Density-Based Projected Clustering of Data Streams

Marwan Hassani[1], Pascal Spaus[1], Mohamed Medhat Gaber[2], and Thomas Seidl[1]

[1]Data Management and Data Exploration Group
RWTH Aachen University, Germany
lastname@cs.rwth-aachen.de
[2]School of Computing
University of Portsmouth, UK
mohamed.gaber@port.ac.uk

Abstract. In this paper, we have proposed, developed and experimentally validated our novel subspace data stream clustering, termed *PreDeConStream*. The technique is based on the two phase mode of mining streaming data, in which the first phase represents the process of the online maintenance of a data structure, that is then passed to an offline phase of generating the final clustering model. The technique works on incrementally updating the output of the online phase stored in a micro-cluster structure, taking into consideration those micro-clusters that are fading out over time, speeding up the process of assigning new data points to existing clusters. A density based projected clustering model in developing *PreDeConStream* was used. With many important applications that can benefit from such technique, we have proved experimentally the superiority of the proposed methods over state-of-the-art techniques.

1 Introduction

Data streams represent one of the most famous forms of massive uncertain data. The continuous and endless flow of streaming data results in a huge amount of data. The uncertainty of these data originates not only from the uncertain characteristics of their sources as in most of the scenarios, but also from their ubiquitous nature. The most famous examples of uncertain data streams are sensor streaming data which are available in everyday applications. These applications start from home scenarios like the smart homes to environmental applications and monitoring tasks in the health sector [12], but do not end with military and aerospace applications. Due to the nature, the installation and the running circumstances of these sensors, the data they collect is in most cases uncertain. Furthermore, the pervasive flow of data and the communication collision additionally leverage the certainty of collected data. The latter fact is the reason why some other types of data streams can also be uncertain although they are not produced from sensor data (e.g. streaming network traffic data).

Clustering is a well known data mining technique that aims at grouping similar objects in the dataset together into same clusters, and dissimilar ones into different clusters, where the similarity is decided based on some distance function. Thus, objects separated by far distances are dissimilar and thus belong to different clusters. In many applications of streaming data, objects are described by using multiple dimensions (e.g. the Network Intrusion Dataset [1] has 42 dimensions). For such kinds of data with higher dimensions, distances grow more and more alike due to an effect termed *curse of dimensionality* [5] (cf. the toy example in Figure 1). Applying traditional clustering algorithms (called in this context: *full-space* clustering algorithms) over such data objects will lead to useless clustering results. In Figure 1, the majority of the black objects will be grouped in a single-object cluster (outliers) when using a full-space clustering algorithm, since they are all dissimilar, but apparently they are not as dissimilar as the gray objects. The latter fact motivated the research in the domain of *subspace* and *projected clustering* in the last decade which resulted in an established research area for static data. For streaming data on the other hand, although a considerable research has tackled the full-space clustering (cf. Section 2.2), very limited work has dealt with subspace clustering (cf. Section 2.3).

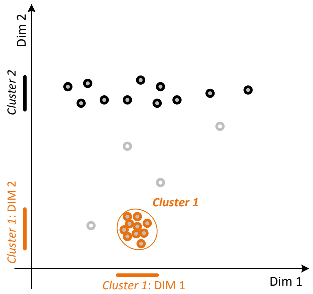


Fig. 1. An example of subspce clustering.

Most full-space stream clustering algorithms use a two-phased (online-offline) model (e.g CluStream [2] and DenStream [7], cf. Section 2.2). While the online part summarizes the stream into groups called *microclusters*, the offline part performs some well-known clustering algorithm over these summaries and gives the final output as clustering of the data stream. Usually, the offline part represents the bottleneck of the clustering process, and when considering the projected clustering which is inherently more complicated compared to the full-space clustering, the efficiency of a projected or subspace stream clustering algorithm becomes a critical issue. In this paper we present a density-based projected clustering over streaming data. Our suggested algorithm *PreDeConStream* tries to find clusters over subspaces of the evolving data stream instead of searching over the full space merely. The algorithm uses the famous (online-offline) model, where in the offline phase, it efficiently maintains the final clustering by localizing the part of the clustering result which was affected by the change of the stream input within a certain time, and then sustaining only that part. Additionally, the algorithm specifies the time intervals within which a guaranteed no-change of the clustering result can be given.

The remainder of this paper is organized as follows: Section 2 gives a short overview of the related work from different neighboring areas. Section 3 introduces some required definitions and formulations to the problem. Our algorithm PreDeConStream is introduced in Section 4, and then thoroughly evaluated in Section 5. Then we conclude the paper with a short outlook in Section 6.

2 Related Work

In this section, we list the related work from three areas: subspace clustering of static data, full-space stream clustering, and finally subspace stream clustering.

2.1 Subspace Clustering Algorithms over Static Data

According to [16], one can differentiate between two main classes of subspace clustering algorithms that deal with static data:

- ***Subspace*** **clustering algorithms** [14] which aim at detecting all possible clusters in all subspaces. In this algorithm class, each data object can be part of multiple subspace clusters.
- ***Projected*** **clustering algorithms** [6] which assign each data object to at most one cluster. For each cluster, a subset of projected dimensions is determined which represents the projected subspace.

SubClu [14] is a subspace clustering algorithm that uses the DBSCAN [9] clustering model of density connected sets. SubClu computes for each subspace all clusters which DBSCAN would have found as well if applied on that specific subspace. The subspace clusters are generated in a bottom-up way and for the sake of efficiency, a monotonicity criteria [14] is used. If a subspace T does not contain a cluster, then no higher subspace S with $T \subseteq S$ can contain a cluster.

PreDeCon [6] is a projected clustering algorithm which adapts the concept of density based clustering [9]. It uses a specialized similarity measure based on the subspace preference vector (cf. Definition 6) to detect the subspace of each cluster. Different to DBSCAN, a preference weighted core point is defined in PreDeCon as the point whose number of preference dimensions is at most λ and the preference weighted neighborhood contains at least μ points.

IncPreDeCon [15] is an incremental version of the algorithm PreDeCon [6] designed to handle accumulating data. It is unable to handle evolving stream data since it does not perform any removal or forgetting of aging data. Additionally, the solution performs the maintenance after each insertion, which makes it considerably inefficient, especially for applications with limited memory. The algorithm we present in this paper adopts in some parts of its offline phase the insertion method of IncPreDeCon, but fundamentally differs from IncPreDeCon by maintaining the summaries of drifting streaming data, applying PreDeCon on the microcluster level, including a novel deletion method, and carefully performing the maintenance of the clustering after some time interval and not after each receiving of an object.

2.2 Full-space Clustering Algorithms over Streaming Data

There is a rich body of literature on stream clustering. Approaches can be categorized from different perspectives, e.g. whether convex or arbitrary shapes are found, whether data is processed in chunks or one at a time, or whether it is a single algorithm or it uses an online component to maintain data summaries and an offline component for the final clustering. Convex stream clustering approaches are based on a k-center clustering [2, 11]. Detecting clusters of arbitrary shapes in streaming data has been proposed using kernels [13], fractal dimensions [17] and density based clustering [7, 8]. Another line of research considers the anytime clustering with the existence of outliers [10].

2.3 Subspace Clustering Algorithms over Streaming Data

Similar to the offline clustering algorithms, two types of stream clustering algorithms exist: subspace and projected stream clustering algorithms. However there is, to the best of our knowledge, only one subspace clustering algorithm and two projected clustering ones over streaming data.

Sibling Tree [19] is a grid-based *subspace* clustering algorithm where the streaming distribution statistics is monitored by a list of grid-cells. Once a grid-cell is dense, the tree grows in that cell in order to trace any possible higher dimensional cluster.

HPStream [3] is a k-means-based *projected* clustering algorithm for high dimensional data stream. The relevant dimensions are represented by a d-dimensional bit-vector D, where 1 marks a relevant dimension and 0 otherwise. HPStream uses a projected distance function, called Manhattan Segmental distance MSD [4], to determine the nearest cluster. HPStream cannot detect arbitrary cluster shapes and a parameter for the number of cluster k has to be given by the user, which is in not intuitive in most scenarios. Additionally, as a k-means based approach, HPStream is a bit sensitive to outliers. The model described in this paper is able to detect arbitrarily shaped and numbered clusters in subspaces and, due to its density-based method, is less sensitive to outliers.

HDDStream [18] is a recent density-based projected stream clustering algorithm that was developed simultaneously with PreDeConStream, and published after the first submission of this paper. HDDStream performs an online summarization of both points and dimensions and then, similar to PreDeConStream it performs a modified version of PreDeCon in the offline phase. Different from our algorithm, HDDStream does not optimize the offline part which is usually the bottleneck of subspace-stream clustering algorithm. In the offline phase, our algorithm localizes effects of the stream changes and maintains the old clustering results by keeping non-affected parts. Additionally, our algorithm defines the time intervals where a guaranteed no-change of the clustering result exists, and organizes the online summaries in multiple lists for a faster update.

3 Problem Formulation and Definitions

In this section, we formulate our related problems and give some definitions and data structures that are needed to introduce our PreDeConStream algorithm. In its online phase, our algorithm adopts the microcluster structure used in most other streaming algorithms [2], [7] with an adaptation to fit our problem (cf. Definitions 2-4). Later, we introduce a data structure and some definitions which are related to the offline phase (cf. Definition 5). Since the algorithm uses a density-based clustering over its online and offline phases, similar notations that appear in both phases are differentiated with an F subscript for the offline phase and N for the online phase.

3.1 Basic Definitions

Definition 1. ***The Decaying Function*** *The fading function [7] used in PreDeConStream is defined as $f(t) = 2^{-\lambda t}$, where $0 < \lambda < 1$ The weight of the data stream points decreases exponentially over time, i.e the older a point gets, the less important it gets. The parameter λ is used to control the importance of the historical data of the stream.*

Definition 2. ***Core Microcluster*** *A core microcluster at time t is defined as a group of close points $p_1, \ldots, p_n$ with timestamps $t_1, \ldots, t_n$. It is represented by a tuple $CMC(w, c, r)$ with:*

1. *Weight, $w = \sum_{j=1}^{n} f(t - t_j)$, with $w \geq \mu_N$*
2. *Center, $c = \frac{\sum_{j=0}^{n} f(t-t_j) p_j}{w}$*
3. *Radius, $r = \frac{\sum_{j=0}^{n} f(t-t_j) dist(p_j, c)}{w}$, with $r \leq \varepsilon_N$*

The weight and the statistical information about the stream data decay according to the fading function (cf. Definition 1). The maintenance of the microclusters is discussed in Definition 4. Two additional types of microclusters are also given, the potential microcluster and the outlier microcluster, to allow the algorithm to quickly recognize changes in the data stream.

Definition 3. ***Potential and Outlier microcluster*** *A* ***potential*** *microcluster $PMC = (\overline{CF^1}, \overline{CF^2}, w, c, r)$ is defined as follows:*

1. *Weight, $w = \sum_{j=1}^{n} f(t - T_j)$ with $w \geq \beta \mu_N$*
2. *Linear weighted sum of the points, $\overline{CF^1} = \sum_{j=1}^{n} f(t - T_j) p_j$*
3. *linear weighted squared sum of the points, $\overline{CF^2} = \sum_{j=1}^{n} f(t - T_j) p_j^2$*
4. *Center $c = \frac{\overline{CF^1}}{w}$*

5. *Radius* $r = \sqrt{\frac{|\overline{CF^2}|}{w} - \left(\frac{|\overline{CF^1}|}{w}\right)^2}$

An ***outlier*** *microcluster* $OMC = (\overline{CF^1}, \overline{CF^2}, w, c, r, t_0)$ *is defined as PMC with the following modifications:*

1. *Weight* $w = \sum_{j=1}^{n} f(t - T_j)$ *with* $w < \beta\mu_N$
2. *An additional entry with the creation time* t_0*, to decide whether the outlier microcluster is being evolving or is fading out.*

The parameter β controls how sensitive the algorithm is to outliers.

Definition 4. ***Microclusters Maintenance*** *With the progress of the evolving stream, any core, potential, or outlier microcluster at time t* $MC_t = (\overline{CF^1}, \overline{CF^2}, w)$ *is maintained as follows: If a point p hits MC at time* $t+1$ *then its statistics become:* $MC_{t+1} = (2^{-\lambda} \cdot \overline{CF^1} + p, 2^{-\lambda} \cdot \overline{CF^2} + p^2, 2^{-\lambda} \cdot w + 1)$ *Otherwise, if no point was added to MC for any time interval* δt*, the microcluster can be updated after any time interval* δt *as follows:* $MC_{t+\delta t} = (2^{-\lambda\delta t} \cdot \overline{CF^1}, 2^{-\lambda\delta t} \cdot \overline{CF^2}, 2^{-\lambda\delta t} \cdot w)$.

It should be noted that this updating method is different from that in DenStream [7]. The modification considers the decaying of the other old points available in MC, even if MC was updated. This makes the algorithm faster in adapting to the evolving stream data. Additionally, this gives our microcluster structure an upper bound for the weight (w_{max}) of the microcluster which will be useful for the maintenance of the offline part as we will see in Section 3.2.

Lemma 1. *The maximum weight* w_{max} *of any microcluster MC is* $\frac{1}{1-2^{-\lambda}}$.

Proof. Assuming that all the points of the stream hit the same microcluster MC. The definition of the weight $w = \sum_{t'=0}^{t} 2^{-\lambda(t-t')}$ can be transformed with the sum formula for geometric series as following:

$$w = \sum_{t'=0}^{t} 2^{-\lambda(t-t')} = \frac{1 - 2^{-\lambda(t+1)}}{1 - 2^{-\lambda}} \tag{1}$$

Thus, the maximum weight of a microcluster is:
$w_{max} = \lim_{t\to\infty} w = \lim_{t\to\infty} \frac{1-2^{-\lambda(t+1)}}{1-2^{-\lambda}} = \frac{1}{1-2^{-\lambda}}$.

Any newly created microcluster needs a minimum time T_p to grow into a potential microcluster, during this time the microcluster is considered as an outlier microcluster. Similarly, there is a minimum time T_d needed for a potential microcluster to fade into an outlier microcluster.

Lemma 2. ***A)*** *The minimum timespan for a newly created microcluster to grow into a potential microcluster is:* $T_p = \left\lceil \frac{1}{\lambda} \log_2 \left(\frac{1}{1-\beta\mu_N(1-2^{-\lambda})} \right) - 1 \right\rceil$.
B) *the minimum timespan needed for a potential microcluster to fade into an outlier microcluster is:* $T_d = \left\lceil \frac{1}{\lambda} \log_2(\beta\mu_N) \right\rceil$.

Proof. **A)**The minimum timespan needed for a newly created microcluster to become potential is $T_p = t_p - t_0$, where t_p is the first timestamp where the microcluster becomes potential and t_0 the creation time of the *outlier* microcluster. According to Def. 3, a microcluster becomes potential when its weight w becomes $w \geq \beta\mu_N$. Thus, from Equation 1: $w = \sum_{t'=t_0}^{T_p} 2^{-\lambda(t-t')} = \frac{1-2^{-\lambda(T_p+1)}}{1-2^{-\lambda}} \geq \beta\mu_N$.
$\Rightarrow T_p = \left\lceil \frac{1}{\lambda} \log_2 \left(\frac{1}{1-\beta\mu_N(1-2^{-\lambda})} \right) - 1 \right\rceil$.
B)Let $T_d = t_d - t_p$ be the minimum timespan needed for a potential microcluster to be deleted, where t_p is the last timestamp where the microcluster was still potential, and t_d is the time when it is deleted. For the deletion, the weight of an outlier microcluster has to be less than $w_{min} = 1$, because the start weight of a newly created microcluster is 1. Let w_p be the last time when the microcluster was potential, according to Def. 4, T_d is the smallest *no-hit* interval that is needed for a potential microcluster to become outlier. Thus: T_d is the smallest value which makes: $w_p \cdot 2^{-\lambda T_d} < 1$. But we know that $w_p = \beta\mu_N \Rightarrow T_d = \lceil \frac{1}{\lambda} \log_2(\beta\mu_N) \rceil$.

Definition 5. ***Minimum Offline Clustering Validity Interval*** *The minimum validity interval of an offline clustering T_v defines the time within which PreDeConStream does not need to update the offline clustering since it is still valid because no change of the status of any microcluster status happened. It is defined as: $T_v = \min\{T_p, T_d\}$*

Definition 6. ***Subspace Preference Vector*** w_c *[6] For each dimension i, if the variance of the microclusters c of the Euclidean ε-neighborhood $\mathcal{N}_{\varepsilon_F}(c)$ is below a user defined threshold δ, then the i-th entry of the preference subspace vector w_c is set to a constant $\kappa \gg 1$, otherwise the entry is set to 1.*

3.2 a Data Structure to Manage the Microclusters

A data structure is needed to manage the updated and non-updated microclusters at each timestamp in an efficient and effective way. The main idea is that the algorithm does not need to check for all *potential* microclusters, at each timestamp, whether the potential microcluster remains potential or fades into a *deleted* microcluster. Therefore a data structure is introduced where only a subset of all the potential microclusters needs to be checked. We group the microclusters into multiple lists according to their weight. The borders between these lists are selected as below (cf. also Figure 2) such that all microclusters in a list that are not hit in the previous timestamp will fade to the lower-weighted list. Thus, only the weight of the one which was hit needs to be checked. There are two types of lists: outlier lists l_j^o, and potential lists: l_i^p. The borders of the lists are: $W_d = 1, W_{min} = \beta\mu_N, W_{max} = \frac{1}{1-2^{-\lambda}}$. The internal borders are selected as: $w_i^p = \frac{w_{i-1}^p}{2^{-\lambda}}$ for the potential lists, and $w_i^o = 2^{-\lambda} w_{i-1}^o + 1$ for the outlier lists. It should be noted, that in this case, only the lists around W_{min} from both outlier and potential sides (cf. Figure 2) need to be checked each T_v to see whether the current offline clustering is still valid as we will see in Section 4.

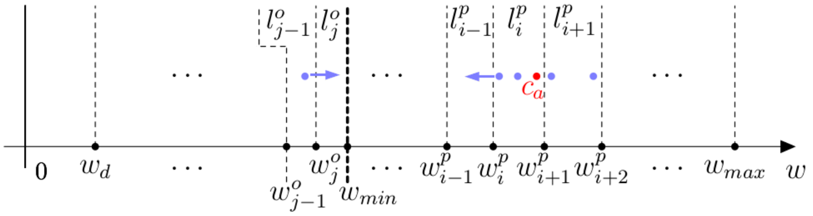


Fig. 2. An example of outlier and potential lists visualized w.r.t. their weights.

4 The PreDeConStream Algorithm

Initialisation Phase In lines 1-4 of Algorithm 1, the minimum timespan T_v based on the user's parameter setting is computed. Furthermore, PreDeConStream needs an initial set of data stream points to generate an initial set of microclusters for the online part. Therefore a certain amount of stream data is buffered and on this initial data, the points are found, whose neighborhood contains at least $\beta\mu_N$ points in its ε_F-neighborhood. If a point p is found, a potential microcluster is created by p and all the points in its neighborhood and they are removed from the initial points. This is repeated until no new potential microcluster is found. Finally the generated initial potential microclusters are inserted into the corresponding lists and the initial clustering is computed with an adapted version of PreDeCon [6].

Algorithm 1 PreDeConStream$(DS, \varepsilon_N, \mu_N, \lambda, \varepsilon_F, \mu_F, \beta, \tau)$

```
1:  $T_p \leftarrow \left\lceil \frac{1}{\lambda}\log_2\left(\frac{1}{1-\beta\mu_N(1-2^{-\lambda})}\right) - 1 \right\rceil$;
2:  $T_d \leftarrow \left\lceil \frac{1}{\lambda}\log_2(\beta\mu_N) \right\rceil$;
3:  $T_v \leftarrow \min\{T_p, T_d\}$;
4:  initialisation phase
5:  repeat
6:      get next point $p \in DS$ with the current timestamp $t_c$
7:      process(p);
8:      maintain microclusters in data structure
9:      if $(t_c \mod T_v) == 0$ then
10:         $C \leftarrow$ updateClustering(C);
11:     end if
12:     if user request clustering is received then
13:         return clustering $C$
14:     end if
15: until data stream terminates
```

Offline: Maintenance of the Resulting Subspace Clustering In Algorithm 2, the new arriving data points $p \in DS$ of the stream data within timestamp t are merged with the existing microclusters. In Lines 1-4 of Algorithm 2, the nearest potential microcluster c_p is searched for in all the lists of the potential microclusters l^p. The algorithm clusters the incoming point $p \in DS$ tentatively to c_p to check, if the point actually fits into the potential microcluster c_p. If the radius r_p of the temporary microcluster c_p is still less than ε_F, the point can be clustered into c_p without hesitation. If p does not fit into the nearest potential microcluster c_p, (cf. Lines 5-12 of Algorithm 2), the algorithm searches for the nearest outlier microcluster c_o in the outlier lists l^o. The algorithm checks again if its radius r_o of c_o is still less than ε_N, when the point is tentatively added to c_o. If the point fits into c_o and it is in the highest list of the outliers l^o, the algorithm checks if the weight w_o is greater than or equal to $\beta\mu_N$. If that is the case, the microcluster c_o is inserted into the lowest list $l^p_{\min}$ of the potential microclusters and has to be considered in the offline part. If the point does not fit into any existing microcluster, a new outlier microcluster is created with this point and is inserted into the outlier microcluster list l^o_0, (cf. Line 14).

Algorithm 2 process(data point p)

```
 1: search nearest potential microcluster $c_p$ in all the lists $l^p$
 2: merge $p$ tentatively into $c_p$
 3: if $r_p \leq \varepsilon_N$ then
 4:    insert $p$ into $c_p$
 5: else
 6:    search nearest outlier microcluster $c_o$ in all the lists $l^o$
 7:    merge $p$ tentatively into $c_o$
 8:    if $r_o \leq \varepsilon_N$ then
 9:       insert $p$ into $c_p$
10:       if $w_o \geq \beta\mu_N$ then
11:          insert $c_o$ into potential list $l^p_{\min}$ and remove it from outlier list $l^o_{\max}$
12:       end if
13:    else
14:       create new outlier microcluster with $p$
15:    end if
16: end if
```

Online: Processing of the Data Stream In Lines 1-10 of Algorithm 3, for each newly created potential microcluster c_p, its subspace preference vector w_{c_p} is computed. Furthermore, for each potential microcluster $c_q \in \mathcal{N}_{\varepsilon_F}(c_p)$, the preference subspace vector of each c_q is updated and checked if its core member property has changed. If that is the case, it is added to the $UPDSEED_i$ set. In Lines 11-19 of Algorithm 3, all the potential microclusters are found which are affected by removing the potential microclusters which faded out in the online part. For each potential microcluster $c_q \in \mathcal{N}_{\varepsilon_F}(c_d)$, the preference subspace vector of each c_q is updated and added to $UPDSEED_d$ if the core member property of c_q has changed because of deleting c_d out of its ε_F-neighborhood. If all the affected potential microclusters were found, $UPDSEED_i$ and $UPDSEED_d$ can be merged to $UPDSEED$. Finally in Lines 20-22, the potential microclusters of $UPDSEED$ need to be reinserted into the clustering. Starting from a poten-

Algorithm 3 updateClustering(C)

```
for all $c_p \in Inserted\_PMC$ do
    compute the subspace preference vector $w_{c_p}$
    for all $c_q \in \mathcal{N}_{\varepsilon_F}(c_p)$ do
        update the subspace preference vector of $c_q$
        if core member property of $c_q$ has changed then
            add $c_q$ to $AFFECTED\_CORES_i$
        end if
    end for
    compute $UPDSEED_i$ based on $AFFECTED\_CORES_i$
end for
for all $c_d \in Deleted\_PMC$ do
    for all $c_q \in \mathcal{N}_{\varepsilon_F}(c_d)$ do
        update the subspace preference vector of $c_q$
        if core member property of $c_q$ has changed then
            add $c_q$ to $AFFECTED\_CORES_d$
        end if
    end for
    compute $UPDSEED_d$ based on $AFFECTED\_CORES_d$
end for
$UPDSEED \leftarrow UPDSEED_i \cup UPDSEED_d$
for all $c_p \in UPDSEED$ do
    call expandCluster() of PreDeCon [6] by considering the old cluster structure;
end for
```

tial microcluster in $UPDSEED$, the function *expandClusters()* of the algorithm PreDeCon [6] is called under consideration of the existing clustering. This is repeated until all the potential microclusters $c_p \in UPDSEED$ are clustered into a cluster or marked as noise.

5 Experimental Evaluation

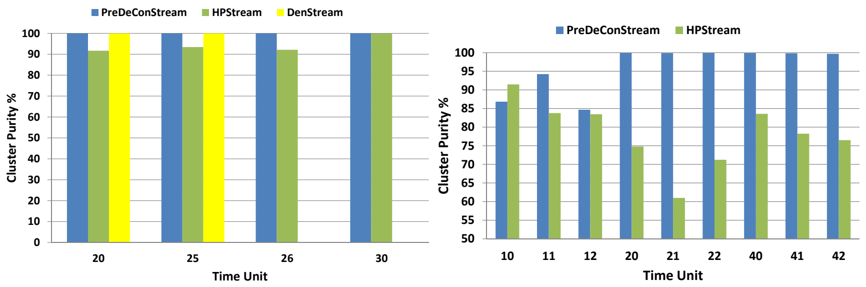
In this section, the experimental evaluation of PreDeConStream is presented. PreDeConStream, as well as the two comparative algorithms, HPStream as a k-means based projected algorithm and DenStream as a fullspace density based algorithm, were implemented in Java. All the experiments were done on a Linux operating system with a 2.4 GHz processor and 3GB of memory.

Datasets For the evaluation of PreDeConStream several datasets were used:
1. *Synthetic Dataset: SynStream3D* consists of 3-dimensional 4000 objects without noise that form at the beginning two arbitrarily shaped clusters over full space. After some time, the data stream evolves so that for each cluster different dimensions of both clusters become irrelevant.
2. *Synthetic Dataset: N100kC3D50R40* generated similar to [7] with 100000 data objects forming 3 clusters with 40 relevant dimensions out of 50.

3. *Real Dataset: Network Intrusion Detection data set KDD CUP'99 (KDD-cup)[1]* used to evaluate several stream clustering algorithms [3, 7] with 494021 TCP connections, each represents either a normal connection, or any of 22 different types of attacks (cf. Figure 4). Each connection consists of 42 dimensions.
4. *Real Dataset: Physiological Data, ICML'04 (PDMC)* is a collection of activities which was collected by test subjects with wearable sensors over several months. The data set consists of 720792 data objects, each data object has 15 attributes and consists of 55 different labels for the activities and one additional label if no activity was recorded.

Evaluation measure and Parameter settings To evaluate the quality of the clustering results, the cluster purity measure [3, 7] is used. For the efficiency, the runtime in seconds was tested. In PreDeConStream the offline parameters ε_F and μ_F specify the density threshold that the clusters must exceed in the offline algorithm part. A lower bound for ε_F is the online parameter ε_N. In the experiments, ε_F was set to at least $2 \times \varepsilon_N$. Unless otherwise mentioned, the parameters for PreDeConStream were set similar to [7] as follows: decay factor $\lambda = 0.25$, initial data object $Init = 2000$, and horizon $H = 5$.

Experiments Purity and runtime were tested for the three algorithms.
A. Evaluation of Clustering Quality: Using the SynStream3D dataset for both PreDeconStream and Denstream, the online parameters are set to $\varepsilon_N = 4$, $\mu_N = 5$ and $\lambda = 0.25$. For PreDeConStream, the maximal preference dimensionality is set to $\tau = 2$ and $\mu_F = 3$. The stream speed was set to 100 points per time unit. With this speed setting, the data stream evolves at timestamp 26, i.e. one dimension for each cluster becomes irrelevant. It can be seen from Figure 3(a) that the cluster purity of PreDeConStream and DenStream is 100% until the data stream changes, which is not the case for HPStream. This is because both can detect clusters with arbitrarily cluster shapes. Beginning from time unit: 26 the stream evolvs such that in each cluster one dimension is no longer relevant and thus DenStream as a fullspace clustering algorithm, does not detect any cluster. Similarly, Figure 3(b) shows the purity results of both algorithms over the N100C3D50R40 dataset. It can be seen that PreDeConStream outperforms HPStream. The time units are selected in such a way that the changes of the cluster purity can be observed when the stream evolves. It can be observed that HPStream has problems with detecting the changes in the stream. That is because the radius of the projected clusters might be too high and the new points are clustered wrong. PreDeConStream adapts to the changes in the stream fast and keeps a high cluster purity. On the Network Intrusion data set, the stream speed was set to 1000 points per time unit. Since the Network Intrusion data set was already used in [3, 7], the same parameter settings are chosen for DenStream and HPStream as in [3, 7]. Since PreDeConStream also builds on a microcluster structure, similar parameters settings for the online part of PreDeConStream are chosen, to have a fair comparison. For PreDeConStream: $\beta = 1.23$, $\mu_F = 5$, and $\tau = 32$. For all the three algorithms, the decaying factor λ is set to 0.25.



(a) (b)

Fig. 3. Clustering purity for: (a) SynStream3D dataset, (b) N100C3D50R40 dataset.

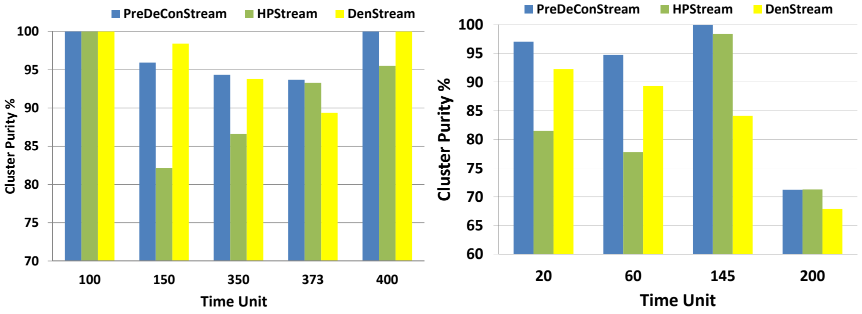
Figure 5(a) shows the purity results for the KDDcup Dataset. It can be seen that PreDeConStream produces the best possible clustering quality. For the evaluation, measurements at timestamps where some attacks exist were selected. The data recordings at timestamp 100 and all the recordings within the horizon 5 were only attacks of the type "smurf". At this time unit any algorithm could achieve 100% purity. The attacks that appeared within horizon $H = 5$ in different timestamps are listed in Figure 4. By comparing Figures 4 and 5(a), one can observe that PreDeConStream is also resistant against outliers. At the timestamp 350 and 400 there were some outlier attacks within the horizon which affected other algorithms less than PreDeConStream.

Normal or	Objects within horizon $H = 5$ at time unit			
attack Type	150	350	373	400
normal	4004	4097	892	406
satan	380	0	0	0
buffer overflow	7	1	2	0
teardrop	99	99	383	0
smurf	143	0	819	2988
ipsweep	52	182	0	0
loadmodule	6	0	0	1
rootkit	1	0	0	1
warezclient	307	0	0	0
multihop	0	0	0	0
neptune	0	618	2688	1603
pod	0	1	99	0
portsweep	0	1	117	1
land	0	1	0	0
sum	5000	5000	5000	5000

Fig. 4. Labels of KDDcup data stream within the horizon $H = 5$, stream speed = 1000.

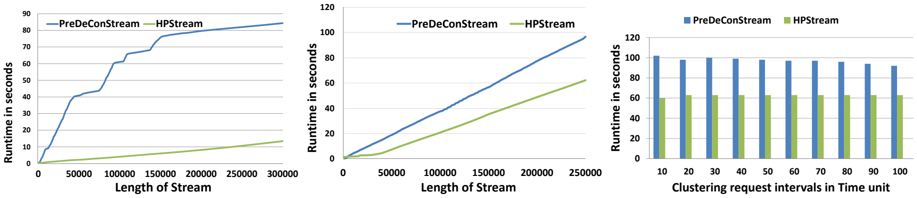
Figure 5(b) shows the purity results over the Physiological dataset. The stream speed = 1000 and $H = 1$. Again, the timestamps were selected in such a way that there are different activity labels within one time unit. It can be seen from Figure 5(b) that PreDeConStream has the highest purity.

B. Evaluation of Efficiency: The real datasets are used to test the efficiency of PreDeConStream against HPStream. The parameters were set the same way as for the previous experiments on these datasets and the results are shown in Figure 6. Although it is unfair to compare the runtime of a completely density-based approach against a k-means based one, but Figures 6(a) and 6(b) show



(a) (b)

Fig. 5. Clustering purity for: (a) KDDcup dataset, (b) PDMC dataset.



(a) (b) (c)

Fig. 6. Runtime results for: (a) KDDcup dataset with a clustering request at each time unit, (b) PDMC dataset with a clustering request at each 20th time unit, and (c) PDMC dataset with different clustering request intervals.

a considerable positive effect of our clustering maintenance model when the clustering requests frequency decreases. Usually, clustering requests are not extremely performed at each timestamp or even at each 20th timestamp. This fact motivated a further experiment where we tested the performance of the two algorithms for different clustering frequencies. The result which is depicted in Figure 6(c) confirms our assumption. Due to its clustering maintenance model, PreDeConStream performs better with higher clustering requests intervals. A further experimental evaluation of PreDeConStream was done in [20].

6 Conclusions and Future Work

In this paper we presented a novel algorithm termed *PreDeConStream*. Based on the two phase process of mining data streams, our technique builds a micro-cluster-based structure to store an online summary of the streaming data. The technique is based on subspace clustering targeting applications with high dimensionality of data. For the first time we have utilised projection, density-based clustering, and cluster fading. As a result our technique has proved experimentally its superiority over state-of-the-art techniques. In the future we plan to deploy the technique in a real sensor network testbed, in order to prove its feasibility. Furthermore, a thorough experimental study with different configuration of clusters in the network is planned.

References

1. KDD Cup 1999 Data: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
2. Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proc. of VLDB '03*, pages 81–92, 2003.
3. Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. of VLDB '04*, pages 852–863, 2004.
4. Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, pages 61–72, 1999.
5. Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proc. of ICDT '99*, pages 217–235, 1999.
6. Christian Bohm, Karin Kailing, Hans-Peter Kriegel, and Peer Kroger. Density connected clustering with local subspace preferences. In *Proc. of ICDM '04*, pages 27–34, 2004.
7. Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proc. of SDM '06*, pages 328–339, 2006.
8. Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proc. of KDD '07*, pages 133–142, 2007.
9. Martin Ester, Hans-Peter Kriegel, S Jörg, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*, pages 226–231, 1996.
10. Marwan Hassani, Philipp Kranen, and Thomas Seidl. Precise anytime clustering of noisy sensor data with logarithmic complexity. In *Proc. SensorKDD '11 Workshop in conj. with KDD '11*, pages 52–60, 2011.
11. Marwan Hassani, Emmanuel Müller, and Thomas Seidl. EDISKCO: energy efficient distributed in-sensor-network k-center clustering with outliers. In *Proc. SensorKDD '10 Workshop in conj. with KDD '09*, pages 39–48, 2009.
12. Marwan Hassani and Thomas Seidl. Towards a mobile health context prediction: Sequential pattern mining in multiple streams. In *Proc. MDM '11*, pages 55–57, 2011.
13. Ankur Jain, Zhihua Zhang, and Edward Y. Chang. Adaptive non-linear clustering in data streams. In *Proc. of CIKM '06*, pages 122–131, 2006.
14. Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *Poc. of SDM '04*, pages 246–257, 2004.
15. Hans-Peter Kriegel, Peer Kröger, Irene Ntoutsi, and Arthur Zimek. Towards subspace clustering on dynamic data: an incremental version of predecon. In *Proc. of StreamKDD workshop in conj. with KDD '10*, pages 31–38, 2010.
16. Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. on Knowledge Discovery from Data*, pages 1:1–1:58, 2009.
17. Guopin Lin and Leisong Chen. A grid and fractal dimension-based data stream clustering algorithm. In *ISISE '08*, pages 66 –70, 2008.
18. Irene Ntoutsi, Arthur Zimek, Themis Palpanas, Peer Kröger, and Hans-Peter Kriegel. Density-based projected clustering over high dimensional data streams. In *Proc. of SDM '12*, pages 987–998, 2012.
19. Nam Hun Park and Won Suk Lee. Grid-based subspace clustering over data streams. In *Proc. of CIKM '07*, pages 801–810, 2007.
20. Pascal Spaus. Density based model for subspace clustering on stream data. *Bachelor's thesis, Dept. of Computer Science, RWTH Aachen University*, May 2011.