DyClee: Dynamic clustering for tracking evolving environments

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

DyClee: Dynamic clustering for tracking evolving environments

Nathalie Barbosa Roa[a,b,c], Louise Travé-Massuyès[a,*], Victor H. Grisales-Palacio[b]

[a] *LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
[b] *Dpt. of Mechanical and Mecatronics Engineering, Universidad Nacional de Colombia. Bogotá, Colombia*
[c] *now with Continental Automotive France. Toulouse, France*

Abstract

Evolving environments challenge researchers with non stationary data flows where the concepts – or states – being tracked can change over time. This requires tracking algorithms suited to represent concept evolution and in some cases, e.g. real industrial environments, also suited to represent time dependent features. This paper proposes a unified approach to track evolving environments that uses a two-stages distance-based and density-based clustering algorithm. In this approach data samples are fed as input to the distance based clustering stage in an incremental, online fashion, and they are then clustered to form μ-clusters. The density-based algorithm analyses the micro-clusters to provide the final clusters: thanks to a forgetting process, clusters may emerge, drift, merge, split or disappear, hence following the evolution of the environment. This algorithm has proved to be able to detect high overlapping clusters even in multi-density distributions, making no assumption about cluster convexity. It shows fast response to data streams and good outlier rejection properties.

Keywords: Dynamic Clustering, Data Mining, On-line Learning, Time-series, Data Streams, Multi-density clustering

1. Introduction

"The ability to adapt to the environment is an essential quality for any intelligent mechanism" [1]. Technological advances of past decades have resulted in production and service infrastructures highly adaptive to constantly changing environments and now instrumented to deliver massive data flows. There is a crucial need for monitoring these big infrastructures and track the condition of their components precisely from these data. For example, in process industries, components are mostly subject to hazardous environments, such as severe shocks, vibration, heat, friction, dust, etc. [2]. These environments cause component health state to change and the process behavior to evolve, as has been widely reported in the cases of tool wear [3, 4] and pipe clogging or leaking [5, 6]. Even though aging acts on the process components, "normal" conditions may be maintained regardless of some evolution. In telecommunication networks, changing environments may result in evolution of the structure of the network itself (e.g. adding servers) causing in turn the normal operation mode to evolve. Hence, in most real-world applications, processes are not stationary. Among the methods that have been used to track process behavior, machine learning approaches have been chosen in cases where data is abundant and/or the process is too complex to be modeled [4, 7]. In this framework, a specific process behavior is associated with a *pattern* in the multidimensional space whose axes correspond to the *features* used to characterize the behavior. A pattern hence corresponds to a *concept*, for example "normal operation mode" in a monitoring context.

Machine learning algorithms build a model, formed of a set of patterns, based on a previously prepared data set called the *training data set*. This data set should be representative of the different behaviors of the process. Most machine learning approaches train the algorithm off-line to develop the model that can be used later online. If the

*Corresponding author
Email addresses: nabarbosa@unal.edu.co (Nathalie Barbosa Roa), louise@laas.fr (Louise Travé-Massuyès), vhgrisalesp@unal.edu.co (Victor H. Grisales-Palacio)

available knowledge allows one to label every object of the training data set by a concept, the goal is to learn the contour of the classes corresponding to the different concepts, which refers to *supervised classification*. If the training data set is unlabeled, the goal is to cluster the objects based on a similarity criterion or based on the objects density, which refers to *non-supervised classification*, also known as *clustering*. Assigning a concept to each cluster is then left to the domain expert. In both cases, one obtains a model in the form of a set of classes. In this work, the word *classifier* is used to refer to this model. When referring to the classification algorithm, we may use *classifier system*.

The standard assumption made by most machine learning approaches (*e.g.* classifiers, neural networks, etc.) is that the data has *stationary or non-biased distribution* [8, 9], that is both training and testing samples encountered during the online stage are drawn from the same distribution [10]. Under this assumption, the algorithm can go through the training set as many times as needed but once the model is deployed online, it remains unchanged over time (its structure and sometimes even its parameters) [4]. If the performance of the classifier decreases, the classifier may be re-trained off-line but it does not adapt automatically online. In other words, new objects assigned to a class do not imply a change of the classifier. This type of classification is called *static classification* [11]. Classic static approaches include a variety of statistical classifier systems and clustering algorithms. The interested reader is referred to [12] for a detailed review. In contrast, learning algorithms modeling time-varying processes should be able to adapt the model online, i.e. when tracking the dynamic behavior [13]. The new objects may cause the classifier to change in time, dynamically [14, 15]. One then refers to *dynamic classification* or more precisely *dynamic clustering* because, obviously, new entering data is not labeled.

Dynamism in the classifier is achieved when not only the parameters but the classifier structure changes according to input data in an automatic way. One difficulty is to deal with the *novelty detection* problem, also called *concept drift* [16], which implies to distinguish outlier data samples and noise from concept drift [1] when the input data falls outside the already learned behavior. A dynamic clustering algorithm can be qualified according to the type of process change that it is able to capture. Abrupt changes in data are captured by cluster creation and elimination. Smooth changes are usually reflected as cluster drifts and less frequently as cluster merging and splitting. We will call a clustering algorithm *dynamic* if it achieves at least two of these desired cluster operations, and *fully dynamic* if it achieves all the desired cluster operations (i.e. creation, elimination, drift, merge, and split). Full dynamicity is obviously a highly desirable feature of any dynamic clustering algorithm.

In view of the difficulty in predicting the underlying distribution of data arriving in an on-line fashion, another desired feature is to be able to deal with clusters of any shape, in particular non convex. The aim of our work is to take up this challenge while addressing at the same time the need of processing fast data streams online and coping with big data in real time [17].

This paper presents ***DyClee***, a **Dy**namic **Cl**ustering algorithm for tracking **E**volving **E**nvironments. ***DyClee*** is a distance and density based algorithm that features several properties like handling non convex, multi-density clustering with outlier rejection, and it achieves full dynamicity. All these properties are not generally found together in the same algorithm and ***DyClee*** hence pushes forward the state of the art in this respect.

This paper subsumes and extends [6] and [18], which can be seen as setting the foundations of the version of ***DyClee*** presented in this paper. However, [6] has a different scope because it puts the focus on a preprocessing stage, called episode abstraction, used to generate additional features that capture the underlying dynamics of the signals forming the data stream. In [6], ***DyClee*** is shown to be applicable to process supervision and diagnosis. The core algorithms of ***DyClee*** were at an early stage and did not implement many of the features presented in this paper, for instance no multi-density clustering was available at that time. ***DyClee*** algorithms were also substantially improved compared to [18]. They were significantly optimized, for instance the second stage is now based on a spatial indexing method that speeds up the final clustering. In addition, intensive benchmarking has been completed and a significant amount of additional tests are presented in this paper, which helps assessing the performances of ***DyClee*** compared to other algorithms of the literature.

This paper is organized as follows. Section 2.1 presents some related work and positions the contributions of this paper. Section 3 presents the ***DyClee*** two-stages algorithm articulating the distance-based stage that produces μ-clusters in Subsection 3.1 and the density-based stage in Subsection 3.3. In this later subsection, we present how groups of connected μ-clusters are formed using spatial indexing in Subsection 3.3.1 then proceed presenting two configurable options, the global density based approach in Subsection 3.3.2 and the local density based approach that achieves multi-density clustering in Subsection 3.3.3. The main functionalities of ***DyClee*** are presented in Section 4, including how the novelty detection problem is approached in subsection 4.1 thanks to a forgetting mechanism de-

scribed in 4.1.2. Subsection 5 summarizes the algorithm and presents the different possible configurations of ***DyClee***. These are illustrated in Section 6 in which the results of extensive benchmarking are given and compared to several specialized algorithms of the literature. Section 7 then demonstrates the use of ***DyClee*** for process monitoring on two real data sets. The first data set refers to a gasifier and the second to the boiler subsystem of a steam generation process. Finally Section 8 concludes the paper and presents future research directions.

2. Work positioning

2.1. Related work

Dynamic clustering has been tackled by several authors. Among the main methods we can mention connectionist approaches like Self-Adaptive Feed-Forward Neural Networks (SAFN) [19] and the evolving cascade connectionist system of [20], fuzzy logic approaches among which Evolving Clustering [17, 21], the algorithm $pClass$ to extract fuzzy rules [22] and also [23, 24], neo-fuzzy approaches [25], Growing Gaussian Mixture Models (2G2M) [26], and random forests [27]. Deep learning architectures have also been proposed [28]. Finally, we must mention the *oDP* algorithm [29] that is the online version of the *DP* algorithm [30] based on identifying density peaks. These approaches classify time-dependent data seen as points in a d-dimensional space. Some of these methods imply high requirements in terms of memory and computational power and are hence not suitable for handling online large amounts of data, such as data arriving in a stream.

All the above mentioned works consider adaptation to abrupt changes through cluster creation but [23] and [26] do not consider elimination. Only SAFN [19] is fully dynamic. [21] considers cluster replacement to handle cluster drift. Cluster replacement involves cluster creation around the new focal point followed by the old cluster elimination, which may be poorly adapted to smooth changes. On the other hand, [29] considers clustering on sliding windows in which one object is inserted and another is deleted at each time step. All the data outside the current window is simply ignored in the clustering, which also seems inadequate to represent how the concepts drift.

Concept drift refers to a subtle change of the underlying data distribution [1]. It can be categorized into two types: virtual concept drift or drift in data distribution, and real concept drift or drift in the conditional distribution of the target concept, while the distribution of the input may – or may not – stay unchanged [7]. The dynamic classifiers [17, 21, 26, 19, 23] manage only virtual concept drift. However, the ability to handle both, virtual and real concept drift, is a highly desirable feature of any clustering algorithm.

The possibility to cope with complex patterns, in particular non-convex clusters, has been pointed as another desired feature. It has been mostly considered by graph-theoretic algorithms. Path-based clustering [31, 32], spectral clustering [33], expectation-maximization using minimal spanning tree [34], agglomerative clustering [35] are some of the non-convex clustering algorithms among others. Unfortunately, these algorithms work in an off-line fashion and with the stationary distribution assumption. To the best of our knowledge, only the quite recent dynamic classifier $Cedas$ proposed in [36] assure the detection of non convex sets.

Clustering techniques for data streams have emerged usually as two stages algorithms [37, 38, 39, 36]. In the first stage data samples are collected, preprocessed, and compressed in μ-clusters along a distance-based clustering approach. In the second stage, μ-clusters are grouped into actual clusters that can be linked to the concepts of the domain. ***DyClee*** is going that way. Through optimized algorithms at each step, it achieves fast processing and detection of non convex sets. This properties are also achieved by the dynamic classifier $Cedas$ [36]. However, two important distinctions must be mentioned. At the distance based stage, ***DyClee*** uses the L_1 norm or Manhattan distance while $Cedas$ uses the L_2 norm or Euclidean distance. Given that we want the selected distance measure to work in high dimensional spaces, the L_1 norm is more appropriate than the L_2 norm as shown in several works that are discussed in Section 3.1. At the density based stage, ***DyClee*** achieves multi-density clustering while $Cedas$ uses only two levels of density. In the case of high and low density clusters, $Cedas$ would reject all the samples of low density clusters as outliers, which is obviously not desired.

2.2. Summary of contributions

The main disadvantages of many of the previous approaches consist in high requirements in terms of memory and/or computational power that are not suitable for online stream clustering, poor adaptation to smooth concept drift, impossibility to cope with complex patterns, and impossibility to deal with multi-density data configurations. In regard to these, the contributions of this paper are the following:

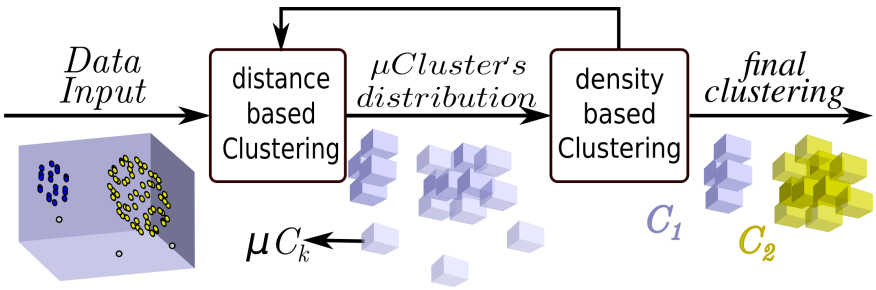


Figure 1: Principle of ***DyClee***

1. We introduce a novel online clustering algorithm named ***DyClee*** for streaming data based on a distance-based and a density-based stage that can run in parallel and execute at a high and lower frequency respectively;
2. We demonstrate that ***DyClee*** achieves several interesting properties, in particular handling non convex, multi-density clustering with outlier rejection, and full dynamicity;
3. Compared to [6], we have improved ***DyClee*** with several new features that can be used by the user in an optional way, like multi-density clustering;
4. Compared to [18], we have optimized the density-based stage based on spatial indexing;
5. We provide concrete demonstrations of ***DyClee*** clustering on several synthetic data sets and on two real data sets in the engineering domain.

3. The *DyClee* algorithm

The dynamic clustering algorithm that we present uses the two stages distance and density-based clustering approach proposed in [6] modified to be able to discover clusters exhibiting different levels of density without any assumption on linearity or convexity. The use of spatial indexing allows fast access to μ-clusters in the first stage and speeds-up connectivity search in the second stage. Both stages are on-line but operate at different time scales. In our proposal, μ-clusters of similar densities can form clusters of any shape and any size. The multi-density feature allows the detection of novelty behavior in its early stages when only a few objects giving evidence of this evolution are present.

DyClee was developed to work under the unsupervised learning paradigm in an incremental fashion. Many existing learning algorithms do not emulate the way in which humans learn. Humans experience the world on the fly, and carry out learning in an incremental way. Often such learning is unsupervised, with the world itself as the teacher [40]. ***DyClee*** learns in a similar way. It makes no *a priori* assumption about the data structure but discovers it progressively, changing the clusterer structure, and hence the representation of concepts, to describe the received data.

DyClee first stage operates at the rate of the data stream and creates μ-clusters putting together data samples that are close in the sense of the L_1-norm. μ-clusters are stored in the form of summarized representations including statistical and temporal information.

DyClee second stage takes place at a lower frequency and analyses the μ-clusters distribution. Its cycle is a multiple of the first stage cycle given by t_{global}, where t_{global} specifies the number of samples that have been processed by the first stage algorithm. The density of a μ-cluster is considered as low, medium or high (cf. subsections 3.3.2 and 3.3.3) and is used to create the final clusters by a density-based approach. Similarly to [41], a cluster is defined as the set of connected μ-clusters where every inside μ-cluster presents high density and every boundary μ-cluster exhibits either medium or high density.

Both stages work as parallel threads and exchange information to mutually improve performance. The distance- and density-based parallel structure allows the algorithm to find the final clusters even in evolving environments described in high dimensional spaces and it has proved outlier rejection capabilities [39, 6]. A global graphical description of ***DyClee*** in block diagrams can be seen in Figure 1. The two stages are further explained in subsections 3.1 and 3.3.

3.1. Distance-based clustering stage

The first clustering stage aims at forming groups of objects of small controlled size, also called μ-clusters, based on object's similarity. As most methods for data analysis we use an L_p norm as dissimilarity measure. Since Aggarwal et al. [42] proved that the meaningfulness of the L_p norm worsens faster with increasing dimensions for higher values of p, we use Manhattan distance, i.e. L_1 norm. In the classification context, this norm outperforms all the other norms in the L_p family as is shown in [43].

3.1.1. μ-clusters

Let us consider that an object $X = [x^1, \ldots, x^d]$ marked with a time stamp t_X and qualified by d features. A μ-cluster μC_k, is the representation of a group of objects *close in all dimensions* whose location is identified by an index id_k and whose information is summarized in a characteristic feature vector CF_k of the following form:

$$CF_k = (n_k, LS_k, SS_k, t_{lk}, t_{sk}, D_k, Class_k) \tag{1}$$

where $n_k \in \mathbb{N}$ is the number of objects in the μ-cluster k, $LS_k \in \Re^d$ is the vector containing the linear sum of each feature over the n_k objects, $SS_k \in \Re^d$ is the square sum of features over the n_k objects, $t_{lk} \in \Re$ is the time when the last object was assigned to that μ-cluster, $t_{sk} \in \Re$ is the time when the μ-cluster was created, D_k is the μ-cluster density and $Class_k$ is the μ-cluster label if known. Using LS_k, SS_k and n_k the variance of the group of objects assigned to μ-cluster μC_k can be calculated. Feature vector was introduced in [44] and has been widely used since for on-line clustering, see *e.g.* [37, 38, 39] .

The data is normalized according to the data context, i.e. the feature range $[min^i, max^i]$ of each feature $i, i = 1, \ldots, d$. If no context is available in advance, it may be established on-line. μ-clusters are shaped as a d-dimensional boxes since the L_1-norm is used as distance measure. The size of the hyperboxes S^i along each dimension i is set as a fraction of the corresponding feature range. The hyperbox size per feature is hence found according to (2), where ϕ^i is a constant establishing the fraction:

$$S^i = \phi^i |max^i - min^i|, \quad \forall i = 1, \ldots, d. \tag{2}$$

The density of a μ-cluster μC_k is calculated using the current number of objects n_k and the current hypervolume of the bounding hyperbox $V = \prod_{i=1}^{d} S^i$, as shown in (3):

$$D_k = \frac{n_k}{V}. \tag{3}$$

3.2. Object insertion

DyClee structure starts empty and is built by successive sample insertions. The first arrived sample becomes the center of the first μ-cluster, and from then on, the insertion mechanism described next is applied. This mechanism uses the notion of *reachable μ-cluster* as defined below.

Definition 3.1 (Reachable μ-cluster). *A μ-cluster μC_k is reachable from a data sample $X = [x^1, \ldots, x^d]$ if*

$$L_\infty(X, c_k) \equiv \max_i \left| x^i - c_k^i \right| < \frac{S^i}{2}, \forall i = 1, \ldots, d$$

where $c_k = [c_k^1, \ldots, c_k^d]^T$ is the center of μC_k and the c_k^i's are calculated from LS_k as $c_k^i = LS_k^i / n_k$.

Data samples are inserted into the closest (w.r.t. Manhattan distance) reachable μ-cluster. The distance between the object $X = [x^1, \ldots, x^d]^T$ and a μ-cluster μC_k, named as $dis(X, \mu C_k)$, is calculated as the sum of the distances between the μC_k vector center $c_k = [c_k^1, \ldots, c_k^d]^T$ and the object value for each feature as shown below:

$$dis(X, \mu C_k) = L_1(X, c_k) = \sum_{i=1}^{d} \left| x^i - c_k^i \right| \tag{4}$$

If there is a tie between several μ-clusters, the data sample is mapped to the denser one. Once found, the feature vector of the selected μ-cluster is updated with the data sample information.

In order to speed-up the retrieval of the closest μ-cluster when a new sample arrives, μ-clusters are stored in one of two lists. The first list is the 'Active μ-cluster' list, the A-list, in which medium and high density μ-clusters are stored. The A-list hence contains the μ-clusters in which sample are more frequently mapped. This list is the first source queried for reachable čs, giving priority in search to the high and medium density μ-clusters over the low density μ-clusters. The second list, the O-list, hence contains the low density μ-clusters. This list is queried only if the data sample does not reach any of the μ-clusters in the A-list.

If the data sample does not reach any of the existing clusters in the two lists, a new μ-cluster is created with the data information and using its time-stamp as cluster time of creation. It is appended to the O-list.

The algorithm 1 given in Figure 2 provides the pseudo-code for μ-clusters formation, i.e. for the distance-based stage of ***DyClee***.

Algorithm 1: Distance-based clustering stage

```
while X in queue do
    if Message in com_pipe then
        com_pipe: receive Message from density-based stage
        Update μC lists with Message
    end
    if no μC exist then
        create μC with X info
        append μC to O-list
    else
        Reachables = μC's in A-list that can reach X                    // L∞ distance
        if Reachables not null then
            find closest μC in Reachables                               // Manhattan distance
            update μC with X
        else
            Reachables = μC's in O-list that can reach X                // L∞ distance
            if Reachables not null then
                find closest μC in Reachables                           // Manhattan distance
                update μC with X
            else
                create μC with X info
                append μC to O-list
            end
        end
    end
    Write trend to log
    if curr_t − t_last_message >= t_global then
        com_pipe: Send lists to density-based stage
        t_last_message = curr_t
    end
end
close com_pipe
```

Figure 2: Pseudo-code of the distance-based clustering stage

3.3. Density-based clustering

In this stage, density-based clustering is executed over the μ-clusters, which allows the algorithm to find clusters of arbitrary shape. A μ-cluster is qualified according to its density as one of three options: dense μ-cluster ($\mathbb{D}\mu$-cluster), semi-dense μ-cluster ($\mathbb{S}\mu$-cluster) or low density (outlier) μ-cluster ($\mathbb{O}\mu$-cluster). The μ-cluster type is decided automatically based on the density of surrounding μ-clusters as explained in subsections 3.3.2 and 3.3.3). $\mathbb{S}\mu$-clusters may result from an increase in the number of outliers from which one can expect a cluster creation, so they have to be updated more frequently than other μ-clusters.

As stated at the beginning of this section, a cluster is defined as a set of *connected* μ-clusters where every inside μ-cluster is a $\mathbb{D}\mu$-cluster and every boundary μ-cluster is either $\mathbb{S}\mu$-cluster or $\mathbb{D}\mu$-cluster. The definition of connected μ-clusters is given by Definition 3.3 that relies on Definition 3.2 defining the notion of *directly connected* μ-clusters. This definition allows us to define the notion of *group* of μ-clusters that is used to define the clusters.

Definition 3.2 (Directly connected μ-clusters). *Let μC_{k_α} and μC_{k_β} be two μ-clusters, then μC_{k_α} and μC_{k_β} are said to be* directly connected *if their hyperboxes overlap in all but φ dimensions, where φ is an integer. The parameter φ establishes the feature selectivity of the classifier and can be set by the user (cf. Section 5.2).*

The μ-clusters μC_j directly connected to a μ-cluster μC_k are those that are at a given distance with respect to the L_∞ norm of μC_k, i.e. whose center c_j satisfies the following condition:

$$\exists I_\varphi = \{i_1, \ldots, i_{d-\varphi}\} \subseteq \{1, ..., d\} \text{ such that } \max_i \left| c_j^i - c_k^i \right| < \frac{S^i}{2}, \forall i \in I_\varphi \tag{5}$$

Definition 3.3 (Connected μ-clusters). *A μ-cluster μC_{k_1} is said to be connected to μC_{k_n} if there exists a chain of μ-clusters $\{\mu C_{k_1}, \mu C_{k_2}, \cdots, \mu C_{k_n}\}$ such that μC_{k_i} is directly connected to $\mu C_{k_{i+1}}$ for $i = 1, 2, \cdots, n-1$.*

Definition 3.4 (μ-clusters group). *A set of connected μ-clusters is defined as a group.*

Definition 3.5 (Cluster). *A set of μ-clusters is defined as a cluster if they belong to the same group, i.e. they are connected, and every inside μ-cluster is a $\mathbb{D}\mu$-cluster and every boundary μ-cluster is either $\mathbb{S}\mu$-cluster or $\mathbb{D}\mu$-cluster.*

3.3.1. Group search and μ-cluster indexing

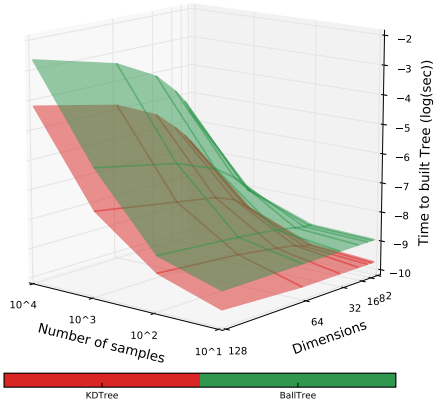
The task of finding all the groups of μ-clusters can be seen as that of finding all the connected μ-clusters to a μ-cluster μC_k. Finding the all the connected μ-clusters comes back to finding all the directly connected μ-clusters, that can be seen as *neighbors*, and then finding all the directly connected μ-clusters to its neighbors and so on. The task of finding the neighbors, which is computationally expensive, is similar to that of finding the nearest neighbors [45], which has been widely studied and specific strategies have emerged to improve this search [46].

Spatial indexing methods have been widely investigated to speed-up the NN-query. These methods sort data based on its spatial occupancy. Specifically, they use hierarchical data structures to decompose the data space into regions which can be further decomposed into sub-regions, and so forth. These regions, usually represented as a tree structure, facilitate the NN query and also intersection queries [47].

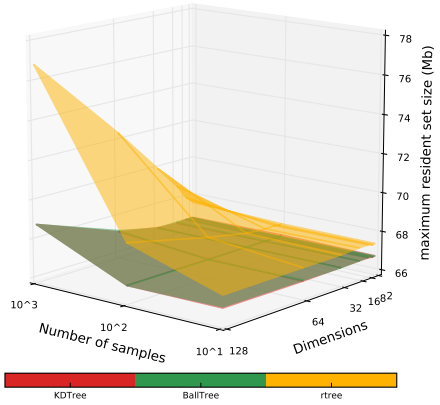
Since we use hyperboxes to represent data samples the use of a spatial index seems appropriate. Multi-dimensional indexing structures have proved to reduce similarity search time in low to medium dimensional spaces. This indexing should allow fast access to μ-clusters in the first stage and speed-up connectivity search in the second stage. Several options were analyzed in order to find the index best suited to our research problem. Three different tree data distributions were tested in order to find the index structure best suited to our goals. The selected structures are: the 'R*Tree' [48], the 'BallTree' [49] and the 'KDTree' [50]. These structures were tested using three data sets exhibiting different spatial behavior: a uniformly distributed data set and two geometrically distributed data sets. Tests were performed varying the number of samples among the values $N = \{10, 100, 1000, 1000\}$ and the number of dimensions as $d = \{2, 4, 8, 16, 32, 64, 128\}$.

The R*Tree showed the poorest performance in terms of both computation time and memory usage. The BallTree was found to be an order of magnitude slower than the KDTree. The KDTree was selected as spatial index since it proved to be more efficient in μ-clusters group retrieval and index construction. Some graphical results are summarized in Figure 3 and the complete test results are reported in [51]. It can be seen from Figure 3 that the tree-like indexes perform poorly when the number of indexed elements is bigger than a thousand. Consequently, in the algorithm implementation a limit of a thousand μ-clusters should be considered if the analyzed dimensions scale up to a hundred.

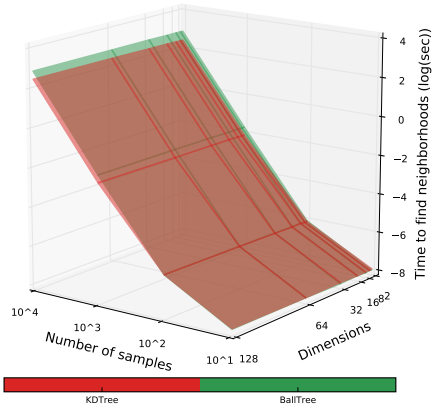
DyClee implements two different approaches to establish the dense character of μ-clusters, named *global-density approach* and *local-density approach*. The former approach allows to detect clusters according to two levels of density while the later allows the detection of clusters with varied densities. ***DyClee*** global- and local-density approaches are explained in the following subsections.



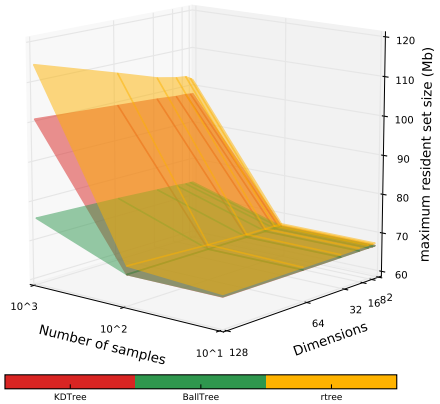
(a) Tree building Computation time ($log_{10}(sec)$)



(b) Tree building Maximum resident set size (Mb)



(c) Group retrieval Computation time ($log_{10}(sec)$)



(d) Group retrieval Maximum resident set size (Mb)

Figure 3: Comparison between different tree indexes for the tree building task (Top) and group retrieval task (Bottom) in a spherical distribution. The R*Tree was removed from (a) and (c) to illustrate the difference between KDTree and BallTree

3.3.2. Global-density approach to clustering

In the global-density approach, density is considered as a μ-cluster characteristic with respect to all the other μ-clusters. Two measures are considered as representative of the μ-clusters, the average of μ-cluster's density and the median. These measures work as thresholds for establishing the dense character of a μ-cluster, with no need of any user defined parameter. The intuition behind the selection of these measures is that the median and average densities of an heterogeneous set of μ-clusters are significantly different, although, if the set of μ-clusters is uniformly dense, these two quantities are equal.

Formally, a μ-cluster μC_k is said to be dense at time t if its density is greater than or equal to both global measures, i.e. the median and the average. On the contrary if its density is bigger or equal to one of the two measures and lower

than the other, the μ-cluster is said to be semi-dense ($\mathbb{S}\mu$-cluster). Finally if the μ-cluster μC_k has a density below both thresholds it is said to be an $\mathbb{O}\mu$-cluster. These conditions are given by inequalities (6) to (8), where D_i is the density of the μ-cluster μC_i and K is the total number of μ-clusters:

$$\mu C_k \text{ dense} \Leftrightarrow D_k \geq median\,(D_1, \cdots, D_K) \wedge D_k \geq avg\,(D_1, \cdots, D_K)\,, \tag{6}$$

$$\mu C_k \text{ semi-dense} \Leftrightarrow D_k \geq median\,(D_1, \cdots, D_K) \vee D_k \geq avg\,(D_1, \cdots, D_K)\,, \tag{7}$$

$$\mu C_k \text{ outlier} \Leftrightarrow D_k < median\,(D_1, \cdots, D_K) \wedge D_k < avg\,(D_1, \cdots, D_K)\,. \tag{8}$$

As stated before, in order to find the final clusters, we take into account the dense character of the μ-clusters and their connections. Connections are accounted for by the notion of *group*. Every group of μ-clusters is analyzed one after the other in order to identify the clusters. A cluster is created if every inside μ-cluster of the group is a $\mathbb{D}\mu$-cluster and every border μ-cluster is either a $\mathbb{D}\mu$-cluster or a $\mathbb{S}\mu$-cluster.

To better illustrate the cluster formation process, let us consider a group of connected μ-clusters exhibiting two areas of high density as shown at the top left of Figure 4. In the figure, μ-clusters density is represented by opacity. ***DyClee*** searches the $\mathbb{D}\mu$-clusters inside the group and all the μ-clusters surrounding them with medium or high density. This subgroup of μ-clusters is qualified as a cluster as depicted in red in Figure 4 (top right). The remaining μ-clusters of the group are then analyzed in a similar manner. Since some $\mathbb{D}\mu$-clusters are found, another cluster is formed as shown in blue at the bottom right of Figure 4. The pseudo-code of the second stage algorithm is given in Figure 5 as Algorithm 2.

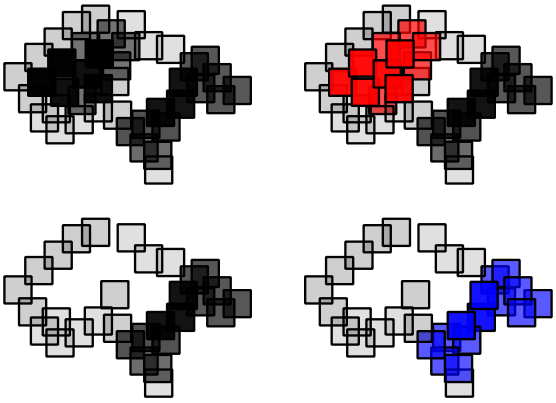


Figure 4: Global-density clustering. Top left: original μ-clusters group. Top right: the first cluster is identified. Bottom left: The remaining μ-clusters of the group to be analyzed. Bottom right: the second cluster is identified, ending the clustering process.

3.3.3. Local-density approach to clustering

Density-based algorithms as DBSCAN [52], DenStream [39], the initial version of ***DyClee*** [6] or the global density approach presented in Section 3.3.2 group data samples according to a notion of density that remains the same in all the data space. The problem of this global point of view to identify a sample as dense appears when regions of varied densities are present in the same data set as can be seen in Figure 6, where density is represented by μ-cluster opacity. In this case, global density-based algorithms may misclassify low density clusters, like the bottom cluster of Figure 6, as noise.

While it is deemed desirable to detect clusters of multiple densities, it is also important to maintain the ability to reject outliers. ***DyClee***'s solution to multi-density clustering, as named in [34], is based on local analysis. The dense character of each μ-cluster is assessed with respect to the density of the other μ-clusters *in the same group*.

The average and the median of the μ-clusters within the same group are chosen as thresholds for this group. In other words, for each group G_l, the μ-clusters having their density higher than or equal to the average density of the

Algorithm 2: Clustering using global-density analysis

```
DMC ← μC_k such that D_k > D_avg ∧ D_k > D_median                    // dense
already_seen= [ ]
for μC_k in DMC do
    if μC_k not in already_seen then
        already_seen.append(μC_k)
        if μC_k.label=Unclass then
            cid ← get next cluster label
            assign cid as current μC_k label
        end
        Connected_μC_k ←all μC in neighborhood overlapped with current μC_k
        while Connected_μC_k not empty do
            neighbor←Connected_μC_k.pop()
            if neighbor is dense then
                neighbor.label← cid
                already_seen.append(neighbor)
                NewConnected_μC_k←all μC overlapped in neighborhood of neighbor
                for newneighbor in NewConnected_μC_k do
                    if newneighbor is dense then
                        Connected_μC_k.append(newneighbor)
                    end
                    newneighbor.label← cid
                end
            end
        end
    end
end
```

Figure 5: Density-based clustering pseudo-code

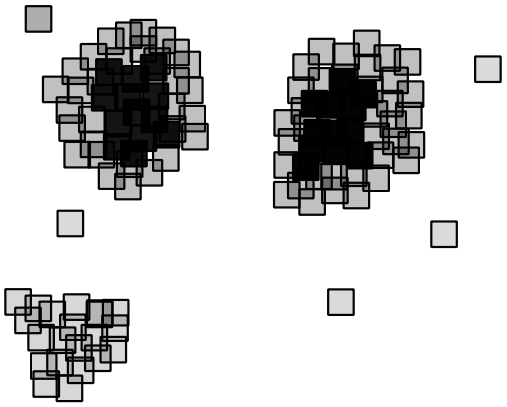


Figure 6: μ-clusters groups of varied densities

group ($avg(D_{G_l})$) and higher than or equal to the median density of the group ($median(D_{G_l})$) are considered as dense. μ-clusters having a density higher than or equal to only one of those measures (either average or median) are considered as $\mathbb{S}\mu$-clusters and those with density below both measures are considered as $\mathbb{O}\mu$-clusters. Assuming that μC_k is a μ-cluster within the group G_l, this is summarized by:

$$\mu C_k \text{ dense} \Leftrightarrow D_k \geq median\left(D_{G_l}\right) \wedge D_k \geq avg\left(D_{G_l}\right), \tag{9}$$

$$\mu C_k \text{ semi-dense} \Leftrightarrow D_k \geq median\left(D_{G_l}\right) \vee D_k \geq avg\left(D_{G_l}\right), \tag{10}$$

$$\mu C_k \text{ outlier} \Leftrightarrow D_k < median\left(D_{G_l}\right) \wedge D_k < avg\left(D_{G_l}\right). \tag{11}$$

Interestingly, the subsequent procedure is the same as in the global density approach presented in Section 3.3.2. A cluster is created if every inside μ-cluster of the group is a $\mathbb{D}\mu$-cluster and every border μ-cluster is either a $\mathbb{D}\mu$-cluster or an $\mathbb{S}\mu$-cluster (in a local sense). After this stage, the group G_l is reduced to the μ-clusters that do not contribute to any cluster and the same procedure is applied recursively.

The μ-clusters groups shown in Figure 6 are analyzed in this manner in Figure 7: first, for each group, clusters are formed with the denser connected μ-clusters resulting in the red and blue top clusters of the figure. Each group is then analyzed in turn with respect to the remaining μ-clusters following the same method. The final classification is shown on the right of Figure 6. The group or groups with the lowest density are taken as outliers.

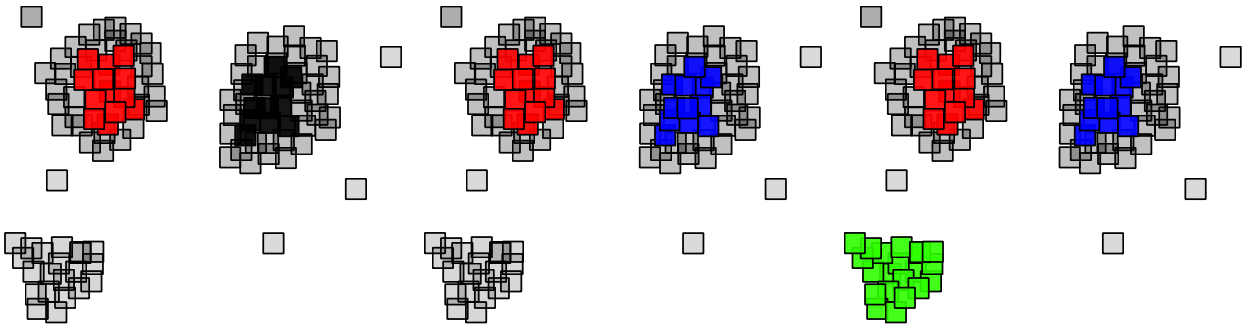


Figure 7: Local-density clustering. Left: the first cluster is identified. Center: The second cluster is identified. Right: the low density cluster is identified, ending the clustering process.

It is worth noting that global-density approaches as the ones proposed in [6], [52] or [39] are unable to detect the green cluster because the densities of the μ-clusters in that group are low with respect to those in the other groups. In fact, the main drawback of those approaches can be observed in overlapping distributions with contiguous regions of different density. Global density algorithms may cluster those regions as one big cluster or consider all the samples in the lowest density regions as noise, missing the characterization of those samples. On the other hand, multi-density approaches may fail to reject noise, since they can cluster it in one low density cluster. In order to avoid the inclusion of outliers in multi-density distributions, ***DyClee*** considers the cluster with the lowest density as noise. Local-density analysis is optional in ***DyClee***, and can be set on or off according to the user's needs (cf. Section 5.2, Table 6).

4. Main functionalities of *DyClee*

4.1. Novelty detection and cluster evolution

Novelty detection imposes itself as a key factor in clustering and classification problems and work has been done on detecting and tracking data changes. [23, 26] reflect data evolution with possible creation of new clusters. Another track of work was done to adapt the whole cluster structure automatically. Cluster changes, creation and elimination, are tracked based on the study of the spatio-temporal properties of a cluster. Cluster evolution is relevant for many applications as customer relationship management [53, 54], fraud detection and marketing [14], network security [55], etc. In order to maintain an up-to-date structure, more weight should be given to incoming data to make it more relevant to the algorithm.

4.1.1. Representation of a forgetting process

DyClee implements a forgetting process in order to cope with cluster evolution. Specifically, μ-clusters are weighted with a decay function dependent on the current time t and the last assignation time t_{lk}. This function

$f\left(t, t_{lk}\right)$ emulates a forgetting process. When a new d-dimensional object $X = \left[x^1, \ldots, x^d\right]^T$ is assigned to a μ-cluster μC_k at t, the cluster last assignation time is updated to $t_{lk} = t$. The other attributes of the feature vector are updated as follows:

$$n_k(t) = n_k(t-1) f\left(t, t_{lk}\right) + 1 \quad \forall k \tag{12}$$

$$LS_k^i(t) = LS_k^i(t-1) f\left(t, t_{lk}\right) + x^i \;\; \forall\, i, i = 1, \ldots, d. \tag{13}$$

$$SS_k^i(t) = SS_k^i(t-1) f\left(t, t_{lk}\right) + (x^i)^2 \;\; \forall\, i, i = 1, \ldots, d. \tag{14}$$

In general, if no object is added to a μ-cluster during the time interval $(t, t + \Delta t)$, its CF at $t + \Delta t$ can be calculated from $CF^{(t)}$ using the decay function for the weighted parameters as follows:

$$CF_k^{(t+\Delta t)} = f\left(\Delta t\right) CF_k^{(t)}. \tag{15}$$

Subsection 4.1.2 shows several options for the decay function $f\left(t, t_{lk}\right)$. If the object cannot be assigned to any of the existing μ-clusters, a new $\mathbb{O}\mu$-cluster is formed using the object information as feature vector. We assume that a μ-cluster with low density ($\mathbb{O}\mu$-cluster) is either outlier or a potential cluster in an emerging state. The later case reveals itself with an increment in the μ-cluster density and consequently, the μ-cluster "grows" into a $\mathbb{S}\mu$-cluster as new data is provided as input.

4.1.2. Decay functions

As stated above, in order to follow the data evolution, data in clusters are subject to a forgetting process. Numerous machine learning methods have implemented some kind of forgetting process through a decay function to be able to detect or track concepts that drift or shift over time. Table 1 presents the set of decay functions implemented in ***DyClee***, hence allowing proper adaptation to all kinds of process evolution underlying the incoming data. In the last column, Table 1 also refers to previous works that used these functions for clustering. These functions are plotted for $t_{w=0} = 6000$, $t_{\sim a} = 2000$ and $t_{lk} = 0$ in Figure 8. The choice of the decay function is left to the user. She/he may also select the default configuration in which no forgetting process is applied (cf. Section 5.2).

Decay function	Equation	Also used by
Linear decay	$f_1\left(t, t_{lk}\right) = \begin{cases} 1 - m\left(t - t_{lk}\right) & t - t_{lk} \leq t_{w=0} \\ 0 & t - t_{lk} > t_{w=0} \end{cases}$	concept drift adaptation [56]
Trapezoidal decay	$f_2\left(t, t_{lk}\right) = \begin{cases} 1 & t - t_{lk} \leq t_{\sim a} \\ \frac{m-t}{m-t_{\sim a}} & t_{\sim a} \leq t - t_{lk} \leq t_{w=0} \\ 0 & t - t_{lk} > t_{w=0} \end{cases}$	Monitoring parameter for clustering [14]
Z-shape function	$f_3\left(t, t_{lk}\right) = \begin{cases} 1 & t \leq t_{\sim a} \\ 1 - 2\frac{t - t_{\sim a}}{t_{w=0} - t_{\sim a}} & t_{\sim a} \leq t - t_{lk} \leq \frac{t_{\sim a} + t_{w=0}}{2} \\ 2\frac{t - t_{w=0}}{t_{w=0} - t_{\sim a}} & \frac{t_{\sim a} + t_{w=0}}{2} \leq t - t_{lk} \leq t_{w=0} \\ 0 & t - t_{lk} > t_{w=0} \end{cases}$	
Exponential decay	$f_4\left(t, t_{lk}\right) = e^{-\lambda_d (t - t_{lk})}$	Stream clustering [37], [38]
Half life function	$f_5\left(t, t_{lk}\right) = \beta^{-\lambda_d (t - t_{lk})}$	Fuzzy classification [24]
Sigmoidal decay	$f_6\left(t, t_{lk}\right) = \frac{1}{1 + e^{-a(t-c)}}$	

Table 1: Decay functions implemented in ***DyClee*** to emulate a forgetting process

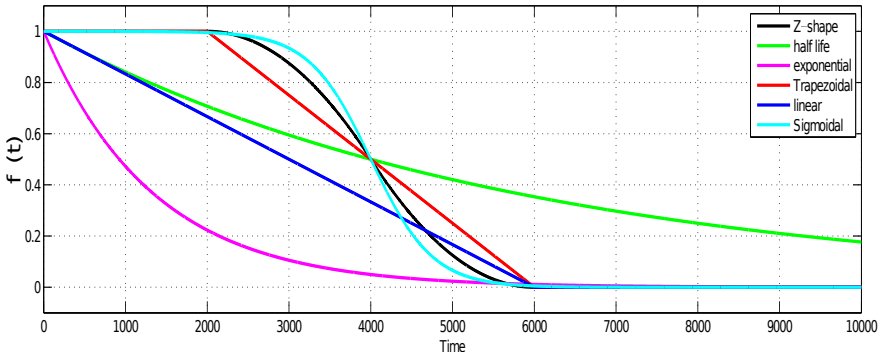


Figure 8: Profile of the decay functions implemented in ***DyClee*** (cf. Table 1)

The simplest forgetting function $f_1\left(t, t_{lk}\right)$ corresponds to a linear decay. The function slope m could be inversely proportional to the time it takes to the function to go from one to zero, $m = 1/t_{w=0}$. If a non-forgetting time range is appended to a linear decay, we get a trapezoidal decay profile. This profile is often used in electronic applications. In the trapezoidal decay function $f_2\left(t, t_{lk}\right)$, $t_{\sim a}$ represents the no forgetting time and $t_{w=0}$ the time when the function reaches zero. Fuzzy logic makes use of the Z-function $f_3\left(t, t_{lk}\right)$. This spline-based function uses only two parameters, the extremes of the sloped portion of the graph $t_{\sim a}$ and $t_{w=0}$.

Statistical processes use overall the exponential decay function $f_4\left(t, t_{lk}\right)$, where λ_d is a positive rate known as exponential decay constant. This type of functions is the most widely used to model decay since it has applications in all fields of science. A generalization of exponential decay given by the change in the base from e to any value β provides interesting properties to $f_5\left(t, t_{lk}\right)$. For example, if β is chosen to be $\beta = 2\psi$, then the time at which half of the data is forgotten, is $\frac{1}{\psi\lambda_d}$. This function is known as the half life function and is widely used in biological processes.

Although exponential decay has been widely used, several processes do not follow this imminent decay dynamics. Logistic functions are the common alternative when dead zones or saturation make part of the process behavior. The Sigmoid function $f_6\left(t, t_{lk}\right)$ is a particular case of the logistic function which has been widely used in ecology, statistics, medicine, machine learning, etc. to describe, for example, growing processes. It introduces a dead zone, which in our case, is used to describe the time gap during which cluster's data should not be affected by the forgetting process.

In ***DyClee***the forgetting process (cf. Table 2) impacts the μ-cluster feature vector, in particular the density of the μ-clusters. Every time t_{global} samples have been processed to form μ-clusters, the density of all μ-clusters is updated as explained in Section 4.1.1. Changes in density imply that the μ-cluster type may change: density decrease may result in $\mathbb{D}\mu$-clusters becoming $\mathbb{S}\mu$-clusters and $\mathbb{S}\mu$-clusters becoming $\mathbb{O}\mu$-clusters, and vice-versa. This is the feature at the core of clusters evolution.

4.1.3. Long-term and short-term memory

We claim that clusters that are $\mathbb{D}\mu$-clusters at some point in time should be given particular attention because they represent well-established concepts. Therefore they should not be forgotten only because the concepts they describe do not correspond to the incoming data anymore. Indeed, some data sets, in particular those coming from dynamic processes, could go through cycles and hence they could return to previous states that were defined previously.

In order to improve the clustering evolution process, the user can configure ***DyClee*** (cf. Table 3) to implement short-term and long-term memory. Short-term memory refers to the μ-clusters describing the most recent behavior, i.e μ-clusters in the A-list and in the O-list (cf. Section 3.1.1). Short-term memory is subject to the forgetting process as previously described. If a $\mathbb{O}\mu$-cluster was once dense, it is stored in a database of old known concepts, i.e. in the long-term memory. If the density of an $\mathbb{O}\mu$-cluster drops below the low-density threshold, it is eliminated only if it has never been $\mathbb{D}\mu$-cluster before. The long-term memory is accessed when an input object is not found to belong to any μ-clusters in the short-term memory. This action verifies if the input object belongs to a previously learned concept, speeding up its recognition in case the concept was already learned before.

4.2. Outlier detection

Outlier detection has been widely investigated in several knowledge disciplines, including statistics, data mining and machine learning and several approaches have been proposed ([57],[16],[58]). This problem is related to other problems such as novelty detection, anomaly detection, deviation detection, etc. A classical, intuitive, definition of 'outlier' was given by Hawkins in [57]: *'an observation, which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism'*. In general, outliers indicate the occurrence of unknown events or unexpected patterns resulting from system evolution or reconfiguration. Nevertheless, outliers can also be an indication of noise, damage or errors.

Outlier detection methods can be classified according to several characteristics: the use of training (pre-labeled) data, the assumption of a standard statistical distribution, the type of data set, the dimension of detected outliers, the type of detected outliers (global/local), among others.

Most commonly used outlier detection approaches are those classified as distribution-based, distance-based [59], density-based [60] or angle-based [61]. If the probability density function (or functions) is (are) known and alleged to follow a normal distribution, the simplest approach to outlier detection is to compute the probability of a sample to belong to a class (or to classes), and then use a threshold to establish its outlier character. This approach is equivalent to finding the distance of the sample to class means and threshold on the basis of how many standard deviations away the sample is [16]. Unfortunately, in most real world applications the data distribution is not known *a priori*, as a result non parametric methods are generally preferred. Non parametric approaches like distance-based, density-based or angle-based, make no assumption about the statistical properties of the data which makes them more flexible than parametric methods.

Since no formal unique definition of outlier exists, the notion of outlier may greatly differ from one outlier detection technique to another. In this work, the outlierness or not of a sample is related to the outlierness of the μ-cluster that contains the sample. Explicitly, for ***DyClee***, an *outlier sample* is defined as: *'a sample contained in an $\mathbb{O}\mu$-cluster* and an *$\mathbb{O}\mu$-cluster* is defined as *a representation of samples that, due to their position and density, do not correspond to any of the identified clusters'*.

The distance and density analysis that ***DyClee*** performs allows it to detect both, global and local multivariate outliers, the former in all configurations and the latter when multi-density clustering is used. The outlier character of a sample is binary, that is, a sample can only be considered as outlier or not outlier. Nevertheless, the outlierness of a μ-cluster may be associated with its density which provides a score about how much outlier a μ-cluster is. In other words, the set of all $\mathbb{O}\mu$-clusters can exhibit different densities although their densities remain in the same relative lower level with respect to the clusters densities that are composed of $\mathbb{S}\mu$-clusters and $\mathbb{D}\mu$-clusters.

If global density as presented in Section 3.3.2 is used, ***DyClee*** detects only global outliers. In this approach, a μ-cluster is considered as outlier if its density is lower than its group median density and its group average density, being its group the set of all the μ-clusters connected (directly or indirectly) to it (cf. Definitions 3.2 and 3.3). This is given by equation (8) repeated below:

$$\mu C_k \text{ outlier (global)} \Leftrightarrow D_k < median\left(D_1 \cdots D_K\right) \wedge D_k < avg\left(D_1 \cdots D_K\right). \tag{16}$$

If multi-density clustering as presented in Section 3.3.3 is used, the equation (11) (repeated below) shows that an μ-cluster is considered as outlier if and only if its density is lower than its group median density and its group average density:

$$\mu C_k \text{ outlier (local)} \Leftrightarrow D_k < median\left(D_{G_k}\right) \wedge D_k < avg\left(D_{G_k}\right). \tag{17}$$

Summarizing, ***DyClee*** may detect global or local outliers depending on the configuration. Its ability to detect local outliers is optional, activated only when the local-density approach is used. The next subsection explains it in more details as it provides a presentation of the main algorithmic loop and of the different clustering options offered by ***DyClee***.

5. DyClee operation

5.1. Summary of the main algorithmic loop

When a new object X is given as input, ***DyClee*** maps the object into the closest reachable μ-cluster. If no reachable μ-cluster exists, it creates a μ-cluster using the object attributes. If there is a tie between several μ-clusters, the data sample is mapped to the denser μ-cluster. The feature vector of the selected μ-cluster is then updated with the data sample information. If no of the existing μ-clusters, including those in the long-term memory, is reachable from X, a new μ-cluster is created with the data sample and it is be appended to the O-list. This is performed by the distance based clustering stage (cf. Section 3.1).

Every time t_{global} samples have been processed, the density based algorithm enters into play (cf. Section 3.3). The first task is to update the μ-clusters density types according to their current density. In particular, $\mathbb{D}\mu$-clusters may become to $\mathbb{S}\mu$-clusters, $\mathbb{S}\mu$-clusters may become $\mathbb{O}\mu$-clusters or $\mathbb{O}\mu$-cluster and $\mathbb{O}\mu$-cluster(s) may become $\mathbb{S}\mu$-clusters or be eliminated (or put in the long-term memory if configured). Then, the groups are formed and the clusters are identified according to the configuration selected by the user, i.e. global or local density procedure.

5.2. **DyClee** *parametrization*

DyClee requires only one parameter, which is the parameter `relative-size` that provides the value of all the ϕ^is of Equation (2), i.e. `relative-size` $= \phi^i$, $i = 1, \ldots, d$. This parameter specifies the size of the hyperboxes shaping the μ-clusters (cf. section 3.1) by defining a portion of the total range along every dimension [1]. `relative-size` is a real number in the interval $[0, 1]$. Although values of `relative-size` > 0.5 can be used to deal with specific cases, they are not recommended because they drastically limit the number of μ-clusters and the clustering grain is too rough. Note that low relative size may result in too many clusters because of an excess sensitivity to density gradients. On the other hand, too large relative size leads to incorrect merging of clusters because μ-clusters tend to be dense and connected (cf. experiment in Section 6.2.5).

Although ***DyClee*** has a default configuration, it can be configured differently when the user has specific knowledge about the process that underlies the data and/or the distribution of the data. The different options might also be used to analyze and mine the data. They are though to help the user discover the patterns hidden in the data.

DyClee optional features are listed below:

- `forget_method` [Forgetting function]: String input.

Configuration	Decay function used by ***DyClee***
`forget_method=none (default)`	none
`forget_method=linear`	Linear decay function
`forget_method=expo`	Exponential decay function
`forget_method=sigmo`	Sigmoidal decay function
`forget_method=z_func`	Z_shape decay function
`forget_method=trapezoidal`	Trapezoidal decay function
`forget_method=half_life`	Half life decay function

Table 2: Configuration of the forgetting process

DyClee implements six different decay functions that can be used to achieve correct tracking of the data evolution process as explained in Subsection 4.1.2. The default value is `none` that implies that ***DyClee*** does not use any forgetting process. This configuration is to be used when there is no dynamic process underlying the data, i.e. the data does not refer to time. The choice of any of the other options depends on the knowledge about the dynamic process underlying the data. The reader can refer to Subsection 4.1.2 to select the most appropriate decay function.

- `ltm` [Long-term memory]: Boolean value.

[1] In the current implementation of ***DyClee***, the ϕ^is are all set to the same value given by `relative-size` but this would be easy to modify if necessary.

Configuration	Behavior of ***DyClee***
`ltm=false (default)`	Long-term memory deactivated
`ltm=true`	Long-term memory activated

Table 3: Configuration of the long-term memory

If the long-term memory is activated, $\mathbb{O}\mu$-clusters that have once been $\mathbb{D}\mu$-clusters, i.e. which are in the long-term memory, are not eliminated even though their density drops below a low density threshold. The long-term memory should be activated for data coming from processes that may undergo long cyclic behaviors.

- `Unclass_accepted` [Non-informative label accepted]: Boolean value.

Configuration	Behavior of ***DyClee***
`Unclass_accepted=true (default)`	Outlier rejection
`Unclass_accepted=false`	No outlier rejection

Table 4: Configuration of the outlier rejection procedure

In the default configuration, outliers are detected as explained in Section 4.2. This configuration should be used when the data could be corrupted by noise [23] or when data samples that do not fit the clusters distributions should be detected, e.g. in anomaly detection applications. However, in some applications, it is desirable to label all samples as described by Pyon et al. [54]. That is the case of credit card issuing process for renewal, where all samples should be labelled as *approved* or *rejected*. In this case `Unclass_accepted` should be set to `false`.

- `minimum_mc` [Minimum amount of μ-clusters]: Boolean value.

Configuration	Behavior of ***DyClee***
`minimum_mc=false (default)`	All clusters are retained
`minimum_mc=true`	Clusters with few μ-clusters are discarded

Table 5: Configuration of the sparse cluster procedure

In the default configuration, all clusters are retained in the final clustering. The user can choose the sparse cluster procedure option by setting `minimum_mc` to `true`. This option should be used when the focus of the analysis is on populated cluster profiles. On the other hand, in some situations such as fault diagnosis applications, the default configuration should be used because faulty behavior is far less frequent than normal behavior. Discarding sparse clusters could lead to late fault detection and even non-detection of faulty situations.

- `multi-density` [Local-density analysis]: Boolean value.

Configuration	Behavior of ***DyClee***
`multi-density=false (default)`	Global density procedure is applied
`multi-density=true`	Local density procedure is applied

Table 6: Configuration of the density based procedure

Global density and local density based procedures are explained in subsection 3.3.3 ***DyClee***. In the default configuration, the global density procedure is applied and the clustering is based on two levels of density. The user should set `multi-density` to `true` when she/he wants to obtain clusters with varied density. In this case, density thresholds are found locally using the groups densities.

- Data context (`context`): Matrix.
  The user can choose to provide a matrix for `context`. This matrix should contain the maximum and minimum

values of each feature. The context is used for normalization purposes. If not provided minimum and maximum values of the features are found on-line by ***DyClee***.

- `t_global`[Period of the density-based clustering cycle]: Integer value.
  t_{global} specifies the number of input samples between two runs of the density-based clustering algorithm. In the default configuration, `t_global`=1, which means that the density-based clustering procedure is run for every input sample. t_{global} establishes how often the cluster structure is updated as an output for the user. It is worth noting that the runs of the density-based clustering stage ensue in parallel with the distance-based clustering stage and its results (cluster labels) are returned to the later after each run.

- `uncdim` [Uncommon dimensions]: Integer value.

Configuration	Behavior of ***DyClee***
`uncdim=0 (default)`	μ-clusters are directly connected when their hyperboxes overlap in all dimensions
`uncdim=`φ	μ-clusters are directly connected when their hyperboxes overlap in all dimensions but φ

Table 7: Configuration of the μ-clusters direct connectivity

The `uncdim` option allows the user to set the feature selectivity of the clusterer. By default `uncdim=0`, thus direct connection is established when hyperboxes overlap in all dimensions (cf. Definition 3.2 of Section 3.3; `uncdim` corresponds to the parameter φ).

6. *DyClee* performance

In this section ***DyClee*** is compared with other known algorithms in order to assess its performance. As stated in subsection 5.2, ***DyClee*** requires only one user pre-defined parameter that determines the size of the hyperboxes that shape the μ-clusters. Nevertheless, if the user has some knowledge about the data to be clustered, ***DyClee*** can be configured to achieve the clustering that best fits user's expectations, as seen in subsection 5.2.

6.1. Complexity analysis

Since ***DyClee*** has two stages that work independently, its complexity analysis is also made independently. The complexity of the distance-based stage is $O(dm)$ for each analyzed d-dimensional element, with m being the average amount of μ-clusters per list (A-list and O-list). Since ***DyClee*** is an incremental algorithm, in general $m << n$, where n is the total amount of elements to be clustered, it results a total complexity of $O(n)$. In the optimistic scenario, data is to be mapped to the most representative μ-clusters, so distance comparison is performed only against the μ-clusters in the A-list, leaving out all the $\mathbb{O}\mu$-clusters that are in the O-list. The worst case scenario refers to high dimensional data sets with small number of samples and sparse distribution, and also all the μ-clusters in the same list A-list or O-list. In this case $m << n$ and $d << n$ do not stand any more. In this case, the complexity of the distance-based stage is $\Theta(dmn)$. Note however that data sets with small number of samples are not the target of ***DyClee***.

The complexity of the density-based stage is that of the group retrieval and density analysis. As said in subsection 3.3.1, the KDTree is used as spatial index to improve μ-clusters group retrieval. A KDTree for a set of p μ-clusters can be build in $O(p\,log\,(p))$ time [62, 63]. Looking for μ-cluster's groups is querying for those μ-clusters whose volume intersect the volume of the query point. The query time is bounded by $O\left(p^{(1-1/d)} + k\right)$ with k being the number of reported μ-clusters [64].

6.2. **DyClee** *comparison on literature synthetic test cases*

In Figures 17 to 22, we show ***DyClee*** clustering results for different test cases. Clusters are represented using a combination of colors and glyphs. As a result, samples belonging to the same cluster have both the same color and the same glyph. For example, in Figure 17 (right), there are four green clusters and in Figure 19 (right) there are four blue clusters (even if the tonality is different); however, each of these clusters are represented using different glyph

($\triangleright$, $\star$, $\pentagon$, $\circ$ for 17 and $\diamond$, $\circ$, $\times$ and $+$ for 19), so they denote different clusters. Detected outliers are represented as black dots.

Test cases will show that ***DyClee*** performance is up to the performance of several existing specialized clustering algorithms. Let us note that the real value of ***DyClee*** is to put together in one unique algorithm all the features achieved by different specialized algorithms. Hence its performance is not to be evaluated on each experiment separately but on the whole set of experiments. These experiments are also used to illustrate ***DyClee*** optional features. For those experiments for which visual inspection is not enough to assess the results, the following metrics are used to evaluate the clustering:

- T_{p_i} denotes the number of true positive samples for cluster i, i.e. the number of samples assigned to cluster i whose label does correspond to the dominant label in this cluster,
- F_{p_i} denotes the number of false positive samples for cluster i, i.e. the number of samples assigned to cluster i whose label does not correspond to the dominant label in this cluster,
- F_{n_i} denotes the number of false negative samples for cluster i, i.e. the number of samples not assigned to cluster i whose label corresponds to the dominant label of cluster i,
- *Purity* is defined as the percentage of the total number of samples that were classified correctly:

$$Purity = \frac{1}{n}\sum_{i=1}^{K} T_{p_i} \tag{18}$$

  where n is the total amount of samples clustered and K the number of clusters.

- *Precision* represents the proportion of retrieved samples which are relevant, i.e. the proportion of true positive samples:

$$Precision = \frac{1}{K}\sum_{i=1}^{K} \frac{T_{p_i}}{T_{p_i} + F_{p_i}} \tag{19}$$

- *Recall* represents the proportion of relevant samples found, defined as the number of true positives over the number of true positives plus the number of false negatives:

$$Recall = \frac{1}{K}\sum_{i=1}^{K} \frac{T_{p_i}}{T_{p_i} + F_{n_i}} \tag{20}$$

6.2.1. Detection of convex and non-convex distributions

When data is organized according to non-convex sets, classic clustering algorithms like k-means and stream algorithms like CluStream [37] and ClusTree [38] tend to fail at clustering them. This section presents ***DyClee***'s capabilities to separate neighboring non-convex shaped clusters and to account for local as well as global outliers. To show these properties, we have chosen to cluster some synthetic sets available in the *scikit-learn* Python module [65]. Our algorithm is compared to scikit-learn module provided implementations of different clustering algorithms, namely MiniBatch KMeans (MBK-m), Agglomerative Clustering, Affinity Propagation, DBSCAN [52] and BIRCH [44].

Four noisy data sets were evaluated: concentric circles, moons, blobs and random data with no structure. These data sets were generated using the scikit-learn data set module. Each distribution, is composed of 1500 samples each and is time invariant. In consequence, ***DyClee***'s forgetting process was disabled for this test.

The parameters for each algorithm for the test were: MBK-m (# of clusters), AC (linkage `"average"`, affinity `"cityblock"`, # of clusters, connectivity [2]), AP (`damping`=0.9, `preference`=-200), BIRCH (# of clusters), DBSCAN (`eps`=0.2), ***DyClee*** (`relative-size`=0.06).

[2] connectivity estimated using scikit function kneighbors_graph with n_neighbors=10.

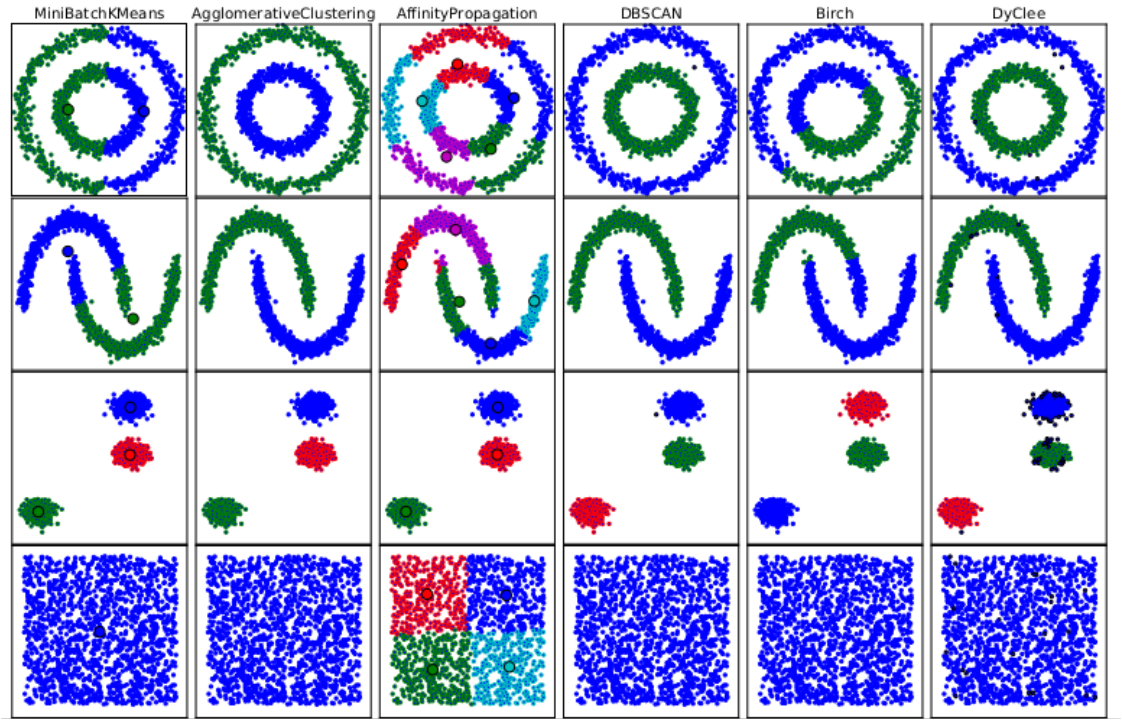


Figure 9: Algorithm comparison in convex and non convex sets. Samples belonging to the same cluster have the same color. Detected outliers appear as black dots. For the MBK-means and the Affinity Propagation algorithms, cluster centers are depicted as black circles.

The results of the test are reported in Figure 9. Note that the AC algorithm, as well as the MBK-m and BIRCH require the number of clusters as initial parameter, which is quite restrictive.

Figure 9 shows that MBK-m, BIRCH and Affinity Propagation are not able to cluster the non-convex sets properly, even if MBK-m, BIRCH are given as input the desired number of clusters. The Affinity Propagation algorithm does not perform well at all in this distribution, creating a high number of clusters. Agglomerative Clustering, DBSCAN and ***DyClee*** are able to detect the non-convex distribution, nevertheless, Agglomerative Clustering is unable to detect outliers and DBSCAN rejects less outliers than ***DyClee***.

6.2.2. Multi-density Clustering

In order to illustrate the multi-density feature, two test sets are used. The first one is taken from [66] and shown on the left of Figure 10. The second is a synthetically generated set shown in Figure 13. ***DyClee*** results for the test case of [66] with the global-density option and the local-density option are shown in Figure 11, on the left and center positions respectively. By default, ***DyClee*** sets the μ-cluster in the lowest density cluster as outliers (black dots) and that's why the center image of Figure 11 identifies all the samples on the right as outliers. If the user wants to detect the lowest density class, it is enough to deactivate the `Unclass_accepted` option (cf. Table 4) to obtain the complete clustering. This result is shown on the right of Figure 11.

The CluStream [37] and Denstream [39] algorithms were confronted to this data set for comparison. in Figure 12 (left), CluStream micro-clusters are depicted in green and the final clustering result in red. DenStream micro-clusters are depicted in green in Figure 12 (right), and the final clusters are represented changing color to blue. It can be seen that DenStream can only detect the two denser clusters; samples in yellow and cyan are not clustered. In addition, even if in this example the target classes are convex sets, CluStream fails to correctly cluster them due mainly to the overlapping of the distributions. Table 8 summarizes the clustering results and show that ***DyClee*** is outperforming the two other algorithms.

Interestingly, using the local-density scheme helps to better shape clusters that share frontiers, that is, clusters that

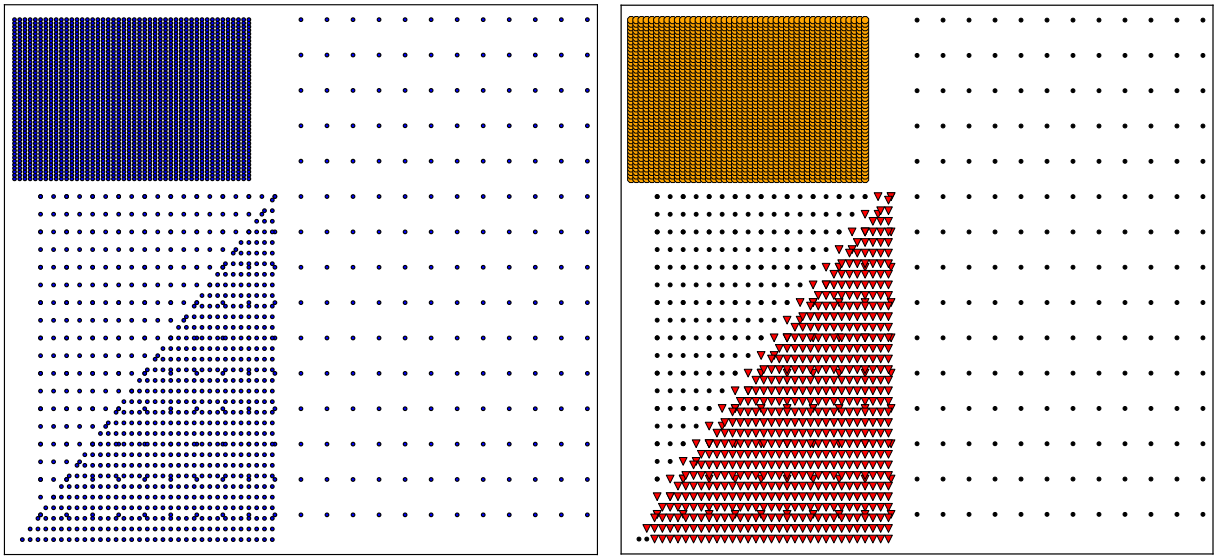


Figure 10: On the left, clusters of varied density, from [66]. On the right, DBSCAN clustering results with $eps = 0.1$

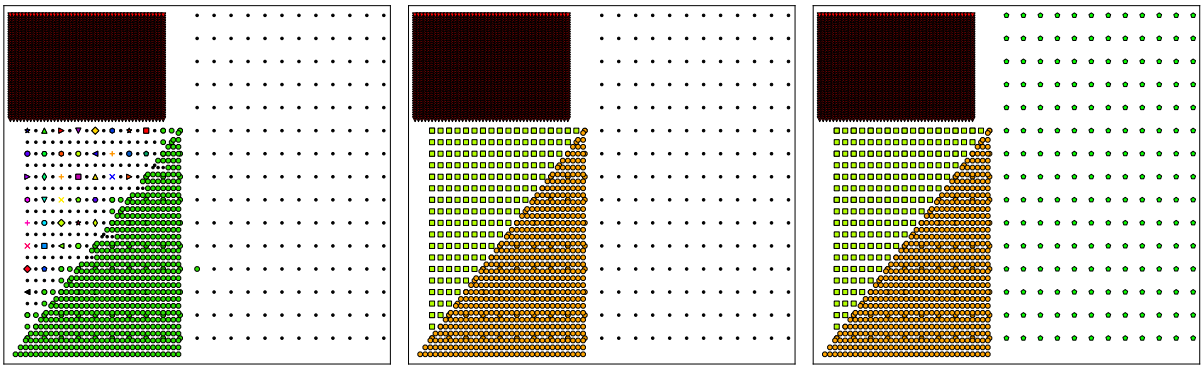


Figure 11: ***DyClee*** clustering results for the test case of [66] shown on the left of Figure 10: without multi-density option (left), with multi-density option (center) and `Unclass_accepted=False` (right)

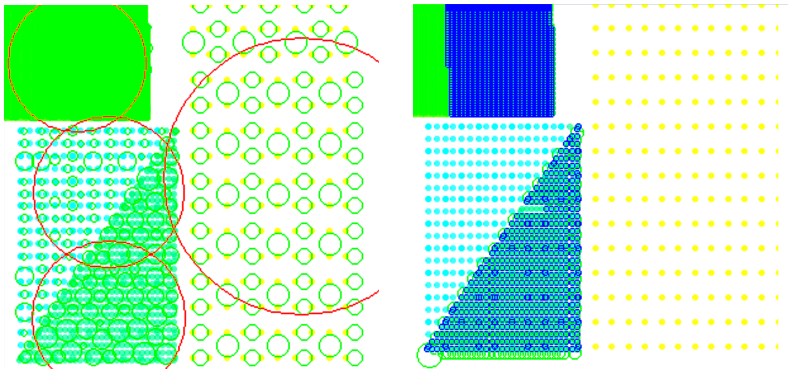


Figure 12: Clustering results from CluStream (left) and DenStream (right) (screenshot from MOA software [67].)

Algorithm	Purity	Precision	Recall	NumClusters
Clustream	0.83	1.0	0.82	4
Denstream	0.70	1.0	0.69	2
DyClee	1.0	1.0	1.0	4

Table 8: Clustering evaluation of streaming methods over the multi-density test case

are side by side, in highly overlapping situations. To do this, ***DyClee*** first finds all possible clusters, allowing multiple densities. Second, it finds the border of the clusters and then analyzes every μ-cluster in the border. Border μ-clusters are assigned to the connected cluster that has the most similar average density. In that way, cluster frontiers can be precisely drawn, which is key in highly overlapping distributions. To further illustrate this, a test case based on cluster distributions exhibiting different densities is shown in Figure 13 (left). ***DyClee*** clustering results are shown in Figure 13 (center) and DBSCAN results in Figure 13 (right) with the best possible configurations for the two algorithms (***DyClee*** `relative-size`=0.07, DBSCAN `eps` $= 0.15$). Glyphs are set according to the original distribution and color to the current clustering. Visual inspection is enough to assess that ***DyClee*** outperforms DBSCAN both in outlier rejection and sparse sample recognition.

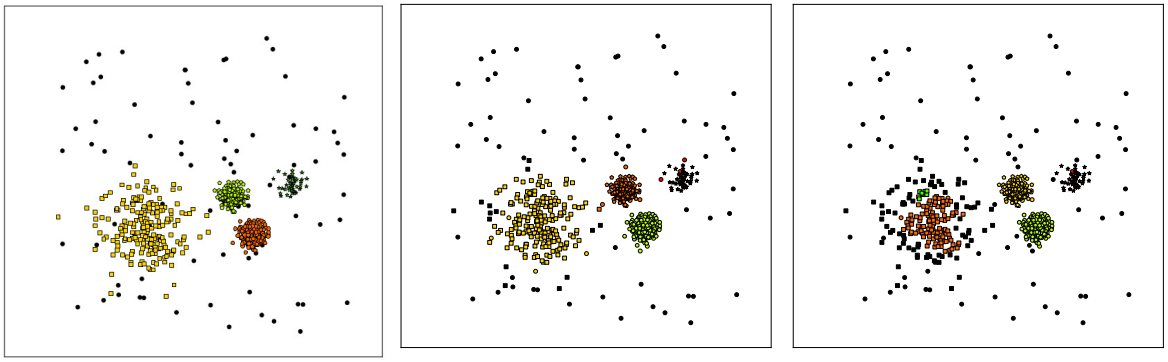


Figure 13: Multi-density test case (left), ***DyClee*** clustering results (center) and DBSCAN clustering results (right)

6.2.3. Robust path based clustering

Classical clustering approaches have powerful capabilities in modeling compact data. Nevertheless they mostly fail in detecting elongated structures. For this kind of challenge, connectivity based or graph based approaches such as [68] behave better but they are often very sensitive to outliers. Figure 14 shows the three spirals unnoisy distribution used in [32] to test what is called *robust path-based clustering*. In the center of each spiral, samples are more abundant and they become sparser as the spirals grow out. This kind of distribution is path based, which makes its clustering specially difficult for an algorithm only based on distance or only based on density. Even more, path based clustering methods found in the literature work over the entire data set which makes them unsuitable for real-time, large-data applications.

Since ***DyClee*** uses an incremental distance-based *and* density-based approach, it can overcome this kind of challenge, achieving results comparable to those of the original paper by Chang and Yeung [32] that uses the so-called robust path-based spectral clustering method. ***DyClee*** clustering results and the original results of [32] are shown in Figure 14. These results are achieved forcing ***DyClee*** to cluster all samples (option `Unclass_accepted=False`)(cf. Table 4).

Robust path-based clustering [32] gives a measure of the inter-point similarity arguably robust in presence of noise. The concentric spirals example was also considered with noise in [32] and the robust path-based clustering is unable to correctly label the noisy samples as outliers. The main advantage of ***DyClee*** over path-based methods like [32] is its ability to recognize outliers. ***DyClee*** does not only work under a unsupervised paradigm which does not

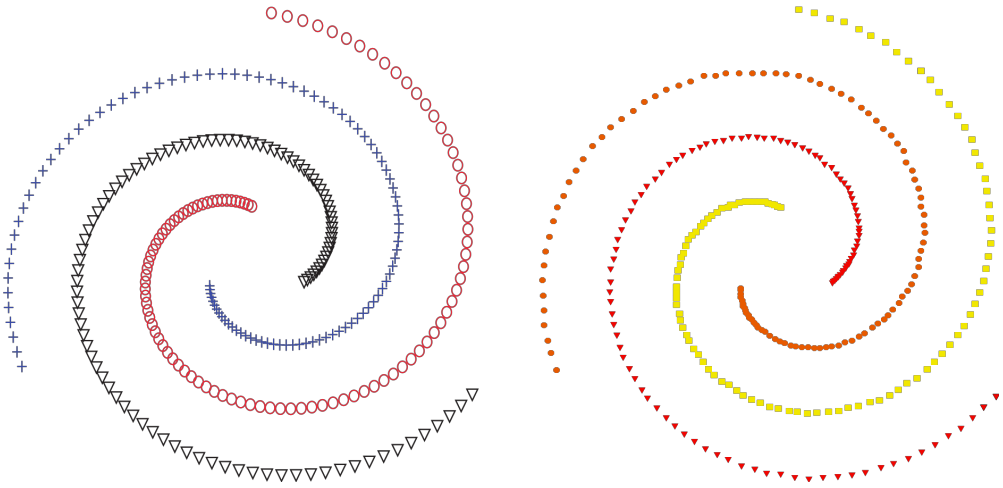


Figure 14: Chang and Yeung robust path-based spectral clustering (left) and ***DyClee*** clustering results (right) for test case in [32].

require a priori knowledge of the number of clusters, but it also excludes outlier samples from the clusters thanks to $\mathbb{O}\mu$-clusters.

The CluStream [37] and Denstream [39] algorithms were also confronted to this data set, setting the time horizon as the length of the samples, hence giving the same importance to all the samples. This scenario was also the test scenario for our algorithm. CluStream micro-clusters in Figure 15 (left) are depicted in green and the final clustering result in red. It can be seen that CluStream actually creates the micro-cluster representation for the samples but the final clusters mix samples of the three spirals. DenStream micro-clusters are depicted in green in Figure 15 (right) and the final clusters are represented changing points color to blue. DenStream does not mix elements of several spirals, but it creates 32 clusters putting apart samples of the same spiral. Table 9 summarizes the experimental results.

Algorithm	Purity	Precision	Recall	NumClusters
Clustream	0.43	1.0	0.82	3
Denstream	1.0	1.0	0.6	32
DyClee	1.0	1.0	1.0	3

Table 9: Clustering evaluation of streaming methods over the path-based test case

Using the three spirals distribution, the value of ***DyClee***'s `minimum_mc` optional parameter (cf. Table 5) can be illustrated. Figure 16 shows ***DyClee*** results with `minimum_mc` disabled (on the left) and enabled (on the right). As can be seen in the figures, the density of samples increases in the middle of the spirals leading the algorithm to group middle samples as belonging to a denser cluster (Figure 16 left). If the `minimum_mc` option is activated, no cluster including less than half of the average amount of μ-clusters is allowed. By default, the μ-clusters belonging to this unaccepted clusters are return as "unclassified".

Images in Figure 16 show several samples at the head of the spirals that are classified as outliers due to the difference in density with μ-clusters in the middle of the spiral. If the option `Unclass_accepted` is set to `False` (cf. Table 4), the unclassified samples are forced to belong to the closest cluster. The activation of this option is

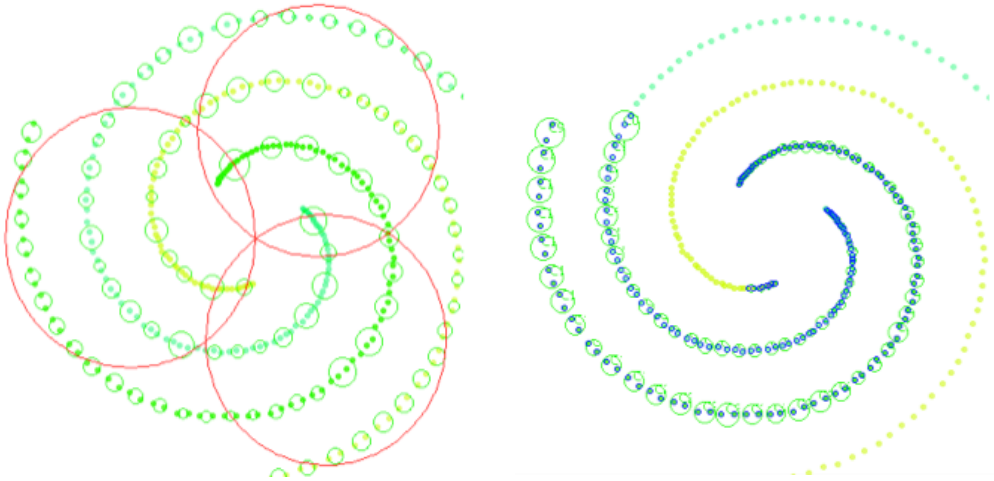


Figure 15: CluStream (left) and DenStream (right) tested against the three spirals distribution. Figures are screen shots from MOA software [67].

recommended when no noise is present in the data or when clustering all samples is mandatory. In the particular case of this dataset, the activation of the `Unclass_accepted` option leads to the clustering result shown on the right side of Figure 14, which is just the right result.

6.2.4. Clustering aggregation problem

The clustering aggregation problem was presented by Gionis *et al.* in [35]. This problem is defined as finding a single clustering that agrees as much as possible with the information gathered from other clusterings used as input. Clustering aggregation is claimed to be useful in improving the robustness of clustering. Figure 17 shows the test case used by Gionis *et al.* [35]. An intuitively good clustering for this data set consists of the seven perceptually distinct groups of samples. The authors of [35] ran five different clustering algorithms implemented in MATLAB (single linkage, complete linkage, average linkage, Ward's clustering, and k-means), setting the number of clusters to 7 in each case. Nevertheless, the correct clusters distribution was not found by any of the named algorithms. Gionis *et al.* [35] then proposed several solutions using clustering aggregation, in particular deriving a clustering distribution that merges the results of the five clustering algorithms used in the previous stage. Unfortunately the aggregation techniques they consider are NP-hard, which makes their implementation unfeasible for practical problems.

DyClee has properties that make it efficient for the test case presented in [35]. As can be seen in Figure 17, ***DyClee*** correctly clusters the test set, achieving almost the same result as Gionis *et al.* whereas Gionis *et al.* solution requires to aggregate the results of five different clustering methods. ***DyClee*** does not even require the number of clusters as input parameter.

Using the same data set one interesting aspect of ***DyClee*** can also be exemplified, which is its ability to reject clusters composed of few μ-clusters. If the clusters distribution is assumed to be relatively uniform, rejecting such clusters can improve outlier detection. When the user selects to enable the minimum μ-clusters cluster option but setting `minimum_mc=true` (cf. Table 5), clusters with few μ-clusters are not accepted and the corresponding samples are taken as noise/outliers. Figure 18 shows ***DyClee***'s clustering results with this option. As can be seen from the figure, this option causes sparse clusters to be eliminated and their samples identified as noise (black dots).

6.2.5. Clustering in highly overlapping situations

In order to illustrate ***DyClee***'s performance in presence of highly overlapping classes, we selected the R15 data set from [69]. This data set of 600 samples is generated by 15 similar 2D Gaussian distributions. ***DyClee*** deals with high overlapping data distributions by design. Its implementation of μ-clusters of different densities allows it

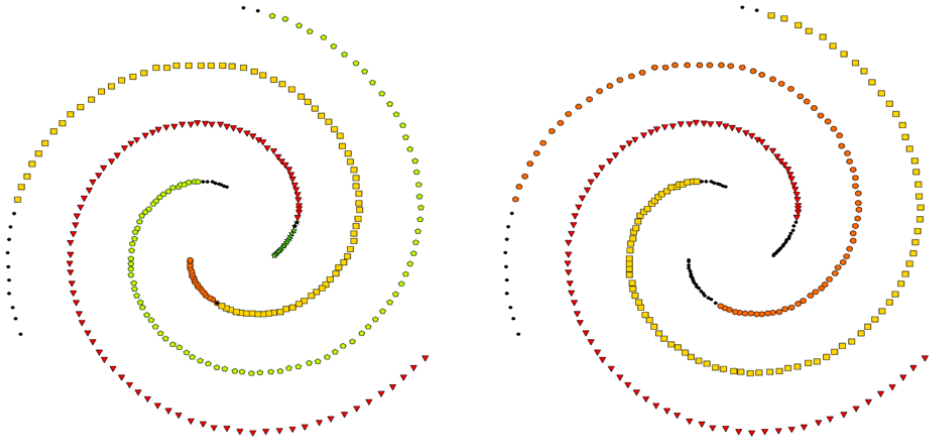


Figure 16: ***DyClee*** clustering results for test case in [32] with `minimum_mc` option set to `False` on the left and set to `True` on the right

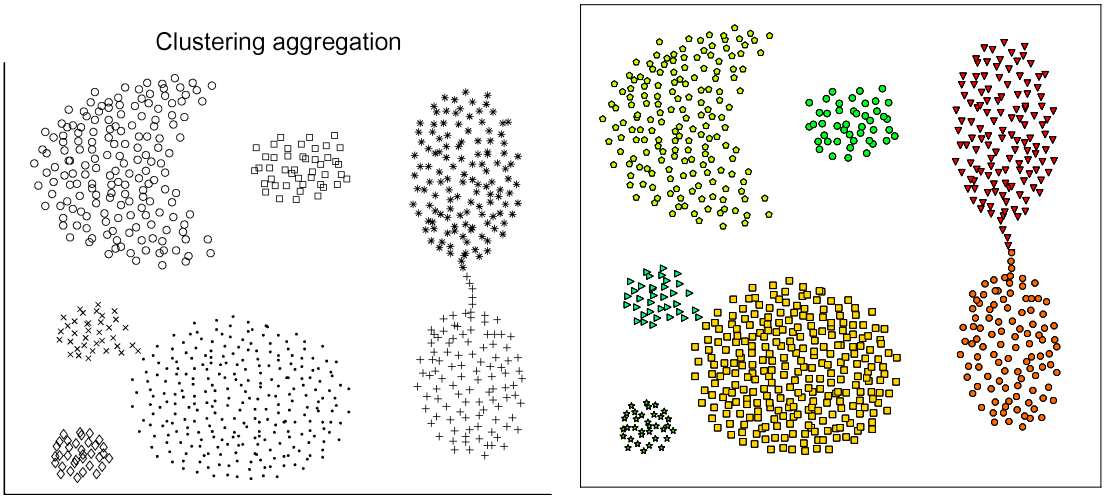


Figure 17: Gionis *et al.* clustering aggregation results (left) and ***DyClee*** clustering results for test case in [35].

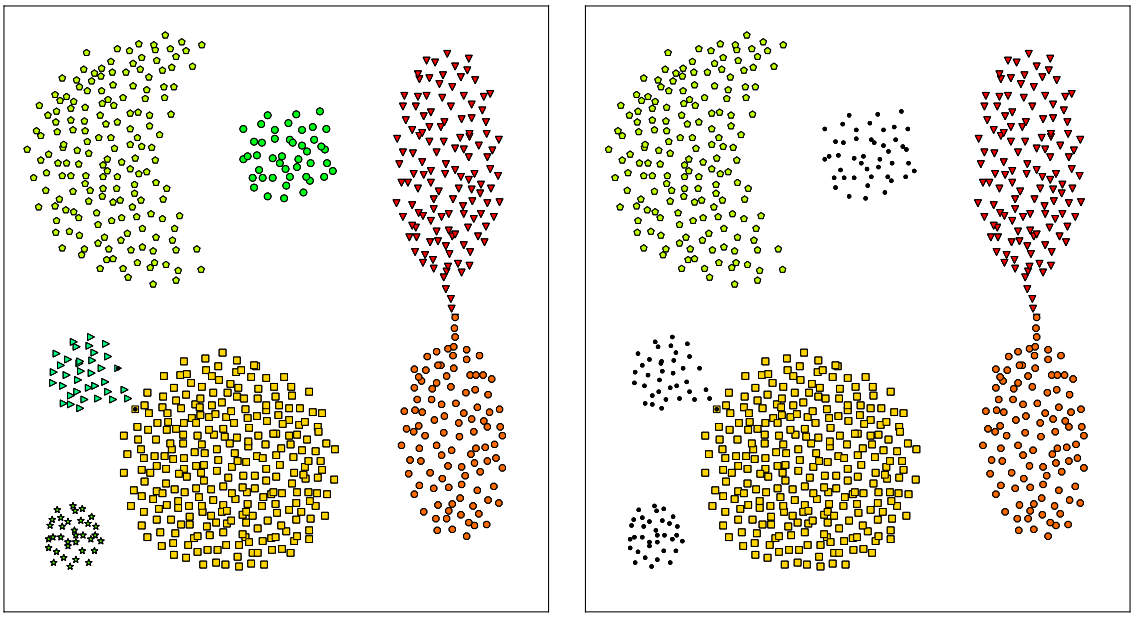


Figure 18: ***DyClee*** clustering results for test case in [35]. On the left the `minimum-mc` option is off, on the right it is on.

to detect low density regions between connected clusters which subsequently enables the detection of adjacent clusters. Figure 19 (right) shows ***DyClee***'s clustering results compared to those of the original paper (left) [69]. The 15 classes are correctly recognized and outliers are detected. For this test, ***DyClee*** was forced to cluster all samples (`Unclass_accepted`=False)(cf. Table 4). If the user wants to improve cluster characterization by including detection of non representative samples (outliers), he/she can turn off the `Unclass_accepted` parameter.

This test allows us to show ***DyClee***'s sensitivity to changes in the `relative-size` parameter, that is, the relative size of the μ-clusters. It is worth remembering that `relative-size` is the only mandatory parameter for running ***DyClee*** and it can take values from 0 to 1. Figures 20 and 21 show ***DyClee***'s clustering results when the μ-cluster relative size varies from 0.01 to 0.06 of the feature range. For this test we set `Unclass_accepted=true`, hence allowing the detection of outliers. Figures 20 and 21 show that ***DyClee*** can correctly cluster the data set using μ-cluster relative size from 0.03 up to 0.05. When the μ-clusters relative size is too low, values under 0.02, more clusters are created due to density changes inside the cluster, in other words regions of low relative density are found between regions of higher density, forming two clusters as can be seeing on the left and middle images of Figure 20. On the other hand, if the μ-cluster relative size is taken too large (as on the right of Figure 21) clusters can be incorrectly merged due to the proximity of μ-clusters of high density.

6.2.6. Clustering Chameleon data sets

Finally we consider the $t4.8k$ data set from the Chameleon data sets[3]. Chameleon is a hierarchical clustering algorithm developped by Karypis *et al.* [70]. The $t4.8k$ data set has six clusters of different size, shape, and orientation, as well as random noise samples and special artifacts such as streaks running across clusters making clustering particularly difficult. For this data set, Chameleon finds eleven clusters, out of which six correspond to the genuine clusters in the data set, and the rest contains outliers. Chameleon clustering results over this data set are the poorest over the five data sets proposed in [71] [4] and this test case was excluded from the final paper [70] reporting Chameleon. Nevertheless, this data set remains an interesting test case for clustering algorithms and has been extensively used for clustering validation [72, 66, 73, 74]. ***DyClee*** results for this test case are given in Figure 22, showing the correct

[3] Available at `http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download`

[4] The original technical report [71] which includes the test data set is available on the page of the author (`http://www-users.cs.umn.edu/han/dmclass/chameleon.pdf`).

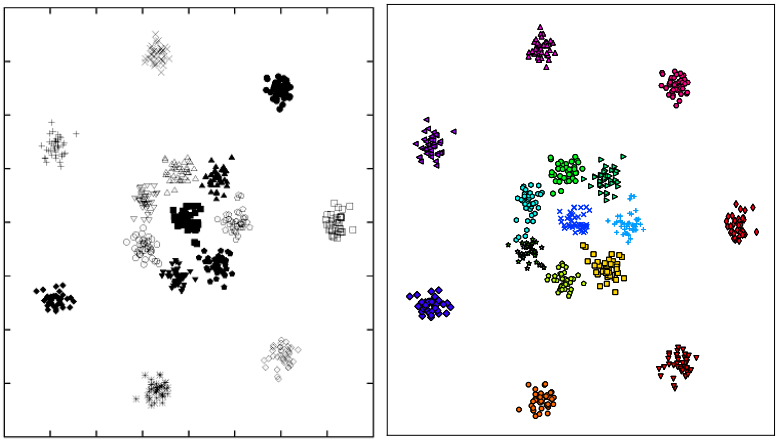


Figure 19: Veenman Maximum Variance Cluster Algorithm (left) and ***DyClee*** clustering results (right) for this test case [69].

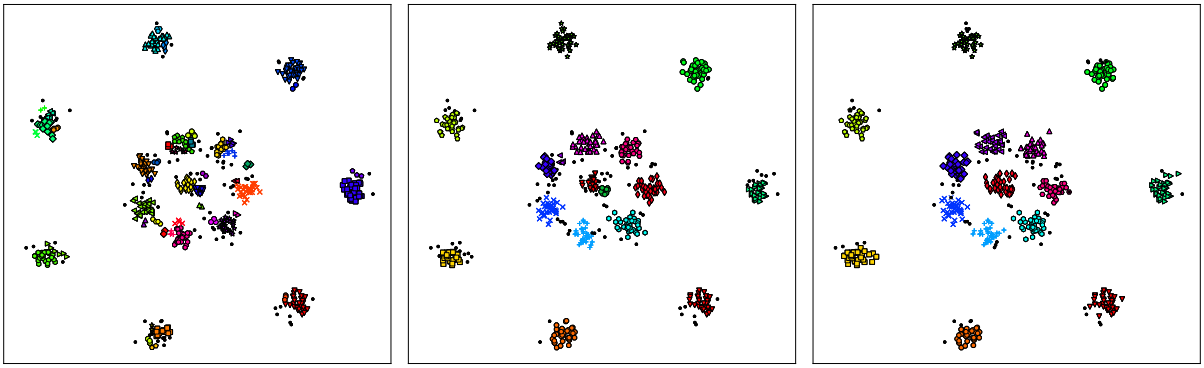


Figure 20: ***DyClee***'s clustering results. μ-cluster relative size of 0.01 (left), 0.02 (middle) and 0.03 (right)

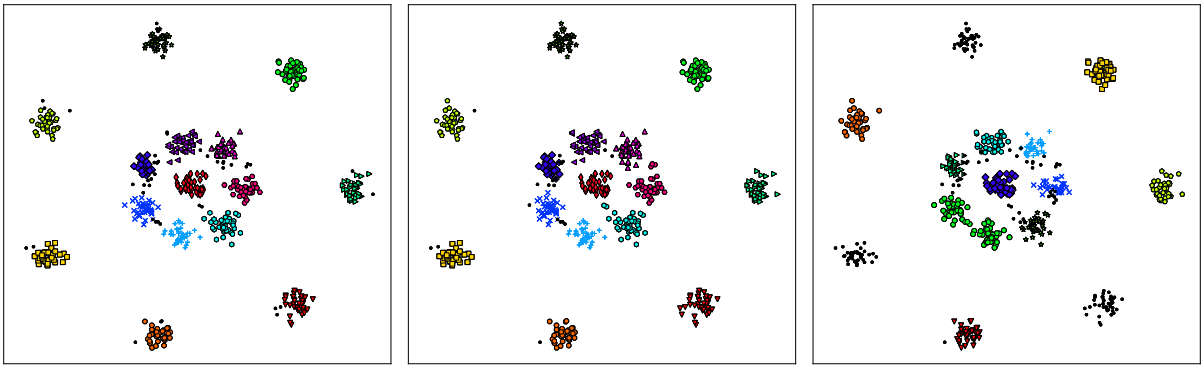


Figure 21: ***DyClee***'s clustering results. μ-cluster relative size of 0.04 (left), 0.05 (middle) and 0.06 (right)

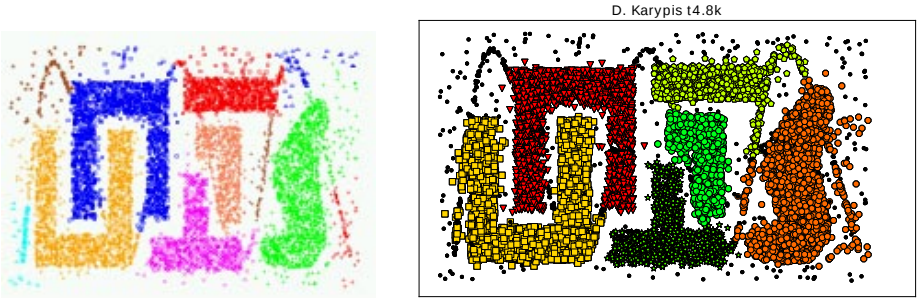


Figure 22: Chameleon (left) and ***DyClee*** clustering results (right) for the test cases in [71].

detection of six natural clusters, which correspond to the genuine clusters, but including some outlier samples in the natural clusters. ***DyClee*** results are hence quite good for this difficult clustering problem.

6.3. Concept drift problem

In the previous section, ***DyClee*** features were shown in static distributions that exhibit interesting challenges. Since in most of the real world applications, non-stationarity is typical, data is expected to evolve over time, that is, its underlying distribution can change.

In this section, algorithms are confronted with slowly time varying distributions, or as usually called in the online learning scenarios, with concept drift [7]. As explained in the introduction, concept drift refers to a subtle change of the data distribution and it can be categorized in two types: virtual and real concept drift.

In this test, algorithms are confronted with real concept drift. Three clearly differentiated distributions are used to form three clusters. Two of these distributions drift in time: synthetic data are generated changing the center of the two distributions each 100 samples. In consequence, the positions of the corresponding clusters significantly rotate, as can be seen in Figure 23 in which the x-axis corresponds to the algorithm number of iterations.

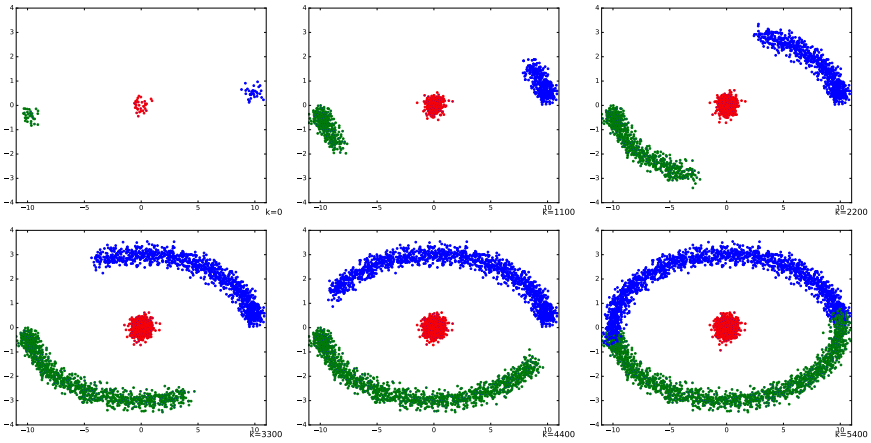


Figure 23: Concept Drift toy example – The x-axis corresponds to the algorithm number of iterations, start on top left and end on bottom right.

Dynamic data cause several clusterers to fail in finding clusters distribution since they cannot cope with evolution. Lets take as example the DBSCAN algorithm. In the considered test case, note that samples at $k = 5000$ in cluster 2 (green in Figure 23) take the position of samples at $k = 0$ in cluster 1 (blue in Figure 23). DBSCAN clustering results with `min_samples`= 5 and `eps`= 0.2 are shown on the left side of Figure 24. Since the DBSCAN implementation does not consider time evolution and takes all data samples as input independently of time, the two drifting clusters are merged in one single cluster.

An incremental implementation of DBSCAN was also tested with this data set, using the same values for `eps` and `min_points` that were used for the standard DBSCAN. The clustering results can be seen in the middle of Figure 24. Incremental DBSCAN finds out 27 different clusters showing that it is able to find the temporal change but it is unable to link this change with cluster evolution. We also tested the MiniBatchKMeans algorithm, an alternative online implementation of k-means that does incremental updates of the center positions using mini-batches. The number of clusters were given to the algorithm and also the size of the batch. MiniBatchKMeans results are shown on the right side of Figure 24. One of the cluster's distribution is time invariant (the cluster in the middle of the figure), however MiniBatchKMeans links the samples of this cluster to those of drifting cluster and it is unable to capture the clusters' evolution.

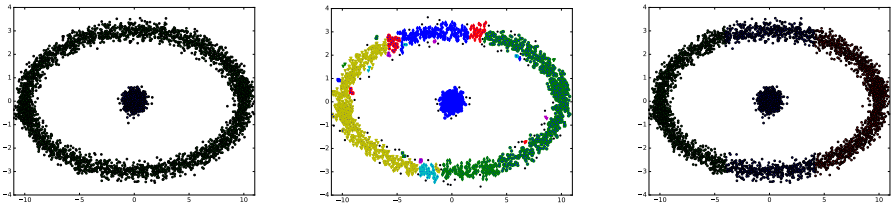


Figure 24: Clustering results on the concept drift example. Left DBSCAN results, middle Incremental DBSCAN, right MiniBatchKMeans results.

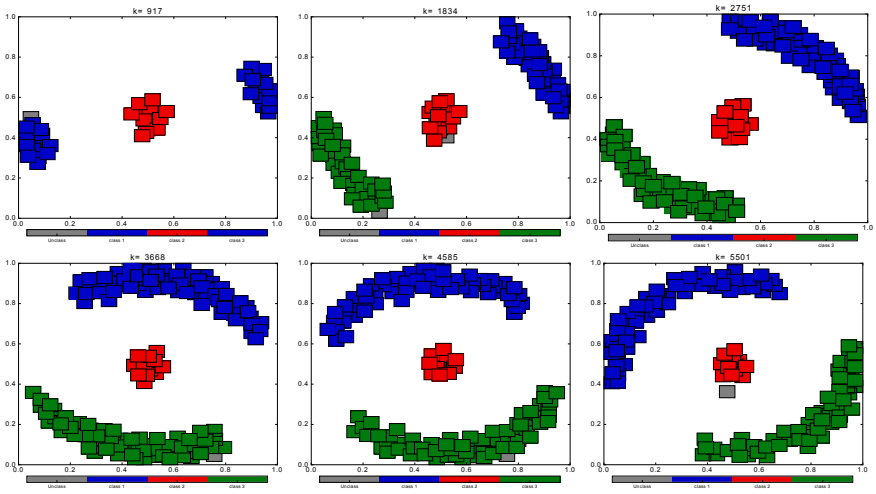


Figure 25: ***DyClee*** on the concept drift toy example – Chronological snapshots from top left to bottom right.

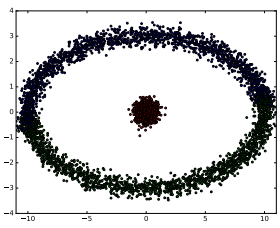


Figure 26: ***DyClee*** clustering results on the concept drift example.

DyClee's results are shown in Figures 25 and 26. Snapshots showing the distribution of μ-clusters at several time

instants are depicted in Figure 25, ordered from top left to bottom right. It can be seen how two of the clusters evolve thanks to the drift of some of the existent μ-clusters and to the creation of new μ-clusters. Growth in the amount of μ-clusters can be clearly seen between the snapshots on the top of the figure. $\mathbb{O}\mu$-clusters are represented as light gray boxes. To provide a visual comparison of ***DyClee***'s results with DBSCAN and MiniBatchKMeans results, the labelling of all the samples processed over time for this data set are shown in Figure 26. As opposed to incremental DBSCAN and MiniBatchKMeans which are presented as online batch implementations, ***DyClee*** can correctly follow the clusters evolution and face the (real) concept drift problem.

7. Experimental evaluation on real case studies

In this section, we consider three case studies and show the results of ***DyClee*** on real data sets.

7.1. Gasifier case study

A gasifier is a chemical reactor that heats and decomposes biomass into synthesis gas, a mixture of hydrogen (H2), carbon monoxide (CO), and carbon dioxide (CO2), in a restricted oxygen (O2) environment. The key reaction is pyrolysis, a combination of thermal decomposition and incomplete combustion, typically from air or pure oxygen. Gasifiers can operate on any carbon-containing feedstock, and are more typically used with coal or waste wood. A sketch of the gasifier is given in Figure 27.

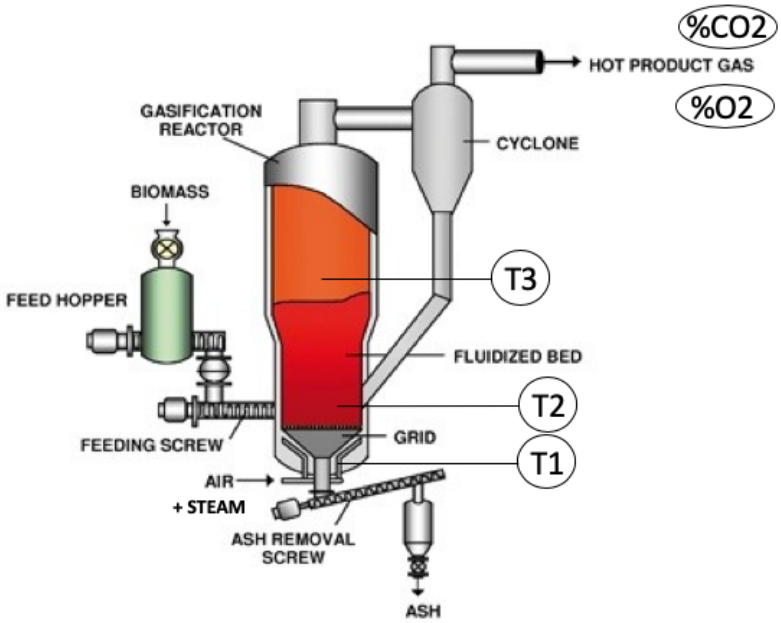


Figure 27: Gasifier

The fluidised bed reactor operates continuously at 930C and 1200hPa. Biomass is fed through an electronically feeding screw at the bottom of the reactor over the gas distributor. The gasifying agent, air plus steam, is fed at the reactor bottom side, allowing solid fluidisation. The inlet steam is produced by the addition of water with a membrane pump to a pre-heater. The desired product is obtained at the top outlet. Its composition is measured indirectly by on-line continuous monitoring of CO2 and O2 concentrations in the output gases. The gasifier is equipped with two pressure sensors, two flow-meters, four thermocouples and an on-line analyzer. It has also actuators for the heating system, the rotary screw and the pumps. The features available for clustering are temperature of steam in the inlet (Tl); temperature in the base of the reactor (T2); reactor temperature (T3), CO2 and O2 concentrations in the product gas.

The scenario that has been provided to ***DyClee*** is given in Figure 28 (top). Ground truth indicates that it includes four operating states: Normal operation (mainly represented by samples 110 to 380, also 430 to 540), High steam flow state (corresponding to the temperature peaks of samples around 70 and 420), No Solid Feed state (samples 540 to 590), and Shut-down state (samples 590 to 630). ***DyClee*** was set with the parameter `relative-size`=0,05, the default configuration was taken for all the other parameters but `Unclass_accepted` was set to false (cf. Table 4), which means that there was no outlier rejection.

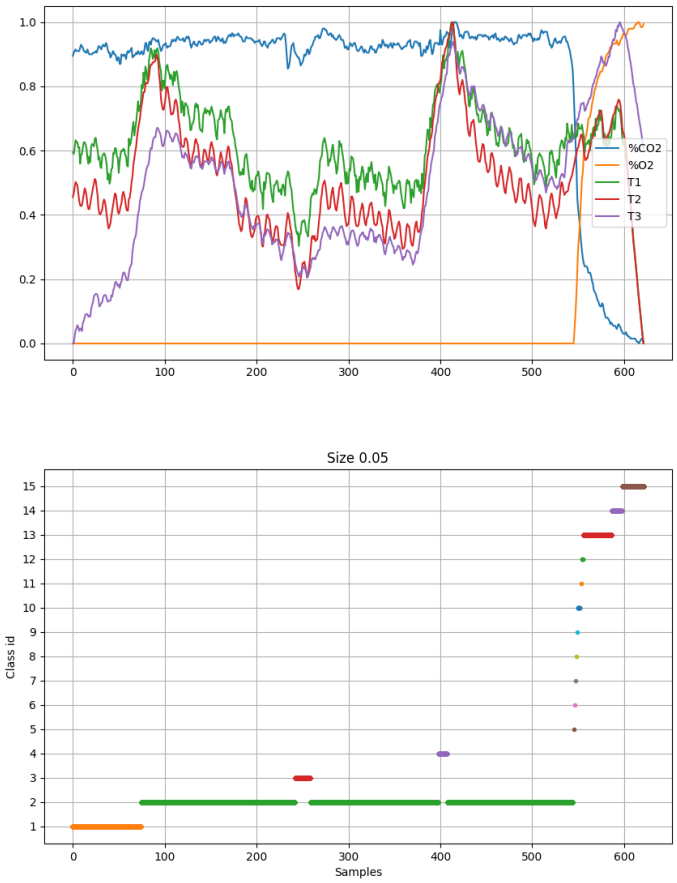


Figure 28: Gasifier normalized data scenario (top) and clustering from ***DyClee*** (bottom)

Cluster Id	**Identification**
Class 1	Normal operation in start up mode (temperature T3 in the reactor is not yet quite high)
Class 2	Normal operation
Class 3	Detection of combustion problem
Class 4	High steam flow state
Class 13	No solid feed state
Class 15	Shut down state
Classes 5 to 12 and 14	Transitory operating states

Table 10: Identification of the clusters provided by ***DyClee*** for the gasifier scenario

The clustering provided by ***DyClee*** is shown in Figure 28 (bottom). It identifies six main clusters as reported in

Table 10. It is consistent with the ground truth and it is even more precise since it makes the difference between normal state in start up mode and normal state at operation (Class 1). Also, it detects a combustion problem that results in the temperatures dropping down (Class 3).

7.2. Steam generation process case study

DyClee has also been applied to the data acquired from the boiler of a steam generation process illustrated in Figure 29.

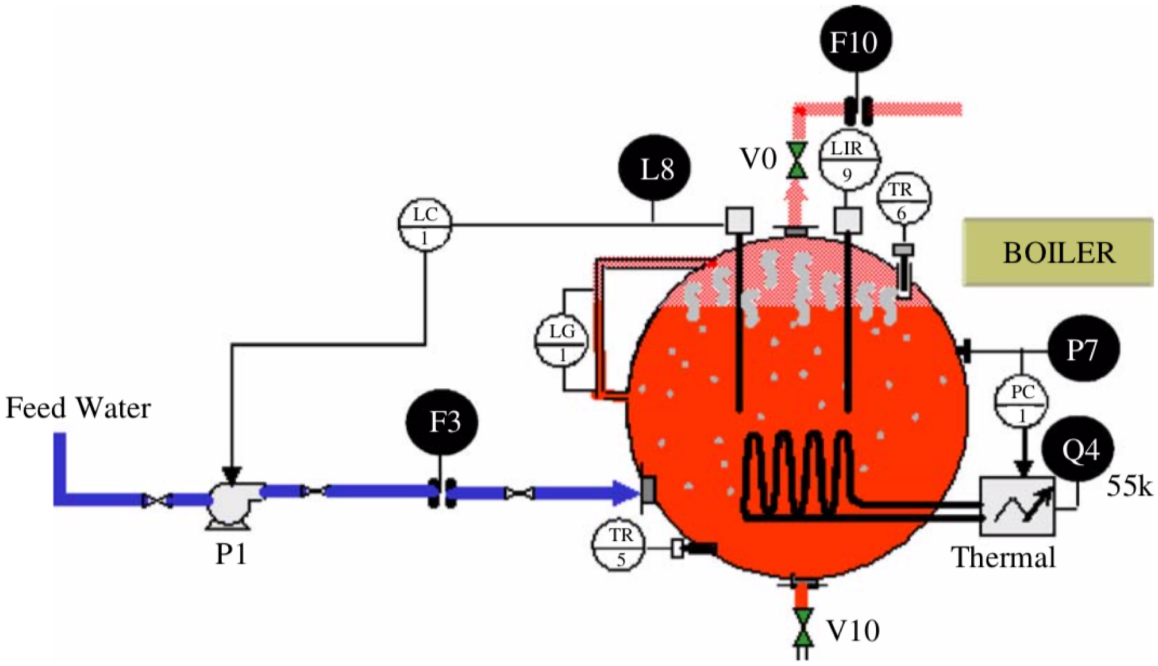


Figure 29: Steam generator boiler subsystem (from [75])

The boiler is heated by a thermal resistor. The feed water flow F3 is pumped to the boiler via the pump P1. To maintain a constant water level L8 in the boiler, an on-off controller acts on the pump. The heat power value Q4 depends on the pressure P7 in the steam accumulator. When the accumulator pressure drops below a minimal value, the resistor delivers maximum heating power, and when the accumulator reaches maximal pressure the resistor heating is cut off so that the pressure is maintained within $\pm$ 0.2 bar of the set-point. The generated steam flow is measured with sensor F10. The scenario that has been given to ***DyClee*** (cf. Figure 30 (top) is quite atypical. The pump and its controller suffer some fault resulting in a pumping problem : water is pumped into the system (around sample 100) but then drops to zero, which results in the boiler operating correctly but the level L8 not being maintained and dropping down to zero. Only then the pump controller reacts and water is injected into the boiler (around sample 1700) but the input water flow is cut almost immediately again. ***DyClee*** was set with the parameter `relative-size`=0,2, the default configuration was taken for all the other parameters but `Unclass_accepted` was set to false (cf. Table 4), which means that there was no outlier rejection, and a linear forgetting function was used (cf. Table 2).

The clustering provided by ***DyClee*** is shown in Figure 30 (bottom). It identifies four main clusters as reported in Table 11. The clustering is consistent with the behavior of the boiler. The water injection states are identified by Class 4. Class 1 and 2 alternate correctly detecting P7 pressure drops and intermittent heating power Q4. It should be noticed that, thanks to the forgetting function, Class 2 is identified over all the boiler normal operating period although the water level L8 is constantly decreasing.

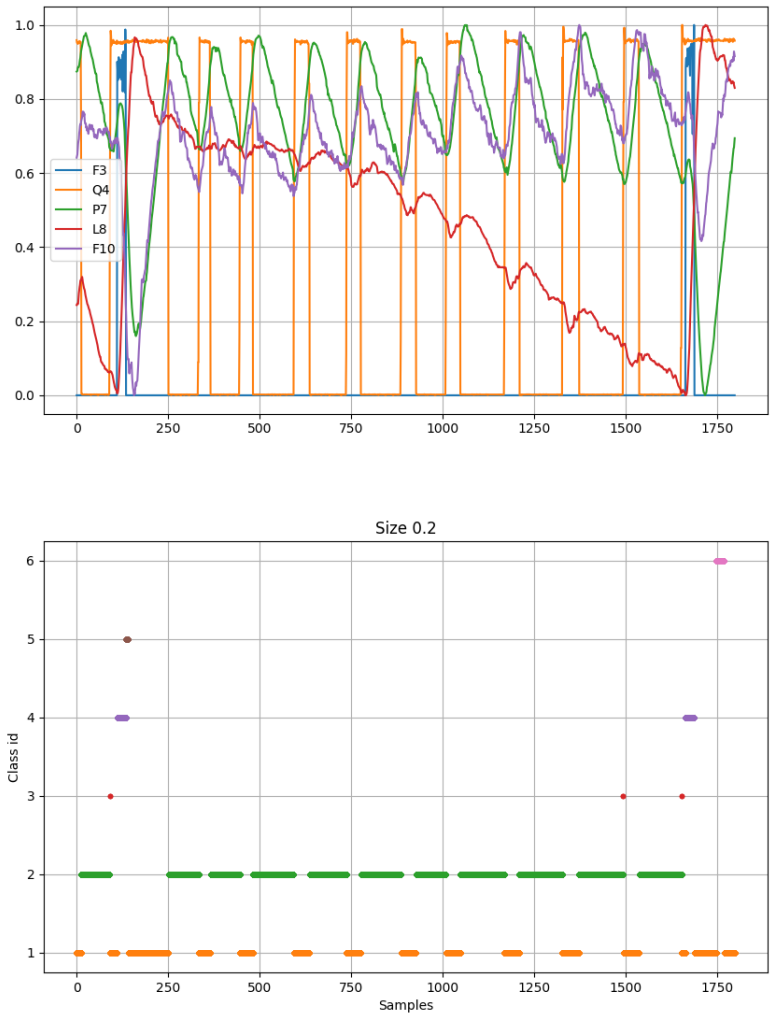


Figure 30: Boiler normalized data scenario (top) and clustering from ***DyClee*** (bottom)

Cluster Id	**Identification**
Class 1	Normal state heating off
Class 2	Normal state heating on
Class 4	Input water injection
Class 5 & 6	Activity recovery state
Classes 3	Transitory operating states

Table 11: Identification of the clusters provided by ***DyClee*** for the boiler scenario

8. Conclusions

In this paper, we have presented ***DyClee***, a fully **Dy**namic **Cl**ustering algorithm for tracking **E**volving **E**nvironments, that handles non-convex, multi-density clustering with outlier rejection even in highly overlapping situations. This algorithm includes multi-density clustering that allows to detect rare behaviors and implements a novelty detection procedure. The use of a two stages algorithm, distance-based and density-based, cycling at different frequencies allows ***DyClee*** to cluster data arriving at fast rates, since the heavy tasks of the density-based algorithm are not applied at the input data rate.

The clustering performance of ***DyClee*** has been tested on several test cases from the literature obtaining comparable results to those of specialized clustering systems in path based detection, clustering aggregation and non-convex cluster detection, hence proving that ***DyClee*** achieves a set of valuable properties that are not generally found together in the same algorithm. ***DyClee*** has also been tested on two real case studies in the engineering domain, a gasifier and the boiler subsystem of a steam generation process, for which it has demonstrate good performance at monitoring the operating state.

Future work includes to implement a post-processing stage that analyzes ***DyClee***'s results for better semantic characterization. Barbosa et al. [76] present some work in this direction, generating online a Markov chain model from ***DyClee*** clustering results. In addition, cluster drift is interesting to characterize system evolution and can provide important information. Another interesting work is to provide a feature selection algorithm targeting dynamic clustering and able to rank the available features on-line according to their discriminability power. The idea behind this work is to change not only the selected inputs to the classifier but also the number of inputs dynamically.

Acknowledgement

The authors are grateful to Ahmed Fahim for providing the multi-density dataset shown in Figure 10 and to Renaud Pons for helping with the experimental evaluation on real case studies. This work was partially supported by the Colombian administrative department of science, technology and innovation COLCIENCIAS.

References

[1] J. Schlimmer, J. Granger, RichardH., Incremental learning from noisy data, Machine Learning 1 (3) (1986) 317–354. doi:10.1007/BF00116895.

[2] S. Rajakarunakaran, P. Venkumar, D. Devaraj, K. S. P. Rao, Artificial neural network approach for fault detection in rotary system, Applied Soft Computing 8 (1) (2008) 740 – 748.

[3] V. Rajesh, V. N. Namboothiri, Flank wear detection of cutting tool inserts in turning operation: application of nonlinear time series analysis, Soft Computing 14 (9) (2010) 913–919.

[4] B. Kilundu, P. Dehombreux, X. Chiementin, Tool wear monitoring by machine learning techniques and singular spectrum analysis, Mechanical Systems and Signal Processing 25 (1) (2011) 400–415.

[5] C. Verde, J. Rojas, Iterative scheme for sequential leaks location, IFAC-PapersOnLine: Proceedings of the 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, Paris, France. 48 (21) (2015) 726–731.

[6] N. A. Barbosa, L. Travé-Massuyès, V. H. Grisales, Trend-based dynamic classification for on-line diagnosis of time-varying dynamic systems, IFAC-PapersOnLine: Proceedings of the 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, Paris, France. 48 (21) (2015) 1224–1231.

[7] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys (CSUR) 46 (4) (2014) 44.

[8] J. Gao, W. Fan, J. Han, On appropriate assumptions to mine data streams: Analysis and practice, in: Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on, IEEE, 2007, pp. 143–152.

[9] B. Zadrozny, Learning and evaluating classifiers under sample selection bias, in: Proceedings of the twenty-first international conference on Machine learning, ACM, 2004, p. 114.

[10] D. Cieslak, N. V. Chawla, et al., Detecting fractures in classifier performance, in: Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on, IEEE, 2007, pp. 123–132.

[11] A. Joentgen, L. Mikenina, R. Weber, H. Zimmermann, Dynamic fuzzy data analysis based on similarity between functions, Fuzzy Sets and Systems 105 (1) (1999) 81–90.

[12] V. Venkatasubramanian, R. Rengaswamy, S. Kavuri, K. Yin, A review of process fault detection and diagnosis. Part III: Process history based methods, Computers & Chemical Engineering 27 (3) (2003) 327–346.

[13] R. Sebastiao, J. Gama, A study on change detection methods, in: Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence, EPIA, 2009, pp. 12–15.

[14] L. Angstenberger, Dynamic fuzzy pattern recognition with applications to finance and engineering, Ph.D. thesis, Rheinisch-Westflische Technische Hochschule Aachen (2001).

[15] J. A. Guajardo, R. Weber, J. Miranda, A model updating strategy for predicting time series with seasonal patterns, Applied Soft Computing 10 (1) (2010) 276–283.
[16] M. Markou, S. Singh, Novelty detection: a review–part 1: statistical approaches and part 2: neural network based approaches, Signal processing 83 (12) (2003) 2481–2497.
[17] P. Angelov, X. Zhou, Evolving fuzzy-rule-based classifiers from data streams, Fuzzy Systems, IEEE Transactions on 16 (6) (2008) 1462–1475.
[18] N. A. Barbosa, L. Travé-Massuyès, V. H. Grisales, A novel algorithm for dynamic clustering: Properties and performance, in: Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on, IEEE, 2016, pp. 565–570.
[19] K. Li, F. Yao, R. Liu, An online clustering algorithm, in: Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on, Vol. 2, IEEE, 2011, pp. 1104–1108.
[20] Y. V. Bodyanskiy, O. K. Tyshchenko, D. S. Kopaliani, An evolving connectionist system for data stream fuzzy clustering and its online learning, Neurocomputing 262 (2017) 41–56.
[21] P. Angelov, Fuzzily connected multimodel systems evolving autonomously from data streams, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 41 (4) (2011) 898–910.
[22] M. Pratama, S. G. Anavatti, M. Joo, E. D. Lughofer, pclass: an effective classifier for streaming examples, IEEE Transactions on Fuzzy Systems 23 (2) (2015) 369–386.
[23] T. Kempowsky, A. Subias, J. Aguilar-Martín, Process situation assessment: From a fuzzy partition to a finite state machine, Engineering Applications of Artificial Intelligence 19 (5) (2006) 461–477.
[24] A. Bouchachia, Fuzzy classification in dynamic environments, Soft Computing 15 (5) (2011) 1009–1022.
[25] Y. V. Bodyanskiy, O. K. Tyshchenko, D. S. Kopaliani, Adaptive learning of an evolving cascade neo-fuzzy system in data stream mining tasks, Evolving Systems 7 (2) (2016) 107–116.
[26] A. Bouchachia, C. Vanaret, Incremental learning based on growing gaussian mixture models, in: Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on, Vol. 2, IEEE, Elsevier, 2011, pp. 47–52.
[27] C. Hu, Y. Chen, L. Hu, X. Peng, A novel random forests based class incremental learning method for activity recognition, Pattern Recognition 78 (2018) 277–290.
[28] S. R. Young, A. Davis, A. Mishtal, I. Arel, Hierarchical spatiotemporal feature extraction using recurrent online clustering, Pattern Recognition Letters 37 (2014) 115–123.
[29] L. Ulanova, N. Begum, M. Shokoohi-Yekta, E. Keogh, Clustering in the face of fast changing streams, in: Proceedings of the 2016 SIAM International Conference on Data Mining, SIAM, 2016, pp. 1–9.
[30] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (6191) (2014) 1492–1496.
[31] B. Fischer, T. Zller, J. Buhmann, Path based pairwise data clustering with application to texture segmentation, in: M. Figueiredo, J. Zerubia, A. Jain (Eds.), Energy Minimization Methods in Computer Vision and Pattern Recognition, Vol. 2134 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001, pp. 235–250. doi:10.1007/3-540-44745-8_16.
[32] H. Chang, D.-Y. Yeung, Robust path-based spectral clustering, Pattern Recognition 41 (1) (2008) 191–203.
[33] U. Von Luxburg, A tutorial on spectral clustering, Statistics and computing 17 (4) (2007) 395–416.
[34] P. Mitra, S. K. Pal, M. A. Siddiqi, Non-convex clustering using expectation maximization algorithm with rough set initialization, Pattern Recognition Letters 24 (6) (2003) 863 – 873. doi:http://dx.doi.org/10.1016/S0167-8655(02)00198-8.
[35] A. Gionis, H. Mannila, P. Tsaparas, Clustering aggregation, ACM Trans. Knowl. Discov. Data 1 (1). doi:10.1145/1217299.1217303.
[36] R. Hyde, P. Angelov, A. MacKenzie, Fully online clustering of evolving data streams into arbitrarily shaped clusters, Information Sciences 382 (2017) 96–114.
[37] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th international conference on Very large data bases-Volume 29, VLDB Endowment, 2003, pp. 81–92.
[38] P. Kranen, I. Assent, C. Baldauf, T. Seidl, The ClusTree: indexing micro-clusters for any stream mining, Knowledge and information systems 29 (2) (2011) 249–272.
[39] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: SIAM (Ed.), SIAM International Conference on Data Mining, 2006, pp. 328–339.
[40] G. A. Carpenter, S. Grossberg, Encyclopedia of machine learning, in: C. Sammut, G. I. Webb (Eds.), Encyclopedia of Machine Learning, Springer Science & Business Media, 2010, Ch. Adaptive resonance theory, pp. 22 – 35. doi:10.1007/978-0-387-30164-8_11.
[41] Y. Chen, L. Tu, Density-based clustering for real-time stream data, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and data mining, 2007, pp. 133–142.
[42] C. C. Aggarwal, A. Hinneburg, D. A. Keim, On the surprising behavior of distance metrics in high dimensional space, in: J. Van den Bussche, V. Vianu (Eds.), Database Theory - ICDT 2001, Vol. 1973 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2001, pp. 420–434. doi:10.1007/3-540-44503-X_27.
[43] Y. Rubner, J. Puzicha, C. Tomasi, J. M. Buhmann, Empirical evaluation of dissimilarity measures for color and texture, Computer Vision and Image Understanding 84 (1) (2001) 25 – 43. doi:http://dx.doi.org/10.1006/cviu.2001.0934.
[44] T. Zhang, R. Ramakrishnan, M. Livny, Birch: A new data clustering algorithm and its applications, Data Mining and Knowledge Discovery 1 (2) (1997) 141–182.
[45] G. Gutin, A. Yeo, A. Zverovich, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp, Discrete Applied Mathematics 117 (1-3) (2002) 81–86.
[46] K. M. Kumar, A. R. M. Reddy, A fast dbscan clustering algorithm by accelerating neighbor searching using groups method, Pattern Recognition 58 (2016) 39–48.
[47] Y. Fang, M. Friedman, G. Nair, M. Rys, A.-E. Schmid, Spatial indexing in microsoft sql server 2008, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 1207–1216. doi:10.1145/1376616.1376737.
[48] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The r*-tree: An efficient and robust access method for points and rectangles, in:

Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, SIGMOD '90, ACM, New York, NY, USA, 1990, pp. 322–331. doi:10.1145/93597.98741.

[49] S. M. Omohundro, Five balltree construction algorithms, Tech. rep., International Computer Science Institute Berkeley (1989).

[50] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.

[51] N. Barbosa Roa, Comparison of spatial indexes, Tech. Rep. 16631, LAAS–CNRS, 13p (2016). doi:hal-01701560.

[52] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: KDD, 1996, pp. 226–231.

[53] Z.-Y. Chen, Z.-P. Fan, M. Sun, A hierarchical multiple kernel support vector machine for customer churn prediction using longitudinal behavioral data, European Journal of Operational Research 223 (2) (2012) 461–472.

[54] C. U. Pyon, J. Y. Woo, S. C. Park, Service improvement by business process management using customer complaints in financial service industry, Expert Systems with Applications 38 (4) (2011) 3267 – 3279. doi:http://dx.doi.org/10.1016/j.eswa.2010.08.112.

[55] J. Mazel, P. Casas, Y. Labit, P. Owezarski, Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection, in: Proceedings of the 7th International Conference on Network and Services Management, International Federation for Information Processing, 2011, pp. 73–80.

[56] I. Koychev, Gradual forgetting for adaptation to concept drift, in: Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning, ECAI, 2000, pp. 101–106.

[57] D. Hawkins, M. Douglas, Identification of outliers, Vol. 11, Springer, 1980.

[58] A. Zimek, E. Schubert, H.-P. Kriegel, A survey on unsupervised outlier detection in high-dimensional numerical data, Statistical Analysis and Data Mining 5 (5) (2012) 363–387.

[59] E. M. Knorr, R. T. Ng, V. Tucakov, Distance-based outliers: algorithms and applications, The VLDB Journal?The International Journal on Very Large Data Bases 8 (3-4) (2000) 237–253.

[60] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander, Lof: Identifying density-based local outliers, SIGMOD Rec. 29 (2) (2000) 93–104. doi:10.1145/335191.335388.

[61] H.-P. Kriegel, M. S hubert, A. Zimek, Angle-based outlier detection in high-dimensional data, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08, ACM, New York, NY, USA, 2008, pp. 444–452. doi:10.1145/1401890.1401946.

[62] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, R. E. Tarjan, Time bounds for selection, J. Comput. Syst. Sci. 7 (4) (1973) 448–461.

[63] I. Wald, V. Havran, On building fast kd-trees for ray tracing, and on doing that in o (n log n), in: Interactive Ray Tracing 2006, IEEE Symposium on, IEEE, 2006, pp. 61–69.

[64] T. H. Cormen, Introduction to algorithms, MIT press, 2009.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[66] A. Fahim, A.-E. Salem, F. Torkey, M. Ramadan, G. Saake, et al., Scalable varied density clustering algorithm for large datasets, Journal of Software Engineering and Applications 3 (06) (2010) 593.

[67] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: massive online analysis, Journal of Machine Learning Research 11 (2010) 1601–1604.

[68] H. Zanghi, C. Ambroise, V. Miele, Fast online graph clustering via erdős–rényi mixture, Pattern recognition 41 (12) (2008) 3592–3599.

[69] C. J. Veenman, M. J. Reinders, E. Backer, A maximum variance cluster algorithm, Pattern Analysis and Machine Intelligence, IEEE Transactions on 24 (9) (2002) 1273–1280.

[70] G. Karypis, E.-H. Han, V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer 32 (8) (1999) 68–75.

[71] G. Karypis, E.-H. Han, V. Kumar, Chameleon: A hierarchical clustering algorithm using dynamic modeling, Tech. Rep. 99-007, University of Minnesota - Department of Computer Science and Engineering, 4-192 EECS Bldg., 200 Union St. SE Minneapolis, MN 55455, USA (1999). URL http://www-users.cs.umn.edu/~han/dmclass/chameleon.pdf

[72] B. Borah, D. K. Bhattacharyya, A clustering technique using density difference, in: 2007 International Conference on Signal Processing, Communications and Networking, 2007, pp. 585–588.

[73] V. A. Tran, O. Hirose, T. Saethang, L. A. T. Nguyen, X. T. Dang, T. K. T. Le, D. L. Ngo, M. Kubo, Y. Yamada, K. Satou, D-impact: A data preprocessing algorithm to improve the performance of clustering, Journal of Software Engineering and Applications 2014.

[74] V. V. Thang, D. V. Pantiukhin, A. I. Galushkin, A hybrid clustering algorithm: The fastdbscan, in: 2015 International Conference on Engineering and Telecommunication (EnT), 2015, pp. 69–74.

[75] T. Kempowsky, A. Subias, J. Aguilar-Martin, Process situation assessment: From a fuzzy partition to a finite state machine, Engineering Applications of Artificial Intelligence 19 (5) (2006) 461 – 477. doi:https://doi.org/10.1016/j.engappai.2005.12.012.

[76] N. A. Barbosa, L. Travé-Massuyès, V. H. Grisales, Diagnosability improvement of dynamic clustering through automatic learning of discrete event models, IFAC-PapersOnLine: Proceedings of the 20th IFAC World Congress, Toulouse, France. 50 (1) (2017) 1037–1042.

Nathalie
Barbosa Roa holds a position as Data Scientist and Engineer at Continental. She received the Electronic Engineering degree from Universidad Distrital Francisco José de Caldas in 2008, and the M.Sc. degree in Industrial Automation from the Universidad Nacional de Colombia in 2011. She receives her Ph.D. in Automation of Université Paul Sabatier (2016) and in Electrical Engineering from Universidad Nacional de Colombia (2017)

Louise Travé-Massuyès holds a position of Research Director at CNRS, France, working as part of the Diagnosis and Supervisory Control (DISCO) research team of the LAAS-CNRS laboratory in Toulouse, France. She received her Ph.D. in control from INSA in 1984, then an HDR from the University of Toulouse in 1998, France. Her research interests are in dynamic systems diagnosis and supervision with special focus on set-membership methods, hybrid systems, structural/qualitative/event based models and machine learning.

Victor
Hugo Grisales Palacio Professor in automatics at Universidad Nacional de Colombia, Dpt of Mechanical and Mechatronics Engineering. He received his Ph.D. in Automatic Control in 2007 from Université Paul Sabatier in Toulouse, France. His research interests are data science, education in engineering, computational intelligence and industrial automation.