie G (2018) A fuzzy shape-based anomaly detection and its application to electromagnetic data. IEEE J Sel Top Appl Earth Obs Remote Sens 11(9):3366–3379. https://doi.org/10.1109/JSTARS.2018.2854865
Citi Bike NYC (2013) Citi Bike: NYC's official bike sharing system. https://www.citibikenyc.com/. Accessed 25 Mar 2018
Citi Bike System Data (2013) https://www.citibikenyc.com/system-data. Accessed 25 Mar 2018
Dang XH, Lee VCS, Ng WK, Ong KL (2009) Incremental and adaptive clustering stream data over sliding window. In: Bhowmick SS, Küng J, Wagner R (eds) Database and expert systems applications. Springer, Berlin, pp 660–674
Din SU, Shao J, Kumar J, Ali W, Liu J, Ye Y (2020) Online reliable semi-supervised learning on evolving data streams. Inf Sci 525:153–171. https://doi.org/10.1016/j.ins.2020.03.052
Ding S, Wu F, Qian J, Jia H, Jin F (2015) Research on data stream clustering algorithms. Artif Intell Rev 43(4):593–600
Duan L, Xiong D, Lee J, Guo F (2006) A local density based spatial clustering algorithm with noise. Inf Syst 32:4061–4066. https://doi.org/10.1109/ICSMC.2006.384769
Ester M, Kriegel HP, Sander J, Wimmer M, Xu X (1998) Incremental clustering for mining in a data warehousing environment. In: Proceedings of the 24rd international conference on very large data bases, VLDB '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 323–333
Fahy C, Yang S, Gongora M (2018) Ant colony stream clustering: a fast density clustering algorithm for dynamic data streams. IEEE Trans Cybern 49(6):2215–2228
Fisher D (1996) Iterative optimization and simplification of hierarchical clustering. J Artif Intell Res 4:147–178. https://doi.org/10.1613/jair.276
Gaber MM, Zaslavsky A, Krishnaswamy S (2009) Data stream mining. In: Maimon O, Rokach L (eds) Data mining and knowledge discovery handbook. Springer, Berlin, pp 759–787

Gama J, Rodrigues PP, Lopes L (2011) Clustering distributed sensor data streams using local processing and reduced communication. Intell Data Anal 15(1):3–28
Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):44:1–44:37
Gedik B, Andrade H (2012) A model-based framework for building extensible, high performance stream processing middleware and programming language for IBM InfoSphere Streams. Softw Pract Exp 42(11):1363–1391
Ghesmoune M, Lebbah M, Azzag H (2016) State-of-the-art on clustering data streams. Big Data Anal 1(1):13
Google Cloud Stream (2012) Streaming analytics for real time insights—Google Cloud. https://cloud.google.com/solutions/big-data/stream-analytics/. Accessed 25 Mar 2018
Hassani M, Spaus P, Seidl T (2014) Adaptive multiple-resolution stream clustering. In: Machine learning and data mining in pattern recognition, pp 134–148
Hassani M, Spaus P, Cuzzocrea A, Seidl T (2015) Adaptive stream clustering using incremental graph maintenance. In: Proceedings of the 4th international conference on big data, streams and heterogeneous source mining: algorithms, systems, programming models and applications , BIGMINE'15, vol 41, pp 49–64
Hassani M, Spaus P, Cuzzocrea A, Seidl T (2016) I-hastream: density-based hierarchical clustering of big data streams and its application to big graph analytics tools. In: 2016 16th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid), pp 656–665
Hyde R, Angelov P, MacKenzie A (2017) Fully online clustering of evolving data streams into arbitrarily shaped clusters. Inf Sci 382–383:96–114
Infosphere IBM (1996) Streaming analytics—overview—IBM Cloud. https://www.ibm.com/cloud/streaming-analytics. Accessed 25 Mar 2018
Isaksson C, Dunham M, Hahsler M (2012) Sostream: self organizing density-based clustering over data stream. vol 7376. https://doi.org/10.1007/978-3-642-31537-4_21
Janardan Mehta S (2017) Concept drift in streaming data classification: algorithms, platforms and issues. Procedia Comput Sci 122:804–811. https://doi.org/10.1016/j.procs.2017.11.440
Karypis G, Han EH, Kumar V (1999) Chameleon a hierarchical clustering algorithm using dynamic modeling. Computer 32:68–75. https://doi.org/10.1109/2.781637
Kaufman L, Rousseeuw PJ (1990) Chapter 3: Clustering large applications (Program CLARA). Wiley, Hoboken, pp 126–163. https://doi.org/10.1002/9780470316801.ch3
Keogh E, Lin J, Fu A (2005) Hot sax: efficiently finding the most unusual time series subsequence. In: Proceedings of the fifth IEEE international conference on data mining, ICDM '05, IEEE Computer Society, USA, pp 226–233. https://doi.org/10.1109/ICDM.2005.79
Kim T, Park CH (2020) Anomaly pattern detection for streaming data. Exp Syst Appl 149:113252. https://doi.org/10.1016/j.eswa.2020.113252
Kong X, Bi Y, Glass DH (2019) Detecting anomalies in sequential data augmented with new features. Artif Intell Rev 53:625–652
Kremer H, Kranen P, Jansen T, Seidl T, Bifet A, Holmes G, Pfahringer B (2011) An effecive evaluation measure for clustering on evolving data streams. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '11, pp 868–876
Kumar P (2016) Data stream clustering in internet of things. SSRG Int J Comput Sci Eng 3(8):1–14
Liu L, Huang H, Guo Y, Chen F (2009) rDenStream, a clustering algorithm over an evolving data stream. In: 2009 International conference on information engineering and computer science, pp 1–4
Lu Y, Sun Y, Xu G, Liu G (2005) A grid-based clustering algorithm for high-dimensional data streams. In: Li X, Wang S, Dong ZY (eds) Advanced data mining and applications. Springer, Berlin, pp 824–831
Mahdiraji AR (2009) Clustering data stream: a survey of algorithms. Int J Knowl-Based Intell Eng Syst 13(2):39–44
Mansalis S, Ntoutsi E, Pelekis N, Theodoridis Y (2018) An evaluation of data stream clustering algorithms. Stat Anal Data Min ASA Data Sci J 11(4):167–187
Massive Online Analysis (MOA) (2014) MOA—machine learning for data streams. https://moa.cms.waikato.ac.nz/. Accessed 25 Mar 2018
Meesuksabai W, Kangkachit T, Waiyamai K (2011) Hue-stream: evolution-based clustering technique for heterogeneous data streams with uncertainty, pp 27–40. https://doi.org/10.1007/978-3-642-25856-5_3
Meetup (2002) We are what we do | Meetup. https://www.meetup.com/. Accessed 25 Mar 2018
Meetup Stream (2002) Extend your community | Meetup. https://www.meetup.com/meetup_api/docs/stream/2/rsvps/. Accessed 25 Mar 2018
Merino JA (2015) Streaming data clustering in MOA using the leader algorithm. PhD thesis, Universitat Politècnica de Catalunya

Microsoft Azure Stream Analytics (2012) Stream analytics—real time data analytics—Microsoft Azure. https://azure.microsoft.com/en-us/services/stream-analytics/. Accessed 25 Mar 2018
MOA Stream Generators (2014) MOA: Package moa.stream.generators. https://www.cs.waikato.ac.nz/~abifet/MOA/API/namespacemoa_1_1streams_1_1generators.html. Accessed 25 Mar 2018
Modi KD, Oza PB (2017) Outlier analysis approaches in data mining. Int J Innov Res Technol 3:6–12
Mousavi M, Bakar A, Vakilian M (2015) Data stream clustering algorithms: a review. Int J Adv Soft Comput Appl 7:1–15
Mouss H, Mouss D, Mouss N, Sefouhi L (2004) Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In: 2004 5th Asian control conference (IEEE Cat. No.04EX904), vol 2, pp 815–818
Namadchian A, Esfandani G (2012) Dsclu: a new data stream clustring algorithm for multi density environments. In: 2012 13th ACIS international conference on software engineering, artificial intelligence, networking and parallel/distributed computing, pp 83–88
National Weather Service (NWS) (1870) National Weather Service. https://www.weather.gov/. Accessed 25 Mar 2018
Nguyen HL, Woon YK, Ng WK (2015) A survey on data stream clustering and classification. Knowl Inf Syst 45(3):535–569
NWS Public Alerts (n.d.) NWS Public Alerts. https://alerts.weather.gov/. Accessed 25 Mar 2018
O'Callaghan L, Meyerson A, Motwani R, Mishra N, Guha S (2002) Streaming-data algorithms for high-quality clustering. In: Proceedings of the 18th international conference on data engineering, ICDE '02, pp 685–694
Ordonez C (2003) Clustering binary data streams with k-means. In: Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, DMKD '03, Association for Computing Machinery, New York, NY, USA, pp 12–19, https://doi.org/10.1145/882082.882087
Parzen E (1962) On estimation of a probability density function and mode. Ann Math Stat 33(3):1065–1076
Prasad BR, Agarwal S (2016) Stream data mining: platforms, algorithms, performance evaluators and research trends. Int J Database Theory Appl 9(9):201–218
Puschmann D, Barnaghi P, Tafazolli R (2017) Adaptive clustering for dynamic IoT data streams. IEEE Internet Things J 4(1):64–74
R (1993) R—the R Project for statistical computing. https://www.r-project.org/. Accessed 25 Mar 2018
Ramesh N (2013) Apache Samza, LinkedIn's framework for stream processing—The New Stack. https://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing/. Accessed 25 Mar 2018
Ramirez-Gallego S, Krawczyk B, Garcia S, Wozniak M, Herrera F (2017) A survey on data preprocessing for data stream mining: current status and future directions. Neurocomputing 239:39–57
RapidMiner (2001) Data Sicence Platform—RapidMiner. https://rapidminer.com/. Accessed 25 Mar 2018
Rodrigues P, Gama J, Pedroso JP (2006) Odac: hierarchical clustering of time series data streams. https://doi.org/10.1137/1.9781611972764.48
Rosenblatt M (1956) Remarks on some nonparametric estimates of a density function. Ann Math Statist 27(3):832–837
Sadik S, Gruenwald L (2014) Research issues in outlier detection for data streams. SIGKDD Explor Newsl 15(1):33–40
Satyanarayanan M (2017) The emergence of edge computing. Computer 50(1):30–39. https://doi.org/10.1109/MC.2017.9
Sheikholeslami G, Chatterjee S, Zhang A (2000) Wavecluster: a wavelet-based clustering approach for spatial data in very large databases. VLDB J 8(3–4):289–304. https://doi.org/10.1007/s007780050009
Shi W, Dustdar S (2016) The promise of edge computing. Computer 49(5):78–81
Shi W, Cao J, Zhang Q, Li Y, Xu L (2016) Edge computing: vision and challenges. IEEE Internet Things J 3(5):637–646
Silva JA, Faria ER, Barros RC, Hruschka ER, Carvalho ACPLFd, Ja Gama (2013) Data stream clustering: a survey. ACM Comput Surv 46(1):13:1–13:31
Song Q, Kasabov N (2001) ECM–a novel on-line, evolving clustering method and its applications. In: Posner MI (ed) Foundations of cognitive science. The MIT Press, Cambridge, pp 631–682
Souiden I, Brahmi Z, Toumi H (2016) A survey on outlier detection in the context of stream mining: review of existing approaches and recommadations. In: Advances in intelligent systems and computing
Streaming Spark (2012) Apache spark streaming. https://spark.apache.org/streaming/. Accessed 25 Mar 2018
Sun Y, Lu Y (2006) A grid-based subspace clustering algorithm for high-dimensional data streams. In: Feng L, Wang G, Zeng C, Huang R (eds) Web information systems–WISE 2006 workshops. Springer, Berlin, pp 37–48

Tasoulis D, Ross G, Adams N (2007) Visualising the cluster structure of data streams, vol 4723, pp 81–92. https://doi.org/10.1007/978-3-540-74825-0_8

Thakkar P, Vala J, Prajapati V (2016) Survey on outlier detection in data stream. Int J Comput Appl 136(2):13–16

Udommanetanakit K, Rakthanmanon T, Waiyamai K (2007) E-stream: Evolution-based technique for stream clustering. vol 4632, pp 605–615. https://doi.org/10.1007/978-3-540-73871-8_58

Waikato Environment for Knowledge Analysis (1993) Weka 3—data mining with open source machine learning software in Java. https://www.cs.waikato.ac.nz/ml/weka/. Accessed 25 Mar 2018

Wan L, Ng WK, Dang XH, Yu PS, Zhang K (2009) Density-based clustering of data streams at multiple resolutions. ACM Trans Knowl Discov Data 3(3):1–28. https://doi.org/10.1145/1552303.1552307

Wang H, Yu Y, Wang Q, Wan Y (2012) A density-based clustering structure mining algorithm for data streams. In: Proceedings of the 1st international workshop on big data, streams and heterogeneous source mining: algorithms, systems, programming models and applications, BigMine'12, Association for Computing Machinery, New York, NY, USA, pp 69–76. https://doi.org/10.1145/2351316.2351326

Wang W, Yang J, Muntz RR (1997) Sting: a statistical information grid approach to spatial data mining. In: Proceedings of the 23rd international conference on very large data bases, , VLDB '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp 186–195

Xu J, Wang G, Li T, Deng W, Gou G (2017) Fat node leading tree for data stream clustering with density peaks. Knowl-Based Syst 120:99–117. https://doi.org/10.1016/j.knosys.2016.12.025

Yasumoto K, Yamaguchi H, Shigeno H (2016) Survey of real-time processing technologies of iot data streams. J Inf Process 24(2):195–202

Yin C, Xia L, Zhang S, Sun R, Wang J (2017) Improved clustering algorithm based on high-speed network data stream. Soft Comput 22(13):4185–4195

Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. SIGMOD Rec 25(2):103–114

Zhang X, Furtlehner C, Germain-Renaud C, Sebag M (2014) Data stream clustering with affinity propagation. IEEE Trans Knowl Data Eng 26(7):1644–1656

Zhang KS, Zhong L, Tian L, Zhang XY, Li L (2017) DBIECM—an evolving clustering method for streaming data clustering. AMSE J 60(1):239–254

Zhou A, Cao F, Yan Y, Sha C, He X (2007) Distributed data stream clustering: a fast em-based approach. In: 2007 IEEE 23rd international conference on data engineering, pp 736–745

Zhou A, Cao F, Qian W, Jin C (2008) Tracking clusters in evolving data streams over sliding windows. Knowl Inf Syst 15(2):181–214

Zhu XH (2010) Stream data mining repository. http://www.cse.fau.edu/~xqzhu/stream.html. Accessed 25 Mar 2018