Bus Inf Syst Eng 61(3):277–297 (2019)
https://doi.org/10.1007/s12599-019-00576-5

STATE OF THE ART

Optimizing Data Stream Representation: An Extensive Survey on Stream Clustering Algorithms

Matthias Carnein · Heike Trautmann

Received: 20 June 2018 / Accepted: 13 November 2018 / Published online: 21 January 2019

Abstract Analyzing data streams has received considerable attention over the past decades due to the widespread usage of sensors, social media and other streaming data sources. A core research area in this field is stream clustering which aims to recognize patterns in an unordered, infinite and evolving stream of observations. Clustering can be a crucial support in decision making, since it aims for an optimized aggregated representation of a continuous data stream over time and allows to identify patterns in large and high-dimensional data. A multitude of algorithms and approaches has been developed that are able to find and maintain clusters over time in the challenging streaming scenario. This survey explores, summarizes and categorizes a total of 51 stream clustering algorithms and identifies core research threads over the past decades. In particular, it identifies categories of algorithms based on distance thresholds, density grids and statistical models as well as algorithms for high dimensional data. Furthermore, it discusses applications scenarios, available software and how to configure stream clustering algorithms. This survey is considerably more extensive than comparable studies, more up-to-date and highlights how concepts are interrelated and have been developed over time.

Keywords Stream clustering · Data streams · Online clustering · Pattern recognition · Decision support · Data representation

Accepted after one revision by the editors of the special issue.

M. Carnein (✉) · H. Trautmann
Information Systems and Statistics, University of Münster, Leonardo-Campus 3, 48149 Münster, Germany
e-mail: matthias.carnein@uni-muenster.de

1 Introduction

Cluster analysis is an unsupervised learning technique which aims to find groups of similar objects. It is a common tool to support decision makers by structuring the large and high-dimensional data into manageable groups and thus generating an optimized data representation. Common application scenarios include identifying profitable market segments, anomaly and fraud detection or sensor analysis. Most clustering algorithms require a fixed set of data and evaluate each point multiple times to generate the clusters. In practice, however, many systems are continuously generating new observations. As an example, sensors generate thousands of observations each second and countless of interactions happen every day in social networks. In order to account for new data points and the possible shift in cluster structures, classical clustering algorithms need to be run periodically. This is computationally expensive and requires that all relevant data is stored for periodic re-evaluation. A more suitable approach is to update existing clusters and integrate new observations into the existing model by identifying emerging structures and removing outdated structures incrementally. This is the goal of stream clustering where data points are assumed to arrive as a continuous stream of new observations where the order cannot be influenced. This stream is possibly unbounded which makes unlimited storage and re-evaluation of data points infeasible (Silva et al. 2013). In this scenario, the main task is to optimize the number and location of clusters in order to represent the underlying data best and to extract the relevant information from a stream without storing and re-evaluating all observations.

The underlying optimization task of these algorithms is to find clusters such that the within-cluster similarity is high. At the same time, similarity between-cluster similarity should be low. In the stream clustering scenario, these two optimization tasks are subject to restrictions with regard to the availability, order, speed and volume of the data as well as limitation regarding available memory and computational power. In the following we discuss algorithms which are able to deal with these restrictions. The main contribution of our survey is as follows: We provide an overview of available algorithms for stream clustering. Our survey is much more extensive and up-to-date than comparable studies and includes all algorithms applicable to numerical data that we are aware of. The only category which we consider out of scope for this paper are stream clustering algorithms for textual data since they usually rely on considerably different approaches. In addition, we identify different research threads and highlight interrelations between algorithms making it easier to understand how the field developed and what kind of trends exist. Furthermore, we discuss problems when applying stream clustering in practice. In particular, we see that most algorithms require numerous parameters which are unintuitive and difficult to choose appropriately. We discuss automatic algorithm configuration as one approach on how to tackle this problem. As an accompanying document to this paper we also provide a website[1] which compiles relevant information about stream clustering such as common datasets, available implementations and a curated list of algorithms and corresponding publications in the field.

The remainder of this paper is organized as follows: Sect. 2 introduces the basic concepts of stream clustering. Then, Sect. 3 gives an overview of related work and introduces a number of related surveys. Next, four different categories of algorithms are identified and a total of 51 algorithms presented. First, Sect. 4 introduces algorithms that use a distance threshold to build clusters. Next, Sect. 5 presents algorithms that utilize density-grids to map observations into a discrete space. Then, Sect. 6 presents algorithms that rely on statistical models and Sect. 7 discusses algorithms that deal with high dimensional data streams. Section 8 presents available software tools and implementations of the algorithms and also discusses common problems during application and how to overcome them. Finally, Sect. 9 concludes with a summary of the findings.

2 Methodological Background

In this section, we introduce the basics of stream clustering. Most importantly, we describe how data streams are typically aggregated and how algorithms adapt to changes over time. For a consistent notation, we denote vectors by boldface symbols and formally define a data stream as an infinite sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ where $\boldsymbol{x}_t$ is a single observation with d dimensions at time t. To calculate the distance between clusters, an appropriate distance measure needs to be used. For numerical data, the Euclidean distance between the centroids of the clusters is common. However, for binary, ordinal, nominal or text data, appropriate distance measures such as the Jaccard index, simple matching coefficient or Cosine similarity could be used.

In general, finding a good clustering solution is defined as an optimization task. The underlying goal is to maximize intra-cluster homogeneity while simultaneously maximizing inter-cluster heterogeneity. This ensures that objects within the same cluster are similar but different clusters are well separated. There are various strategies that aim to achieve this task. Popular strategies include minimizing intra-cluster distances, minimizing radii of clusters or finding maximum likelihood estimates. A popular example is the k-means algorithm which minimizes the within-cluster sum of squares, i.e., the distance from data points to their cluster centroids.

In a streaming scenario, these optimization objectives are subject to several restrictions regarding availability and order of the data as well as resource and time limitations. For example, the large volume of data makes it undesirable or infeasible to store all observations of the stream. Typically, observations can only evaluated once and are discarded afterwards. This requires to extract sufficient information from observations before discarding them. Similarly, the order of observations cannot be influenced. As an illustrative example, let us consider the case of eye tracking which is typically used in order to analyze how people perceive content such as websites or advertisements. It records the movement of the eye and detects where a person is looking. An example of a stream of eye tracking data is visualized in Fig. 1, showing the pupil positions at three different points in times (grey points) (-Steil et al. 2018). In this context, stream clustering can be applied in order to find the areas of interest or subjects that the person is looking at.

Throughout this paper, we discuss common strategies that can be used to identify clusters under the streaming restrictions. For example, we could use similarity thresholds in order to decide whether an observation fits into an existing cluster (Fig. 2a). Alternatively, we could split the data space into a grid and only store the location of densely populated cells (Fig. 2b). Other approaches include fitting a model to represent the observed data (Fig. 2c) or projecting high-dimensional data to a lower dimensional space (Fig. 2d).

Generally, these strategies allow to capture the location of dense areas in the data space. These regions can be

[1] http://www.matthias-carnein.de/streamClustering.

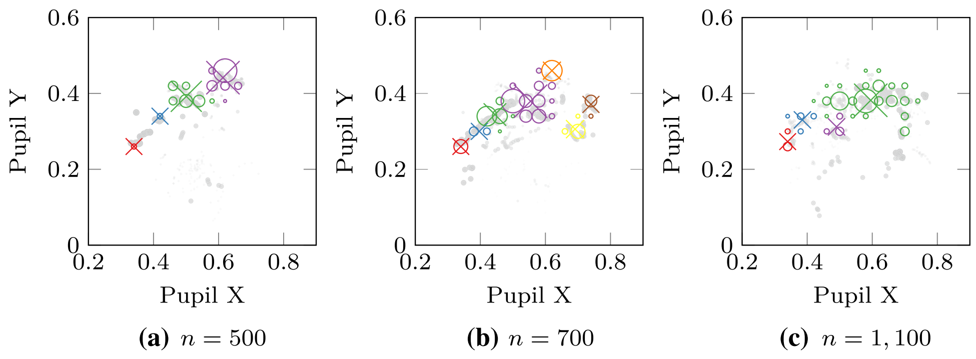


Fig. 1 Stream of eye tracking data (Steil et al. 2018) at three different points in time. Grey points denote the normalized pupil centers and their opacity and size is relative to their recency. Circles mark the centers of micro-clusters and crosses the centers of macro-clusters. Both are scaled relative to the number of observations assigned to them

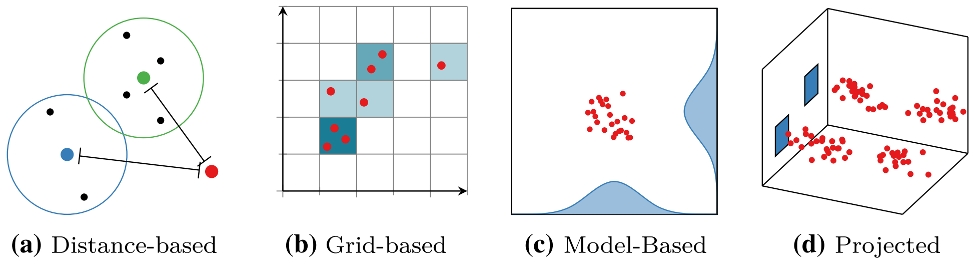


Fig. 2 Categories of stream clustering algorithms

considered clusters and they can even be merged when they become too similar over time. However, it is not possible to ever split a clusters again since the underlying data was discarded and only the centre of the dense region was stored (Aggarwal 2007). To avoid this problem, many stream clustering algorithms divide the process in two phases: an online and an offline component (Aggarwal et al. 2003).

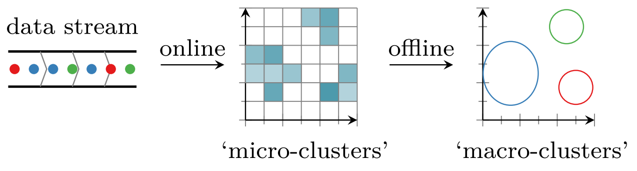


Fig. 3 Exemplary two-phase stream clustering using a grid-based approach (Carnein et al. 2017a)

2.1 Two-Phase Clustering

In the two-phase clustering approach, an online component evaluates arriving data points in real time and captures relevant summary statistics as outlined above. The result is a number of *micro-clusters* that represent a large number of preliminary clusters in the stream (Circles in Fig. 1). The number of micro-clusters is much smaller than the number of data points in the stream but larger than the final number of clusters. This gives sufficient flexibility to merge or split clusters, without the need to store all observations. Note that some publications refrain from using the term micro-clusters for grid-based approaches to highlight the different type of information that is maintained.

Upon request, an offline component then 'reclusters' the micro-clusters to derive a final set of *macro-clusters* (Crosses in Fig. 1). This process is usually not considered time-critical which allows to use variants of existing clustering algorithms. While most algorithms explicitly specify an offline component, the online and offline steps can usually be combined arbitrarily. The two-phase clustering approach is visualized in Fig. 3 by summarizing the data in a grid-structure. While the vast majority of algorithms apply such a two-phase process, some rely on incremental approaches where macro-clusters are generated incrementally without an intermediate step.

2.2 Time Window Models

As shown in our eye tracking example, the underlying distribution of the stream will often change over time. This is also known as drift or concept-shift. To handle this, algorithms can employ time window models. This approach aims to 'forget' older data to avoid that historic data is biasing the analysis to outdated patterns. There exist four main types of time window models (Fig. 4) (Silva et al. 2013; Nguyen et al. 2015).

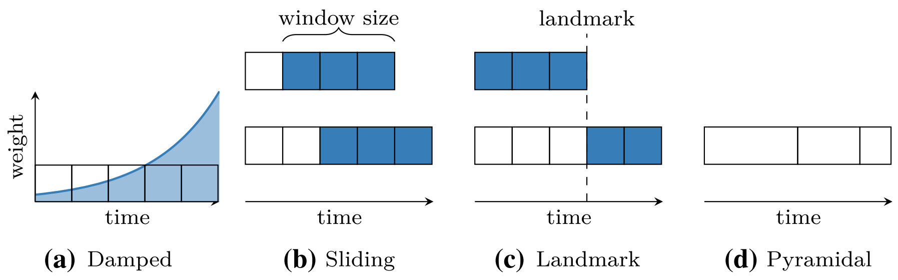


Fig. 4 Overview of time window models (Zhu and Shasha 2002; Silva et al. 2013)

The *damped time window* assigns a weight to each micro-cluster based on the number of observations assigned to it. In each iteration, the weight is faded by a factor such as $2^{-\lambda}$, where decay factor λ influences the rate of decay. Since fading the weight in every iteration is computationally costly, the weight can either be updated in fixed time intervals (Cao et al. 2006) or whenever a cluster is updated (Chen and Tu 2007). In this case, the fading can be performed with respect to the elapsed time $\omega(\Delta t) = 2^{-\lambda \Delta t}$ (Cao et al. 2006), where Δt denotes the time since the cluster was last updated. In Fig. 1, we applied the same fading function to reduce the size and opacity of older data. In some cases, clusters are implicitly decayed over time by considering their weight relative to the total number of observations (Gao et al. 2005; Amini et al. 2012).

An alternative is the *sliding time window* which only considers the most recent observations or micro-clusters in the stream. This is usually based on a First-In-First-Out (FIFO) principle, where the oldest data point in the window is removed once a new data point becomes available. The size of this window can be of fixed or variable length. While a small window size can adapt quickly to concept drift, a larger window size considers more observations and can be more accurate for stable streams.

In addition, a *landmark time window* is a very simple approach which separates the data stream into disjunct chunks based on events. Landmarks can either be defined based on the passed time or other occurrences. The landmark time window summarizes all data points that arrive after the landmark. Whenever a new landmark occurs, all the data in the window is removed and new data is captured. This category also includes algorithms that do not specifically consider changes over time and therefore require the user to regularly restart the clustering.

Finally, the *pyramidal time model* (Aggarwal et al. 2003) or *tilted time window* (Nguyen et al. 2015) uses different granularity levels based on the recency of data. This approach summarizes recent data more accurately whereas older data is gradually aggregated.

3 Related Work

Due to the increasing relevance of stream clustering, a number of survey papers began to summarize and structure the field. Most notably Amini et al. (2014b) provide an overview of the two largest research threads, namely distance-based and grid-based algorithms. In total, the authors discuss ten distance-based approaches, mostly extensions of `DenStream` (Cao et al. 2006), and nine grid-based approaches, mostly extensions of `D-Stream` (Chen and Tu 2007; Tu and Chen 2009). The authors describe the algorithms, name input parameters and also empirically evaluate some of the algorithms. In addition, the authors highlight interrelations between the algorithms in a timeline. We utilize this timeline and extend it with more algorithms and additional categories. However, their paper focusses only on distance and grid-based algorithms while we have taken more categories and more algorithms into account.

Additionally, Silva et al. (2013) introduced a taxonomy that allows to categorize stream clustering algorithms, e.g., regarding the reclustering algorithm or used time window model. The authors describe a total of 13 stream clustering algorithms and categorize them according to their taxonomy. In addition, application scenarios, data sources and available toolsets are presented. However, a drawback is that many of the discussed algorithms are one-pass clustering algorithms that need extensions to suit the streaming case.

In Ghesmoune et al. (2016) the authors discuss 19 algorithms and are among the first to highlight the research area of Neural Gas (NG) for stream clustering. However, only a single grid-based algorithm is discussed and other popular algorithms are missing. Further, the authors in Nguyen et al. (2015) focus on stream clustering and stream classification and present a total of 17 algorithms. Considerably shorter overviews are also provided in Mousavi et al. (2015), Ma (2014), Amini and Wah (2011, 2012) and Amini et al. (2011).

In this survey, we cover a total of 51 different stream clustering algorithms. This makes our survey much more exhaustive than all comparable studies. In addition, our paper identifies four common work streams and how they

Table 1 Overview of relevant characteristics of this survey

Number of algorithms	51
Years	1987–2018
Algorithm categories	4
Algorithm implementations	26
Software frameworks	3

developed over time. We also focus on common problems when applying stream clustering. As an example, we point to a total of 26 available algorithm implementations, as well as three different frameworks for data stream clustering. Furthermore, we address the problem of configuring stream clustering algorithms and present automatic algorithm configuration as an approach to address this problem. Table 1 briefly summarizes the relevant dimensions of our survey.

In previous work (Carnein et al. 2017a), we have also performed a rigorous empirical comparison of the most popular stream clustering algorithms. In total, we evaluated ten algorithms on four synthetic and three real-world data sets. In order to obtain the best results, we performed extensive parameter configuration. Our results have shown that DBSTREAM (Hahsler and Bolaños 2016) produces the highest cluster quality and is able to detect arbitrarily shaped clusters. However, it is sensitive to the insertion order and has many parameters which makes it difficult to apply in practice. As an alternative, D-Stream (Chen and Tu 2007; Tu and Chen 2009) can produce competitive results, but often requires more micro-clusters due to its grid based approach.

4 Distance-Based Approaches

Many approaches in stream clustering are *distance-based*. These algorithms typically threshold the distance of a new observation to existing micro-clusters and either insert it or initialize a new cluster. The main challenge for algorithms

Table 2 Overview of distance-based stream clustering algorithms

Algorithm	Year	Time window model	Offline clustering
BIRCH (Zhang et al. 1996)	1996	Landmark	Hierarchical clustering
ScaleKM (Bradley et al. 1998)	1998	Landmark	–
Single-pass k-means (Farnstrom et al. 2000)	2000	Landmark	–
STREAM (O'Callaghan et al. 2002)	2002	Landmark	k-median
CluStream (Aggarwal et al. 2003)	2003	Pyramidal	k-means
HCluStream (Yang and Zhou 2006)	2006	Pyramidal	k-means
DenStream (Cao et al. 2006)	2006	Damped	DBSCAN
E-Stream (Udommanetanakit et al. 2007)	2007	Damped	–
SWClustering (Zhou et al. 2007a)	2007	Pyramidal	k-means
Olindda (Spinosa et al. 2007)	2007	Landmark	k-means
RepStream (Lühr and Lazarescu 2009)	2008	Sliding	–
C-DenStream (Ruiz et al. 2009)	2009	Damped	C-DBSCAN (Ruiz et al. 2007)
rDenStream (Liu et al. 2009)	2009	Damped	DBSCAN
HDenStream (Lin and Lin 2009)	2009	Damped	DBSCAN
SDStream (Ren and Ma 2009)	2009	Pyramidal	DBSCAN
ClusTree (Kranen et al. 2009)	2009	Damped	Not specified
LiarTree (Hassani et al. 2011)	2011	Damped	Not specified
HUE-Stream (Meesuksabai et al. 2011)	2011	Damped	–
SOStream (Isaksson et al. 2012)	2012	Damped	–
StreamKM++ (Ackermann et al. 2012)	2012	Pyramidal	k-means
FlockStream (Forestiero et al. 2013)	2013	Damped	–
BICO (Fichtenberger et al. 2013)	2013	Landmark	k-means
LeaDen-Stream (Amini and Wah 2013)	2013	Damped	DBSCAN
G-Stream (Ghesmoune et al. 2014)	2014	Damped	–
Improved BIRCH (Ismael et al. 2014)	2014	Landmark	Hierarchical clustering
DBSTREAM (Hahsler and Bolaños 2016)	2016	Damped	Shared density
A-BIRCH (Lorbeer et al. 2017)	2017	Landmark	Hierarchical clustering
evoStream (Carnein and Trautmann 2018)	2018	Damped	Evolutionary algorithm

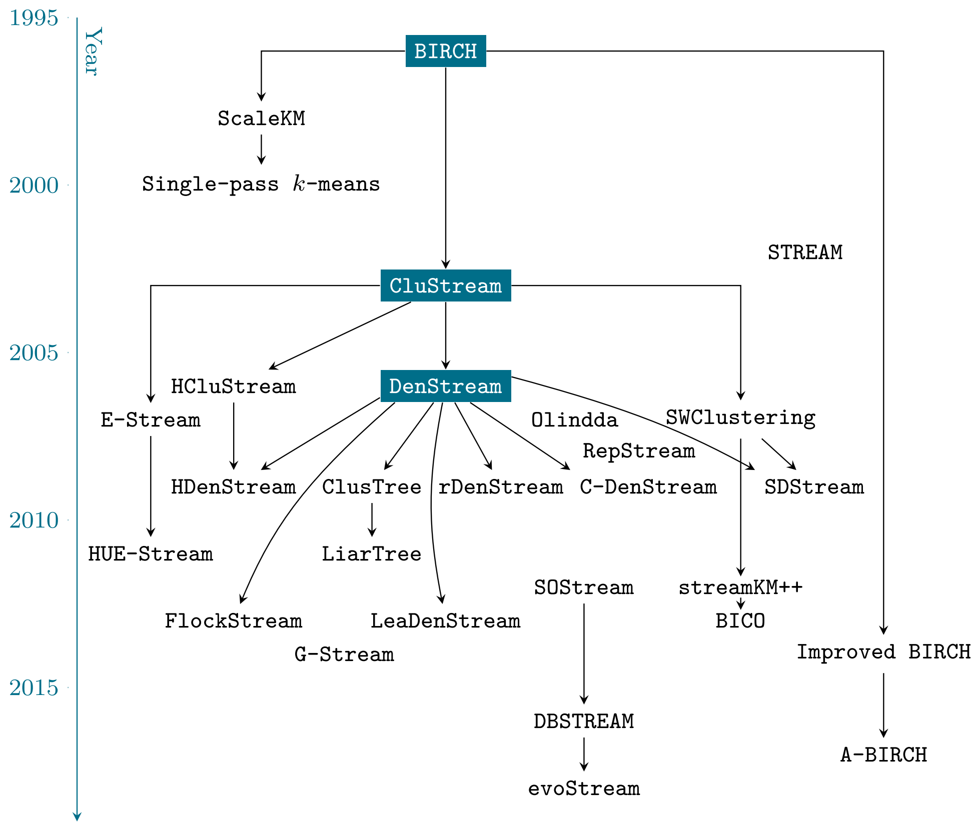


Fig. 5 Development of distance-based stream clustering algorithms

in this category is to summarize and maintain clusters over time without storing each individual observation. Common strategies include the use of a synopsis data structure which allows to calculate location and radius. Alternatively, the centroids or representatives of clusters can be maintained directly. More recently also competitive learning strategies became popular which can update the centers of clusters over time. Table 2 gives an overview of 28 popular distance-based stream clustering algorithms. In the following, each algorithm and its clustering strategy is discussed in more detail. In addition, Fig. 5 highlights the relationship between the algorithms and shows how concepts have been refined and improved over time. It becomes obvious that the vast majority of algorithms use concepts introduced by BIRCH, CluStream or DenStream.

4.1 Clustering Feature

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) (Zhang et al. 1996, 1997) is one of the earliest algorithms applicable to stream clustering. It reduces the information maintained about a cluster to only a few summary statistics stored in a so called Clustering Feature (CF). The CF consists of three components: $(n, \boldsymbol{LS}, SS)$, where n is the number of data points in the cluster, $\boldsymbol{LS}$ is a d-dimensional vector that contains the linear sum of all data points for each dimension and SS is a scalar that contains the sum of squares for all data points over all dimensions. Some variations of this concept also store the sum of squares per dimension, i.e., as a vector $\boldsymbol{SS}$. A CF provides sufficient information to calculate the centroid $\boldsymbol{LS}/n$ and also a radius, i.e., a measure of deviation from the centroid. In addition, a CF can be easily updated and merged with another CF by summing the individual components.

To maintain the CFs, BIRCH incrementally builds a balanced-tree as illustrated in Fig. 6, where each node can contain a fixed number of CFs. Each new observation descends the tree by following the child of its closest CF

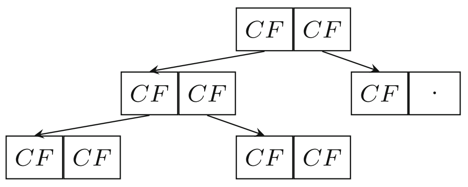


Fig. 6 Structure of a CF tree with at most 2 Clustering Features per node

until a leaf node is reached. The observation is either merged with its closest leaf-CF or used to create a new leaf-CF. For reclustering, all leaf-CF can be used as an input to a traditional algorithm such as k-means or hierarchical clustering.

`Improved BIRCH` (Ismael et al. 2014) is an extension which uses different distance thresholds per CF which are increased based on entries close to the radius boundary. Similarly, `A-BIRCH` (Lorbeer et al. 2017) estimates the threshold parameters by using the Gap Statistics (Tibshirani et al. 2001) on a sample of the stream.

`ScaleKM` (Bradley et al. 1998) is an incremental algorithm to cluster large databases which uses the concept of CFs. The algorithm fills a buffer with initial points and initializes k clusters as with standard k-means. The algorithm then decides for every point whether to discard, summarize or retain it. First, based on a distance threshold to the cluster centers and by creating a worst case perturbation of cluster centers, the algorithm identifies points that are unlikely to ever change their cluster assignments. These points are summarised in a CF per cluster and then discarded. Second, the remaining points are used to identify a larger number of micro-clusters by applying k-means and merging clusters using agglomerative hierarchical clustering. Each cluster is again summarised using a CF. All remaining points are kept as individual points. The freed space in the buffer is then filled with new points to repeat the process.

`Single pass` k `-means` (Farnstrom et al. 2000) is a simplified version of `scaleKM` where the algorithm discards all data points with every iteration and only the k CFs are maintained.

4.2 Extended Clustering Feature

`CluStream` (Aggarwal et al. 2003) extends the CF from `BIRCH` which allows to perform clustering over different time-horizons rather than the entire data stream. The extended CF is defined as $(\boldsymbol{LS}, \boldsymbol{SS}, LS^{(t)}, SS^{(t)}, n)$, where $LS^{(t)}$ and $SS^{(t)}$ are the linear and squared sum of all timestamps of a cluster. The online algorithm is initialized by collecting a chunk of data and using the k-means algorithm to create q clusters. When a new data point arrives, it is absorbed by its closest micro-cluster if it lies within an adaptive radius threshold. Otherwise, it is used to create a new cluster. In order to keep the number of micro-clusters constant, outdated clusters are removed based on a threshold on their average time stamp. If this is not possible, the two closest micro-clusters are merged.

To support different time-horizons, the algorithm regularly stores snapshots of the current CFs following a pyramidal scheme. While some snapshots are regularly updated, others are less frequently updated to maintain information about historic data. A desired portion of the stream can be approximated by subtracting the current CFs from a stored snapshot of previous CFs. The extracted micro-clusters are then used to run a variant of k-means to generate the macro-clusters.

`HCluStream` (Yang and Zhou 2006) extends `CluStream` for categorical data by storing the frequency of attribute-levels for all categorical features. Based on this, it defines a separate categorical distance measure which is combined with the traditional distance measure for continuous attributes.

`SWClustering` (Zhou et al. 2007a) uses the extended CF and pyramidal time window from `CluStream`. The algorithm maintains CFs in an Exponential Histogram of Cluster Features (EHCF) which stores data in different levels of granularity, depending on their recency. While the most recent observation is always stored individually, older observations are grouped and summarized. In particular, this step is organized in granularity levels. Once more than $1/\epsilon + 1$ CFs of a granularity level exist, the next CF contains twice as many entries (cf. Fig. 7). A new observation is either inserted into its closest CF or used to initialize a new one based on a radius threshold, similar to `BIRCH`. If the initialization creates too many individual CFs, the oldest two individual CFs are merged and this process cascades down the different granularity levels. An old CF is removed if its time-stamp is older than the last N observed time stamps. To generate the final clustering all CFs are used for reclustering, similar to `BIRCH`.

`SDStream` (Ren and Ma 2009) combines the EHCF from `SWClustering` to represent the potential core and outlier micro-clusters from `DenStream`. The algorithm also enforces an upper limit on the number of micro-clusters by either merging the two most similar micro-clusters or deleting outlier micro-clusters. The offline

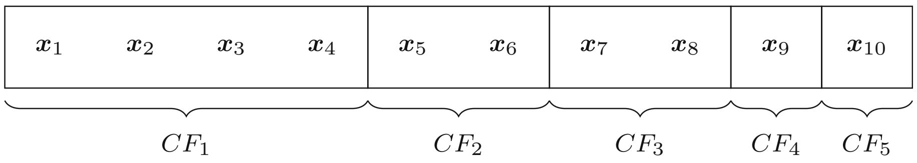

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
CF_1
CF_2
CF_3
CF_4
CF_5

Fig. 7 Granularity levels in an EHCF with $\epsilon = 1$. Recent observations are stored individually, whereas older data points are iteratively summarized

component applies DBSCAN to the centers of the potential core-micro clusters.

4.3 Time-Faded Clustering Feature

DenStream (Cao et al. 2006) presents a temporal extension of the CFs from BIRCH. It maintains two types of clusters: Potential core micro-clusters are stable structures that are denoted using a time-faded CF $\left(\boldsymbol{LS}^{(\omega)}, \boldsymbol{SS}^{(\omega)}, n^{(\omega)}\right)$. The superscript (ω) denotes that each entry of the CF is decayed over time using a decay function $\omega(\Delta t) = \beta^{-\lambda \Delta t}$. In addition, their weight $n^{(\omega)}$ is required to be greater than a threshold value. Outlier micro-clusters are unstable structures whose weight is less than the threshold and they additionally maintain their creation time.

At first, DBSCAN (Ester et al. 1996) is used to initialize a set of potential core micro-clusters. Similar to BIRCH, a new observation is assigned to its closest potential core micro-cluster if the addition does not increase the radius beyond a threshold. If it does, the same attempt is made for the closest outlier-cluster and the outlier-cluster is promoted to a potential core if it satisfies the weight threshold. If both cannot absorb the point, a new outlier-cluster is initialized. In regular intervals, the weight of all micro-clusters is evaluated. Potential core-micro clusters that no longer have enough weight are degraded to outlier micro-clusters and outlier micro-clusters that decayed below a threshold based on their creation time are removed. Macro-clusters are generated by applying a variant of DBSCAN (Ester et al. 1996) to potential core micro-clusters.

C-DenStream (Ruiz et al. 2009) is an extension of DenStream which allows to include domain knowledge in the form of instance-level constraints into the clustering process. Instance-level constraints describe observations that must or cannot belong to the same cluster.

Another extension is rDenStream (Liu et al. 2009). Instead of discarding outlier micro-clusters which cannot be converted into potential core micro-clusters, the algorithm temporarily stores them away in an outlier buffer. After the offline component, the algorithm attempts to relearn the data points that have been cached in the buffer in order to refine the clustering.

HDenStream (Lin and Lin 2009) combines D-Stream with the categorical distance measure of HCluStream to make it applicable to categorical features.

E-Stream (Udommanetanakit et al. 2007) uses the time-faded CF from DenStream in combination with a histogram which bins the data points. New observations are either added to their closest cluster or used to initialize a new one. Existing clusters are split if one of the dimensions shows a significant valley in their histogram. When a cluster is split along a dimension, the other dimensions are weighted by the size of the split. Additionally, clusters can be merged if they move into close proximity.

HUE-Stream (Meesuksabai et al. 2011) is an extension of E-Stream which also supports categorical data and can also handle uncertain data streams. To model uncertainty, each observation is assumed to follow a probability distribution. In this case, the vectors of linear and squared sum become the sum of expectation, faded over time.

ClusTree (Kranen et al. 2009, 2011a) uses the time-faded CF and applies it to the tree structure of BIRCH. Additionally, it allows to handle data streams where entries arrive faster than they can be processed. A new entry descends into its closest leaf where it is inserted as a new CF. Whenever a node is full, it is split and its entries combined in two groups such that the intra-cluster distance is minimized. However, if a new observation arrives before a node could be split, the new entry is merged with its closest CFs instead. If a new observation arrives while an entry descends the tree, that entry is temporarily stored in a buffer at its current location. It remains there until another entry descends into the same branch and is then carried further down the tree as a ‘hitchhiker’. Again, the leafs can be used as an input to a traditional algorithm to generate the macro-clusters.

LiarTree (Kranen et al. 2011b; Hassani et al. 2011) is an extension of ClusTree with better noise and novelty handling. It does so by adding a time-weighted CF to each node of the tree which serves as a buffer for noise. Data points are considered noise with respect to a node based on a threshold on their distance to the node’s mean, relative to the node’s standard deviation. The noise buffer is promoted to a regular cluster when its density is comparable to other CFs in the node.

FlockStream (Forestiero et al. 2013) employs a flocking behavior inspired by nature to identify emerging flocks and swarms of objects. Similar to DenStream, the algorithm distinguishes between potential core and outlier micro-clusters and uses a time-faded CF. It projects a batch of data onto a two-dimensional grid where each data point is represented by a *basic agent*. Each agent then makes movement decisions solely based on other agents in close proximity. The movement of agents is similar to the behavior of a flock of birds in flight: (1) Agents steer in the same direction as their neighbors; (2) Agents steer towards the location of their neighbors; (3) Agents avoid collisions with neighbors. When agents meet, they can be merged depending on a distance or radius threshold. After a number of flocking steps, the next batch of data is used to fill the grid with new agents in order to repeat the process.

LeaDen-Stream (Amini and Wah 2013) (Leader Density-based clustering algorithm over evolving data

Stream) can choose multiple representatives per cluster to increase accuracy when clusters are not uniformly distributed. To do so, the algorithm maintains two different granularity levels. First, Micro Leader Clusters (MLC) correspond to the concept of traditional micro-clusters. However, they maintain a list of more fine granular information in the form of Mini Micro Leader Clusters (MMLC). These mini micro-clusters contain more detailed information and are represented by a time-faded CF. For new observations, the algorithm finds the closest MLC using the Mahalanobis distance. If the distance is within a threshold, the closest MMLC within the MLC is identified. If it is also within a distance threshold, the point is added to the MMLC. If one of the thresholds is violated, a new MLC or MMLC is created respectively. For reclustering all selected leaders are used to run DBSCAN.

4.4 Medoids

An alternative to storing Clustering Features is to maintain medoids of clusters, i.e., representatives. RepStream (Lühr and Lazarescu 2008, 2009), for example, incrementally updates a graph of nearest neighbors to identify suitable cluster representatives. New observations are inserted as a new node in the graph and edges are inserted between the node and its nearest-neighbors. The point is assigned to an existing cluster if it is mutually connected to a representative of that cluster. Otherwise it is used as a representative to initialize a new cluster. Representatives are also inserted in a separate representative graph which maintains the nearest neighbors only between representatives. To split and merge existing clusters, the distance between them is compared to the average distance to their nearest neighbors in the representative graph. In order to reduce space requirements, non-representative points are discarded using a sliding time window. In addition, if a new representative is found but space limitations prevent it from being added to the representative graph, it can replace an existing representative depending on its age and number of nearest neighbors.

streamKM++ (Ackermann et al. 2012) is a variant of k-means++ (Arthur and Vassilvitskii 2007) which computes a small weighted sample that represents the data called *coreset*. The coreset is constructed in a binary tree by using a divisive clustering approach. The tree is initialized by selecting a random representative point from the data. To split an existing cluster, the algorithm starts at the root node and iteratively chooses a child node relative to their weights until a leaf is reached. From the selected leaf, a data point is chosen as a second centre based on its distance to the initial centre of the cluster. Finally, the cluster is split by assigning each data point to the closest of the two centers.

To handle data streams, the algorithm uses a similar approach as SWClustering (see Sect. 4.2). First, new observations are inserted into a coreset tree. Once the tree is full, all its points are moved to the next tree. If the next tree already contains points, the coreset between the points in both trees is computed. This cascades further until an empty tree is found. For reclustering, the union of all points is used to compute a coreset and the representatives are used to apply the k-means++ algorithm (Arthur and Vassilvitskii 2007).

BICO (Fichtenberger et al. 2013) combines the data structure of BIRCH (see Sect. 4.1) with the coreset of streamKM++. BICO maintains the coreset in a tree structure where each node represents one CF. The algorithm is initialized by using the first data point in the stream to open a CF on the first level of the empty tree and the data point is kept as a representative for the CF. For every consecutive point, the algorithm attempts to insert it into an existing CF, starting on the first level. The insertion fails if the distance of the new point to the representative of its closest CF is larger than a threshold. In this case, a new CF is opened on the same level, using the new point as the reference point. Additionally, the insertion fails if the cluster's deviation from the mean would grow beyond a threshold. In this case the algorithm attempts to insert the point into the children of the closest CF. The final clustering is generated by applying k-means++ to the representatives of the leafs.

4.5 Centroids

A simpler approach to maintain clusters is to store their centroids directly. However, this makes it generally more difficult to update clusters over time. As an example, STREAM (O'Callaghan et al. 2002; Guha et al. 2003) only stores the centroids of k clusters. Its core idea is to treat the k-Median clustering problem as a facility planning problem. To do so, distances from data points to their closest cluster have associated costs. This reduces the clustering task to a cost minimization problem in order to find the number and position of facilities that yield the lowest costs. In order to generate a certain number of clusters, the algorithm adjusts the facility costs in each iteration by using a binary search for the costs that yield the desired number of centers k.

To deal with streaming data, the algorithm processes the stream in chunks and solves the k-Median problem for each chunk individually. Assuming n different chunks, a total of nk clusters are created. To generate the final clustering or if available storage is exceeded, these intermediate clusters are again clustered using the same approach.

OLINDDA (Spinosa et al. 2007) (Online Novelty and Drift Detection Algorithm) relies on cluster centroids to identify new and drifting clusters in a data stream. Initially, *k*-means is used to generate a set of clusters. For each cluster the distance from its centre to its furthest observation is considered a boundary. Points that do not fall into the boundary of any cluster are considered as an unknown concept and kept in a buffer. This buffer is regularly scanned for emerging structures using *k*-means. If an emerging cluster is of similar variance as the existing cluster, it is considered valid. To distinguish a new cluster from a drifting cluster, the algorithm assumes that drifts occur close to the existing clusters whereas new clusters form further away from the existing model.

4.6 Competitive Learning

More recently, algorithms also use competitive learning strategies in order to adapt the centroids of clusters over time. This is inspired by Self-Organizing Maps (SOMs) (Kohonen 1982) where clusters compete to represent an observation, typically by moving cluster centers towards new observations based on their proximity. `SOStream` (Isaksson et al. 2012) (Self Organizing density based clustering over data Stream) combines `DBSCAN` (Ester et al. 1996) with Self-Organizing Maps (SOMs) (Kohonen 1982) for stream clustering. It stores a time-faded weight, radius and centre for the cluster directly. A new observation is merged into its closest cluster if it lies within its radius. Following the idea of competitive learning, the algorithm also moves the *k*-nearest neighbors of the absorbing cluster in its direction. If clusters move into close proximity during this step, they are also merged.

`DBSTREAM` (Hahsler and Bolaños 2016) (Density-based Stream Clustering) is based on `SOStream` (see Sect. 4.6) but uses the shared density between two micro-clusters in order to decide whether micro-clusters belong to the same macro-cluster. A new observation $\boldsymbol{x}$ is merged into micro-clusters if it falls within the radius from their centre. Subsequently, the centers of all clusters that absorb the observation are updated by moving the centre towards $\boldsymbol{x}$. If no cluster absorbs the point, it is used to initialize a new micro-cluster. Additionally, the algorithm maintains the shared density between two micro-clusters as the density of points in the intersection of their radii, relative to the size of the intersection area. In regular intervals it removes micro-clusters and shared densities whose weight decayed below a respective threshold. In the offline component, micro-clusters with high shared density are merged into the same cluster.

`evoStream` (Carnein and Trautmann 2018) (Evolutionary Stream Clustering) makes use of an evolutionary algorithm in order to bridge the gap between the online and offline component. Evolutionary algorithms are inspired by natural evolution where promising solutions are combined and slightly modified to create offsprings which can yield an improved solution. By iteratively selecting the best solutions, an evolutionary pressure is created which improves the result over time. `evoStream` uses this concept in order to iteratively improve the macro-clusters through recombinations and small variations. Since macro-clusters are created incrementally, the evolutionary steps can be performed while the online components waits for new observations, i.e., when the algorithm would usually idle. As a result, the computational overhead of the offline component is removed and clusters are available at any time. The online component is similar to `DBSTREAM` but does not maintain a shared-density since it is not necessary for reclustering.

`G-Stream` (Ghesmoune et al. 2014, 2015) (Growing Neural Gas over Data Streams) utilizes the concept of Neural Gas (Martinetz et al. 1991) for data streams. The algorithm maintains a graph where each node represents a cluster. Nodes that share similar data points are connected by edges. Each edge has an associated age and nodes maintain an error term denoting the cluster's deviation. For a new observation $\boldsymbol{x}$ the two nearest clusters C_1 and C_2 are identified. If $\boldsymbol{x}$ does not fit into the radius of its closest cluster C_1, it is temporarily stored away and later re-inserted. Otherwise, it is inserted into C_1. Additionally, the centre of C_1 and all its connected neighbors are moved in the direction of $\boldsymbol{x}$. Next, the age of all outgoing edges of C_1 are incremented and an edge from C_1 to C_2 is either inserted or its weight is reset to zero. The age of edges serves a similar purpose as a fading function. Edges who have grown too old, are removed as they contain outdated information. In regular intervals, the algorithm inserts new nodes between the node with the largest deviation and its neighbor with the largest deviation.

4.7 Summary

Distance-based algorithms are by far the most common and popular approaches in stream clustering. They allow to create accurate summaries of the entire stream with rather simple insertion rules. Since it is infeasible to store all observations within the clusters, distance-based algorithms usually summarize the observations associated with a cluster. A popular example of this are Clustering Features which only store the information required to calculate the location and radius of a cluster. Alternatively, some algorithms maintain medoids, i.e., representatives of clusters or store the cluster centroids directly. In order to update cluster centroids over time, some algorithms also make use of competitive learning strategies, similar to Self-Organizing Maps (SOM) (Kohonen 1982). Generally, distance-based algorithms are computationally inexpensive and will suit the majority of stream clustering scenarios well. However, they often rely on many parameters such as distance and weight thresholds, radii or cleanup intervals. This makes it more difficult to apply them in practice and requires either expert knowledge or extensive parameter configuration. Another common issue is that distance-based algorithms can often only find spherical clusters. However, this is usually due to the choice of offline component which can be easily replaced by other approaches that can detect arbitrary clusters such as `DBSCAN` or hierarchical clustering with single linkage. While the popular algorithms `BIRCH`, `CluStream` and `DenStream` face many problems, either due to lack of fading or complicated maintenance steps, we find newer algorithms such as `DBSTREAM` particularly interesting due to their simpler design.

5 Grid-Based Approaches

An alternative to distance-based approaches is to capture the density of observations in a grid. A grid separates the data space along all dimensions into intervals to create a number of grid-cells. By mapping data points to the cells, a density estimate can be maintained. Macro-clusters are typically found by grouping adjacent dense cells. The main challenge for algorithms of this category is how to construct the grid-cells, i.e., how often cells are partitioned and how to choose the size of cells. Table 3 gives an overview of the 13 approaches that are discussed in the following. In addition, Fig. 8 shows how density-based algorithms developed over time. By far the most popular and influential algorithm of this category has been `D-Stream` (Chen and Tu 2007).

5.1 One-Time Partitioning

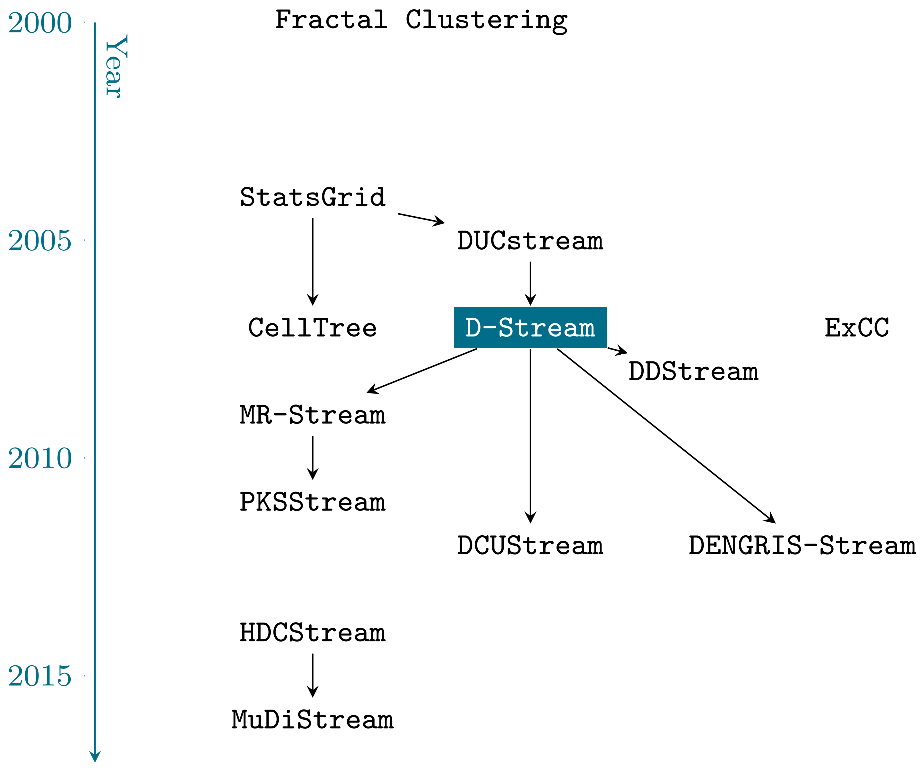
DUCstream (Gao et al. 2005) (Dense Units Clustering for data streams) is one of the earliest grid-based algorithms. It partitions the data space once into grid-cells of fixed size. To initialize the clustering, the algorithm processes a first chunk of data and maintains all cells with sufficient density. The density of cells is calculated relative to the total number of observations. All dense cells that are connected by common faces are placed in the same macro-cluster. This initial result is then updated incrementally as more chunks are processed and new dense cells arise while others fade. Each new dense cell is absorbed by a macro-cluster if it shares a face with a cell in that cluster. Additionally, if it shares faces with cells in different clusters, those clusters are merged. Finally, if it does not share any faces, it is used to create a new cluster. Each removed dense cell is also removed from its corresponding cluster. If this leaves the cluster empty, the cluster is deleted. Alternatively, if this disconnects two cells in the same cluster, the cluster is split.

`D-Stream` (Chen and Tu 2007) is among the most popular stream clustering algorithms and uses a fixed grid structure. The algorithm distinguishes between three types of cells: dense cells, sparse cells and transitional cells whose weight lies between the other two types. The algorithm maps new data points to its respective cell and is initialized by assigning all dense cells to individual clusters. These clusters are extended with all neighboring

Table 3 Overview of density-based stream clustering algorithms. Macro-clusters are typically generated from neighboring dense cells

Algorithm	Year	Time window model	Partitioning	Grid size
`Fractal Clustering` (Barbará and Chen 2000)	2000	Landmark	One-time	Fixed
`Stats-Grid` (Park and Lee 2004)	2004	Landmark	Recursive	Dynamic
`DUCstream` (Gao et al. 2005)	2005	Damped	One-time	Fixed
`D-Stream` (Chen and Tu 2007)	2007	Damped	One-time	Fixed
`Cell-Tree` (Park and Lee 2007a)	2007	Damped	Recursive	Fixed
`ExCC` (Bhatnagar and Kaur 2007)	2007	Landmark	One-time	Fixed
`DDStream` (Jia et al. 2008)	2008	Damped	One-time	Fixed
`MR-Stream` (Wan et al. 2009)	2009	Damped	Recursive	Fixed
`PKSStream` (Ren et al. 2011)	2011	Damped	Recursive	Fixed
`DCUStream` (Yang et al. 2012)	2012	Damped	One-time	Fixed
`DENGRIS-Stream` (Amini et al. 2012)	2012	Sliding	One-time	Fixed
`HDCStream` (Amini et al. 2014a)	2014	Damped	One-time	Fixed
`MuDi-Stream` (Amini et al. 2016)	2016	Damped	One-time	Fixed

Fig. 8 Development of density-based stream clustering algorithms



transitional grids or are merged with the clusters of neighboring dense cells. In regular intervals, the clustering evaluates the weight of each cell and incorporates the changes in cell types into the clustering.

In Tu and Chen (2009), the authors extended their concept by a measure of attraction that incorporates positional information of data within a grid-cell. This variant only merges neighboring cells if they share many points at the cell border.

`DD-Stream` (Jia et al. 2008) is a small extension on how to handle points that lie exactly on the grid boundaries. For such a point, the distance to adjacent cell centers is computed and the point is assigned to its closest cell. If the observation has the same distance to multiple cells, it is assigned to the one with higher density. If this also does not break the tie, it is inserted into cell that has been updated more recently.

`ExCC` (Bhatnagar and Kaur 2007; Bhatnagar et al. 2014) (Exclusive and Complete Clustering) constructs a grid where the number of cells and grid boundaries are chosen by the user. This allows to handle categorical data, where the number of cells is chosen to be equal to the number of attribute levels. Clusters are identified as neighboring dense cells. Cells of numeric variables are considered neighbors if they share a common vertex. Cells of categorical variables employ a threshold on a similarity function between the attribute levels. To form macro-clusters, the algorithm iteratively chooses an unvisited dense cell and initializes a new cluster. Each neighboring grid-cell is then placed in the same cluster. This is repeated until all cells have been visited.

DCUStream (Yang et al. 2012) (Density-based Clustering algorithm of Uncertain data Stream) aims to handle uncertain data streams, similar to `HUE-Stream` (see Sect. 4.3), where each observation is assumed to have an existence probability. The algorithm is initialized by collecting a batch of data and mapping it to a grid of fixed size. The density of a cell is defined as the sum of all existence probabilities faded over time. A grid is considered dense when its density is above a dynamic threshold. To generate a clustering, the algorithm selects the dense-cell with highest density and assigns all its neighboring cells to the same cluster. neighboring sparse-cells are considered the boundary of a cluster. This is repeated for all dense cells.

DENGRIS-Stream (Amini et al. 2012) (Density Grid-based algorithm for clustering data streams over Sliding window) is a grid-based algorithm that uses a sliding window model. New observations are mapped into a fixed size grid and the cell's densities are maintained. Densities are implicitly decayed by considering them relative to the total number of observations in the stream. In regular intervals, cells whose density decayed below a threshold or cells that are no longer inside the sliding window are removed. Macro-clusters are formed by grouping neighboring dense cells into the same cluster.

Fractal Clustering (Barbará and Chen 2000, 2003) follows an usual grid-based approach. It uses the concept of fractal dimensions (Theiler 1990) as a measure of size for a set of points. A common way to calculate the fractal dimension is by dividing the space into grid-cells of size ϵ and counting the number of cells that are occupied by points in the data $N(r)$. Then, the fractal dimension can be calculated as:

$$D = \lim_{\epsilon \to 0} \frac{\log N(\epsilon)}{\log \frac{1}{\epsilon}}. \quad (1)$$

Fractal Clustering is first initialized with a sample by recursively placing close points into the same cluster (similar to DBSCAN). For a new observation, the algorithm then evaluates which influence in fractal dimension the addition of the point would have for each cluster. It then inserts the point into the cluster whose fractal dimension changes the least. However, if the change in fractal dimension is too large, the observation is considered noise instead.

5.2 Recursive Partitioning

Stats-Grid (Park and Lee 2004) is an early algorithm which recursively partitions grid-cells. The algorithm begins by splitting the data into grid-cells of fixed size. Each cell maintains its density, mean and standard deviation. The algorithm then recursively partitions grid-cells until cells become sufficiently small *unit cells*. The aim is to find adjacent unit cells with large density which can be used to form macro-clusters. The algorithm splits a cell in two subcells whenever it has reached sufficient density. The size of the subcells is dynamically adapted based on the distribution of data within the cell. The authors propose three separate splitting strategies, for example choosing the dimension where the cell's standard deviation is the largest and splitting at the mean. Since the weight of cells is calculated relative to the total number of observations, outdated cells can be removed and their statistics returned to the parent cell.

Cell-Tree (Park and Lee 2007a) is an extension of Stats-Grid which also tries to find adjacent unit cells of sufficient density. In contrast to Stats-Grid, subcells are not dynamically sized based on the distribution of the cell. Instead, they are split into a pre-defined number of evenly sized subcells. The summary statistics of the subcells are initialized by distributing the statistics of the parent cell following the normal distribution. To efficiently maintain the cells, the authors propose a siblings list. The siblings list is a linear list where each node contains a number of grid-cells along one dimension as well as a link to the next node. Whenever a cell is split, the created subcells replace their parent cell in its node. To maintain a siblings list over multiple dimensions, a first-child/next-sibling tree can be used where subsequent dimensions are added as children of the list-nodes.

The splitting strategy of MR-Stream (Wan et al. 2009) is similar but splits each dimension in half, effectively creating a tree of cells as shown in Fig. 9. New observations start at the root cell and are recursively assigned to the appropriate child-cell. If a child does not exist yet, it is created until a maximum depth is reached. If the insertion causes a parent to only contain children of high density, the children are discarded since the parent node is able to represent this information already. Additionally, the tree is regularly pruned by removing leafs with insufficient weight and removing children of nodes that only contain dense or only sparse children. To generate the macro-clusters, the user can choose a desired height of the tree. For every unclustered cell, the algorithm initializes a new macro-cluster and adds all neighboring dense cells. If the size and weight of the cluster is too low, it is considered noise.

PKSStream (Ren et al. 2011) is similar to MR-Stream but does not require a subcell on all heights of the tree. It only maintains intermediate nodes when there are more than $K-1$ non-empty children. Each observation is iteratively descended down the tree until either a leaf is reached or the child does not exist. In the latter case a new cell is initialized. In regular intervals, the algorithm evaluates all leaf nodes and removes those with insufficient weight. The offline component is the same as in MR-Stream for the leafs of the tree.

5.3 Hybrid Grid-Approaches

HDCStream (Amini et al. 2014a) (hybrid density-based clustering for data stream) first combined grid-based algorithms with the concept of distance-based algorithms. In particular, it maintains a grid where dense cells can be promoted to become micro-clusters as known from distanced-based algorithms (see Sect. 4). Each observation in

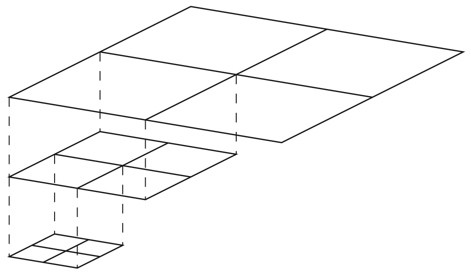


Fig. 9 Tree structure in MR-Stream

the stream is assigned to its closest micro-cluster if it lies within a radius threshold. Otherwise, it is inserted into the grid instead. Once a grid-cell has accumulated sufficient density, its points are used to initialize a new micro-cluster. Finally, the cell is no longer maintained, as its information has been transferred to the micro-cluster. In regular intervals, all micro-clusters and cells are evaluated and removed if their density decayed below a respective threshold. Whenever a clustering request arrives, the micro-clusters are considered virtual points in order to apply DBSCAN.

Mudi-Stream (Amini et al. 2016) (Multi Density Data Stream) is an extension of HDCStream that can handle varying degrees of density within the same data stream. It uses the same insertion strategy as HDCStream with both, grid-cells and micro-clusters. However, the offline component applies a variant of DBSCAN (Ester et al. 1996) called M-DBSCAN to all micro-clusters. M-DBSCAN only requires a *MinPts* parameter and then estimates the ϵ parameter from the mean and standard deviation around the centre.

5.4 Summary

Grid-based approaches are a popular alternative to density-based algorithms due to their simple design and support for arbitrarily shaped clusters. While many distance-based algorithms are only able to detect spherical clusters, almost all grid-based algorithms can identify cluster of arbitrary shape. This is mostly because the grid-structure allows an easy design of an offline-component where dense cells with common faces form clusters. The majority of grid-based algorithms partition the data space once into cells of fixed size. However, some algorithms do this recursively to create a more adaptive grid. Less common are algorithms where the size of cells is determined dynamically, mostly because of the increased computational costs. Lastly, some algorithms employ a hybrid strategy where a grid is used to establish distance-based approaches. Generally, the grid structure is less efficient than distance-based approaches due to its inflexible structure. For this reason, grid-based approaches often have higher memory requirements and need more micro-clusters to achieve the same quality as distance-based approaches. Empirical evidence (Carnein et al. 2017a) has also shown this to be true for the most popular grid-based algorithm D-Stream.

6 Model-Based Approaches

A different approach to stream clustering is to summarize the data stream as a statistical model. Common areas of research are based on the Expectation Maximization (EM) algorithm. Table 4 gives an overview of 6 model-based algorithms. This class of algorithms is highly diverse and few interdependencies exist between the presented algorithms.

Table 4 Overview of model-based stream clustering algorithms

Algorithm	Year	Time window model
COBWEB (Fisher 1987)	1987	Landmark
ICFR (Motoyoshi et al. 2004)	2004	Damped
WStream (Tasoulis et al. 2006)	2006	Damped
CluDistream (Zhou et al. 2007b)	2007	Landmark
SWEM (Dang et al. 2009a)	2009	Sliding
SVStream (Wang et al. 2013)	2013	Damped

CluDistream (Zhou et al. 2007b) uses the EM algorithm to process distributed data streams. At each location, it maintains a number of Gaussian mixture distributions and a coordinator node is used to combine the distributions. For each location, the stream is processed in chunks and the first chunk is used to initialize a new clustering using EM. For subsequent chunks, the algorithm checks whether the current models can represent the chunk sufficiently well. This is done by calculating the difference between the average log-likelihood of the existing model and the average log-likelihood of the chunk under the existing model. If the difference is less than a threshold, the weight of the model is incremented. Else, the current model is stored and a new model is initialized by applying EM to the current chunk. Whenever the weight of a model is updated or a new model is initialized, the coordinator receives the update and incorporates the new information into a global model by merging or splitting the Gaussian distributions.

SWEM (Dang et al. 2009a, b) (Sliding Window with Expectation Maximization) applies the EM to chunks of data. Starting with random initial parameters, a set of m distributions is calculated for the first chunk and points are assigned to their most likely cluster. Each cluster is then summarized using a CF and k macro-cluster are generated by applying EM again. For a new chunk, the algorithm sets the initial values to the converged values of the previous chunk and incrementally applies EM to generate m new distributions. If a cluster grows too large or too small during this phase, the corresponding distributions can be split or merged. Finally the m new clusters are summarized in CFs and used with the existing k clusters to apply EM again.

COBWEB (Fisher 1987) maintains a classification tree where each node describes a cluster. The tree is built incrementally by descending a new entry $\boldsymbol{x}$ from the root to a leaf. On each level the algorithm makes one of four clustering decisions that yields the highest clustering quality: (1) Insert $\boldsymbol{x}$ into most fitting child, (2) Create a new cluster for $\boldsymbol{x}$, (3) Combine the two nodes that can best absorb $\boldsymbol{x}$ and add existing nodes as children of the new node, (4) Split the two nodes that can best absorb $\boldsymbol{x}$ and move its children up one level. The quality of each decision is evaluated using a measure called Category Utility (CU) which defines a trade-off between intra-class similarity and inter-class distance.

`ICFR` (Motoyoshi et al. 2004) (Incremental Clustering using F-value by Regression analysis) uses concepts from linear regression in order to cluster data streams. The algorithm assigns points to existing clusters based on their cosine similarity. To merge clusters the algorithm finds the two closest clusters based on the Mahalanobis distance. If the merged clusters yield a greater F-value than the sum of individual F-values, the clusters are merged. The F-value is a measure of model validity in linear regressions. If the clusters cannot be merged, the next closest two clusters are evaluated until the closest pair exceeds a distance threshold.

`WStream` (Tasoulis et al. 2006) uses multivariate kernel density estimates to maintain a number of rectangular windows in the data space. The idea is to use local maxima of a density estimate as cluster centers and the local minima as cluster boundaries. `WStream` transfers this approach to data streams. New data points are either assigned to an existing window and their centre is moved towards the new point or it is used to initialize a new window of default size. Windows can enlarge or contract depending on the ratio of points close to their centre and close to their border.

`SVStream` (Wang et al. 2013) (Support Vector based Stream Clustering) is based on Support Vector Clustering(SVC) (Ben-Hur et al. 2001). SVC transforms the data into a higher dimensional space and identifies the smallest sphere that encloses most points. When mapping the sphere back to the input space, the sphere forms a number of contour lines that represent clusters. `SVStream` iteratively maintains a number of spheres. The stream is processed in chunks and the first chunk is used to run SVC. For each subsequent chunk, the algorithm evaluates what portion of the chunk does not fall into the radius of existing spheres. If too many do not fit the current spheres, these values are used to initialize a new sphere. The remaining values are used to update the existing spheres.

6.1 Summary

Model-based stream clustering algorithms are far less common than distance-based and grid-based approaches. Typically strategies try to find a mixture of distributions that fits the data stream, e.g. `CluDiStream` or `SWEM`. Unfortunately, no implementation of model-based algorithms is readily available which limits their usefulness in practice. In addition, they are often more computationally complex than comparable algorithms from the other categories.

7 Projected Approaches

A special category of stream clustering algorithms deals with high dimensional data streams. These types of algorithms address the curse of dimensionality (Beyer et al. 1999), i.e., the problem that almost all points have an equal distance in very high dimensional space. In such scenarios, clusters are defined according to a subset of dimensions where each cluster has an associated set of dimensions in which it exists. Even though these algorithms often use concepts from distance and grid-based algorithms their application scenarios and strategies are unique and deserve their own category. Table 5 summarizes 4 projected clustering algorithms and Fig. 10 shows the relationship between the algorithms. Despite their similarity, `HDDStream` and `PreDeConStream` have been developed independently.

`HPStream` (Aggarwal et al. 2004) (High-dimensional Projected Stream clustering) is an extension of `CluStream` (see Sect. 4.2) for high dimensional data. The algorithm uses a time-faded CF with an additional bit vector that denotes the associated dimensions of a cluster. The algorithm normalizes each dimension by regularly sampling the current standard deviation and adjusting the existing clusters accordingly. The algorithm initializes with k-means and associates each cluster with the l dimensions in which it has the smallest radius. The cluster assignment

Table 5 Overview of projected stream clustering algorithms

Algorithm	Year	Time window model	Offline clustering
`HPStream` (Aggarwal et al. 2004)	2004	Damped	k-means
`SiblingTree` (Park and Lee 2007b)	2007	Damped	–
`HDDStream` (Ntoutsi et al. 2012)	2012	Damped	`PreDeCon` (Bohm et al. 2004)
`PreDeConStream` (Hassani et al. 2012)	2012	Damped	`PreDeCon` (Bohm et al. 2004)

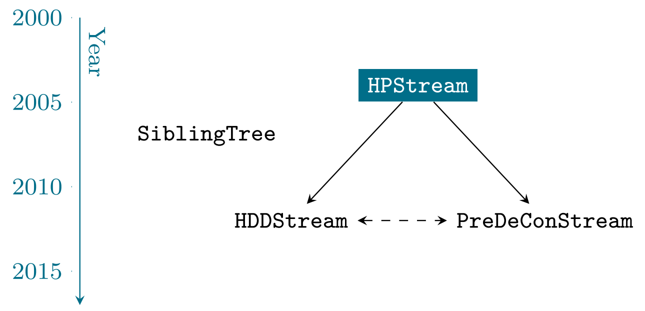


Fig. 10 Development of projected stream clustering algorithms

is then updated by only considering the associated dimensions for each cluster. Finally, the process is repeated until the cluster and dimensions converge. A new data point is tentatively added to each cluster to update the dimension association and added to its closest cluster if it does not increase the cluster radius above a threshold.

SiblingTree (Park and Lee 2007b) is an extension of CellTree (Park and Lee 2007a) (see Sect. 5.2). It uses the same tree-structure but allows for subspace clusters. To do so, the algorithm creates a siblings list for each dimension as children of the root. New data points are recursively assigned to the grid-cells using a depth first approach. If a cell's density increases beyond a threshold, it is split as in CellTree. If a unit cell's density increases beyond a threshold, new sibling lists for each remaining dimension are created as children of the cell. Additionally, if a cell's density decays below a density threshold, its children are removed and it is merged with consecutive sparse cells. Clusters in the tree are defined as adjacent unit-grid-cells with enough density.

HDDStream (Ntoutsi et al. 2012) (Density-based Projected Clustering over High Dimensional Data Streams) is initialized by collecting a batch of observations and applying PreDeCon (Bohm et al. 2004). PreDeCon can be considered a subspace version of DBSCAN. The update procedure is similar to DenStream (see Sect. 4.3): A new observation is assigned to its closest potential core micro-cluster if its projected radius does not increase beyond a threshold. Else, the same attempt is made for the closest outlier-cluster. If both cannot absorb the observation, a new cluster is initialized. Periodically, the algorithm downgrades potential core micro clusters if their weight is too low or if the number of associated dimensions is too large. Outlier-clusters are removed as in DenStream. To generate the macro-clusters a variant of PreDeCon (Bohm et al. 2004) is used.

PreDeConStream (Hassani et al. 2012) (Subspace Preference weighted Density Connected clustering of Streaming data) was developed simultaneously to HDDStream (see Sect. 7) and both share many concepts. The algorithm is also initialized using the PreDeCon (Bohm et al. 2004) algorithm and the insertion strategy is the same as in DenStream (see Sect. 4.3). Additionally, the algorithm adjusts the clustering in regular intervals using a modified part of the PreDeCon algorithm on the micro-clusters that were changed during the online phase.

7.1 Summary

Projected stream clustering algorithms serve a niche for high dimensional data streams where it is not possible to perform prior feature selection in order to reduce the dimensionality. In general, these algorithms have added complexity associated with the selection of subspaces for each cluster. In return, they can identify clusters in very high dimensional space and can gracefully handle the *curse of dimensionality* (Beyer et al. 1999). The most influential and popular algorithm of this category has been HPStream.

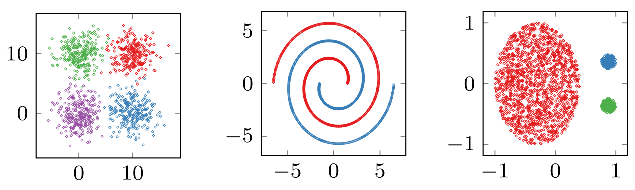
8 Application and Software

An increasing number of physical devices these days is interconnected. This trend is generally described as the Internet of Things (IoT) where every-day devices are collecting and exchanging data. Popular examples of this are Smart Refrigerators that remind you to restock or Smart Home devices such as thermostats, locks or speakers which can remote control your home. Due to this, many modern applications produce large and fast amounts of data as a continuous stream. Stream Clustering is a way to analyze this data and extract relevant information from it. The resulting clusters can help decision makers to understand the different groups. For example, IoT enables Predictive Maintenance where necessary maintenance tasks are predicted from the sensors of the devices. Clustering can help to find a cluster of devices that are likely to fail next. This can help to prevent machine failures but also reduce unnecessary maintenance tasks. Additionally, stream clustering could be applied for market or customer segmentation where customers that have similar preference or behavior are identified from a stream of transactions. These segments can be engaged differently using appropriate marketing strategies. Stream clustering has also been successfully applied to mine conversational topics from chat data (Carnein et al. 2017b) or analyze user behavior based on web click-streams (Wang et al. 2016). In addition, it was used to analyze transactional data in order to detect fraudulent plastic card transactions (Tasoulis et al. 2008) and to detect malicious network connections from computer network data (Hahsler and Bolaños 2016; Ackermann et al. 2012; Amini et al. 2016; Guha et al. 2003).

Further, it was used to analyze sensor readings (Hahsler and Bolaños 2016), social network data (Gao et al. 2015), weather monitoring (Motoyoshi et al. 2004), telecommunication data (Ali et al. 2011), stock prices (Kontaki et al. 2008) or the monitoring of automated grid computing, e.g. for anomaly detection (Zhang et al. 2010; Zhang and Wang 2010). Other application scenarios include social media analysis or the analysis of eye tracking data as in our initial example.

Unfortunately, there is not a single solutions that can fit all application scenarios and problems. For this reason, the choice of algorithm depends on the characteristics and requirements of the stream. An important characteristic is the speed of the stream. For very fast streams, more efficient algorithms are required. In particular, anytime algorithms such as `ClusTree` (Kranen et al. 2009) or `evoStream` (Carnein and Trautmann 2018) are able to output a clustering result at anytime during the stream and handle faster streams better. On the other hand, some algorithms store additional positional information alongside micro-clusters. While this often helps to achieve better clustering results, it makes algorithms such as `LeaDen-Stream` (Amini and Wah 2013), `D-Stream` with attraction (Tu and Chen 2009) and `DBSTREAM` less suitable for faster streams.

Another important characteristic is the desired or expected shape of clusters. For example, many algorithms can only recognize compact clusters, as shown in Fig. 11a. This type of clusters often corresponds to our natural understanding of a cluster and is usually well recognised by distance-based approaches such as `BICO` (Fichtenberger et al. 2013) or `ClusTree` (Kranen et al. 2009). Some streams, however, consists of mostly long and straggly clusters as shown in Fig. 11b. These clusters are generally easier to detect for grid-based approaches where clusters of arbitrary shape are formed by dense neighboring cells. Nevertheless, distance-based approaches can also detect these clusters when using a reclustering algorithm that can identify arbitrary shapes, e.g., as used by `DBSTREAM` (Hahsler and Bolaños 2016). In addition, clusters may be of different density as as shown in Fig. 11c. This is a niche problem and only `MuDi-Stream` (Amini et al. 2016) currently addresses it.



(a) `Compact` **(b)** `Connected` **(c)** `Multi-Density`

Fig. 11 Visualisation of different cluster types

Furthermore, the dimensionality of the problem plays an important role. Generally, faster algorithms are desirable as the dimensionality increases. However, for very high-dimensional data, projected approaches such as `HPStream` (Aggarwal et al. 2004) are necessary in order to find meaningful clusters.

Finally, the expected amount of concept-shift of the stream is important. If the structure of clusters changes regularly, an algorithm that applies a damped time-window model should be used. This includes algorithms such as `DenStream` (Cao et al. 2006), `D-Stream` (Chen and Tu 2007), `ClusTree` (Kranen et al. 2009) and many more. For streams without concept-shift, most algorithms are applicable. For example, algorithms using a damped time window model can set the fading factor $\lambda = 0$. Note, however, that algorithms such as `DenStream` rely on the fading mechanism in order to remove noise (Bolaños et al. 2014).

8.1 Software

An important aspect to apply stream clustering in practice is available software and tools. In general, availability of stream clustering implementations is rather scarce and only the most prominent algorithms are available. Only few authors provide reference implementations for their algorithms. As an example, `C`, `C++` or `R` implementations are available for `BIRCH` (Zhang et al. 1997), `STREAM` (Guha et al. 2003; Ackermann et al. 2012), `streamKM++` (Ackermann et al. 2012), `BICO` (Fichtenberger et al. 2013) and `evoStream` (Carnein and Trautmann 2018). Previously, also an implementation of `RepStream` (Lühr and Lazarescu 2009) was available.

More recently, several projects aim to create unified frameworks for stream data mining, including implementations for stream clustering. The most popular framework for data stream mining is the Massive Online Analysis (MOA) (Bifet et al. 2010) framework. It is implemented in Java and provides the stream clustering algorithms `Cob-Web`, `D-Stream`, `DenStream`, `ClusTree`, `Clu-Stream`, `streamKM++` and `BICO`.

For faster prototyping there also exists the `stream` package (Hahsler et al. 2018) for the statistical programming language R. It contains general methods for working with data streams and also implements the `D-Stream`, `DBSTREAM`, `BICO`, `BIRCH` and `evoStream` algorithm. There is also an extension package `streamMOA` (Hahsler et al. 2015) which interfaces the MOA implementations of `DenStream`, `ClusTree` and `CluStream`.

For working with data in high-dimensional space, the Subspace MOA framework (Hassani et al. 2013) provides Java implementations for `HDDStream` and `PreDeCon-Stream`. Again, the R-package subspaceMOA (Hassani et al. 2016) interfaces both methods to make them accessible with the `stream` package.

Alternatively, the `streamDM` (Huawei Noah's Ark Lab 2015) project provides methods for data mining with Spark Streaming which is an extension for the Spark engine. Currently it implements the `CluStream` and `streamKM++` algorithms with plans to extend the project with more stream clustering algorithms.

8.2 Algorithm Configuration

Streaming data in general (Bifet et al. 2018) pose considerable challenges for respective algorithms, especially due to the requirement of real-time capability, the high probability of non-stationary data and the lack of availability of the original data over time. Moreover, many clustering approaches in general require standardized data. In order to standardize a data stream which evolves over time, one could either estimate the values for centering and scaling from an initial portion of the stream (Hahsler et al. 2018). Alternatively, in a more sophisticated manner, CF based approaches can also incrementally adapt the values for scaling and update the existing micro-clusters accordingly (Aggarwal et al. 2004).

Specifically, as we have seen throughout the discussion of available stream clustering algorithms, most of them require a multitude of parameters to be set by the user a-priori. These settings control the behavior and performance of the algorithm over time. Usually, density-based algorithms require at least a distance or radius threshold and grid-based algorithms need the grid's size. The same applies to their extensions for projected stream clustering and model-based algorithms mostly make use of a similarity-threshold. In practice, such parameters are often unintuitive to choose appropriately even with expert knowledge. As an example, it might be possible to find appropriate distance thresholds for a given scenario but choosing appropriate weight thresholds or cleanup intervals tends to be very difficult for a users, especially considering possible drift of the stream. A notable exception from this problem is the `ClusTree` (see Sect. 4.3) algorithm which at least makes an effort to be parameter-free.

Therefore, a systematic online approach for automated parameter configuration is required. However, state-of-the art automated parameter configuration approaches such as `irace` (López-Ibáñez et al. 2016), `ParamILS` (Hutter et al. 2007, 2009) or `SMAC` (Hutter et al. 2011) are not perfectly suited for the streaming data scenario. First of all, they are mostly set-based, thus not focussed on online learning on single, specific data. Moreover, they require static and stationary data so that they can only be applied in a prequential manner, i.e. in regular intervals or on an initial sample of the stream in order to determine and adjust appropriate settings over time which does not really meet the efficiency requirement of the real-time capability.

However, an initial approach on configuring and benchmarking stream clustering approaches based on `irace` (López-Ibáñez et al. 2016) has been presented by Carnein et al. (2017a). Very promising are ensemble-based approaches, both for algorithm selection and configuration on data streams, which have successfully been applied in the context of classification algorithms already (van Rijn et al. 2014, 2018).

9 Conclusion

Analyzing data streams is becoming extremely important as most applications today create a continuous flow of new observations. An interesting aspect of analyzing streaming data is clustering, where homogeneous groups are identified. It supports decision making in large and possibly unstructured data by identifying manageable groups of similar observations. Possible application scenarios include the analysis of sensor data, click stream data, network data or identifying market segments in customer relationship management applications (Wedel and Kamakura 2000). Stream clustering aims to find clusters within an evolving data stream without the need to revisit or store all observations. It has been a very active research topic over the past decades and has produced a multitude of algorithms following different approaches. Most algorithms rely on a two-phase approach where an online component extracts relevant information from the stream. An offline component then uses this information to derive a final set of clusters. The underlying optimization task needs to build a suitable summary of the stream. The interplay between the online and offline component is then crucial for decision making since it helps to reveal hidden structures and dependencies within the data streams. In this paper we summarized and reviewed a total of 51 available algorithms. To the best of our knowledge our survey is the most extensive and thorough study of its kind, discussing almost every available stream clustering algorithm. This paper is supported by our website[2] which serves as a repository for algorithms, literature and data sets in the field of data stream clustering.

In addition, we identified categories of algorithms and research threads. First, we identify algorithms that used density-threshold and either assign new observations to the closest cluster or use it to initialize a new cluster. A milestone algorithm in this area is `BIRCH` which proposed to store summary statistics of a cluster. These can be incrementally updated and allow to calculate location and

[2] http://www.matthias-carnein.de/streamClustering.

deviation of a cluster. `CluStream` and `DenStream` have refined this concept to account for concept drift of a data stream. Next, density-based algorithms use grids to identify dense regions. The grid-cells are typically of fixed size but can also be dynamically calculated. The most important algorithm employing this strategy is `D-Stream`. A third category is based on statistical models. Many algorithms utilize the Expectation Maximization (EM) algorithm to fit a mixture of distributions to the data. Lastly, we identified subspace clustering algorithms aimed at high dimensional data streams.

A crucial challenge when applying stream clustering algorithms is the appropriate choice of parameter settings. Systematic automated algorithm configuration is required but the streaming data scenario is very challenging, even state-of-the art configuration approaches are not perfectly suited as they require an appropriate learning phase and would have to be able to deal with drifts or structural changes of the stream.

Future work should systematically benchmark and configure prominent stream clustering algorithms and determine respective strengths and weaknesses, e.g., regarding cluster structure, computational complexity and clustering quality. We have already published experimental results for the most popular algorithms (Carnein et al. 2017a). In addition, real-life use cases and application examples should be highlighted and compared to traditional approaches in the same scenario.

Acknowledgements The authors would like to thank for the support provided by Karsten Kraume and the ERCIS Omni-Channel Lab – powered by Arvato (https://omni-channel.ercis.org/).

References

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

Aggarwal CC (2007) Data streams: models and algorithms, vol 31. Springer, Berlin

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, volume 29 of VLDB '03, VLDB Endowment, Berlin, pp 81–92

Aggarwal CC, Han J, Wang J, Yu PS (2004) A framework for projected clustering of high dimensional data streams. In: Proceedings of the thirtieth international conference on very large data bases, volume 30 of VLDB '04, VLDB Endowment, Toronto, pp 852–863

Ali MH, Sundus A, Qaiser W, Ahmed Z, Halim Z (2011) Applicative implementation of D-stream clustering algorithm for the real-time data of telecom sector. In: International conference on computer networks and information technology, pp 293–297

Amini A, Wah TY (2011) Density micro-clustering algorithms on data streams: a review. In: Proceeding of the international multiconference of engineers and computer scientists (IMECS)

Amini A, Wah TY (2012) A comparative study of density-based clustering algorithms on data streams: micro-clustering approaches. Springer, US, Boston, pp 275–287

Amini A, Wah TY (2013) LeaDen-Stream: a leader density-based clustering algorithm over evolving data stream. J Comput Commun 01(05):26–31

Amini A, Wah TY, Saybani MR, Yazdi SRAS (2011) A study of density-grid based clustering algorithms on data streams. In: Eighth international conference on fuzzy systems and knowledge discovery (FSKD) 3:1652–1656

Amini A, Wah TY, Teh YW (2012) DENGRIS-Stream: a density-grid based clustering algorithm for evolving data streams over sliding window. In: Proceedings of the international conference on data mining and computer engineering, pp 206–210

Amini A, Saboohi H, Wah TY, Herawan T (2014a) A fast density-based clustering algorithm for real-time internet of things stream. Sci World J 2014:1–11

Amini A, Wah TY, Saboohi H (2014b) On density-based data streams clustering algorithms: a survey. J Comput Sci Technol 29(1):116–141

Amini A, Saboohi H, Herawan T, Wah TY (2016) MuDi-Stream: a multi density clustering algorithm for evolving data stream. J Netw Comput Appl 59:370–385

Arthur D, Vassilvitskii S (2007) K-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, SODA '07, Society for Industrial and Applied Mathematics, New Orleans, pp 1027–1035

Barbará D, Chen P (2000) Using the fractal dimension to cluster datasets. In: Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '00, ACM, Boston, pp 260–264

Barbará D, Chen P (2003) Using self-similarity to cluster large data sets. Data Min Knowl Discov 7(2):123–152

Ben-Hur A, Horn D, Siegelmann HT, Vapnik V (2001) Support vector clustering. J Mach Learn Res 2:125–137

Beyer K, Goldstein J, Ramakrishnan R, Shaft U (1999) When is "nearest neighbor" meaningful? Springer, Berlin, pp 217–235

Bhatnagar V, Kaur S (2007) Exclusive and complete clustering of streams. Springer, Berlin, pp 629–638

Bhatnagar V, Kaur S, Chakravarthy S (2014) Clustering data streams using grid-based synopsis. Knowl Inf Syst 41(1):127–152

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. J Mach Learn Res 11:1601–1604

Bifet A, Gavaldà R, Holmes G, Pfahringer B (2018) Machine learning for data streams with practical examples in MOA. MIT Press, Cambridge

Bohm C, Kailing K, Kriegel H-P, Kroger P (2004) Density connected clustering with local subspace preferences. In: Proceedings of the fourth IEEE international conference on data mining, ICDM '04, IEEE Computer Society, Washington, DC, pp 27–34

Bolaños M, Forrest J, Hahsler M (2014) Clustering large datasets using data stream clustering techniques. In: Spiliopoulou M, Schmidt-Thieme L, Janning R (eds) Data analysis, machine learning and knowledge discovery, studies in classification, data analysis, and knowledge organization. Springer, Berlin, pp 135–143

Bradley PS, Fayyad U, Reina C (1998) Scaling clustering algorithms to large databases. In: Proceedings of the 4th international conference on knowledge discovery and data mining (KDD'98). AAAI Press, pp 9–15

Cao F, Ester M, Qian W, Zhou A (2006) Density-based clustering over an evolving data stream with noise. In: Conference on data mining (SIAM '06), pp 328–339

Carnein M, Trautmann H (2018) evoStream—evolutionary stream clustering utilizing idle times. Big Data Res 14:101–111. https://doi.org/10.1016/j.bdr.2018.05.005

Carnein M, Assenmacher D, Trautmann H (2017a) An empirical comparison of stream clustering algorithms. In: Proceedings of the ACM international conference on computing frontiers (CF '17). ACM, pp 361–365

Carnein M, Assenmacher D, Trautmann H (2017b) Stream clustering of chat messages with applications to twitch streams. In Proceedings of the 36th international conference on conceptual modeling (ER'17). Springer International Publishing, pp 79–88

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, ACM, San Jose, pp 133–142

Dang XH, Lee V, Ng WK, Ciptadi A, Ong KL (2009a) An EM-based algorithm for clustering data streams in sliding windows. In: Zhou X, Yokota H, Deng K, Liu Q (eds) Proceedings of the 14th international conference on database systems for advanced applications (DASFAA 2009). Springer, Berlin, pp 230–235

Dang XH, Lee VCS, Ng WK, Ong KL (2009b) Incremental and adaptive clustering stream data over sliding window. Springer, Berlin, pp 660–674

Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd international conference on knowledge discovery and data mining. AAAI Press, pp 226–231

Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited. SIGKDD Explor Newsl 2(1):51–57

Fichtenberger H, Gillé M, Schmidt M, Schwiegelshohn C, Sohler C (2013) BICO: BIRCH meets coresets for k-means clustering. In: Algorithms - ESA 2013—Proceedings of 21st annual European symposium, Sophia Antipolis, pp 481–492. http://ls2-www.cs.tu-dortmund.de/grav/de/bico. Accessed 27 Dec 2018

Fisher DH (1987) Knowledge acquisition via incremental conceptual clustering. Mach Learn 2(2):139–172

Forestiero A, Pizzuti C, Spezzano G (2013) A single pass algorithm for clustering evolving data streams based on swarm intelligence. Data Min Knowl Discov 26(1):1–26

Gao J, Li J, Zhang Z, Tan P-N (2005) An incremental data stream clustering algorithm based on dense units detection. Springer, Berlin, pp 420–425

Gao X, Ferrara E, Qiu J (2015) Parallel clustering of high-dimensional social media data streams. arXiv:1502.00316

Ghesmoune M, Azzag H, Lebbah M (2014) G-Stream: growing neural gas over data stream. In: Loo CK, Siah YK, Wong KW, Jin AT, Huang K (eds) Proceedings of neural information processing: 21st international conference, ICONIP 2014, Kuching, Malaysia, November 3–6, 2014, Part I. Springer International Publishing, pp 207–214

Ghesmoune M, Lebbah M, Azzag H (2015) Clustering over data streams based on growing neural gas. Springer, Berlin, pp 134–145

Ghesmoune M, Lebbah M, Azzag H (2016) State-of-the-art on clustering data streams. Big Data Anal 1(1):13

Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: theory and practice. IEEE Trans Knowl Data Eng 15(3):515–528

Hahsler M, Bolaños M (2016) Clustering data streams based on shared density between micro-clusters. IEEE Trans Knowl Data Eng 28(6):1449–1461

Hahsler M, Bolanos M, Forrest J (2015) streamMOA: interface for MOA stream clustering algorithms. https://cran.r-project.org/web/packages/streamMOA/. Accessed 27 Dec 2018

Hahsler M, Bolanos M, Forrest J, Carnein M, Assenmacher D (2018) stream: infrastructure for data stream mining. https://cran.r-project.org/web/packages/stream/. Accessed 27 Dec 2018

Hassani M, Kranen P, Seidl T (2011) Precise anytime clustering of noisy sensor data with logarithmic complexity. In: Proceedings of 5th international workshop on knowledge discovery from sensor data (SensorKDD 2011) in conjunction with 17th ACM SIGKDD conference on knowledge discovery and data mining (KDD 2011), ACM, San Diego, pp 52–60

Hassani M, Spaus P, Gaber MM, Seidl T (2012) Density-based projected clustering of data streams. Springer, Berlin, pp 311–324

Hassani M, Kim Y, Seidl T (2013) Subspace MOA: subspace stream clustering evaluation using the MOA framework. Springer, Berlin, pp 446–449

Hassani M, Hansen M, Kim Y, Seidl T (2016) subspaceMOA: interface to 'subspaceMOA'. https://cran.r-project.org/web/packages/subspaceMOA/. Accessed 27 Dec 2018

Huawei Noah's Ark Lab (2015). streamDM. http://huawei-noah.github.io/streamDM/. Accessed 27 Dec 2018

Hutter F, Hoos HH, Stützle T (2007) Automatic algorithm configuration based on local search. In: Proceedings of the twenty-second conference on artifical intelligence (AAAI '07), pp 1152–1157

Hutter F, Hoos HH, Leyton-Brown K, Stützle T (2009) ParamILS: an automatic algorithm configuration framework. J Artif Intell Res 36:267–306

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: Proceedings of LION-5, pp 507–523

Isaksson C, Dunham MH, Hahsler M (2012) SOStream: self organizing density-based clustering over data stream. Springer, Berlin, pp 264–278

Ismael N, Alzaalan M, Ashour W (2014) Improved multi threshold birch clustering algorithm 2(1):1–10. https://doi.org/10.14257/ijaiasd.2014.2.1.01

Jia C, Tan C, Yong A (2008) A grid and density-based clustering algorithm for processing data stream. In: Second international conference on genetic and evolutionary computing (WGEC '08), pp 517–521

Kohonen T (1982) Self-organized formation of topologically correct feature maps. Biol Cybern 43(1):59–69

Kontaki M, Papadopoulos AN, Manolopoulos Y (2008) Continuous trend-based clustering in data streams. Springer, Berlin, pp 251–262

Kranen P, Assent I, Baldauf C, Seidl T (2009) Self-adaptive anytime stream clustering. In: 9th IEEE international conference on data mining (ICDM '09), pp 249–258

Kranen P, Assent I, Baldauf C, Seidl T (2011a) The ClusTree: indexing micro-clusters for anytime stream mining. In: Knowledge and information systems journal (Springer KAIS), Vol 29, Issue 2. Springer, London, pp 249–272

Kranen P, Reidl F, Villaamil FS, Seidl T (2011b) Hierarchical clustering for real-time stream data with noise. Springer, Berlin, pp 405–413

Lin J, Lin H (2009) A density-based clustering over evolving heterogeneous data stream. In: 2009 ISECS international colloquium on computing, communication, control, and management, vol 4, pp 275–277

Liu LX, Huang H, Guo YF, Chen FC (2009) rDenStream, a clustering algorithm over an evolving data stream. In: 2009 international conference on information engineering and computer science, pp 1–4

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Stützle T, Birattari M (2016) The irace package: iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58

Lorbeer B, Kosareva A, Deva B, Softić D, Ruppel P, Küpper A (2017) A-BIRCH: automatic threshold estimation for the BIRCH clustering algorithm. Springer, Berlin, pp 169–178

Lühr S, Lazarescu M (2008) Connectivity based stream clustering using localised density exemplars. Springer, Berlin, pp 662–672

Lühr S, Lazarescu M (2009) Incremental clustering of dynamic data streams using connectivity based representative points. Data Knowl Eng 68(1):1–27

Ma WH (2014) Survey on data streams clustering techniques. In: Manufacture engineering, quality and production system III, volume 933 of Advanced Materials Research. Trans Tech Publications, pp 768–773

Martinetz T, Schulten K et al (1991) A "neural-gas" network learns topologies. University of Illinois at Urbana-Champaign

Meesuksabai W, Kangkachit T, Waiyamai K (2011) Hue-Stream: evolution-based clustering technique for heterogeneous data streams with uncertainty. In: Tang J, King I, Chen L, Wang J (eds) ADMA, volume 7121 of Lecture Notes in Computer Science. Springer, pp 27–40

Motoyoshi M, Miura T, Shioya I (2004) Clustering stream data by regression analysis. In: Proceedings of the second workshop on Australasian information security, data mining and web intelligence, and software internationalisation, volume 32 of ACSW Frontiers '04, Australian Computer Society, Darlinghurst, pp 115–120

Mousavi M, Bakar AA, Vakilian M (2015) Data stream clustering algorithms: a review. Int J Adv Soft Comput Appl 7:1–15

Nguyen H-L, Woon Y-K, Ng W-K (2015) A survey on data stream clustering and classification. Knowl Inf Syst 45(3):535–569

Ntoutsi I, Zimek A, Palpanas T, Kröger P, Kriegel H-P (2012) Density-based projected clustering over high dimensional data streams. In: Proceedings of the 2012 SIAM international conference on data mining, pp 987–998

O'Callaghan L, Mishra N, Meyerson A, Guha S, Motwani R (2002) Streaming-data algorithms for high-quality clustering. In: Proceedings of the 18th international conference on data engineering (ICDE), pp 685–694

Park NH, Lee WS (2004) Statistical Grid-based clustering over data streams. SIGMOD Rec 33(1):32–37

Park NH, Lee WS (2007a) Cell trees: an adaptive synopsis structure for clustering multi-dimensional on-line data streams. Data Knowl Eng 63(2):528–549

Park NH, Lee WS (2007b) Grid-based subspace clustering over data streams. In: Proceedings of the sixteenth ACM conference on conference on information and knowledge management, ACM, New York, pp 801–810

Ren J, Ma R (2009) Density-based data streams clustering over sliding windows. In: 2009 Sixth international conference on fuzzy systems and knowledge discovery, volume 5, pp 248–252

Ren J, Cai B, Hu C (2011) Clustering over data streams based on grid density and index tree. J Converg Inf Technol 6(1):83–93

Ruiz C, Spiliopoulou M, Menasalvas E (2007) C-DBSCAN: density-based clustering with constraints. Springer, Berlin, pp 216–223

Ruiz C, Menasalvas E, Spiliopoulou M (2009) C-DenStream: using domain knowledge on a data stream. Springer, Berlin, pp 287–301

Silva JA, Faria ER, Barros RC, Hruschka ER, de Carvalho AC, Gama J (2013) Data stream clustering: a survey. ACM Comput Surv 46(1):13:1–13:31

Spinosa EJ, de Leon F de Carvalho AP, Gama J (2007) Olindda: a cluster-based approach for detecting novelty and concept drift in data streams. In: Proceedings of the 2007 ACM symposium on applied computing. ACM, pp 448–452

Steil J, Huang MX, Bulling A (2018) Fixation detection for head-mounted eye tracking based on visual similarity of gaze targets. In: Proceedings of international symposium on eye tracking research and applications (ETRA), pp 23:1–23:9

Tasoulis DK, Adams NM, Hand DJ (2006) Unsupervised clustering in streaming data. In: Sixth IEEE international conference on data mining–workshops (ICDMW'06), pp 638–642

Tasoulis D, Adams N, Weston DJ, Hand DJ (2008) Mining information from plastic card transaction streams. In: Proceedings in computational statistics: 18th symposium (COMPSTAT 2008), volume 2, pp 315–322

Theiler J (1990) Estimating fractal dimension. J Opt Soc Am A 7(6):1055–1073

Tibshirani R, Walther G, Hastie T (2001) Estimating the number of clusters in a data set via the gap statistic. J R Stat Soc Ser B (Stat Methodol) 63(2):411–423

Tu L, Chen Y (2009) Stream data clustering based on grid density and attraction. ACM Trans Knowl Discov Data 3(3):12:1–12:27

Udommanetanakit K, Rakthanmanon T, Waiyamai K (2007) E-Stream: evolution-based technique for stream clustering. Springer, Berlin, pp 605–615

van Rijn JN, Holmes G, Pfahringer B, Vanschoren J (2014) Algorithm selection on data streams. In: Džeroski S, Panov P, Kocev D, Todorovski L (eds) Proceedings of the 17th international conference on discovery science (DS), volume 8777 of lecture notes in computer science (LNCS). Springer, pp 325–336

van Rijn J, Nicolaas GH, Pfahringer B, Vanschoren J (2018) The online performance estimation framework: heterogeneous ensemble learning for data streams. Mach Learn 107(1):149–176

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):14:1–14:28

Wang CD, Lai JH, Huang D, Zheng WS (2013) SVStream: a support vector-based algorithm for clustering data streams. IEEE Trans Knowl Data Eng 25(6):1410–1424

Wang G, Zhang X, Tang S, Zheng H, Zhao BY (2016) Unsupervised clickstream clustering for user behavior analysis. In: Proceedings of the 2016 CHI conference on human factors in computing systems, ACM, New York, pp 225–236

Wedel M, Kamakura WA (2000) Market segmentation, 2nd edn. Springer, US

Yang C, Zhou J (2006) HClustream: a novel approach for clustering evolving heterogeneous data stream. In: Sixth IEEE international conference on data mining—workshops (ICDMW'06), pp 682–688

Yang Y, Liu Z, Zhang Jp, Yang J (2012) Dynamic density-based clustering algorithm over uncertain data streams. In: 2012 9th international conference on fuzzy systems and knowledge discovery, pp 2664–2670

Zhang X, Wang W (2010) Self-adaptive change detection in streaming data with non-stationary distribution. Springer, Berlin, pp 334–345

Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the 1996 ACM SIGMOD international conference on management of data, ACM, Montreal, pp 103–114

Zhang T, Ramakrishnan R, Livny M (1997) BIRCH: a new data clustering algorithm and its applications. Data Mini Knowl Discov 1(2):141–182

Zhang X, Germain C, Sebag M (2010) Adaptively detecting changes in autonomic grid computing. In: 2010 11th IEEE/ACM international conference on grid computing, pp 387–392

Zhou A, Cao F, Qian W, Jin C (2007a) Tracking clusters in evolving data streams over sliding windows. Knowl Inf Syst 15(2):181–214

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

Zhu Y, Shasha D (2002) StatStream: statistical monitoring of thousands of data streams in real time. In: Proceedings of the 28th international conference on very large data bases, VLDB Endowment, Hong Kong, pp 358–369