Data Stream Clustering Methods: A Review

Nathalie A Barbosa, Louise Travé-Massuyès, Victor Hugo Grisales

Nathalie Barbosa

Data Stream Clustering Methods: A Review

1 Introduction

This review presents several approaches to stream-data clustering . Among the explored techniques, two phase (online-offline) clustering is a common approach. Some approaches base their online component in the micro-cluster generation concept. These micro-structures are characterized by a feature vector that summarizes the information of all individuals contained therein. The use of this vector makes the algorithms suitable for online applications since the storage of each individual becomes unnecessary. Micro-clusters may be arranged in an ordered manner (e.g. grids or trees) with the purpose of making the belonging cluster location easier. The offline component of the algorithms takes the micro-clusters features as source data for final clustering. K-means based techniques are a common choice in the offline component, since the one-pass only condition (for high stream rates) is not necessary in this stage (advantage given by the offline nature of the component). Density-based clustering algorithms are also used in the offline component, showing an improvement in their capacity to handle any shape clusters including non-convex sets.

The techniques discussed in this report are introduced by the chronological order of their proposal and reported in pink color on the figure 1.

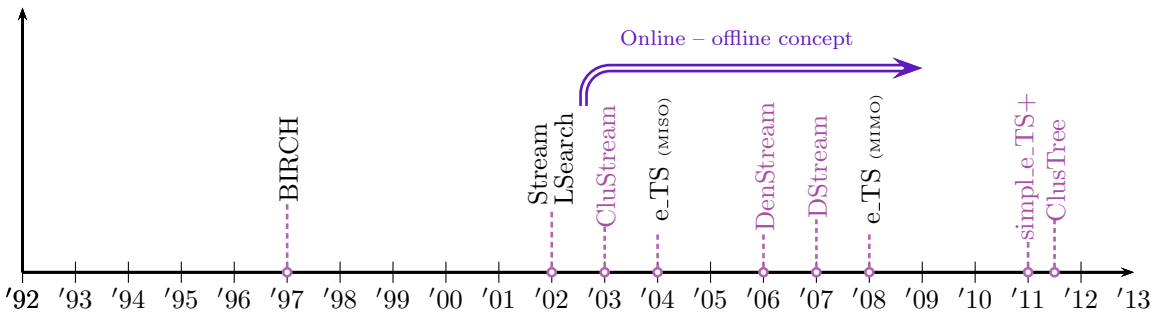


Figure 1: Data stream clustering chronology

This report has the following structure. First the distance based methods are displayed in chronological order. Second some density based approaches are presented. Third some fuzzy evolving approaches are pointed out. Finally, the methods are discussed and compared.

2 CluStream [1]

The framework of the so called Clustream approach is separated in two components: online and offline clustering. In the first component, data are collected, pre-processed and compressed in micro-clusters. In the second component, the micro-clusters are grouped into macro-clusters using a modification of the k-means algorithm.

The micro-cluster phase is used to maintain statistical information about the entire data stream without explicit data storage. Summary statistics suggested in BIRCH [10] and stored in a cluster feature vector, are extended to

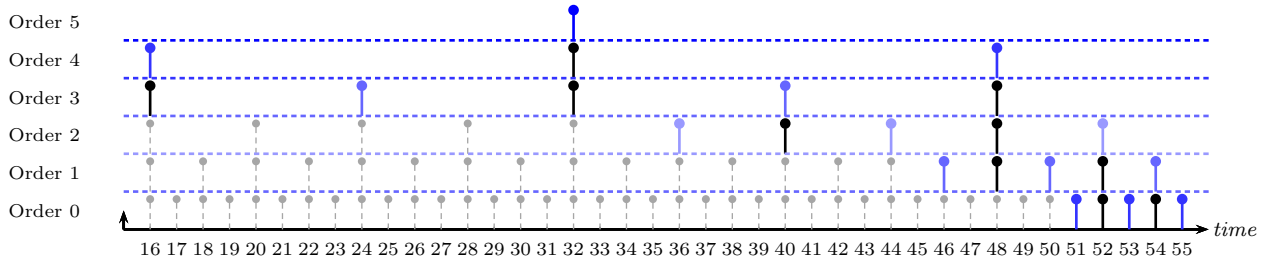


Figure 2: Time Snapshots taken and store in a pyramidal scheme. Adapted from [1]

provide sufficient spatial and temporal information for the offline clustering phase. The BIRCH statistics are: the number of data points inside the micro-cluster n, the linear sum of the data values LS and the sum of the squares of the data values SS. CluStream extends the statistic spatial concept of feature vector to the temporal information including the same statistics for the stream data arrival time-stamps. Consequently, a micro-cluster is defined as the vector that summarizes a set of d-dimensional points $X_{i1} \dots X_{in}$ with time-stamps $T_{i1} \dots T_{in}$ in a $(2 \cdot d + 3)$ tuple $\left(\overline{CF1_x}, \overline{CF2_x}, CF1_t, CF2_t, n\right)$, where $\overline{CF1}$ is the LS of the values in X (d-dimensional) or T (1-dimension) and $\overline{CF2}$ is the SS of those values. Each micro-cluster has an unique id associated with it.

The micro-clusters are stored in memory at particular moments in the stream referred as "'snapshots"'. This snapshots are used in the offline stage to generate the macro-clusters for different, specific time horizons. The Snapshots are stored at different levels of granularity (in a pyramidal scheme) and then classified into different orders varying from 1 to log(T) maximum; being T the time elapsed since the beginning of the stream or the application of the technique. Snapshots of order i are taken at the moment when $mod(t, \alpha^i) = 0$, $\alpha \geq 1$ being α the base for the sampling time. The authors posed that a limited amount of snapshots are sufficient for macro clustering and therefore only the last $\alpha^l + 1$ snapshots are maintained (for each order). The figure 2 exemplifies the framework of pyramidal time framing for $T = 55$, $\alpha = 2$ and $l = 2$. In this case, the storage requirement of the technique corresponds to $(\alpha^l + 1) = 5$ snapshots for each order, with a maximum of 5 orders ($log_\alpha(T) = 5.78$) being ≈ 29 snapshots to store in the worst case scenario. In the actual case taking redundancy into account, only 14 snapshots are necessary to analyse the time horizons with five different levels of granularity.

3 ClusTree [9]

Based also on micro clusters as CluStream, the ClusTree proposes a characteristic vector in the form $CF = (n, LS, SS)$ but in the hierarchical index structure of the R-Tree [7] as illustrated in figure 3. Such structure allows one to maintain stream summaries up to date and provides efficient cluster location for object insertion at different levels of granularity. With the hierarchical structure the algorithm could drop the object in a higher level of the tree (using a buffer space) if no further time is available and then go back after to finally place it in the corresponding leaf node. This rate depending strategy, provides a framework for "anytime clustering". The tree micro-cluster structure is shown in figure 3. The general idea is the following: at the time of arrival of a new data point and given enough time the algorithm descends, searching for the closest micro-cluster (according to euclidean distance to its mean), to reach the leaf level. If a new point comes before the insertion process is finished, the algorithm interrupts the insertion process and stores the object temporarily in a local aggregate (buffer) from which it would be taken along in the future as a "'hitch-hiker"' when a new data arrives looking to be located in a micro-cluster on the same path.

The tree nodes that are no leafs are called inner nodes. An inner node can contain between m and M entries and stores the CF of the objects it summarizes, a CF of the objects in its local buffer and a pointer to its child nodes. A leaf nodes stores only the CF of the objects it represents. The tree is always balanced, that is, all the paths from the root to a leaf have the same length. Since the tree is updated on the fly, splitting nodes is usual. When a leaf node is reached and the insertion of the object is not possible because the leaf micro-cluster is full, the algorithm splits the node based on pairwise distances between the clusters. In order to maintain an up-to-date view, clusters are weighted with an exponential time-dependant decay function ω, $\omega(\Delta t) = \beta^{-\lambda \Delta t}$. The Δt is calculated with the current time

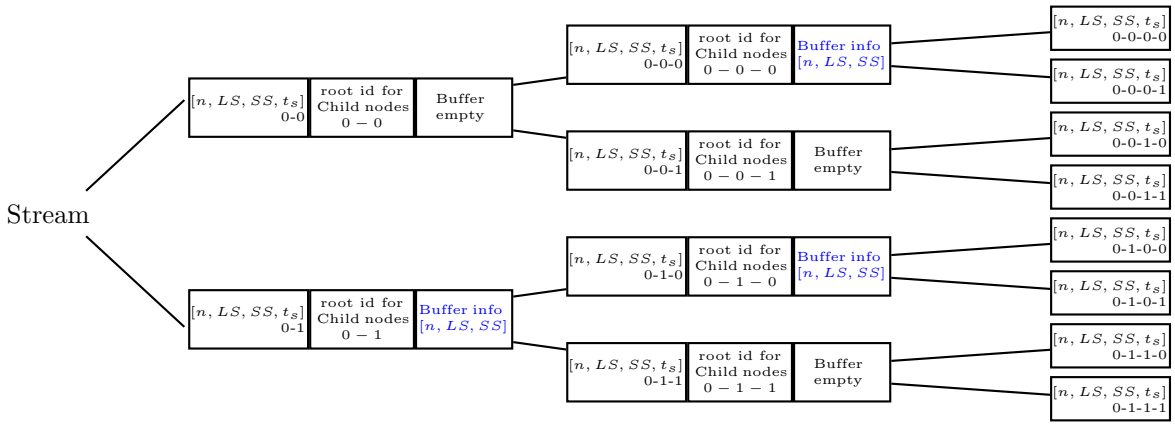


Figure 3: ClusTree micro-cluster hierarchical structure

and the cluster time-stamp ts_i. To safe time each cluster is updated only when a new object passes through it or is assigned to it. The CF components are updated as shown in equations (1) to (3)

$$n^{(t)} = \sum_{i=1}^{n} \omega\left(t - ts_i\right) \tag{1}$$

$$LS^{(t)} = \sum_{i=1}^{n} \omega\left(t - ts_i\right) x_i \tag{2}$$

$$SS^{(t)} = \sum_{i=1}^{n} \omega\left(t - ts_i\right) x_i^2 \tag{3}$$

In general, if no object is added to a cluster's CF during the time interval $(t, t + \Delta t)$, its CF in $t + \Delta t$ can be calculated from $CF^{(t)}$ using the decay function as follows:

$$CF^{(t+\Delta t)} = \omega(\Delta t) CF^{(t)} \tag{4}$$

The elements stored in the buffer should keep their time-stamp until being assigned to a cluster. Upon descending into a node, the CF of the clusters that were followed through the path are updated by position wise multiplication with the decay function and the time-stamps are resetted so that,

$$e_s.CF \leftarrow \omega\left(t_x - e_s.ts\right) e_s.CF \tag{5}$$

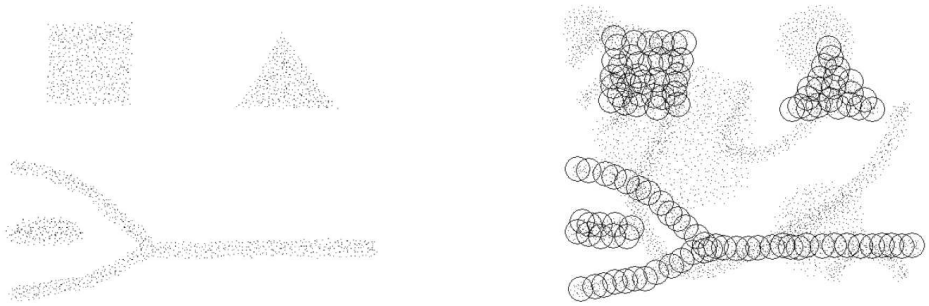
$$e_s.Buffer \leftarrow \omega\left(t_x - e_s.ts\right) e_s.Buffer \tag{6}$$

$$es.ts \leftarrow t_x \tag{7}$$

Inner nodes time-stamps can be calculated from their children ts and the buffer's ts, as:

$$es.CF^{(t+\Delta t)} = \left(\omega\left(\Delta t\right) \sum_{i=1}^{v_s} es_{oi}.CF^{(t)}\right) + es.Buffer^{(t+\Delta t)} \tag{8}$$

This time weighing avoids splitting due to cluster size limitation (as much as possible). If a node is about to split due a new assignation, the algorithm checks whether the last significant cluster associated to it can be deleted, because it is no longer significant. When a cluster has to be deleted, its statistics are subtracted from the corresponding path up to the root.



(a) Original data distribution (b) Core-micro-clusters detection (outliers rejected)

Figure 4: DenStream algorithm results with non-convex clusters

If the algorithm faces exceptionally fast streams, interruption of the insertion process and data temporal storage on local buffers are no longer a feasible solution. The authors propose to speed up through aggregation before the insertion. Their solution is to create several aggregates for dissimilar objects in order to group similar objects together. Ideally objects in the same aggregate should correspond to the same leaf node then only objects within a maximum radius (given by the maximum variance in the leafs) are aggregated. The authors used a deep first approach to reach quickly the leaf nodes, and, when there is still time available (no other data has arrived), alternatives to the first selection are evaluated leading to an optimum path. At the moment of new data arrival, the process of iterative searching is interrupted and the data is assigned in the current optimum.

4 DenStream [3]

This algorithm considers the problem of clustering a data stream over a damped window. Considering also the concept of micro cluster the authors make an extension to consider potential clusters and also clusters of outliers. To summarize clusters with arbitrary shape the concept of core (dense) micro-cluster (CMC) is introduced. The outliers are considered to be grouped also in low density micro-clusters (OMC). For those clusters with densities below the normal but over the outlier limit, potential micro-clusters (PMC) are created and their evolution is monitored to decide whether it is part of a new dense cluster creation or just an increment in the quantity of outliers.

Each CMC is defined at time t as the tuple $CMC = \{w, c, r\}$ that summarizes a group of close points $p_{i1}, \ldots, p_{in}$ with time-stamps $T_{i1}, \ldots, T_{in}$, where w, c and r are the weight of the cluster, its center and radius, respectively.

$$w = \sum_{j=1}^{n} f\left(t - T_{ij}\right) \tag{9}$$

$$c = \frac{\sum_{j=1}^{n} f\left(t - T_{ij}\right) p_{ij}}{w} \tag{10}$$

$$r = \frac{\sum_{j=1}^{n} f\left(t - T_{ij}\right) dist\left(p_{ij}, c\right)}{w} \tag{11}$$

dist is the euclidean distance and $f(t) = 2^{-\lambda t}$ is the fading function representing the exponential decay in the damped window. To be a CMC the cluster must satisfy the following conditions: $w \geq \mu$ and $r \leq \epsilon$, proving to be dense. Both μ and ϵ user defined. A variant of the BIRCH algorithm is applied to find the final clusters, using a set of density-connected non-redundant CMC as virtual points located in the CMC center c with weight equal to w. An illustration of the use of DenStream in non-convex clusters is shown in figure 4.

The potential-micro-cluster is defined as $PMC = \left\{\overline{CF}^1, \overline{CF}^2, w\right\}$, where w is defined as before but satisfying $w \geq \beta\mu$, being $0 < \beta \leq 1$ the parameter that establishes the minimum weight for non dense clusters. $\overline{CF}^1$ is the weighted sum of points $\overline{CF}^1 = \sum_{j=1}^{n} f(t - T_{ij})\, p_{ij}$ and $\overline{CF}^2$ is the weighted squared sum of the points, $\overline{CF}^2 = \sum_{j=1}^{n} f(t - T_{ij})\, p_{ij}^2$. The radius and center of the cluster can be easily calculated from those statistics.

The outlier-micro-cluster is defined as $OMC = \left\{\overline{CF}^1, \overline{CF}^2, w, t_o\right\}$ with $\overline{CF}^1$, $\overline{CF}^2$ and w being the same as before. The weight restriction in this case is $w < \beta\mu$ and $t_o = T_{i1}$ is the cluster creation time-stamp, which is used to determine the life span of the OMC. Outlier micro-clusters are maintained in a separate memory space.

Operation: when a new point p arrives the algorithm verifies which is the closest C or P micro-cluster. If the closest is a CMC, maximum radius condition is verified. If the new radius is $\leq \epsilon$ the point is absorbed by the cluster, otherwise the algorithm tries to merge the point with the closest OMC. If there the radius condition stands the point is merged and after that, the weight of the OMC is verified to decide if the cluster has grown into a PMC. When an OMC becomes denser it is removed from the outlier buffer and with its statistics a new PMC is created. In the case when no cluster is a radius match to the point a new OMC is created with the point as center and its time of arrival is set as OMC time-stamp. PMC and OMC are maintained incrementally. When a point p is absorbed by the cluster, its new statistics are:

$$(O \backslash P)MC = \left\{\overline{CF}^1 + p, \overline{CF}^2 + p^2, w + 1\right\} \tag{12}$$

For each existing micro cluster, if no point is absorbed in a time interval δt, its statistics become:

$$(O \backslash P)MC = \left\{2^{-\lambda\delta t}\overline{CF}^1, 2^{-\lambda\delta t}\overline{CF}^2, 2^{-\lambda\delta t} w\right\} \tag{13}$$

since its weight decays gradually, if it lies below $\beta\mu$, it means that the PMC becomes an OMC, so it should be deleted from the main memory. The clusters weight is checked periodically each T_P time periods, with $T_P = \frac{1}{\lambda} log\left(\frac{\beta\mu}{\beta\mu - 1}\right)$. The age of the OMC is checked with the same periodicity and clusters with weight lower than a limit $\xi(\lambda, T_P, t_o, t_c)$ are deleted from the outlier buffer.

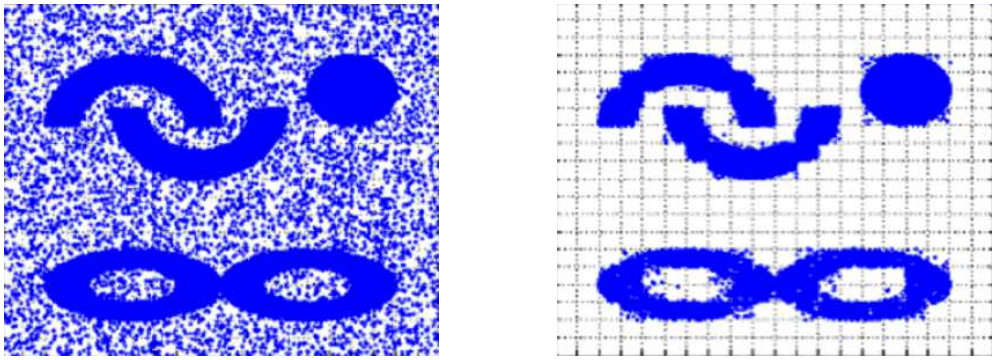
The algorithm is initialized offline applying DBSCAN [6] over a set of points $\{P\}$. Then for each point p in $\{P\}$, if the total number of points in the area $r < \epsilon$ is above $\beta\mu$ then a PMC is created using p and those points and formerly the point are deleted from P, the process continues until P is empty. Once initialized the algorithm updates the micro clusters incrementally.

5 D-Stream [4]

This algorithm based on density analysis, maps the data into a grid in an online procedure. Offline the grid density is calculated and clusters are found based on that density. The offline phase is evaluated every "'gap"' time steps. After the first gap, the algorithm generates the initial cluster and then periodically, it removes low density grids and updates the clusters.

Online, each data x is mapped into a density grid $g(x)$ in the multi-dimensional space. A density coefficient is assigned to the data, which decreases as it ages, following $D(x,t) = \lambda^{t - Ts_x}$, where Ts_x is the input time-stamp and where $\lambda \in (0,1)$ is a constant called the decay factor. For a grid, its density $D(g,t)$ is defined as the sum of the density coefficients of all data records that mapped to it until the given time t; $D(g,t) = \sum_{x \longrightarrow g} D(x,t)$. The density coefficient of each grid is updated only when new data is mapped to the grid, and the time of this insertion is stored as time-stamp for future updates. The update is given by $D(g,tn) = \lambda^{tn - ts} D(g,ts) + 1$, where tn is the new time of arrival and ts is the time stamp of the last assignation. It should be noted that only those grids in the grid list are considered for calculations and updating procedures.

Grids are described by a characteristic vector that permits incremental update. The tuple $\{t_s, t_m, D, label, status\}$ is formed by: t_s, the time-stamp of the last update in g; t_m, the time-stamp from the last removal of g from the grid list as a sporadic grid (if ever). D is the grid density at the last update, *label* is the class label of the grid, and $status = \{SPORADIC, NORMAL\}$ is the grid status in the last update (used to remove grids).



(a) Original distribution (b) Final clustering results

Figure 5: Randomly generated 4 clusters 30K 2-dimensional data set, including 5K outlier data

A grid g is said to be dense at a time y if $D(g,t) \geq \frac{C_m}{N(1-\lambda)}$ where $C_m > 1$ controls the threshold. At time t, a grid g is said to be sparse if $D(g,t) \leq \frac{C_l}{N(1-\lambda)}$, where $0 < C_l < 1$ is used to handle the maximum sparse density. And finally a grid is said to be transitional if its density is between the C_l and C_m thresholds, $\frac{C_l}{N(1-\lambda)} \leq D(g,t) \leq \frac{C_m}{N(1-\lambda)}$.

To find the final clusters the dense character of a cluster and its neighbours is analysed. A grid group id said to be a cluster if every inside grid of the group is a dense grid and every outside grid is either a dense grid or a transitional. Two grids $g_1 = (j_1^1, j_2^1, \cdots, j_d^1)$ et $g_2 = (j_1^2, j_2^2, \cdots, j_d^2)$ are considered as neighbours if there exists k, $1 \leq k \leq d$ such that: $j_i^1 = j_i^2 \ \forall i \neq k$ and $\left|j_k^1 - j_k^2\right| = 1$. This definition allows to quickly find the grid's neighbours but brings with it a limitation: only those grids with $d-1$ equal dimensions can be detected as neighbours (i.e. diagonal grids will not be considered as neighbours in a $2-d$ space). An example of non-convex clustering using DStream is shown in figure 5.

6 simpl_e_Clustering [2]

One approach used for handling large amounts of incoming data is to consider only the data arriving in the current time (no time-window), avoiding to deal with temporal relations or sequential elements. This is the case of evolving rule base clustering. From its beginning as evolving Takagi-Sugeno eTS in 2004, the fuzzy system with evolving structure has changed greatly to become a fast algorithm capable to deal with real time data streams. In such approach an evolving fuzzy rule based algorithm that allows a flexible and evolving system structure is created. One of the main advantages of this method is that it does not need the prespecification of any parameter, not even the number of inputs. The simpl_eTS+ can be interpreted as an evolving set of N linguistic fuzzy rules R^i of the form:

$$R^i : IF\left(x_1 \ is \ x_1^{i*}\right) \ AND \ldots \ AND\left(x_n \ is \ x_n^{i*}\right) \ THEN \ \left(LocalModel \ \left(y^i\right)\right) \tag{14}$$

where $i = [1, N]$, $x = [x_1, x_2, \ldots, x_n]^T$ is the input vector, n the number of fuzzy sets of the ith fuzzy rule (could evolve), x^{i*} is the focal point of the ith rule antecedent, $LM(y_i)$ is the Local Model of the ith fuzzy rule which can be a first order Takagi-Sugeno system (linear output) or into a zero-Order Takagi-Sugeno system (singleton output) and is expressed by the (generally, m-dimensional) output variables $y_i = [y_1^i, y_2^i, \ldots, y_m^i]$. The global output y is given by a weighted sum of local outputs $y = \sum_{i=1}^{N} \lambda^i y^i$ being $\lambda^i = \tau^i / \sum_{j=1}^{N} \tau^j$ the normalized activation level of the rule i and τ^i its firing level. The algorithm includes a gradual evolution in terms of local subsystems as well as in terms of input variables. This gradual evolution which involves drift and/or shift of the data distribution is based in the global density increment. Abrupt changes in the distribution (shift) are reflected in new cluster formation or old cluster elimination. There is a gradual simplification of the rule data base based on a measure of the utility of the rules, in terms of the accumulated time of appearance of the samples that form the cluster which supports that fuzzy rule.

The structure of the simpl_eTS+ is updated according to the following conditions: condition A) a data sample that covers a new area of the data space is represented by the density increment relative to the global mean (equation (15)); condition B) overlap and information redundancy must be avoided.

$$IF \quad (\delta(k) = N) \ THEN \ (x(k) \rightarrow \ new \ center) \tag{15}$$

$$\delta(k) = \left| \sum_{i=1}^{N} sign \sum_{j=1}^{n+m} \gamma_j^{i*}(k) \right| \tag{16}$$

$$\gamma_j^{i*} = \left(z_j^{i*}(k)\right)^2 - z_j^2(k) + 2\left(z_j(k) - z_j^{i*}(k)\right)\bar{z}_j^{i*}(k) \tag{17}$$

where $\delta(k)$ (equation (16)) denotes the total density increment, γ denotes the partial (per cluster and per input) density increment (equation (17)) and $z = [x^T \ y^T]$. It is important to notice that online analysis (Condition A) only requires to know: the current point z_k, the previous focal points $z^{i*}(k)$ and the global mean $\bar{z}$.

It is assumed that if a new data does not bring a density increment to all the existing clusters/rules then it can be interpolated by the existing rules and no structural change in the classifier is necessary. On the other side if the density increment is global $\delta(k) = \pm N$ a new rule should be created taking the point as prototype. The point must not have a high membership degree to any of the existing rules. If the global increment is positive, the new rule represents a focal point with higher density than any previous focal points. If it is negative a new rule is defined by focal points that cover new areas of the data space that cannot be interpolated well using the existing focal points.

Operation: This algorithm begins from scratch (if there is no a priori knowledge) or from an initial rule-base. When x_k arrives, data are standardized online using recursively updated mean and standard deviation, as seen in equation (18).

$$\begin{aligned} x_{st} &= \tfrac{x-\bar{x}}{\sigma_x} \\ \bar{x}(k) &= \tfrac{k-1}{k}\bar{x}(k-1) + \tfrac{1}{k}x(k); \qquad \bar{x}(1) = x(1) \\ \sigma_x^2(k) &= \tfrac{k-1}{k}\sigma_x^2(k-1) + \tfrac{1}{k-1}\left(x(k) - \bar{x}(k)\right)^2 \sigma_x^2(1) \end{aligned} \tag{18}$$

With the x_{st} the density increment $\delta(k)$ is calculated with respect to all the previous existing rules using (16). Then the mean value of all data points, $\bar{x}(k)$ is updated using (18). If condition A (equation (15)) holds a new center is added based on current data point $x^{(N+1)*} \leftarrow x(k)$ and the radius of all clusters are updated per input j using:

$$r_j^i(k)^2 = \beta\left(r_j^i(k-1)\right)^2 + (1-\beta)\left(\sigma_j^i(k)\right)^2; \quad r_j^i(1) = 0.5 \tag{19}$$

where β is the learning step and $\sigma_j^i(k)^2$ is related to the cluster scatter. σ is calculated as:

$$\sigma_j^i(k)^2 = \tfrac{1}{S_k^i}\sum_{l=1}^{S^i(k)} \left\|z^{i*} - z_l\right\|^2 \quad \sigma_j^i(1) = 1; \tag{20}$$

$$IF \left(l = arg\ min_{i=1}^{N} \left\|z(k) - z^{i*}\right\|\right) \ THEN \ \left(S^l(k+1) = S^l(k) + 1\right) \tag{21}$$

being S^i the support of the ith cluster. Then Condition B is evaluated and if the overlapping of the new rule with at least one of the old rules is too high then the previous focal point(s) is(are) removed and replaced with the new data point. The relevance of a cluster can change in time due to the incremental approach. To keep the rule base up to date non relevant rules must be removed from the rule base. The authors propose as a measure of relevance the cumulated weight of the rule contributions to the overall output during the life of the rule, the measure can be seen in equation (22).

$$U^l(k) = \frac{1}{k - t^l(k)} \sum_{l=1}^{t^l(k)} \lambda^l \tag{22}$$

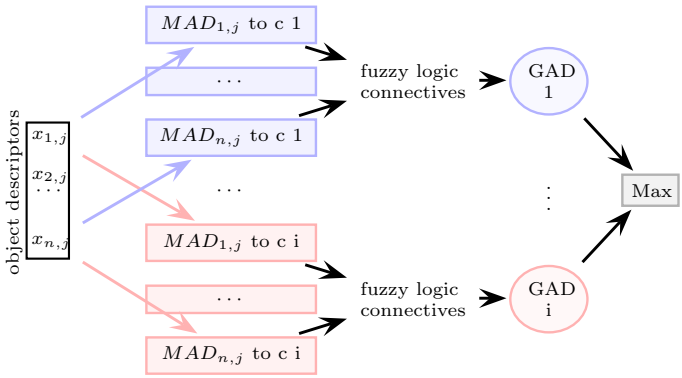


Figure 6: General structure of the LAMDA classification method. Edited from [8]

In simpl_eClustering, only cluster centers and the global mean value are kept in the memory that are $N+1$ values of dimension $(n+m)$. It is worth noting that this algorithm does not use the mean of all data or mean of groups of data to create new clusters, conversely it takes an actual point as prototype for the class, avoiding reaching a non-existing and possibly infeasible point.

7 P3S-LAMDA

P3S is a software for sensor selection and qualitative situation assessment developed inside the DISCO group in the LAAS-CNRS [5]. Made over LAMDA this tool can combine qualitative, quantitative and/or interval information of the process variables and the expert's knowledge to provide guide over sensor selection and functional state identification via clustering. LAMDA algorithm creates and drift classes in an automatic fashion, and assigns one individual to one existing class (or creates a new one) by analysing the contribution of each one of the object descriptors to the characteristics of the classes. This contribution is called Marginal Adequacy Degree (MAD). Once all the MAD's are calculated, the Global Adequacy Degree (GAD) of the element to a given class is calculated using a convex interpolation of fuzzy logic connectives. This procedure is done for each class. Finally, the object is assigned to the class with the maximum GAD. LAMDA allows the classes descriptors to update taking into account the previous data of the class and the values given by the new element. The major difference between the LAMDA methodology and the classical clustering and classification approaches is that LAMDA models the total homogeneity inside the feature space from which the information is extracted [8]. This is done through the Non-Informative Class (NIC) that accepts all items with the same adequacy (GAD). If NIC is chosen as the class with maximum GAD, it is concluded that the object corresponds to a new class that should be created (in the learning stage). A diagram of the LAMDA classification methodology for an object x_j with n descriptors is shown in figure 6.

8 Discussion

In the previous sections several different methods are presented. In this section the strengths and weaknesses of each method are pointed out in order to extract ideas for future works. Table 1 gives an overview of those characteristics.

The online-offline framework is a common approach but some works put their efforts in one of this two stages. This two phase scheme, proposed in CluStream, is very useful when it comes to handling data streams. Another powerful tool introduced in that work, is the pyramidal time framework. On the upside, that tool allows managing arbitrary time horizons with low storage consumption. On the downside, it asks the user to input the target time duration of clustering, which can be problematic if the user has no deep knowledge of the process. The main problem with CluStream is the predefined constant number of micro clusters, that could lead the algorithm to create several clusters

Method	Strength	Weakness
CluStream	• Snapshots (time evolution analysis)	• Can't detect non-convex clusters • ⇑number of user defined parameters • low density MC are considered as outliers • pre-fixed # of MC
ClusTree	• Tree Structure ids save time • buffer and hitch-hiker • exponential decay of clusters (aging)	• No tree construction only updating • MC split and merge due memory • no final clustering proposal • Final MC may not be the optimal
DenStream	•Damped window (aging) •Dense micro-cluster allow non-convex clusters • Handle outliers in specific OMC	• ⇑number of user defined parameters
DStream	• Grid list makes faster grid location and reduces calculations • Sporadic grids to handle outliers	• Grid mapping is not trivial • Need of context • no grid granularity analysis
Simpl_e_clus	• No fix number of inputs • Global density as a measure of structural change	• Utility vs. aging
P3S	• Handle quantitative, qualitative and interval information • NIC improves classification purity	• No online evolution of the structure • wasted information of points assigned to NIC? • Need of context

Table 1: Strengths and weaknesses of streaming clustering

for the outliers, which may leave less space for true clusters. Furthermore, since each cluster can absorb only a limited amount of data, similar data can be located in different micro-clusters and possibly even in different final clusters.

ClusTree improves the limitation of CluStream about the fix number of MC, but establishes a maximum number of MC and when the algorithm reaches this value, has the same limitations as CluStream. ClusTree also keeps the limitation in the number of elements described by a MC. The authors works only in the online micro cluster update and tree modification; initial tree construction is not considered, neither is the final clustering (offline-phase). On the other side, the proposal of a hierarchical id structure saves a lot of time reducing the amount of calculation necessary for MC location. Nevertheless the deep-first descent approach can lead to non optimal classification, since data could be located within a micro cluster that might not be the closest to it.

Between the algorithms that use density based final clustering DenStream makes a good proposal that handles outliers and cluster evolution with the use of outlier and potential micro cluster. It saves time by searching with priority core or potential micro-cluster to locate arriving data. The problem with this algorithm is the high number of user defined parameters, that make it unsuitable for users with little knowledge of the process. That problem is also seen but to a lesser extent in DStream algorithm where context should be provided in order to create a grid fine enough for mapping. DStream algorithm also provides an explicit way to deal with outliers as low density grid areas that are not considered for initial mapping.

The fuzzy approaches can grow from zero (without any user knowledge). In fact P3S, DenStream and simpl_e_TS+ describe explicitly how to built the initial cluster partition from scratch. P3S and DenStream build the cluster structure in an offline fashion, instead, simpl_e_TS+ makes it online. The simpl_e_TS+ algorithm does not consider elimination of rules with a decay factor but instead considers elimination by low rule's contribution in the global output. Only the simpl_e_TS+ approach considers non pre fixed inputs and only P3S considers quantitative and non quantitative data.

References

[1] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.

[2] Plamen Angelov. Fuzzily connected multimodel systems evolving autonomously from data streams. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(4):898–910, 2011.

[3] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.

[4] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and data mining*, pages 133–142, 2007.

[5] DISCO GROUP, LAAS-CNRS. *P3S User's Manual*, Mai 2011.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databdata with noise. In *KDD*, 1996.

[7] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[8] Tatiana Kempowsky, Audine Subias, and Joseph Aguilar-Martin. Process situation assessment: From a fuzzy partition to a finite state machine. *Engineering Applications of Artificial Intelligence*, 19(5):461–477, 2006.

[9] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The ClusTree: indexing micro-clusters for any stream mining. *Knoledge and information systems*, 29(2):249–272, 2011.

[10] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.