TSF-DBSCAN: a Novel Fuzzy Density-based Approach for Clustering Unbounded Data Streams

Alessio Bechini, Francesco Marcelloni, *Member, IEEE,* Alessandro Renda

***Abstract*—In recent years, several clustering algorithms have been proposed with the aim of mining knowledge from streams of data generated at a high speed by a variety of hardware platforms and software applications. Among these algorithms, density-based approaches have proved to be particularly attractive, thanks to their capability of handling outliers and capturing clusters with arbitrary shapes. The streaming setting poses additional challenges that need to be addressed as well: data streams are potentially unbounded and affected by concept drift, i.e. a modification over time in the underlying data generation process. In this paper, we propose Temporal Streaming Fuzzy DBSCAN (TSF-DBSCAN), a novel fuzzy clustering algorithm for streaming data. TSF-DBSCAN is an extension of the well-known DBSCAN algorithm, one of the most popular density-based clustering approaches. Fuzziness is introduced in TSF-DBSCAN to model the uncertainty about the distance threshold that defines the neighborhood of an object. As a consequence, TSF-DBSCAN identifies clusters with fuzzy overlapping borders. A fading model, which makes objects less relevant as they become more remote in time, endows TSF-DBSCAN with the capability of adapting to evolving data streams. The integration of the model in a two-stage approach ensures computational and memory efficiency: during the online stage continuously arriving objects are organized in proper data structures that are later exploited in the offline stage to determine a fine-grained partition. An extensive experimental analysis on synthetic and real world datasets shows that TSF-DBSCAN yields competitive performance when compared to other clustering algorithms recently proposed for streaming data.**

***Index Terms*—Data stream clustering, fuzzy clustering, streaming data, density-based clustering, DBSCAN.**

I. Introduction

NOWADAYS, we are witnessing a wider and wider spread of applications that generate a huge amount of data at a very high speed. For instance, every minute roughly half a million tweets are posted on Twitter, and 4.5 millions searches are handled by Google[1]. *Data streams* are ubiquitous, and knowledge mining from such data is of the utmost importance in many application contexts.

The intrinsic nature of streaming scenarios poses several challenges in the adoption of clustering algorithms, as it has been highlighted in recent works [1], [2], [3], [4]. In particular, the sequence of objects may be *unbounded*. Further, the statistical properties of incoming data may be subject to *concept drift*, due to possibly unpredictable modifications of the underlying, non-stationary data generation process: therefore, the number and/or shape of clusters can change over time [5], [6]. Several taxonomies for concept drift have been recently proposed [1], [5], [6], and a coarse distinction can be made between *sudden* (or abrupt) and *incremental* concept drift. In the former, the underlying distribution switches from one concept to another, e.g. a blackout suddenly alters the measurements of a light sensor. In the latter, the underlying distribution gradually moves from a concept to another [6], exploring the intermediate concepts in between, e.g. the day-night alternation causes gradual and periodic fluctuations in the values recorded by a temperature sensor. For the purpose of this paper we refer to *incremental* concept drift.

Besides the streaming-related challenges, discovering clusters of arbitrary shapes and without prior knowledge of the number of clusters are certainly desirable requirements for a clustering algorithm. These requirements can be satisfied, both in *static* and *streaming* conditions, exploiting density-based approaches. Indeed, some clustering algorithms proposed in the streaming data context extend the well-known DBSCAN algorithm [7], one of the most popular density-based clustering approaches. Behaviour and performance of DBSCAN strongly depend upon the choice of two parameters: ε, which specifies the radius of the neighborhood of an object p, and $MinPts$, which defines the minimum number of objects in the neighborhood of p to elect p as a *core* object. The choice of the two parameters depends on the application and implicitly affects the number and shape of the discovered clusters [8].

DBSCAN cannot capture clusters with overlapping borders and/or variable density distributions [9], which are quite frequent in practical applications. To overcome this limitation, in [10] we have proposed SF-DBSCAN (Streaming Fuzzy DBSCAN) [10], a fuzzy version of DBSCAN for streaming data. SF-DBSCAN discovers arbitrarily-shaped clusters with fuzzy overlapping borders and, as an online approach, it updates the output partition at each newly collected object. SF-DBSCAN gives equal importance to *all* objects, neglecting to consider that the eldest objects might not represent the current concept. Actually the absence of a forgetting mechanism prevents the adaptation to evolving concepts and makes SF-DBSCAN unsuited for handling large volumes of streaming data [11]. Indeed, in realistic settings, the data stream is potentially unbounded, thus making practically impossible to store and use all the objects when determining clusters.

This paper proposes Temporal SF-DBSCAN (TSF-DBSCAN), which extends and improves SF-DBSCAN by introducing a forgetting mechanism by means of a damped window model, thus enabling adaptation to evolving scenarios and ensuring compliance with memory and computational constraints. Furthermore, differently from SF-DBSCAN, the fine-grained clustering procedure takes place in an offline on-

Alessio Bechini, Francesco Marcelloni and Alessandro Renda are with the Department of Information Engineering of the University of Pisa, Largo Lucio Lazzarino 1, 56123 Pisa, Italy. Alessandro Renda is also with the Department of Information Engineering of the University of Florence.

[1] https://www.internetlivestats.com/one-second/ Visited on Jan 03, 2020

Accepted version of the manuscript:

A. Bechini, F. Marcelloni and A. Renda, "TSF-DBSCAN: A Novel Fuzzy Density-Based Approach for Clustering Unbounded Data Streams," in IEEE Transactions on Fuzzy Systems, vol. 30, no. 3, pp. 623-637, March 2022, doi: 10.1109/TFUZZ.2020.3042645.

demand step: we propose the use of simple yet effective data structures for organizing data in the online stage, disengaging from the microcluster abstraction used in recent approaches to data stream clustering while still keeping computational efficiency. As a result, TSF-DBSCAN can efficiently manage potentially unbounded sequences of objects.

The main contributions of this work can be summarized as follows:

- TSF-DBSCAN integrates concepts of fuzzy set theory, forgetting mechanisms and density-based clustering for the analysis of data streams. To the best of our knowledge this combination has not been adequately investigated in streaming clustering literature;
- We formulate a simple yet effective strategy for clustering the most recent objects of the stream, without resorting to traditional summarization techniques (neither grids nor microclusters);
- An extensive experimentation is carried out on synthetic and real-world benchmark datasets to evaluate the performance of TSF-DBSCAN against several crisp and fuzzy streaming clustering algorithms.

The paper is organized as follows: Section II reports related works in the field of data stream clustering algorithms, distinguishing between two-stage and online approaches, and with a focus on fuzzy density-based algorithms. Section III introduces the algorithms that inspired TSF-DBSCAN, highlighting their pros and cons with respect to a set of desirable requirements for data stream clustering. Section IV describes in detail TSF-DBSCAN. In Section V we present the experimental investigation carried out to evaluate the modelling capability of TSF-DBSCAN on synthetic and real-world datasets. Section VI discusses the results, comparing them with those obtained by several recently proposed data stream clustering algorithms. Concluding remarks are drawn in Section VII.

II. Related Works

Two main computational approaches have been proposed in the literature for clustering data streams: we refer to these approaches as *two-stage clustering* and *online clustering*, respectively. In the former, the algorithm keeps track of the received data by calculating a data synopsis or summarization in the *online* stage, and then exploits such an aggregated estimate of local density for the evaluation of the actual partition in the *offline* clustering stage, to be executed on demand or periodically. In the latter, the partition is updated at *each* new occurrence of an object, as in [10].

Surveys on data stream clustering algorithms, challenges and applications have been presented in [1], [2], [3], [12], [13]. In this section we recall the most significant proposals in the context of density-based and fuzzy approaches for clustering data streams, including recent proposals.

A. Two-stage approaches

The first proposal of a summarization technique to handle very large datasets (and also streaming ones) is due to Zhang et al. [14], who introduced the *Clustering Feature (CF) vector*, i.e. a triplet that keeps summarized data information to support clustering operations, with no need to store *all* the objects. A CF vector includes the number, the linear sum and the square sum of the data objects in the cluster. Such an abstraction can be efficiently updated as new objects arrive, and conveniently used to calculate both cluster centroids and radii.

Aggarwal et al. [15] extended the original CF vector by including temporal information in the data summarization step. They proposed CluStream, a framework for clustering evolving data streams. CluStream includes the notions of *microcluster* in a two-stage approach for handling evolving data streams, and of *pyramidal time frame* to let users obtain a data partition over different time-horizons. Just like CF vectors, microclusters are data structures that summarize a set of objects from the stream: less memory is thus required, and the computational efficiency is improved. The macro-clustering process over the pyramidal time frame produces the global partition of data, allowing the user to capture the evolution of clusters over different time periods. The offline stage of CluStream is based on a variant of the k-means algorithm, and this determines two important drawbacks: the number of macroclusters must be specified in advance, and only spherical clusters can be captured.

DenStream [16] adapts the two-stage framework to the case of a density-based clustering algorithm. It is able to uncover arbitrary-shaped clusters without prior specification of the number of clusters. Furthermore, it handles potentially unbounded evolving data streams by adopting a damped window model. For the online summarization step, two types of microclusters are defined: potential-core microclusters (*p-mc*), and outlier microclusters (*o-mc*). For each new object o, o is absorbed in the closest *p-mc*, if the absorption does not increase the radius beyond a predefined threshold ε. Otherwise, o is absorbed in the closest *o-mc* or initializes a new *o-mc*. A *p-mc* differs from an *o-mc* because it exceeds a weight threshold, defined by two user defined parameters β and μ. The arrival of new objects and the weight decay strategy can clearly lead to changes of status for existing microclusters. Thus, periodically the weight of each *p-mc* is checked and a pruning strategy is applied to free the outlier buffer. In the offline stage the notion of density-reachability and density connectedness are adopted for microclusters, similarly to what happens in the classical DBSCAN algorithm [7].

A shared density graph has been recently proposed for better handling the low density regions between nearby microclusters [17]. Such an approach, named DBStream, differently from previous works, does not assume a Gaussian distribution of objects around a microcluster center. The notion of *shared density* is introduced as the weight of objects that lie in the intersection between two microclusters, divided by the area of the intersection itself. The shared density graph is a weighted graph with microclusters as vertices, and edges representing the shared density between pairs of microclusters. Similarly to DenStream, exponential fading is adopted for dealing with evolving data streams. Furthermore, weak microclusters and weak entries in the shared density graph are removed to speed up the computation. During the reclustering stage, two microclusters are assigned to the same macrocluster only if their shared density exceeds a predefined threshold. Thanks

to the definition of a connectivity graph, DBStream can also detect clusters of different density.

Grids represent an alternative to microclusters as a structure for incrementally summarizing the data stream. In grid-based methods the object space is discretized along each dimension, thus defining a grid structure. The clustering operation is performed on the grid cells, which are populated in the online stage with the stream of objects. A representative case of grid-based solutions is *D-Stream* [18], where the importance of each grid reflects the local density of the incoming flow of objects and varies according to the fading model applied to the stream. A later extension [19] deals with the issue of low density regions between adjacent grid cells, introducing also the concept of *attraction*.

MuDi-Stream [20] copes with the challenge of clustering multi-density data in the streaming settings, following the online-offline paradigm. In the online phase, it summarizes information on the data stream by leveraging a hybrid method based on both grids and microclusters; in the offline phase, it applies a variant of DBSCAN for macro-clustering. During the online phase, the evolving data stream is organized in the form of core-miniclusters. The proposed offline component does not require the definition of a constant distance threshold ε, but relies on the notion of local cluster density: intuitively, a core-minicluster is merged to an existing macro-cluster if it has a coherent value of mean distance from its core neighbors that are identified thanks to the grid partitioning of the space. In *MR-Stream* [21], grid cells are recursively halved along each dimension, resulting in a tree that allows discovering clusters at multiple resolutions. Furthermore, the authors propose a technique to trigger the offline stage at proper time, instead of periodically, thus improving the overall performance.

B. Online approaches

The first proposal for an online extension of DBSCAN for dynamic data warehousing environments [22] required a complete partition update whenever an object was moved to/from the dataset.

The CODAS (Clustering Online Data-streams into Arbitrary Shapes) algorithm [23] exploits the microcluster abstraction to perform online clustering. At the occurrence of a new object, data synopses are adjusted, and the overall partition is evaluated by checking overlapping microclusters. Thus, the global cluster membership of the new object can be immediately assessed. Nevertheless, CODAS does not implement any forgetting mechanism and indeed is unsuitable for handling *evolving* data streams. The CEDAS (Clustering Evolving Data-streams into Arbitrary Shapes) algorithm [24] overcomes such limitation by introducing a time-decaying *energy* property for microclusters, thus allowing discarding objects with low energy. A graph structure for representing adjacency of microclusters is exploited to efficiently update the overall partition when microclusters are deleted.

Recently, BOCEDS (Buffer-based Online Clustering for Evolving Data Streams) [25] was proposed as an improvement over the CEDAS algorithm. In addition to adapt the radius of each microcluster, a buffer for temporarily irrelevant microclusters is employed and the energy function is redefined including spatial information.

C. Fuzzy density-based approaches

Fuzzy extensions of density-based clustering algorithms have proven to be very effective in capturing clusters with overlapping borders and/or variable density distributions, and dealing with data streams subject to concept drift [9], [10], [26], [27]. Three possible fuzzy extensions of DBSCAN have been recently proposed [9], differing in the way fuzziness is applied: (i) the *FuzzyCore* version provides a definition of the core objects that employs a soft constraint on the minimum number $MinPts$ of objects in the neighborhood. This solution allows determining clusters with a variable density of core points; (ii) the *FuzzyBorder* version generates clusters with fuzzy borders by introducing a soft constraint on the neighborhood radius ε; (iii) the *FuzzyDBSCAN* version produces clusters with both fuzzy cores and fuzzy borders. Notably, these extensions do not target data streams, but require the whole dataset available offline.

Recently, we proposed SF-DBSCAN [10], an online adaptation to streaming data of the *FuzzyBorder* version mentioned above. Albeit showing superior performance compared to a crisp counterpart, SF-DBSCAN does not take advantage of any aging mechanism, thus being unable to handle evolving and/or unbounded data streams.

The recently proposed d-FuzzStream algorithm [26] extends the traditional offline-online framework by incorporating concepts from fuzzy set theory. During the online step examples are summarized in fuzzy structures, i.e. fuzzy microclusters (*f-mc*s), which differ from crisp microclusters as each object may belong to multiple microclusters with a different membership degree. Compared to the former FuzzStream proposal [28], d-FuzzStream introduces the notions of fuzzy dispersion and similarity of microclusters (adaptations of microcluster radius and mutual distance, respectively) in order to reduce the summarization complexity and to promote absorption of examples and merging of *f-mc*s. d-FuzzStream poses a limit on the number of *f-mc*s kept in memory: whenever such limit is exceeded, the least recently updated *f-mc*s are deleted, thus ensuring both memory compliance and adaptation to evolving settings. The offline step organizes *f-mc*s in fuzzy macroclusters through a weighted fuzzy c-means (WFCM) procedure [29]. Each *f-mc* is represented as a virtual point in the original space, located in the *f-mc* center and weighted by the sum of the membership degrees of the relative examples. WFCM requires to specify the expected number of macroclusters; the *f-mc*s with the highest weights are used to initialize the cluster prototypes. For the purpose of clustering evaluation, each original object is mapped onto its closest microcluster. Similarly, an alternative formulation of fuzzy microclusters has been proposed in [27], maintaining the online-offline approach.

Although most of the works presented in this section exploit the abstraction of microclusters as a synopsis over the received data, they often neglect an issue related to the adoption of microclusters, which has been highlighted in previous works [30] [31]: in order to ensure memory efficiency, an object is

discarded after its assignment to a microcluster, thus making only possible to infer the mapping between *microclusters* and *macroclusters* during the reclustering stage. The fine-grained partition over original objects can only be obtained if the mapping between *objects* and *microclusters* is also maintained.

III. Background on baseline algorithms

In this section we describe the rationale behind the development of the proposed TSF-DBSCAN algorithm. In particular we briefly recall the algorithms that inspired our solution along with their benefits and limitations.

The very forerunner of the algorithms that led to developing TSF-DBSCAN is DBSCAN, proposed by Ester et al. [7]: it determines clusters by inspecting the neighborhood of each generic object o. If the neighborhood of o, identified by the radius ε, contains at least $MinPts$ objects, o is marked as a *core object*. An object y that is not a core object but belongs to the ε-neighborhood of some core object o is denoted as *border object*. Finally, an object that is neither a core nor a border object is identified as a *noise object* or an *outlier*. DBSCAN determines clusters by searching for density connected objects. Two objects o_1 and o_2 are density connected if there exists a core object o_3 such that both o_1 and o_2 are density reachable from o_3, that is, there exists a set of core objects leading from o_3 to o_1 and from o_3 to o_2. DBSCAN determines clusters of arbitrary shape and detects outliers. However, the clustering result strongly depends on the input parameters ε and $MinPts$.

Recently, we have proposed S-DBSCAN [10], an extension of the DBSCAN algorithm aimed at managing data streams, and inspired by [22]. The partition is updated at the arrival of each new object p by properly handling one out of the following scenarios: i) p is labeled as an outlier, ii) p joins an existing cluster, iii) p determines the merging of two existing clusters, iv) p initiates a new cluster.

In the same paper [10], we have also proposed Streaming Fuzzy DBSCAN (SF-DBSCAN), which represents the first fuzzy version of DBSCAN for data stream clustering. The online functionality of S-DBSCAN is preserved. Unlike S-DBSCAN, however, SF-DBSCAN exploits fuzzy borders by relaxing the constraint on the neighborhood size, thus allowing marginal objects of data distribution to be included into fuzzy overlapping borders of clusters. According to the terminology usually adopted for microclusters [24], [25], given an object p, we define the region at a distance from p lower than a threshold ε_{min} as *kernel* region, and the region at a distance from p lower than a threshold ε_{max} but higher than ε_{min} as *shell* region (see Fig. 1). In SF-DBSCAN, the membership of an object to the neighborhood of p when it lies in its shell region decreases linearly with the distance from p, as shown in Fig. 2. As in the first proposal of *Fuzzy Border* [9], only objects within the kernel region are considered for the definition of a *core* object. However, an object belongs to a cluster with a non-zero membership even if it lies in the shell region. Experimental comparisons [10] indicate that SF-DBSCAN is characterized by a higher modelling capability than S-DBSCAN thanks to the introduction of fuzzy borders. Since all data are kept in memory, however, the sequence of objects to analyze has necessarily to be bounded, and the capability to handle non-stationary data streams is compromised.

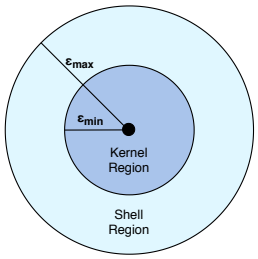


Fig. 1. The neighborhood of an object is assumed to be composed of two distinct areas: the *kernel* region, at a distance lower than ε_{min}, and the *shell* region, at a distance lower than ε_{max} but higher than ε_{min}.

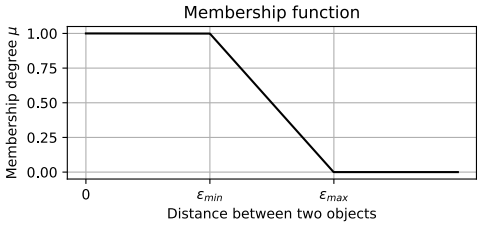


Fig. 2. The function that defines the membership degree of an object to the neighborhood of another object in SF-DBSCAN.

TABLE I
SIGNIFICANT REQUIREMENTS FOR STREAMING DATA CLUSTERING: SUPPORT BY DBSCAN AND SOME OF ITS EXTENSIONS (S-DBSCAN, SF-DBSCAN, AND THE PROPOSED TSF-DBSCAN)

	DBSCAN	S-DBSCAN	SF-DBSCAN	TSF-DBSCAN
Detection of arbitrarily shaped clusters	✓	✓	✓	✓
Number of clusters not required as input parameter	✓	✓	✓	✓
Ability to update clustering sequentially		✓	✓	✓
Enhanced modelling capability due to fuzzy borders			✓	✓
Ability to deal with unbounded data streams				✓
Ability to deal with concept drift				✓

The main features of the three algorithms are summarised in Table I. To better illustrate the different behaviors of S-DBSCAN, SF-DBSCAN and TSF-DBSCAN, we also report the partitions obtained by the three algorithms on two artificially generated scenarios. In both cases the number of objects is limited so that it can be managed by all algorithms, and input data are defined on a bidimensional space to enable a visual comparison of the output partitions. In DS1 [32], 800 objects from four distinct Gaussian distributions evolve over time, shifting clusters at a constant pace towards the center of the bidimensional space. DS2 is composed by two clusters, each with 2500 samples originating from a Gaussian distribution. The centers of the two clusters are initially close to each other

but they separate over time by moving in opposite directions. Indeed, this scenario depicts the phenomenon in which a single cluster splits into two separate clusters. For each dataset we plot the partition obtained by each algorithm after analyzing 25%, 50%, 75% and 100% of the total number of objects. Results are shown in Fig. 3 and Fig. 4.

The four clusters in DS1 are captured by all the algorithms. As highlighted in [10], S-DBSCAN does not model correctly the marginal objects of the distribution with the adopted parameter configuration. On the other hand, increasing the value of the ε parameter to reduce the number of outliers would result in the improper unification of the clusters in the final stage of the analysis, when the four distributions are close to each other in the center of the attribute space. Thanks to the introduction of fuzziness, both SF-DBSCAN and TSF-DBSCAN model correctly the border objects, which are rendered with a transparency inversely proportional to the highest membership degree. S-DBSCAN and SF-DBSCAN keep track of all the points, thus generating *elongated* clusters when the mean value of the distributions drift toward the center of the bidimensional space. On the other hand, the fading mechanism endows TSF-DBSCAN with the capability of adapting to an evolving scenario by progressively removing the oldest objects and leading to the detection of *moving* clusters. The adaptation capability of TSF-DBSCAN is even more evident in the analysis of DS2. The scenario with a single cluster that splits into two distinct clusters cannot be modelled by S-DBSCAN and SF-DBSCAN. Since the two distributions originate from the same point in the space, they will never be distinguished by an algorithm that considers *all* the objects to update the partition. The properly tuned fading mechanism allows TSF-DBSCAN to correctly distinguish the two drifting distributions at the end of the stream.

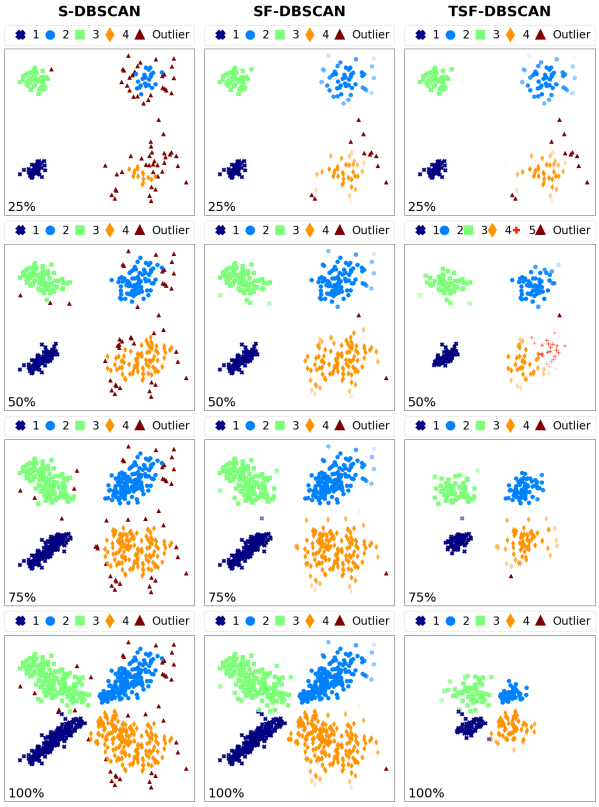


Fig. 3. DS1: partitions obtained by S-DBSCAN, SF-DBSCAN and TSF-DBSCAN after analyzing 25%, 50%, 75% and 100% of the total number of objects.

IV. The Temporal Fuzzy Streaming DBSCAN

The proposed TSF-DBSCAN inherits the basic characteristics from SF-DBSCAN but differs from it in two key aspects. First, TSF-DBSCAN employs a two-stage approach, similarly to the paradigm proposed in [15] and adopted in many recent proposals. Second, time and space efficiency are guaranteed by the adoption of a damped window model. This model associates each object with a weight that decreases over time: objects whose weight is lower than a given threshold are not taken into account any more, so they can be removed from memory. Hence, TSF-DBSCAN is able to handle potentially unbounded sequences of objects, also adapting over time to changes in the data generation process. Notably, no summarization technique is used in the online stage. A fine-grained analysis is carried out by updating, for each not-expired object, the set of its *kernel*-neighbors (those within ε_{min}) and the set of its *shell*-neighbors (those between ε_{min} and ε_{max}).

The on-demand offline clustering procedure rebuilds the partition from scratch in an efficient way, relying on the neighborhood sets updated during the online stage. Clusters are generated by exploiting the concept of density-connectedness. To take the weights into consideration, the definition of core object is properly adapted. An object is a core object if the sum of the weights of the objects in its *kernel*-neighborhood is higher than a pre-fixed threshold. An object that does not satisfy the core condition but lies in the neighborhood of one or more core objects, is assigned to its or their fuzzy neighborhood, respectively. The membership degree of an object to the neighborhood of another object is evaluated according to the function depicted in Fig. 2. The membership degree of a border object to a cluster is determined by its closest core object: such a definition allows obtaining clusters with fuzzy overlapping borders.

The main steps of the proposed approach for the online and offline components are represented in Fig. 5. The pseudo-code is reported in algorithms 1-4; a detailed explanation of the algorithms is given in the following subsections.

A. Data structures and parameter initialization

TSF-DBSCAN stores the most recent objects in the data structure *plist*, which is continuously updated by adding every newly arrived object and periodically pruned, during the offline stage, according to the forgetting mechanism. Each object p of the stream is characterised by the following attributes:

- t: the timestamp of arrival. For the purposes of this paper it can be considered as an incremental index;

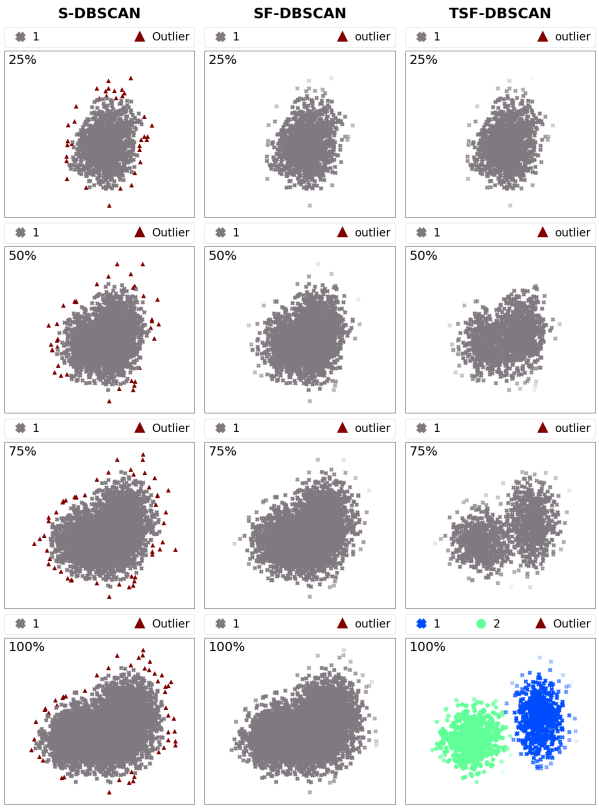


Fig. 4. DS2: partitions obtained by S-DBSCAN, SF-DBSCAN and TSF-DBSCAN after analyzing 25%, 50%, 75% and 100% of the total number of objects.

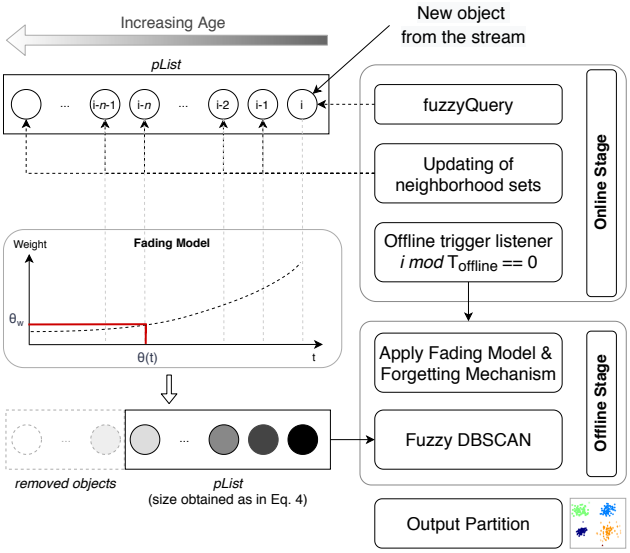


Fig. 5. The main steps of TSF-DBSCAN. The data structure *plist* stores the most recent objects. Each object p in *plist* is associated with two sets, with objects in its kernel-neighborhood and in its shell-neighborhood, respectively.

Algorithm 1 TSF-DBSCAN(p,$plist$)

Given: p - new object
Given: $pList$ - list of valid arrived objects, not yet expired
p.kernel, p.shell = fuzzyQuery($\varepsilon_{min}, \varepsilon_{max}, p, pList$)
for k in p.kernel **do**
  k.kernel = k.kernel $\cup \{p\}$
for s in p.shell **do**
  s.shell = s.shell $\cup \{p\}$
add p to $pList$
if $p.t$ mod $T_{offline} == 0$ **then**
  offline-FDBSCAN($plist$,$p.t$)

Algorithm 2 offline-FDBSCAN($pList$,t)

Given: $pList$ - list of valid arrived objects, not yet expired
Given: t - current timestamp, depending on the last arrived object
$\theta = t + \log_2 \theta_w/\alpha$
for $q \in pList$ **do**
  if $q.t < \theta$ **then**
    $pList = pList \setminus \{q\}$
  else
    label q as outlier
    re-initialize q.memberships as empty
visited = $\emptyset$
for $p \in pList$ **do**
  to_visit = $\emptyset$
  first = True
  if $p \notin$ visited **then**
    to_visit = to_visit $\cup \{p\}$
    for $q \in$ to_visit **do**
      if evaluateCore(q,t) **then**
        if $q \notin$ visited **then**
          visited = visited $\cup \{q\}$
          if first == True **then**
            C = nextCluster
            first = False
          add q to C as core point
          to_visit = to_visit $\cup$ q.kernel
          for s in q.kernel $\cup$ q.shell **do**
            if $\neg$ isCore(s) **then**
              fuzzyBorderUpdate(s,q)

- *kernel*: the set of objects that belong to the *kernel*-neighborhood of p, i.e. that lie at a distance lower than, or equal to, ε_{min} from p;
- *shell*: the set of objects that belong to the *shell*-neighborhood of p, i.e. that lie at a distance between ε_{min} and ε_{max} (included) from p;
- *weight*: the weight of the object, which is set to 1 at the arrival time and decreases over time according to the window model. Details about the adopted damped window model are given in Section IV-C1;
- *memberships*: a data structure that associates the membership degree of a border object p with each of the close clusters. As an example, given a set of three clusters $C1$,

Algorithm 3 evaluateCore(p,t)

Given: p - candidate core object
Given: t - actual timestamp, depending on the last arrived object

```
weightSum = 2^{-α*(p.t-t)}
θ = t + log_2 θ_w/α
for s ∈ p.shell do
    if s.t < θ then
        p.shell = p.shell \{s}
for k ∈ p.kernel do
    if k.t < θ then
        p.kernel = p.kernel \{k}
    else
        k.weight = 2^{-α*(t-k.t)}
        weightSum = weightSum + k.weight
if weightSum ≥ MinWeight then
    return True
return False
```

Algorithm 4 fuzzyBorderUpdate(b, c)

Given: b - border object
Given: c - core object

```
μ_bc = fuzzyMembership(ε_min, ε_max) //as in Fig. 2
C = cluster of object c
b.memberships[C] = max(b.memberships[C], μ_bc)
```

$C2$, $C3$, then

$$p.\text{memberships} = \{C1: \ 0.6, C3: \ 0.1\}$$

means that p belongs to $C1$ with a membership degree of 0.6, and to $C3$ with a membership degree of 0.1. It follows that:

$$p.\text{memberships}[C1] = 0.6$$
$$p.\text{memberships}[C2] = 0$$
$$p.\text{memberships}[C3] = 0.1$$

During the initialization step, the following parameters need to be set.

- Density-Based reclustering related parameters:
  - ε_{min}: the *kernel*-neighborhood radius of an object;
  - ε_{max}: maximum radius that, along with ε_{min}, defines the *shell*-neighborhood of an object;
  - $MinWeight$: minimum sum of weights associated with the obejcts in the *kernel*-neighborhood of an object to define it as a *core*. It is equivalent to the $MinPts$ parameter of the DBSCAN algorithm;
- Stream related parameters:
  - $T_{offline}$: period of the offline cluster evaluation;
  - α: decay factor adopted for the damped window model;
  - θ_w: weight threshold. Objects with a weight lower than θ_w are removed.

B. The online stage: updating neighborhood sets

The *online stage* is described by the pseudo-code in Algorithm 1. The algorithm handles the arrival of each new object of the stream: the goal is to organize quickly and smartly the objects in appropriate data structures so that, during the periodic offline stage, a partition can be generated in a very short time. Instead of resorting to widely used summarization techniques, for each object p in the stream we instantiate two data structures, namely $p.shell$ and $p.kernel$, aimed at keeping track of its *kernel*-neighborhood and *shell*-neighborhood, respectively.

We recall that TSF-DBSCAN, as well as SF-DBSCAN, considers only objects within ε_{min} in the definition of core objects, and relaxes the constraint on the neighborhood size for the definition of fuzzy borders. The two data structures $p.shell$ and $p.kernel$ let us decouple the identification of core objects and the membership assessment of border objects.

At the arrival of each object p of the stream, first of all the $fuzzyQuery$ procedure is called. Given the parameters ε_{min} and ε_{max}, the procedure computes the set of objects to be inserted into $p.kernel$ and $p.shell$. The procedure extends the $RegionQuery$ procedure defined in [7] by relaxing the crisp constraint on the radius ε. Then, for each object k in $p.kernel$, and for each object s in $p.shell$, $k.kernel$ and $s.shell$ are updated by adding object p. Finally, if the timestamp $p.t$ is a multiple of the period $T_{offline}$, then the offline reclustering stage is triggered.

C. The offline stage: forgetting mechanism and density-based clustering

First of all, the offline stage updates object weights and the $pList$ according to the damped window model. Then, it partitions the data by exploiting the information collected during the online stage. The key aspects of the offline stage can be summarized as follows.

1) The fading model - A damped window approach: In several streaming applications it can be assumed that the importance of a single datum decreases with time and therefore recent data are more relevant than old ones. A popular approach for implementing such a concept associates a weight with each object and adopts a damped window model, in which the weight decreases exponentially over time. Given an object p associated with an arrival timestamp $p.t$, the weight of p at time t, with $t \geq p.t$, is calculated as

$$p.\text{weight}(t) = 2^{-\alpha(t-p.t)} \tag{1}$$

The coefficient α represents the *decay factor* and must be greater than zero: the higher its value, the lower the importance given to old data in the stream.

To better adapt to concept drift and to satisfy the real-time requirement of the online stage, the oldest objects are progressively "forgotten", according to the damped window model. To this aim, objects with a weight lower than the threshold value θ_w are removed. Instead of updating the weight value of each object, in practice it is more convenient at time t to evaluate the *oldest valid timestamp* $\theta(t)$; this can be done by solving the equation

$$2^{-\alpha(t-\theta(t))} = \theta_w \tag{2}$$

It follows that at time t:

$$\theta(t) = \frac{\log_2 \theta_w}{\alpha} + t \tag{3}$$

In an offline procedure run at time t, objects with timestamp lower than $\theta(t)$ are deleted from $pList$ (Algorithm 2, line 4) and from all the neighborhood lists to which they belonged, (Algorithm 3, lines 5 and 8). The lifetime Δ_{decay} of each object is therefore:

$$\Delta_{decay} = t - \theta(t) = -\frac{\log_2 \theta_w}{\alpha} \tag{4}$$

The period $T_{offline}$ of the offline evaluation should be set in accordance with the application requirement and must be coherent with the Δ_{decay} value. In particular, $T_{offline}$ must be shorter than Δ_{decay} so as to avoid that the contribution of some objects between two subsequent clustering procedures is lost.

It is worth underlining that the previous considerations are valid both for continuous and discrete time models. In the first case, t represents the actual datetime of arrival of an object and can be used to evaluate the object weight and the time threshold. In the second case, t corresponds to the index of arrival of an object in the stream. The timestamp t of an object is given by the number of instances arrived so far. In our setting a discrete time model is used. Parameters α and θ_w regulate the forgetting mechanism and should be set according to the application requirements (a raplidly evolving stream should forget older objects quickly) and, in any case, to comply with memory and computational constraints.

2) The core election based on the neighborhood set: The election of an object p as a *core* is described in Algorithm 3: the condition on the minimum number of objects in the ε_{min}-neighborhood adopted in DBSCAN, S-DBSCAN and SF-DBSCAN, is replaced with a condition on the sum of the weights of the objects in the neighborhood, compliant with the variable weight of each object due to the damped window model. Indeed, sets $p.shell$ and $p.kernel$ are updated according to the damped window model. For each object k (s) in $p.kernel$ ($p.shell$), if $k.t < \theta(t)$ ($s.t < \theta(t)$), then object k (s) is considered to be expired and therefore removed from the $p.kernel$ ($p.shell$) set; otherwise, the weight associated with object k is updated and added to the sum of the weights of the objects in $p.kernel$. If the sum is higher than or equal to the threshold $MinWeight$, then object p is a *core* object.

3) The overall reclustering procedure: The reclustering procedure is described in Algorithm 2. It evaluates a new partition from scratch: all the non-expired objects in $pList$ are investigated. As soon as an object q meets the *core* condition (line 15) it starts a new cluster (line 19). The algorithm tries to expand the newly created cluster by first visiting the objects that belong to ε_{min}-neighborhood of q (in line 17 the set of objects to be visited is updated). Contextually, non-core objects in the neighborhood of q are assigned to its border according to the procedure described in the following subsection.

4) The fuzzy border assessment: The assessment of border object cluster membership is described in Algorithm 4: when a non-core object b lies in the neighborhood of an existing core object c (i.e. at a distance lower than ε_{max}), it is assigned to the cluster of c. The membership degree (line 1) is evaluated according to the function sketched in Fig. 2. The membership of b to a cluster is calculated on the basis of the closest core object (line 3). It is worth underlining that such definition allows assigning a border object to different clusters, when it lies in the neighborhood of core objects of different clusters. Hence, clusters with fuzzy overlapping borders can be captured.

D. Space and time complexity

Compliance with storage and time requirements is of the utmost importance in streaming clustering applications. In the following, we analyze the space and time complexity of TSF-DBSCAN.

Since no summarization technique is applied, space complexity depends on the number of objects from the data stream that are stored in memory, i.e. the size of $pList$. Thanks to the pruning strategy defined in Algorithm 2 (lines 3 and 4) an upper bound can be defined regardless of the length of the stream and it depends on the decay factor α and the weight threshold θ_w. Assuming that a single object is collected per unit of time (i.e. stream speed = 1) the size of $pList$ can be obtained as in equation 4: $n = |pList| = -\frac{\log_2 \theta_w}{\alpha}$. For each object p we maintain two data structures, namely p.kernel and p.shell. The resulting worst case for space complexity is $\mathcal{O}(n^2)$, but it reflects the unlikely scenario in which all objects lie in the ε_{max} neighborhood of any other object. Let $\hat{k}$ and $\hat{s}$ be the average size of the kernel and shell sets over all the objects in $pList$, respectively. Typically, a proper setting of ε_{min} parameter implies $\hat{k} \ll n$. When $\hat{k} = n$ the clustering procedure is meaningless since a single cluster is discovered. The choice of ε_{max} influences the size of the shell set and leads to a trade-off between cluster border extension and space (and time) complexity. The space complexity for the online maintenance of data structures is $\mathcal{O}(n(\hat{k} + \hat{s}))$. Furthermore, we store the membership degree of each object to each discovered cluster, as the output of the offline fuzzy reclustering stage. Let C be the number of discovered clusters. The memory complexity for storing the membership degrees is $\mathcal{O}(nC)$. We can ultimately formulate the space complexity as $\mathcal{O}(n(\hat{k} + \hat{s} + C))$.

As per the time complexity, we separately analyze the various components of the algorithm. During the online stage, for each newly collected object, the $fuzzyQuery$ procedure described in Algorithm 1 is executed. The procedure is charactersied by a time complexity $\mathcal{O}(d * n)$, where d is the data dimensionality and n is the number of objects in $pList$. Notably, the adoption of a spatial indexing structure would improve the complexity to $\mathcal{O}(d * \log n)$ [7]. The offline component processes the collected objects similarly to how DBSCAN does. However, in TSF-DBSCAN, thanks to the online maintenance of adjacency data structures, no additional distance computation is required. For each object in $pList$ we check the core condition; whenever an object is recognized as a core we assess the fuzzy membership degree of every non-core object to its ε_{max}-neighborhood, and we try to expand its cluster by analyzing unvisited objects in its ε_{min}-neighborhood.

The resulting time complexity for the offline step can be expressed as $\mathcal{O}(n(\hat{k}+\hat{s}))$.

E. Relationship with other algorithms

TSF-DBSCAN has some similarities with DenStream and DBStream, as both of them are shaped according to the two-stage online-offline paradigm, and adopt a density based approach for the reclustering step. Furthermore, TSF-DBSCAN implements a fading mechanism that gives lower importance to older objects, thus adapting the developed model according to the evolution of non-stationary data streams. The mechanism is also combined with a pruning strategy to limit the required memory. Remarkably, in TSF-DBSCAN we discontinue the use of microclusters, which recently have found large consensus and have been exploited also in DenStream and DBStream, in favour of a solution that directly stores the received objects and is able to generate a fine-grained partition over them. This approach turns to be workable thanks to the use of the aging mechanism, properly tuned on the basis of the specific application case and the object arrival rate. Similarly to d-FuzzStream, TSF-DBSCAN leverages concepts from fuzzy set theory pursuing a higher modelling capability. However, d-FuzzStream is substantially different from TSF-DBSCAN, since it exploits the microcluster abstraction in the online stage and adopts a prototype-based algorithm in the offline stage.

V. Experimental Evaluation

The behavior of the proposed approach has been tested by means of a broad experimental evaluation on both synthetic and real-world datasets. Further, we compared TSF-DBSCAN to several related streaming clustering algorithms, namely D-Stream, DenStream, d-FuzzStream[2], and DBStream, making use of implementations available in the MOA (Massive Online Analysis) framework [33] and in the *stream* R package [34]. Experiments have been performed on a server with an Intel Core i7-2670QM CPU @ 2.20GHz and 6 GB of RAM, running Linux Ubuntu 18.04.

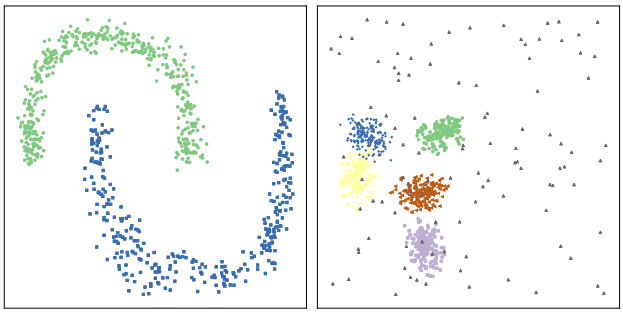
A. Datasets description

The properties of each dataset are briefly described in the following, and are summarized in Table II. The performance of the different clustering algorithms has been assessed in a variety of settings, w.r.t. the nature of the stream (synthetic or real, evolving or static), the dimensionality of the problem, and the number of samples.

DS1, DS2, Banana and RBF are bidimensional synthetic data sets. **DS1** and **DS2** have been already described in Section III. **Banana**[3] contains simple exemplary non-convex structures: although not originally developed as a streaming dataset, in our experiments its objects have been picked up sequentially after a preliminary randomly shuffling. Coversely, **RBF**[4] is a non-stationary data stream with objects originated from multiple Gaussian components that move in a 2-dimensional space, overlapping in some zones. The presence of noise, along with appearance/disappearance of clusters makes the scenario even more challenging. Figure 6 shows the initial portion of the stream extracted from Banana and RBF datasets. Hyperplane, Powersupply and Sensor are hosted in the publicly available Stream Data Mining Repository[5]. **Hyperplane** is a synthetic dataset where concept drift and decision boundaries among classes are generated by using a closed-form expression. **Powersupply** contains hourly power supplies of an Italian electricity company, which records the power from two sources: power supply from main grid, and power transformed from other grids. The class attribute is the hour of the day associated with each record. **Sensor** stream consists of information collected over two months from 54 sensors scattered in the Intel Berkeley Research Lab. Each record reports values for temperature, humidity, light and sensor voltage data, along with the sensor ID. Drift phenomena distinctly show up: the light trend is expected to have a daily period, while the temperature in certain rooms may depend on the presence of people inside. **Covertype** has been widely used for the evaluation of streaming clustering algorithms. It is a collection of cartographic records on observations of 30×30 metres areas in the Roosevelt National Forest in northern Colorado. The available class labels represent seven types of forest cover. Following the preprocessing method adopted in other recent works [17], [31], in our experiments we take into

TABLE II
Datasets description

Dataset	Real / Synth	Samples	Dim.	Max number of classes	Noise	T
DS1	S	800	2	4	N	100
DS2	S	5000	2	2	N	500
Banana	S	4811	2	2	N	800
RBF	S	40000	2	7	Y	1000
Hyperplane	S	100000	10	5	N	1000
Powersupply	R	29928	2	24	N	1000
Sensor	R	2219803	4	55	N	1000
Covertype	R	581012	10	7	N	1000
PricePaid	R	180	1	3	N	10
Insects	R	57018	33	6	N	1000
NOAA	R	18159	8	2	N	1000
MnistVAE	R	60000	5	10	N	1000



(a) Banana (b) RBF

Fig. 6. Bidimensional synthetic datasets: the first 800 and 1000 data points are represented for Banana and RBF datasets, respectively.

[2] https://github.com/Xicks/d-FuzzStream (last visited August 2020)
[3] https://github.com/deric/clustering-benchmark (last visited August 2020)
[4] https://github.com/CIG-UFSCar/DS_Datasets (last visited August 2020)
[5] http://www.cse.fau.edu/~xqzhu/stream.html

account only the numerical attributes of the dataset (10 out of 54). **PricePaid**, from the HM Land Registry, is a collection of house sales in England and Wales. Following a procedure adopted in previous literature [32], we select house sales in London, Liverpool and York, as observed along the decade 2006-2016. The most relevant attribute for clustering purposes is the house price; objects are ordered according to the sale transaction date. Since information on the size of houses is missing, transactions are aggregated over a non-overlapping sliding window of 60 days. Thus, three averaged values (one for each city) evaluated on the two-months window, are sequentially fed to the algorithm. The goal is to recognize the evolution over time of the average house price in different cities. The recently proposed **Insects** dataset [35] contains data collected from sensors in a mosquito trap which are able to measure the flight characteristics of captured insects. For each sample, 33 features are derived from the acquired signal spectrum. Interestingly, different types of concept drift have been induced by purposely varying environmental factors, e.g. temperature. In our analysis, we specifically refer to the *Incremental drift and balanced* version of the dataset. The **NOAA** dataset[6] contains weather measurements by the National Oceanic and Atmospheric Administration, over a time span of 50 years. For each sample, eight weather features (e.g. temperature and wind speed) are associated with a binary label, i.e. rain or not. Finally, as per **MnistVAE** we use a Variational AutoEncoder (VAE) [36] to extract a low-dimensional representation of the MNIST dataset, consisting of handwritten digits images. The class label indicates the represented digit. The VAE projects each input image of size 28×28 (i.e. 784-dimensional) into a k-dimensional latent space. Coherently with previous experiments reported in the literature [36], we set k = 5. Reasonably, the properties of the latent-space might make it a proper target for a fuzzy model, since different handwritings may result in irregular and overlapping clusters. The stream is obtained via a sequential scan of the static MNIST dataset.

B. Evaluation Strategy

Various metrics have been proposed for the evaluation of clustering algorithms. In genuine clustering applications, where the availability of true class labels cannot be assumed, *internal* measures provide indications about the quality of the clustering. Unfortunately, most of them are meaningful only for convex clusters, and thus are not suitable for evaluating the results of density-based clustering algorithms. On the other hand, *external* measures evaluate the consistency between clustering results and ground truth. In many streaming clustering algorithms such an evaluation is not directly applicable, because the original data points are examined and immediately discarded in favor of summarization structures. Nevertheless, the "potential assignment" of each individual object can be recovered by mapping each example to its closest micro-cluster, and inheriting the relative macro-cluster assignment.

Partitions are evaluated using a widely accepted external measure, namely Adjusted Rand Index (ARI). ARI, a corrected-for-chance version of the Rand Index, measures the agreement between clustering results and ground truth labels:

$$\text{ARI} = \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}$$

where a_i, b_j and n_{ij} are the number of objects in class i, the number of objects in cluster j, and the number of objects in class i and cluster j, respectively. ARI is evaluated after a defuzzification process: border objects are assigned to clusters on the basis of their highest membership degree.

It is worth underlining that, for some datasets, no bijective mapping between classes and clusters can be assumed. For example, in the initial portion of DS2, objects from the two classes generate one single promiscuous cluster: later, as a consequence of the occurred concept drift, objects of different classes can be split into different clusters. In this case we can notice that, for any algorithm, the output partition may not show a one-to-one match between clusters and classes, thus yielding low average values for the ARI index. Nevertheless, the adoption of ARI, as a corrected-for-chance *pair counting based* measure, ensures a fair comparison of different algorithms since it is not biased in favour of a large number of discovered clusters (as it happens for other indexes like Purity) [37].

Taking into account the substantial modeling and operating differences across the target algorithms, we carefully designed a common evaluation strategy for an empirical comparison: for each dataset, we periodically trigger the offline reclustering step with a period of T objects; after reclustering, each external measure is evaluated on the last T objects. Such strategy lets us evaluate the clustering quality over time, giving equal importance to every object in the stream for the purpose of metrics computation.

C. Data Preprocessing and Parameter Setting

Although not essential for the operation of TSF-DBSCAN and other algorithms, we opted for working with normalized attribute values, obtained by subtracting the mean and dividing by the standard deviation, adhering to the procedure adopted in [31]. Of course, such an operation may not always be viable in a real streaming setting when a dataset cannot be analyzed in batch. However, in a real application the normalization parameters can be known in advance, estimated by a domain expert, or even evaluated and adapted online as new data are collected [31], [38]. Consider for example the case of the Sensor dataset: a user can immediately identify reasonable values for standardizing attributes either according to her/his experience (temperature and humidity of an indoor environment), or by acquiring information about the operating range of sensors. On the other hand, without normalization, the Euclidean distance between objects is dominated by the features with the largest range of values (e.g. temperature or humidity) and low importance is given to features with values in a smaller range (e.g. sensor voltage).

We evaluate the performance of each algorithm by varying the parameters in a reasonable range around default values. In

[6] https://sites.google.com/view/uspdsrepository

the following, we report the results obtained with the parameter configuration that yields the highest values of average ARI. Of course such sort of supervised parameter tuning is unfeasible in real clustering applications, but can provide insights into the potential ability to accurately model various datasets, and for this reason it has been often used in the clustering literature [17], [31].

For TSF-DBSCAN we vary the ε_{min} and ε_{max} distance thresholds and the $MinWeight$ parameter for the identification of core and border objects. Notably, the values of these parameters in real applications may be inferred during a warm-up phase on the initial portion of the stream resorting to heuristic approaches as proposed in [7]. We also tune the decay factor α in order to obtain a value of Δ_{decay} comparable to the evaluation period T. For d-FuzzStream, we vary the fuzzification parameter m and the merging threshold τ. The minimum and maximum number of allowed FMiCs is left as default, except for the long sensor data stream in which they are set as 100 and 1000, respectively. For the offline weighted Fuzzy-C-Means procedure we set the expected number of clusters as the number of true classes, facilitating the correct modelling of the dataset for d-FuzzStream. Notably, the initialization of cluster prototypes makes the procedure deterministic. For DenStream, we vary the parameters ε, β and μ, which regulate the maintenance of microclusters. We also vary the decay factor λ and the number of objects used for the initialization of the online process. For DStream we vary parameters Cm, Cl and β that regulate the maintenance of sparse, dense and sporadic grids. We also tune the number of grids per dimension and the decay factor.

VI. Results and Discussion

The online-offline paradigm enables the periodic evaluation of the quality of a partition. For each dataset, the period of the evaluation is fixed, regardless of the algorithm, and its value is reported as 'T' in Table II for both static (i.e. Banana and MnistVAE) and non-static datasets. As a consequence we can measure the evolution of ARI over time. Table III reports the average values of ARI and the average runtime over five identical executions of each algorithm. For each execution we measure the total runtime including both the online and the offline phases. This comparison should be considered only as a rough estimate, since most algorithms are implemented in different frameworks. Given the relative stability found in the measurements and the high time requirement of some runs, we deemed it appropriate not to further increase the number of repetitions. For a more insightful analysis of the results, we also report boxplots of ARI values in Figure 7.

In general, TSF-DBSCAN achieves ARI values comparable to or even better than DBStream, which has proven to be a state-of-the-art algorithm for clustering data streams [17],[31]. At the same time, we notice that D-Stream and DenStream cannot yield competitive average ARI values in the majority of the datasets, as highlighted also in the empirical comparison reported in [31]. To the best of our knowledge, this is the first time that d-FuzzStream is investigated in an extensive experimental campaign: it shows superior performance in terms of average ARI in five out of the twelve datasets. Nevertheless we recall that for the offline phase (based on Weighted Fuzzy-c-means) we set the expected number of clusters as the true number of classes, thus facilitating the agreement between the output partition and the ground-truth labels. For all other algorithms, included TSF-DBSCAN, this information is not provided. The computational cost represents another drawback of d-FuzzStream: Table III suggests that the values of execution time of d-FuzzStream are one or two orders of magnitude greater than other algorithms for most datasets. The runtime values of TSF-DBSCAN, on the other hand, are comparable to those of DBStream, the most accurate among the other algorithms.

DS1 is quite well modelled by all algorithms: the best results are obtained by DBStream, d-FuzzStream and TSF-DBSCAN. Clustering DS2 is much more challenging since the two spherical clusters initially overlap (Fig. 4): in the initial portion of the stream, only d-FuzzStream splits the objects in two clusters as the number of classes is set manually; this may explain the highest value of average ARI. Arguably, when concept drift gradually separates the two clusters, the other algorithms are able to distinguish them quite well too. TSF-DBSCAN outperforms other algorithms (except d-FuzzStream) benefiting from fuzzy modelling of cluster borders.

As per the bidimensional synthetic datasets Banana and RBF, TSF-DBSCAN outperforms the other algorithms. The Banana dataset is perfectly modelled at every reclustering step, yielding an ARI value of 1. Notably, in this scenario d-FuzzStream delivers the worst results, motivated by the adoption of a prototype-based algorithm for the offline stage in presence of non-convex structures (Fig. 6a). The high ARI value achieved by TSF-DBSCAN over the RBF dataset suggests that adopting models with fuzzy borders may turn to be beneficial in very dynamic scenarios with clusters that gradually appear, disappear, merge, and split. Similarly, TSF-DBSCAN yields the highest values of ARI on the evolving Hyperplane and Powersupply datasets. However, in these cases the quality of the partitions captured by all algorithms is poor. In the former, this may be explained by the intrinsic complexity of the dataset. In the latter, the fine-grained class labels, i.e. hours of the day, and the consequent strong overlap between the clusters hamper the proper partitioning of streaming objects.

d-FuzzStream achieves the highest performance in Sensor dataset. It is worth underlining that in this case the number of classes is high (54), and it is not guaranteed that all of them are present in each evaluation window, which spans 1000 objects. Although we were aware that it may not be exact, we have set the value of 54 classes for the reclustering algorithm anyway and we have verified that it is realistic: the average number of classes over all the time windows is 47. The lower performance of TSF-DBSCAN and DBStream can be explained by the higher average number of captured clusters (93 and 102, respectively).

Results on Covertype, Insects and NOAA confirm that high dimensional datasets are hard to cluster, as highlighted also in previous works [17], [31]. Notwithstanding, TSF-

TABLE III
ADJUSTED RAND INDEX (ARI), AVERAGED OVER ALL THE EVALUATION TIME WINDOWS, AND AVERAGE RUNTIME MEASUREMENTS (IN SECONDS, INCLUDING BOTH ONLINE/OFFLINE PHASES). BEST VALUES ARE INDICATED IN BOLD.

	DS1	DS2	Banana	RBF	HyperP	PowerS	Sensor	CoverT	PriceP	Insects	NOAA	MnistVAE
							ARI					
DenStream	0.77527	0.31426	0.70235	0.46913	0.02649	0.04776	0.45444	0.09673	0.51120	0.13859	0.04683	0.09695
D-Stream	0.81821	0.32392	0.57710	0.63839	0.00281	0.08959	0.07232	0.07390	0.61730	0.09611	0.03671	0.12140
d-FuzzStream	**0.98043**	**0.59793**	0.44695	0.68210	0.00967	0.02750	**0.66186**	0.08965	**0.75483**	0.12620	0.02141	**0.41938**
DBStream	0.97729	0.41700	**1.00000**	0.67165	0.01773	0.09330	0.60072	0.14142	0.73470	0.13831	0.04926	0.30932
TSF-DBSCAN	0.97085	0.46269	**1.00000**	**0.79529**	**0.03986**	**0.11045**	0.61765	**0.15433**	0.70183	**0.15586**	**0.05740**	0.31460
							Average Runtime					
DenStream	**0.058**	**0.363**	1.633	5.414	32.411	12.282	1638.397	2173.839	0.379	**14.854**	24.928	5.671
D-Stream	0.170	0.754	1.657	4.951	31.847	**1.211**	**165.571**	**27.036**	**0.163**	18.714	**1.249**	**5.167**
d-FuzzStream	0.241	10.323	10.311	73.224	474.775	6.934	2925.813	2524.590	0.242	798.914	1.709	84.092
DBStream	1.247	1.214	**0.846**	**3.131**	37.921	9.064	556.633	99.389	1.105	34.784	8.020	22.955
TSF-DBSCAN	0.088	0.990	2.053	4.665	**6.857**	7.169	1104.651	56.592	0.243	40.681	10.624	36.284

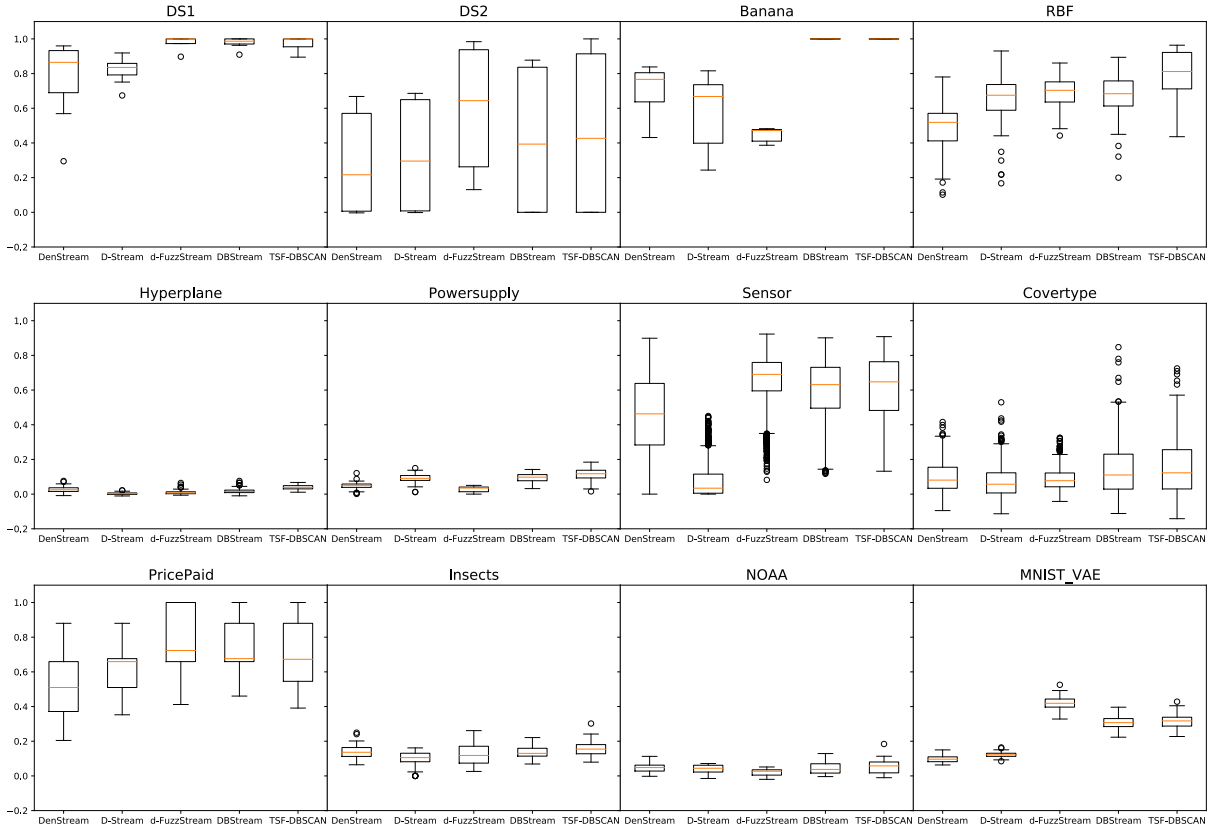


Fig. 7. Boxplots of Adjusted Rand Index values obtained by each algorithm for each dataset.

DBSCAN slightly outperforms all the other algorithms in terms of ARI. As regards D-Stream, its poor performance has been motivated with the limited number of grid cells used to summarize the stream [31]. Similar observations can be made for the MnistVAE dataset. As per the ARI values, reasonable partitions are obtained by fuzzy algorithms (TSF-DBSCAN and d-FuzzStream) and by DBStream, suggesting that the introduction of fuzziness or the careful handling of low density regions between microclusters are crucial for clustering in the latent space of our Variational AutoEncoder.

The monodimensional PricePaid dataset is hard to model because of the diverse fluctuations of the house prices in the three UK cities. The related variability of the ARI values is evident in the boxplots in Fig. 7. For each algorithm, the partition is evaluated every 10 instances, starting from the 20^{th} instance to avoid including the initial measure when a few instances have been collected.

Making use of the results obtained over all the twelve datasets, we perform a statistical test to assess possible statistical differences among algorithms (as regards their deliv-

ered ARI values). TSF-DBSCAN is selected as the control algorithm, and it is pairwise compared with each of the other algorithms through the non-parametric Wilcoxon signed-rank test [39] (SciPy implementation[7]). Results are reported in Table IV.

TABLE IV
RESULTS OF THE WILCOXON SIGNED-RANK TEST ON THE AVERAGE ARI VALUES OBTAINED ON TWELVE DATASETS.

Comparison	Statistic	p-value	Hypothesis
TSF-DBSCAN vs DenStream	0	0.00222	Rejected
TSF-DBSCAN vs D-Stream	0	0.00222	Rejected
TSF-DBSCAN vs d-FuzzStream	32	0.58292	Not-Rejected
TSF-DBSCAN vs DBStream	13.5	0.04546	Rejected

The hypothesis of equivalence is rejected whenever the p-value is lower than the significance level α. With $\alpha = 0.05$, we can state that TSF-DBSCAN outperforms DenStream, D-Stream, and DBStream in terms of ARI. It is worth focusing on the comparison between TSF-DBSCAN and d-FuzzStream: although they are statistically equivalent, in general the former is more efficient (with lower average runtime) and, most importantly, it does not require a prior knowledge of the number of clusters.

A. Sensitivity Analysis

We investigate the sensitivity of TSF-DBSCAN with respect to its most influential input parameters by considering two representative datasets. RBF is taken as an example of evolving data stream, while MnistVAE represents an example of static dataset, i.e. without concept drift. In particular, we explore the bi-dimensional space defined by ε_{min} and ε_{max}, fixing the value of $MinWeight$. Indeed, it has been shown [40] that the traditional DBSCAN algorithm is rather stable across the choice of $MinPoints$, provided that a reasonable combination with the distance threshold ε is adopted. Furthermore, we evaluate the sensitivity with respect to the decay factor α. For both datasets, ε_{min} is varied in the range $[0.05, 0.4]$ with step 0.05, while ε_{max} is expressed as a multiple of ε_{min}. Parameter α, along with the weight threshold θ_w determines the number of objects stored in memory; since for both datasets we consider an evaluation period of 1000 samples, we derive the upper bound for α as 0.0017, at which just the 1021 most recent objects are maintained. We set the lower bound to 0.0005 (3473 objects), varying the parameter with a step of 0.0002. Results are reported in Figures 8 and 9: we measure the average ARI and the runtime of each parameter configuration.

It is evident that TSF-DBSCAN, just like the progenitor DBSCAN, is rather sensitive to the ε_{min} parameter. For RBF, Fig. 8a, we hypothesize that a small kernel region does not enable the identification of core objects, and increasing it above the optimal value (0.1) leads to a gradual degradation of ARI. For MnistVAE the 'acceptable' region is even more confined: we argue that high values of ε_{min} lead to the improper merging of close clusters in the VAE latent space, as confirmed by the fact that the number of macroclusters ranges from ~ 15 for $\varepsilon_{min} = 0.1$, to ~ 3 for $\varepsilon_{min} = 0.4$.

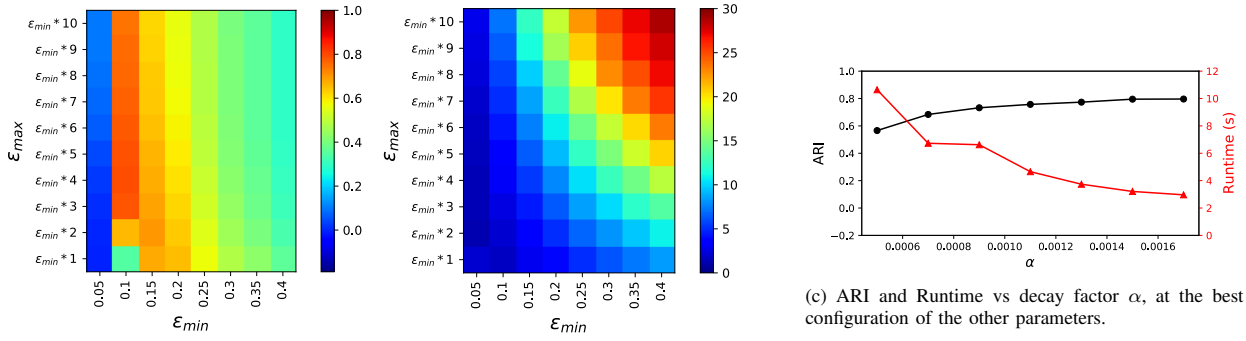
We can also notice that for both datasets the introduction of fuzziness is often beneficial in terms of ARI: the best configuration is never located in the first row, where $\varepsilon_{min} = \varepsilon_{max}$ and TSF-DBSCAN degenerates into its crisp counterpart. Such outcome certifies the major modelling capability gained with the introduction of fuzziness in the considered case studies, i.e. in the presence of concept drift or clusters with fuzzy overlapping borders.

Figures 8b and 9b show that this improvement in performance comes at a cost: as ε_{max} increases, the runtime increases as well. The same applies for ε_{min}, as anticipated in section IV-D. The sensitivity with respect to α depends on the dataset as well. In the non-stationary scenario of RBF, a short-term forgetting mechanism is crucial for good performance (Fig. 8c): when $\alpha = 0.0005$ the ARI drop is -0.23 w.r.t. the best parameter configuration. For the static MnistVAE dataset (Fig. 9c) maintaining more samples lead only to a slight decrease in performance (-0.08). Obviously, for both datasets, the execution time decreases as α increases since the number of objects is directly proportional to α^{-1}.

VII. CONCLUSION

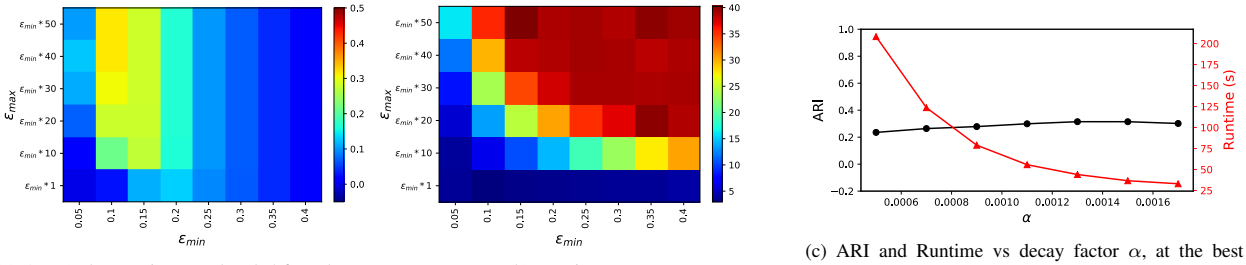
In this paper, we have proposed TSF-DBSCAN, a fuzzy density-based algorithm for clustering data streams. It adopts a two-stage online-offline approach, and its main novelty lies in an efficient fine-grained handling of incoming new objects, instead of summarization methods. This characteristic allows TSF-DBSCAN to manage potentially unbounded evolving data streams. We have discussed the advantages of TSF-DBSCAN over SF-DBSCAN, S-DBSCAN (a non-fuzzy version of SF-DBSCAN) and the original DBSCAN algorithm with respect to the challenges of clustering data streams. The modelling capability of TSF-DBSCAN benefits from the introduction of a fuzzy membership function between objects in the attribute space. Relaxing the crisp constraint on the definition of the neighborhood of an object allows capturing clusters with fuzzy overlapping borders, which is particularly significant when boundaries between adjacent clusters are not well defined. Furthermore, the implementation of a damped window model, which assigns lower weights to older objects, supports the adaptation to evolving settings, where concept drift causes a gradual modification of the data distribution. Simple tests on two bidimensional datasets provided a straightforward visual means to assess the modelling power of TSF-DBSCAN: by introducing a soft constraint on the neighborhood radius, marginal objects of the distribution can be included into the border of the detected clusters, improving the accuracy of the detected structures (with reference to S-DBSCAN). The introduction of the damped window model substantially affects the algorithm operation and behavior: in fact, it turns to be able to uncover cluster movements in the feature space, rather than elongations, by favoring the influence of recent data instances. The modelling capability of TSF-DBSCAN has been compared to that of several crisp and fuzzy clustering algorithms on synthetic and real-world datasets in an

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.wilcoxon.html



(a) ARI (b) Runtime (c) ARI and Runtime vs decay factor α, at the best configuration of the other parameters.

Fig. 8. Parameter sensitivity for the RBF dataset. ARI (a) and Runtime (b) are evaluated for different configurations of ε_{min} and ε_{max}, and w.r.t. to the decay factor α (c).



(a) ARI (colormap is capped at 0.5 for a better visualization) (b) Runtime (c) ARI and Runtime vs decay factor α, at the best configuration of the other parameters.

Fig. 9. Parameter sensitivity on the MnistVAE dataset. ARI (a) and Runtime (b) are evaluated for different configurations of ε_{min} and ε_{max}, and w.r.t. to the decay factor α (c).

unprecedented experimental analysis. In particular, statistical tests on ARI values indicate that the proposed algorithm behaves significantly better than DenStream, D-Stream, and comparably to or even slightly better than DBSTREAM, which is currently regarded as one of the most effective streaming clustering algorithms. Although d-FuzzStream has proved to be statistically equivalent to TSF-DBSCAN and occasionally leads to better ARI values, it requires the prior knowledge of the number of clusters, and it is computationally much more demanding. Finally, a parameter sensitivity analysis carried out on two representative datasets reveals the benefit of the introduction of fuzziness and highlights the main limitation of the proposed algorithm, typical of density-based approaches: the crucial importance of a proper setting for the distance threshold parameter.

ACKNOWLEDGMENTS

This work was partially supported by the Italian Ministry of Education and Research (MIUR) in the framework of the CrossLab project (Departments of Excellence). The authors would like to thank Dr. Andrea Aliperti, who implemented a preliminary version of the algorithm presented in this paper.

REFERENCES

[1] R. H. Moulton, H. L. Viktor, N. Japkowicz, and J. Gama, "Clustering in the presence of concept drift," in *Machine Learning and Knowledge Discovery in Databases. Proc. of ECML PKDD 2018*, M. Berlingerio, F. Bonchi, T. Gärtner, N. Hurley, and G. Ifrim, Eds. Cham: Springer International Publishing, 2019, pp. 339–355.

[2] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, and J. a. Gama, "Data stream clustering: A survey," *ACM Comput. Surv.*, vol. 46, no. 1, pp. 13:1–13:31, Jul. 2013.

[3] A. Amini, T. Y. Wah, and H. Saboohi, "On density-based data streams clustering algorithms: A survey," *Journal of Computer Science and Technology*, vol. 29, no. 1, pp. 116–141, 2014.

[4] A. Zubaroğlu and V. Atalay, "Data stream clustering: a review," *Artificial Intelligence Review*, Jul 2020.

[5] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, Jul 2016.

[6] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, Mar. 2014.

[7] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. of the 2nd Int'l Conf. on Knowledge Discovery and Data Mining*. AAAI Press, 1996, pp. 226–231.

[8] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[9] D. Ienco and G. Bordogna, "Fuzzy extensions of the DBScan clustering algorithm," *Soft Computing*, vol. 22, no. 5, pp. 1719–1730, 2018.

[10] A. Aliperti, A. Bechini, F. Marcelloni, and A. Renda, "A fuzzy density-based clustering algorithm for streaming data," in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, June 2019, pp. 1–6.

[11] J. Gama, *Knowledge discovery from data streams*. CRC Press, 2010.

[12] M. Carnein and H. Trautmann, "Optimizing data stream representation: An extensive survey on stream clustering algorithms," *Business & Information Systems Engineering (BISE)*, vol. 61, pp. 277–297, 2019.

[13] H.-L. Nguyen, Y.-K. Woon, and W.-K. Ng, "A survey on data stream

clustering and classification," *Knowledge and Information Systems*, vol. 45, no. 3, pp. 535–569, Dec 2015.
[14] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.
[15] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proc. of the 29th Int'l Conf. on Very Large Data Bases*, vol. 29. VLDB Endowment, 2003, pp. 81–92.
[16] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proc. of the 2006 SIAM Int'l Conf. on Data Mining*. SIAM, 2006, pp. 328–339.
[17] M. Hahsler and M. Bolaños, "Clustering data streams based on shared density between micro-clusters," *IEEE Trans. on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1449–1461, 2016.
[18] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proc. of the 13th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*. ACM, 2007, pp. 133–142.
[19] L. Tu and Y. Chen, "Stream data clustering based on grid density and attraction," *ACM Trans. on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 3, p. 12, 2009.
[20] A. Amini, H. Saboohi, T. Herawan, and T. Y. Wah, "Mudi-stream: A multi density clustering algorithm for evolving data stream," *Journal of Network and Computer Applications*, vol. 59, pp. 370–385, 2016.
[21] L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, and K. Zhang, "Density-based clustering of data streams at multiple resolutions," *ACM Trans. on Knowledge discovery from Data (TKDD)*, vol. 3, no. 3, p. 14, 2009.
[22] M. Ester and R. Wittmann, "Incremental generalization for mining in a data warehousing environment," in *Advances in Database Technology — EDBT'98*, H.-J. Schek, G. Alonso, F. Saltor, and I. Ramos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 135–149.
[23] R. Hyde and P. Angelov, "A new online clustering approach for data in arbitrary shaped clusters," in *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*, June 2015, pp. 228–233.
[24] R. Hyde, P. Angelov, and A. MacKenzie, "Fully online clustering of evolving data streams into arbitrarily shaped clusters," *Information Sciences*, vol. 382-383, pp. 96 – 114, 2017.
[25] M. K. Islam, M. M. Ahmed, and K. Z. Zamli, "A buffer-based online clustering for evolving data stream," *Information Sciences*, vol. 489, pp. 113 – 135, 2019.
[26] L. Schick, P. de Abreu Lopes, and H. de Arruda Camargo, "d-fuzzstream: A dispersion-based fuzzy data stream clustering," in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2018, pp. 1–8.
[27] S. Laohakiat, P. Ratanajaipan, L. Navaravong, R. Ungrangsi, and K. Maleewong, "A fuzzy density-based incremental clustering algorithm," in *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, July 2018, pp. 1–5.
[28] P. de Abreu Lopes and H. de Arruda Camargo, "Fuzzstream: Fuzzy data stream clustering based on the online-offline framework," in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. IEEE, 2017, pp. 1–6.
[29] R. J. Hathaway and Y. Hu, "Density-weighted fuzzy c-means clustering," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 1, pp. 243–252, 2008.
[30] M. Bolaños, J. Forrest, and M. Hahsler, "Clustering large datasets using data stream clustering techniques," in *Data Analysis, Machine Learning and Knowledge Discovery*, M. Spiliopoulou, L. Schmidt-Thieme, and R. Janning, Eds. Cham: Springer International Publishing, 2014, pp. 135–143.
[31] M. Carnein, D. Assenmacher, and H. Trautmann, "An empirical comparison of stream clustering algorithms," in *Proceedings of the Computing Frontiers Conference*, ser. CF'17. New York, NY, USA: ACM, 2017, pp. 361–366.
[32] D. G. Márquez, A. Otero, P. Félix, and C. A. García, "A novel and simple strategy for evolving prototype based clustering," *Pattern Recognition*, vol. 82, pp. 16–30, 2018.
[33] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.
[34] M. Hahsler, M. Bolaños, and J. Forrest, "Introduction to stream: An extensible framework for data stream clustering research with R," *Journal of Statistical Software*, vol. 76, no. 14, pp. 1–50, 2017.
[35] V. M. A. Souza, D. M. Reis, A. G. Maletzke, and G. E. A. P. A. Batista, "Challenges in benchmarking stream learning algorithms with real-world data," *Data Mining and Knowledge Discovery*, pp. 1–54, 2020.
[36] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv preprint arXiv:1312.6114*, 2013.
[37] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *The Journal of Machine Learning Research*, vol. 11, pp. 2837–2854, 2010.
[38] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004, pp. 852–863.
[39] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
[40] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

Alessio Bechini earned his Laurea degree in Electronics Engineering and the PhD degree in Information Engineering both from the University of Pisa in 1996 and 2003. At present he is a researcher at the Department of Information Engineering of the University of Pisa. His research interests span the fields of concurrent and distributed systems, enterprise information systems, data management and integration, service management, with particular interest in bioinformatics problems. Recently, his work has been focused on issues in data mining for Big Data. He has served as TPC co-chair, general co-chair and program co-chair of ACM international conferences.

Francesco Marcelloni (M'06) is currently a full professor at the Department of Information Engineering of the University of Pisa, Italy. His main research interests include data mining for big data, sentiment analysis and opinion mining, multi-objective evolutionary algorithms, genetic fuzzy systems, fuzzy clustering algorithms, and data compression and aggregation in wireless sensor networks. He has co-edited three volumes, four journal special issues, and is (co-)author of a book and of more than 230 papers in international journals, books and conference proceedings. He has been selected to receive the 2021 IEEE TFS Outstanding Paper award. He has been TPC co-chair, general co-chair and tutorial chair of some international conferences and has held invited talks in a number of events. Currently, he serves as associate editor of IEEE Transactions on Fuzzy Systems (IEEE), Information Sciences (Elsevier) and Soft Computing (Springer), and is on the editorial board of a number of other international journals. He has coordinated various research projects funded by both public and private entities.

Alessandro Renda received the M.Sc. degree in Biomedical Engineering from the University of Pisa, Italy, in 2017. He is currently a Ph.D. candidate at the Department of Information Engineering of the University of Pisa, Italy, enrolled in the Smart Computing programme, jointly awarded by the Universities of Pisa, Florence and Siena, Italy. His current research interests include machine learning algorithms for data streams, applications of deep learning methodologies and affective computing.