Density-based Projected Clustering over High Dimensional Data Streams

Irene Ntoutsi[1], Arthur Zimek[1], Themis Palpanas[2], Peer Kröger[1], Hans-Peter Kriegel[1]

[1]*Institute for Informatics, Ludwig-Maximilians-Universität München, Germany*
{ntoutsi,zimek,kroeger,kriegel}@dbs.ifi.lmu.de

[2]*Information Engineering and Computer Science Department (DISI), University of Trento, Italy*
themis@disi.unitn.eu

Abstract

Clustering of high dimensional data streams is an important problem in many application domains, a prominent example being network monitoring. Several approaches have been lately proposed for solving independently the different aspects of the problem. There exist methods for clustering over full dimensional streams and methods for finding clusters in subspaces of high dimensional static data. Yet only a few approaches have been proposed so far which tackle both the stream and the high dimensionality aspects of the problem simultaneously. In this work, we propose a new density-based projected clustering algorithm, HDDStream, for high dimensional data streams. Our algorithm summarizes both the data points and the dimensions where these points are grouped together and maintains these summaries online, as new points arrive over time and old points expire due to ageing. Our experimental results illustrate the effectiveness and the efficiency of HDDStream and also demonstrate that it could serve as a trigger for detecting drastic changes in the underlying stream population, like bursts of network attacks.

1 Introduction

High dimensional data are collected in many scientific projects, humanity research, or business processes, in order to better understand the phenomena the researchers or managers are interested in. An abundance of attributes is observed and recorded without knowing whether these attributes are actually helpful in describing these phenomena. Among the features of a high dimensional dataset, for any given object of interest, many attributes can be expected to be irrelevant for describing that object. Irrelevant attributes can easily obscure clusters that are clearly visible when we consider only the relevant 'subspace' of the dataset. Furthermore, the relevance of certain attributes may be *local*, i.e., it may differ for different clusters of objects within the same dataset. Thus, clusters may be meaningfully defined by some of the available attributes only. The irrelevant attributes will interfere with the efforts to find these clusters. This problem is aggravated in streaming data, since we have to decide with one single look at the data which attributes may be useful for describing a potential cluster for the current object. Moreover, streams are volatile and the discovered clusters might also evolve over time either slightly (concept drift) or rapidly (concept shift). The problem of high dimensional data has been a topic for clustering since years [17]. Clustering data streams has been a research topic as well [1, 7, 8, 11, 13]. The combination of both issues, i.e., clustering in subspaces of high dimensional stream data, has however gained little attention so far. The only example of such a combined method for streaming data is HPStream [2], a partitioning method for projected clustering. As a k-means-type approach though, it assumes that the number of clusters remains constant over the whole lifetime of the stream. Other approaches to clustering high dimensional data streams are either incremental (requiring access to raw data), or are not able to detect clusters in subspaces, or both.

Here, we propose a new density-based projected clustering algorithm for data streams, HDDStream. Density-based clustering methods are more appropriate for streams than e.g., partitioning methods since they do not make any assumption on the number of clusters, they can discover clusters of arbitrary shapes, and they are invariant to outliers. A challenge though for density-based clustering is that there is no clustering model for the discovered clusters, which are usually described as sets of objects. Such a description however is not applicable to streams, where data objects cannot be accessed in an unlimited fashion. A common way to overcome this issue is to summarize the dataset by some appropriate summary, e.g., micro clusters [1]. We propose a summary that models both objects and

dimensions, since in high dimensional feature spaces not all dimensions might be important for summarizing a set of data objects. We maintain the summaries in an online fashion and we rely upon these summaries in order to derive the final clusters and their relevant dimensions.

To summarize, our main contributions are:

(i) HDDStream is the first algorithm for density-based projected clustering over high dimensional data streams. Although there exist methods for density-based clustering over full dimensional data streams, e.g., [7, 8], and density-based clustering over high dimensional static data, e.g., [6,18], there is no algorithm that simultaneously tackles both the high dimensionality and the stream aspects of the data.

(ii) We propose a summary structure, projected microclusters, for summarizing a set of objects in their relevant dimensions. We propose different types of microclusters in order to allow for the gradual formation of true projected microclusters and the safe removal of outliers. This is important for streams, since in a stream environment clusters may become outliers later on and vice versa. This is in contrast to HPStream [2], where a new point, if it is far away from all existing clusters, will become the seed of a new cluster (even if it is an outlier), and some old cluster is deleted in order to keep the total number of clusters constant.

(iii) HDDStream adapts the number of microclusters to the underlying evolving population thus allowing the end user to monitor the evolution of the population and to detect possible drastic changes in the population. For example, a drastic increase in the variety of activities in a network monitoring application might be actionable as it could correspond to some burst of attacks. Such a property is not possible with HPStream [2] since it maintains a constant number of clusters over time.

In the remainder, we discuss related work in Section 2, and introduce basic concepts in Section 3. The HDDStream algorithm is presented in Section 4. Section 5 presents the evaluation analysis. Section 6 concludes the paper.

2 Related work

2.1 Density-based clustering Density-based clustering [15] can be seen as a non-parametric approach, where clusters are modeled as areas of high density (relying on some unknown density-distribution). In contrast to parametric approaches that try to approximate the unknown density-distribution generating the data by mixtures of k densities (e.g., Gaussian distributions), density-based clustering methods do not require the number of clusters as input and do not make any specific assumptions concerning the nature of the density-distribution. As a result, however, density-based methods do not readily provide models, or otherwise compressed descriptions for the discovered clusters. A computationally efficient method for density-based clustering on static data sets is, e.g., DBSCAN [10]. Incremental variants have been proposed to capture changes of the database over time, e.g., incDBSCAN [9]. Incremental algorithms require access to the raw data for the reorganization of the clustering, which is a requirement that cannot be met in a data streaming context, where access to the entire history of the data is not feasible.

2.2 Clustering high dimensional data Clustering high dimensional data has found a lot of attention, though mostly focused on static data so far. The primary idea is that clusters can no longer be found in the entire feature space because many features are irrelevant for the clustering. Often, in such high dimensional data, we observe the phenomenon of local feature relevance or local feature correlation (i.e., different subsets of features are relevant/correlated for different clusters). Thus, clusters can only be detected in subspaces rather than in the entire feature space. The recent approaches can be classified [17] into methods that search for clusters in arbitrarily oriented subspaces grasping the idea of local feature correlation (thus also called correlation clustering algorithms) as well as methods that search clusters only in axis-parallel subspaces accounting for local feature relevance (called subspace or projected clustering algorithms). Here, we focus on the latter class restricting the search space to axis-parallel subspaces. We can distinguish between methods that search the relevant subspaces for each cluster bottom-up starting with 1D subspaces (like the grid-based approach CLIQUE [4]) and approaches that search for the relevant projections of each cluster top-down (like the k-means like PROCLUS algorithm [3] or the density-based PreDeCon [6]). While the bottom-up approaches usually compute all clusters in all subspaces, allowing multiple cluster memberships (often called subspace clustering methods), the top-down approaches usually compute a disjoint, non-overlapping partition of the data points into clusters where each cluster may aggregate in a different projection (these are often called projected clustering methods). Since subspace clustering algorithms usually provide a lot of redundancy, here, we focus on the problem of projected clustering aiming at a partition into disjoint groups. In particular, we follow the density-based clustering model that has been successfully used for projected clustering in PreDeCon [6] and several variations [19,20] for the static case, and we extend these concepts to highly dynamic data streams.

2.3 Stream clustering in full dimensional space Data streams impose new challenges for the clustering problem since *"it is usually impossible to store an entire data stream or to scan it multiple times due to its tremendous volume"* [12]. Several methods have been proposed that first summarize the data by some summary structure and then apply clustering over these summaries instead of the original raw data. STREAM [13] proposes a technique for clustering stream data that works by segmenting the original data stream in time-chunks. Then, a clustering is produced for each chunk, which serves as its summary, and periodically all these summaries are processed to produce an overall clustering. The CluStreams [1] framework splits the clustering process into an online and an offline part: the online component incrementally maintains a summary of the data stream (the so called 'micro-clusters') and periodically stores them to disk, whereas the offline component applies a variation of k-means over these micro-clusters for the formation of the actual clusters (the so called 'macro-clusters') over a user-defined time horizon. The micro-clusters are based on the concept of clustering features (CFs) originally introduced in BIRCH [22] for summarizing the data through spherical clusters. DenStream [7] follows the online-offline rationale of CluStream [1], but in contrast to CluStream (that is specialized to spherical clusters), it can detect clusters of arbitrary shapes, following the density-based clustering paradigm. Data are summarized through micro clusters and the clusters with arbitrary shapes are described by a set of micro clusters. The algorithm distinguishes between outlier micro clusters and potential core micro clusters, thus allowing noise handling. DStream [8] uses a grid structure to capture the data distribution. The grid is updated online and clusters are extracted as sets of connected dense units. To deal with noise, they distinguish between dense, transitional, and sparse units based on the unit density. DUCStream [11] is based on CLIQUE [4]. It updates incrementally only the full-dimensional grid structure, whereas the clusters are discovered over the (now updated) grid. Also, ClusTree [14], a micro clusters tree-structure that is updated like an index with updates of the data has been proposed for anytime stream clustering.

Though a lot of research has been carried out on clustering stream data, all these clustering approaches tackle clustering in the full-dimensional space only and thus might miss clusters in high dimensional data, where clusters are likely to be present in subspaces only.

2.4 Projected clustering over high dimensional stream data The ideas of CluStream [1] have been extended to high dimensional data streams in HPStream [2], a method for projected clustering over data streams. A summary structure, the 'fading cluster structure', comprises a condensed representation of the statistics of the points inside a cluster and can be updated efficiently as the data stream proceeds. Each summary is associated with a projected subspace, consisting of the dimensions that are the most relevant for the cluster. The algorithm requires as input the number of clusters k and the average cluster dimensionality l. When a new point arrives, it is assigned to its closest cluster or it starts a new cluster. In the latter case, though, the oldest of the existing clusters is deleted in order to keep the total number of clusters constant. Recently, there have been also some approaches to adapt projected clustering for handling dynamic data leading to incremental clustering [16, 21]. These approaches allow for updates or changes of the data but, as opposed to stream clustering, require access to the raw data for updating the clusters accordingly.

2.5 Summary Although a plethora of work tackled subspace or projected clustering in high dimensional static data and several methods have been proposed for clustering over data streams, only HPStream [2] actually deals with the problem of clustering in subspaces of high dimensional stream data. However, HPstream relies on the k-means clustering model and, thus, the number of clusters is required as input and is assumed to remain constant over the complete stream. Note that in a streaming context we do not have the opportunity to try out different values of k or even different runs with the same k in order to get the best result. Since this is usually required for k-means-type approaches in order to select the most convincing solution afterwards, this is a significant limitation. Contrary to this, the proposed HDDStream algorithm is based on a density-based notion of clusters. The number of clusters is variably adjusted over time, depending on the evolution of the underlying dataset. The clusters can be of arbitrary shape, naturally following the data characteristics.

3 Basic concepts

A data stream is defined as an infinite sequence of points $\{p_1, p_2, \ldots p_i, \ldots\}$ arriving over time at different time points $t_1, t_2, \ldots t_i, \ldots$, respectively. Each point is described as a vector $p_i = \langle p_{i_1}, p_{i_2}, \ldots p_{i_d} \rangle$ in the d-dimensional feature space.

An important concept in data streams is data ageing. In order to give a greater level of importance to more recent data, a weight is assigned to every point via an ageing function. In this paper, we adopt the *exponential fading function* that is widely used in temporal applications. According to this function, the

weight of a point decreases exponentially with time t via $f(t) = 2^{-\lambda \cdot t}$, where $\lambda > 0$. The decay rate λ determines the importance of historical data; the higher the value of λ, the lower the importance of old data.

An important characteristic of data streams is the inability to store all data points. However, in density-based clustering, clusters of arbitrary shapes are described in terms of all their member-points. Such a requirement though is prohibitive for data streams. A usual way to overcome this problem is by summarizing the data through an appropriate *summary* structure e.g., [1,2,7,8]. A popular technique employed by several clustering algorithms, e.g., [1, 2, 7], are micro clusters that summarize a set of points in the full dimensional space. When ageing is considered, as in our case, the temporal extension of micro clusters [2, 7] is employed.

We first define the microclusters for summarizing a set of points in the full dimensional feature space. We then introduce the notion of dimension preferences to consider the fact that in high dimensional feature spaces not all dimensions are relevant for a microcluster. Based on dimension preferences we introduce the *projected microclusters* which summarize a set of points in a subspace of the original feature space.

DEFINITION 1. (MICROCLUSTER - MC)
A microcluster at time t for a set of d-dimensional points $C = \{p_1, p_2, \ldots, p_n\}$ arriving at different time points is defined as a tuple $mc(C,t) = (\overline{CF1(t)}, \overline{CF2(t)}, W(t))$ where:

- *$\overline{CF1(t)}$ is a d-dimensional vector of the weighted linear sum of the points in each dimension: $\overline{CF1(t)} = \langle CF1_1(t), CF1_2(t), \ldots, CF1_d(t)\rangle$. Entry $CF1_j(t)$ refers to dimension j and is given by: $CF1_j(t) = \sum_{i=1}^{n} f(t-t_i) \cdot (p_{i_j})$, where p_{i_j} is the value of point p_i in dimension j, t_i is the arrival time of p_i and $f(t-t_i)$ is the weight of p_i at t.*
- *$\overline{CF2(t)}$ is a d-dimensional vector of the weighted square sum of the points in each dimension: $\overline{CF2(t)} = \langle CF2_1(t), CF2_2(t), \ldots, CF2_d(t)\rangle$. Entry $CF2_j(t)$ corresponds to the j^{th} dimension and is given by: $CF2_j(t) = \sum_{i=1}^{n} f(t-t_i) \cdot (p_{i_j})^2$.*
- *$W(t)$ is the sum of the weights of the data points: $W(t) = \sum_{i=1}^{n} f(t-t_i)$.*

A microcluster summarizes a set of points in the full d-dimensional space. Different dimensions though might be of different importance for the microcluster. We evaluate the preference for a dimension on the basis of the variance along this dimension in the microcluster.

DEFINITION 2. (PREFERRED DIMENSION)
Let $mc(C,t) = (\overline{CF1(t)}, \overline{CF2(t)}, W(t))$ be a microcluster, shortly denoted by mc. The microcluster mc prefers the j^{th} dimension iff:

$$\text{VAR}_j(mc) \leq \delta$$

where $\text{VAR}_j(mc)$ is the variance along dimension j:

$$\text{VAR}_j(mc) = \sqrt{\frac{CF2_j(t)}{W(t)} - \left(\frac{CF1_j(t)}{W(t)}\right)^2}$$

and δ is the variance threshold.

Intuitively, a microcluster prefers a dimension, if the members of the microcluster are densely packed along this dimension. The variance threshold parameter δ controls whether a dimension should be considered as a preferred or as a non-preferred dimension.

Based on the preference of a microcluster for a single dimension, we define the dimension preference vector of a microcluster to distinguish between preferred and non-preferred dimensions.

DEFINITION 3. (DIMENSION PREFERENCE VECTOR)
Let mc be a microcluster. The dimension preference vector of mc is defined as:

$$\overline{\Phi(mc)} = \langle \phi_1, \phi_2, \ldots \phi_d \rangle$$

where ϕ_j, $j = 1:d$ is given by:

$$\phi_j = \begin{cases} \kappa, & \text{if } \text{VAR}_j(mc) \leq \delta \\ 1, & \text{otherwise} \end{cases}$$

$\kappa \gg 1$ is a constant.

The number of dimensions that a microcluster prefers, comprises the *projected dimensionality* of the microcluster. For a microcluster mc, its projected dimensionality, denoted by $\text{PDIM}(mc)$, can be easily computed through its dimension preference vector $\overline{\Phi(mc)}$ by counting the κ-valued entries.

A microcluster accompanied with a dimension preference vector is called a *projected microcluster*, where the term 'projected' stands for the fact that the microcluster is defined over a projected subspace of the feature space instead of the whole feature space.

We introduce now the notion of core projected microclusters which is important for density-based clustering. A core projected microcluster is a microcluster that 'compresses' more than μ points within a limited radius ϵ in a projected subspace of maximal dimensionality π. More formally:

DEFINITION 4. (CORE–PMC)
Let mc be a microcluster and let $\overline{\Phi}$ be its dimension preference vector. The microcluster mc is a core projected microcluster iff:

i) $radius^{\overline{\Phi}}(mc) \leq \epsilon$,
ii) $W(t) \geq \mu$ *and,*
iii) $\text{PDIM}(mc) \leq \pi$.

Note that in the above definition, the radius is defined w.r.t. the dimension preference vector $\overline{\Phi}$ of the microcluster. This is the projected radius that takes into account the dimension preferences of the microcluster:

DEFINITION 5. (PROJECTED RADIUS)
Let mc be a microcluster and let $\overline{\Phi}$ *be its dimension preference vector. The projected radius of mc is given by:*

$$radius^{\overline{\Phi}}(mc) = \sqrt{\sum_{j=1}^{d} \frac{1}{\Phi_j} \left(\frac{CF2_j(t)}{W(t)} - \left(\frac{CF1_j(t)}{W(t)} \right)^2 \right)}$$

In evolving data streams, the role of outliers and clusters often exchange and what is now considered to be an outlier might turn later into a cluster or vice versa. To this end, we introduce the notions of pCORE–pMC and o–MC to distinguish between potential core projected microclusters and outlier microclusters.

DEFINITION 6. (pCORE–pMC)
Let mc be a microcluster and let $\overline{\Phi}$ *be its dimension preference vector. The microcluster mc is a potential core projected microcluster iff:*
i) $radius^{\overline{\Phi}}(mc) \leq \epsilon$,
ii) $W(t) \geq \beta \cdot \mu$ *and,*
iii) $\text{PDIM}(mc) \leq \pi$.

The only difference to CORE–pMC lies in condition ii): the density threshold for core projected microclusters is relaxed thus allowing potential core projected microclusters to compress a certain percentage $\beta \in (0,1)$ of μ. The maximal projected dimensionality threshold π is not relaxed though; the reason is that if so, the final microclusters might not be projected.

However, there might be microclusters that do not fulfill the above constraints either because their density is smaller than $\beta \cdot \mu$ or because their projected dimensionality exceeds π. We treat them as outliers.

DEFINITION 7. (OUTLIER MC, o–MC)
Let mc be a microcluster and let $\overline{\Phi}$ *be its dimension preference vector. The microcluster mc is an outlier microcluster iff:*
i) $radius^{\overline{\Phi}}(mc) \leq \epsilon$ *and, either*
ii) $W(t) < \beta \cdot \mu$ *or*
iii) $\text{PDIM}(mc) > \pi$.

The microclusters can be maintained online as new points arrive from the stream and old points expire due to ageing. This is possible due to additivity and temporal multiplicity properties of the microclusters.

PROPERTY 3.1. (ONLINE MAINTENANCE)
Consider a microcluster mc at time t, $mc = (\overline{CF1(t)}, \overline{CF2(t)}, W(t))$.

- *Additivity: If a point p is merged to mc at time t, mc can be updated as follows:* $mc = (\overline{CF1(t)} + p, \overline{CF2(t)} + p^2, W(t) + 1)$.
- *Temporal multiplicity: If no points are added to mc during the interval* $(t, t + \delta t)$*, the components of mc are downgraded by a factor* $2^{-\lambda \cdot \delta t}$ *and the updated microcluster is given by:* $mc = (2^{-\lambda \cdot \delta t} \cdot \overline{CF1(t)}, 2^{-\lambda \cdot \delta t} \cdot \overline{CF2(t)}, 2^{-\lambda \cdot \delta t} \cdot W(t))$.

The additive property allows for the easy integration of new points into an existing microcluster. The multiplicity property allows for the easy update of the recency of microclusters over time.

Being able to maintain the microclusters online is very important for streams since there is no access to the original raw data. Note also that all the features derived from the microclusters, e.g., radius, variance, or dimension preference vector, can be computed online.

4 The HDDSTREAM algorithm

The pseudocode of the HDDSTREAM algorithm is presented in Figure 1. It consists of three main steps:
(i) *Initialization:* After the arrival of the first *initPoints* points from the stream, the initial set of microclusters is extracted (lines 6–9, Figure 1). The initialization step is explained in Section 4.1.
(ii) *Online microcluster maintenance:* The microclusters are maintained online as new points arrive over time and old points expire due to ageing. A new point might be assigned to an existing microcluster, or it might start its own microcluster (lines 10–22, Figure 1). The online step is explained in Section 4.2.
(iii) *Offline clustering:* The final clusters are extracted on demand based on the so far maintained microclusters. The offline step is presented in Section 4.3.

4.1 Initialization We apply PreDeCon [6] on the first *initPoints* points from the stream $\{D\}$, to extract the initial set of microclusters. In particular, for each point $p \in D$, we compute its d-dimensional neighborhood $\mathcal{N}_\varepsilon(p)$ containing all points within distance ε from p. Based on the variance along each dimension in the neighborhood, we compute the dimension preference vector of p, $\overline{\Phi}(p)$ and the preferred neighborhood of p, $\mathcal{N}_\varepsilon^{\overline{\Phi}(\mathbf{p})}(p)$ containing all points within preference weighted distance ε from p. If the projected dimensionality of p does not exceed the maximal projected dimensionality π and the density in its preferred neighborhood is above $\beta\mu$, we create a *potential core projected*

```
Algorithm HDDStream()
Input stream S = p_1, p_2 ..., p_t, ...
1.  while (stream has more instances) do
2.    p: the next point from the stream at t
3.    //initialize upon the arrival of the first initPoints
4.    if (!initialized)
5.         initBuffer.add(p);
6.         if (|initBuffer| == initPoints) then
7.           pCore–pMicroClusters=init(initBuffer);
8.           initialized =true;
9.         end if
10.   else
11.        //try adding p to a potential microcluster
12.        Trial1 = add(p, pCore–pMicroClusters);
13.        if (!Trial1) then
14.          //try adding p to an outlier microcluster
15.          Trial2 = add(p, o–microClusters);
16.        end if;
17.        if (!Trial1 & !Trial2) then
18.          //start a new outlier microcluster
19.           Create a new outlier microcluster omc by p;
20.           o–microClusters.add(omc);
21.        end if;
22.   end if;
23.    //Periodic check pCore–pMicroClusters for
downgrade
24.   //Periodic check o–microClusters for removal
25. end while
```

Figure 1: Pseudo code of the HDDStream algorithm.

microcluster with p and all its preferred neighbors in D, i.e., $\{p \cup \mathcal{N}_{\varepsilon}^{\overline{\Phi}(\mathbf{p})}(p)\}$.

Since these points are already covered by a microcluster, we remove them from D and we repeat the same procedure for the remaining points in D. The result of this step is an initial set of potential core projected microclusters, pCore–pMicroClusters.

4.2 Online microcluster maintenance We maintain online two lists of microclusters: the potential core projected microclusters pCore–pMicroClusters and the outlier microclusters o–microClusters.

A new point p might be assigned to an existing microcluster or it might start its own microcluster; this depends on its proximity to the microclusters. In more detail, when a new point p arrives at time t:

1. We first try to add p to its closest potential core projected microcluster in pCore–pMicroClusters (line 12).

```
Algorithm add(p, pCore–pMicroClusters)
Input a new point p at t
      the list of pCore–pMicroClusters at t
  distances: array of distances w.r.t. p
1.    for each pmc ∈ pCore–pMicroClusters do
2.      //update the dimension preference vector of pmc
3.      prefDim = updateDimensionPreferences(pmc,p);
4.     if (prefDim ≤ π) then
5.        //compute projected distance
6.        dist = computeProjectedDistance(pmc,p);
7.        distances.add(dist);
8.     end if
9.    end for
10.  if (distances not empty) then
11.      //get the closest microcluster
12.      pmc_closest = getClosestSummary(distances);
13.     //check the radius
14.     radius = computeRadius(pmc_closest);
15.    if (radius ≤ ε) then
16.       Add p to pmc_closest;
17.       return true;
18.  end if
19.  return false;
```

Figure 2: Pseudo code of the add procedure

2. If this is not possible, we then try to add p to its closest outlier microcluster in o–microClusters (lines 13–16).

3. If both are not possible, we finally create a new outlier microcluster for p (lines 17–21).

We explain hereafter each step in more detail.

4.2.1 Adding p to pCore–pMicroClusters The procedure of adding p to some potential core projected microcluster in pCore–pMicroClusters consists of three steps: i) updating the dimension preferences of the microclusters, ii) computing the closest microcluster to p and, iii) finalizing the assignment. Each step is explained in detail below. The pseudocode of the algorithm is depicted in Figure 2.

Step 1 – Update dimension preferences We temporarily add p to each microcluster in pCore–pMicroClusters and compute the new projected subspaces of the microclusters (line 3) based on Definition 3. Due to the addition of p to a microcluster $pmc \in$ pCore–pMicroClusters, the projected subspace of pmc might be affected since the variance along some dimension j in pmc might change and thus, the j^{th} dimension might turn now into a preferred/non-

preferred dimension for pmc. In particular, for each dimension j, we compare the variance along this dimension in pmc before and after the addition of p. Three cases might occur:

- If the j^{th} dimension was a non-preferred dimension, it might turn now into a preferred dimension if after the addition of p, $\text{VAR}_j(pmc) \leq \delta$.
- If the j^{th} dimension was a preferred dimension, it might turn now into a non-preferred dimension if after the addition of p, $\text{VAR}_j(pmc) > \delta$.
- No changes in the preference of the j^{th} dimension occur due to the addition of p, that is, j remains either a preferred or a non-preferred dimension.

If after the addition of p, more dimensions turn into preferred dimensions, the projected dimensionality of pmc, $\text{PDIM}(mc)$, increases. Such an increase might violate condition ii) of Definition 6 regarding the maximum projected dimensionality π. If this condition is violated, we do not consider pmc as a candidate for the insertion of p and we proceed with the rest of the microclusters in pCORE–pMICROCLUSTERS (line 4–8). The reason is that we do not want to further 'develop' potential core microclusters that violate their definition.

Step 2 – Find the closest microcluster To find the closest microcluster, we compute the distance between p and every potential core projected microcluster $pmc \in$ pCORE–pMICROCLUSTERS, taking into account the updated projected subspace of pmc (lines 4–8). We call this *projected distance* since it relies on the projected subspace of a microcluster and we define it as follows.

DEFINITION 8. (PROJECTED DISTANCE) *Consider a point p at time t. Let pmc be a projected microcluster at t with dimension preference vector $\overline{\Phi}$. The projected distance between p and pmc is defined as follows:*

$$dist^{\overline{\Phi}}(p, pmc) = \sqrt{\sum_{j=1}^{d} \frac{1}{\Phi_j} (p_j - center_j)^2}$$

where center is the center of the microcluster given by: $\overline{center}(pmc) = \overline{CF1}(t)/W(t)$. The values $center_j$, p_j, Φj refer to the j^{th} dimension.

The projected distance between p and pmc is computed w.r.t. the dimension preferences of pmc. In particular, the value differences between the point and the center of the microcluster in each dimension are weighted based on the preference of the microcluster for that dimension. The closest microcluster $pmc_{closest} \in$ pCORE–pMICROCLUSTERS is chosen (line 12).

Step 3 – Finalize the assignment Although $pmc_{closest}$ was found to be the closest projected microcluster to p, the assignment is possible only if the addition of p to $pmc_{closest}$ does not affect the natural boundary of $pmc_{closest}$. This boundary is expressed by the projected radius of $pmc_{closest}$ (cf. Definition 5). If the radius is below the maximum radius threshold ε, p is added to $pmc_{closest}$ and the statistics of $pmc_{closest}$ are updated according to Property 3.1 (lines 15–18).

4.2.2 Adding p to o–MICROCLUSTERS

If p cannot be added to some potential core projected microcluster in pCORE–pMICROCLUSTERS, we try to add it to some outlier microcluster in o–MICROCLUSTERS (lines 13–16, Figure 1).

The procedure is similar as for adding p to pCORE–pMICROCLUSTERS (cf. Figure 2), so we do not explain here all the details. The difference is that, due to the definition of the outlier microclusters, there is no restriction on the maximal projected dimensionality, so the line 4 of the algorithm in Figure 2 is irrelevant now. We find the closest outlier microcluster $omc_{closest} \in$ o–MICROCLUSTERS for p by comparing p to all microclusters in o–MICROCLUSTERS. If the radius of $omc_{closest}$ does not exceed the radius threshold ε, we add p to $omc_{closest}$ and we update its statistics based on Property 3.1.

Due to the insertion of p, $omc_{closest}$ might turn into a potential core projected microcluster according to Definition 6 if now its weight exceeds $\beta \cdot \mu$ and its projected dimensionality does not exceed π (Note that the radius threshold ϵ should also hold for outlier microclusters.). If this is the case, we remove $omc_{closest}$ from the outlier microclusters list, o–MICROCLUSTERS, and we add it to the potential core projected microclusters list, pCORE–pMICROCLUSTERS.

4.3 Offline clustering

The online maintained projected microclusters capture the density of the stream, however they do not comprise the final clusters. To extract the final clusters, an offline procedure is applied on–demand over these microclusters. This procedure is a variant of PreDeCon [6] applied over microclusters instead of raw data. In traditional PreDeCon, the preference weighted core points are used as seeds for the 'creation' of the clusters. For each such point, an expansion procedure is applied starting with the points in its preferred neighborhood till the full cluster is revealed.

A similar procedure could be employed here: The core projected microclusters in pCORE–pMICROCLUSTERS act as seeds for the extraction of the projected clusters. In particular, for

each microcluster $pmc \in$ pCore–pMicroClusters we check whether it is a core projected microcluster (cf. Definition 4). If so, we start a projected cluster with it and all its neighbor microclusters in pCore–pMicroClusters and we use these neighbors as possible seeds for the further expansion of the cluster. We mark these microclusters as covered and we proceed with the remaining microclusters until all projected clusters are extracted.

4.4 Discussion In this section, we discuss different issues related to the HDDStream algorithm.

During the update of the projected dimensionality of the microclusters, we *temporarily* assign the new point p to each microcluster (Step 1, Section 4.2.1). This way, if p is assigned to that microcluster, the effect of p to the projected subspace of the microcluster is also taken into account. Another reason is that the computation of the variance along each dimension in the microcluster becomes more stable, especially in cases where the microcluster contains only a few points.

At each timepoint, we receive from the stream a certain number of points w, where w is the *window size*. Also, the old data are gradually forgotten based on the exponential fading function $f(t) = 2^{-\lambda \cdot t}$. However, the overall weight W of the data stream is *constant*. Let t_c be the current time, $t_c \to \infty$. The overall weight W of the stream at t_c is given by: $W = w \cdot 2^{-\lambda \cdot (t_c - 0)} + w \cdot 2^{-\lambda \cdot (t_c - 1)} + \ldots + w \cdot 2^{-\lambda \cdot (t_c - (t_c - 1))} + w \cdot 2^{-\lambda \cdot (t_c - t_c)} = w \cdot (2^{-\lambda \cdot 0} + \cdot 2^{-\lambda \cdot 1} + \ldots + \cdot 2^{-\lambda \cdot (t_c - 1)} + 2^{-\lambda \cdot t_c}) = w \cdot \frac{1}{1 - 2^{-\lambda}}$ So, the overall weight of the stream at time t_c depends on the window size w that determines how many data points arrive at each timepoint and on the decay rate λ that determines how fast the old data are forgotten.

In the previous sections, we described the update of a microcluster when a new instance is assigned to the microcluster. However, there might be microclusters that are not 'supported' by new instances. These microclusters should be also updated, since the data are subject to ageing and, thus, the microclusters are also subject to change over time. In particular, potential core projected microclusters may be downgraded to outliers and outlier microclusters may be vanished.

Obviously, updating all microclusters after each timepoint to account for the ageing of data points is not so efficient. To this end, we follow the approach of [7] and check the weight of each microcluster in pCore–pMicroClusters periodically, every T_{span} timepoints. T_{span} is the minimum time span such that a potential core projected microcluster that does not receive any new points from the stream may fade into an outlier microcluster, i.e.,: $2^{-\lambda \cdot T_{span}} \cdot \beta \cdot \mu = \beta \cdot \mu - 1$. By solving the above equation, T_{span} is computed as:

$$T_{span} = \left\lceil \frac{1}{\lambda} \cdot \log \frac{\beta \cdot \mu}{\beta \cdot \mu - 1} \right\rceil$$

So, every T_{span} timepoints we check each microcluster $pmc \in$ pCore–pMicroClusters. If either its projected dimensionality exceeds π or its weight is lower than $\beta \cdot \mu$, pmc is downgraded to an outlier microcluster[1]. This way also the maximum number of projected microclusters in memory at each timepoint is $\frac{W}{\beta \cdot \mu}$, where W is the weight of the stream and $\beta \cdot \mu$ is the minimum density of a potential core projected microcluster.

In case of outlier microclusters, it is more difficult to perform the ageing update, because there is neither some lower limit on their density, not some maximum limit on their projected dimensionality. A current outlier microcluster may be a real outlier that we want to delete or it may be the first point of a cluster. In the latter case, we want to keep it but the problem is that the time where the remaining members of that cluster come in can be far in the future. This would require to store these outlier microclusters for an infinite time. Since this is not feasible, we adopt the heuristic proposed in [7] to differentiate between real outliers and those that shall be upgraded later. In particular, after T_{span} timepoints, we check the real weight of each outlier microcluster $omc \in$ o–microClusters at the current time t with its lower expected weight at t. The lower expected weight of a microcluster at t is subject to its creation time, t_0 and is given by [7]:

$$W_{exp}(t, t_0) = \frac{2^{-\lambda(t - t_0 + T_{span})} - 1}{2^{-\lambda T_{span}} - 1}$$

The expected weight is 1 if $t = t_0$ and it approaches $\beta \cdot \mu$ as time elapses. So, the intuition is that the longer an outlier microcluster exists, the higher its weight should be. So, if omc evolves to become a potential core projected microcluster, its weight will be greater than the expected lower weight. Otherwise, it is most likely a real outlier. We delete omc if $W(omc) < W_{exp}$.

This procedure of ageing the microclusters takes place in lines 23–24 of the Algorithm in Figure 1. The result is an updated set of pCore–pMicroClusters and o–microClusters.

The cost of adding a new point from the stream depends on the number of the online maintained summaries: for each arriving point, we have to check whether it fits into an existing summary, resulting in a time complexity of $O(d \cdot (|$pCore–pMicroClusters$| + |$o–microClusters$|))$.

[1] Note that the weight of those microclusters receiving new points from the stream is updated after each addition.

Table 1: Parameters

Param	Description	Relevant to
λ	decay rate	stream
δ	variance threshold	dimensionality
π	maximal projected dimensionality	dimensionality
ε	radius threshold	clustering
μ	density threshold for Core–pMicroClusters	clustering
β $(\cdot\mu)$	density threshold for pCore–pMicroClusters	clustering

In Table 1 we summarize the different parameters of HDDStream grouped as stream relevant, dimensionality relevant and clustering relevant. Let us note that only β is actually introduced as a new parameter here. λ affects the decay rate of the stream and is commonly used by other stream mining algorithms. δ, π, ε, and μ are inherited from PreDeCon and discussed in [6].

5 Experimental evaluation

We compared HDDStream to HPStream [2] which to the best of our knowledge is the only projected clustering algorithm for high dimensional data streams so far. We experimented with the Network Intrusion and the Forest Cover Type datasets, which are typically used for the evaluation of stream clustering algorithms e.g., [1, 7] and they were also used in the experiments of HPStream [2]. Both algorithms were implemented in JAVA in the MOA framework [5].

We first describe the datasets and the evaluation measures we examined and then we present the results for each dataset.

5.1 Datasets The *Network Intrusion dataset (KDD Cup'99)* contains TCP connection logs from two weeks of LAN network traffic (424,021 records). Each record corresponds to a normal connection or an attack. The attacks fall into 4 main categories and 22 more specific types: DOS (i.e., denial-of-service), R2L (i.e., unauthorized access from a remote machine), U2R (i.e., unauthorized access to local superuser privileges), and PROBING (i.e., surveillance and other probing). We used all 34 continuous attributes as in [1, 2].

The *Forest Cover Type dataset* from UCI KDD Archive contains data on different forest cover types, containing 581,012 records. The challenge in this dataset is to predict the correct cover type from cartographic variables. The problem is defined by 54 variables of different types: 10 quantitative variables, 4 binary wilderness area attributes and 40 binary soil type variables. The class attribute contains 7 different forest cover types. We used all the 10 quantitative variables for our experiments as in [2].

In both cases, the datasets were turned into streams by sorting on the data input order.

5.2 Evaluation criteria Several full dimensional clustering algorithms, e.g., [1, 13] choose the sum of square distances (SSQ) to evaluate the clustering quality. However, "SSQ is not a good measure in evaluating projected clustering" [2] because it is a full dimensional measure. So, as in [2], we evaluate the *clustering quality* by the average purity of clusters, which examines the purity of the clusters w.r.t. the true class labels. The purity is defined as the average percentage of the dominant class label in each cluster [7]:

$$purity(\Theta) = \frac{\sum_{\theta \in \Theta} \frac{|C_\theta^d|}{|C_\theta|}}{|\Theta|}$$

where $\Theta = \{\theta_1, \theta_2, \ldots, \theta_k\}$ is the clustering result. $|C_\theta|$ denotes the number of points assigned to cluster C_θ and C_θ^d is the number of points in C_θ belonging to the dominant class d. In other words, purity describes how pure in terms of the dominant class the clusters are.

Due to the fact that the data points age over time, the purity is computed w.r.t. a set of points that arrive within a predefined window of time from the current time. We refer to this as the *horizon* H in order to remove any ambiguity with the *window size* w that defines the number of data points from the stream that arrive at each time point. So, H describes over how many windows the cluster purity is evaluated.

The *memory usage* is measured by the number of microclusters maintained at each time point.

Unless particularly mentioned, the stream parameters for both algorithms were set as follows: initial number of points $initPoints = 2,000$, decay rate $\lambda = 0.5$ and horizon $H = 1$.

5.3 Network Intrusion Dataset We tested the clustering quality of HPStream and HDDStream on the Network Intrusion Dataset. The parameters for HPStream are chosen to be the same as those adopted in [2]. The parameters for HDDStream were set as follows: density threshold for projected core microclusters $\mu = 10$, density factor for potential core projected microclusters $\beta = 0.5$, radius threshold $\varepsilon = 0.2$, maximal projected dimensionality $\pi = 30$, variance threshold $\delta = 0.001$.

In Figure 3, we display the purity of the clusters during the whole period of stream execution for a specific window size, w=1000 points per time. We can observe the change in the purity over time and also, that HDDStream achieves better purity compared to HPStream. Note that there are cases when both algorithms achieve the maximum purity of 100%. This is due to the

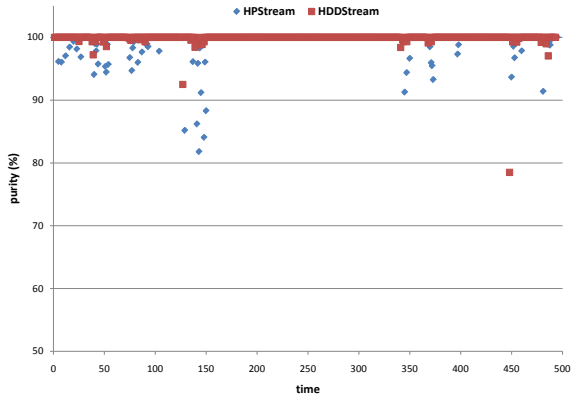

Figure 3: Clustering quality (Network Intrusion dataset, window size $w = 1000$)

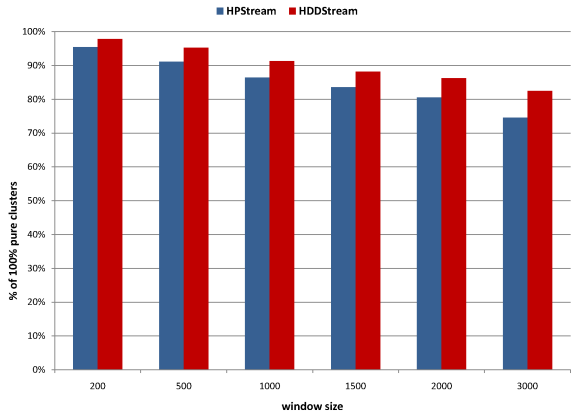

Figure 4: Clustering quality for different window sizes (Network Intrusion dataset)

fact that there are timepoints where all instances belong to the same connection type (e.g., timepoints of normal connections or timepoints with burst of attacks of a specific type). As such, any clustering result would obtain a purity of 100%.

For this reason, in Figure 4 we display the percentage of 100% pure clusters discovered by the two algorithms under different window sizes, varying from $w = 200$ to $w = 3,000$ points per each timepoint. As we can see, HDDStream outperforms HPStream for all window sizes. This is expected since HPStream tries to summarize the stream at each time point with only a constant number of clusters, whereas HDDStream adapts the number of microclusters at each time point to the characteristics of the incoming data. Moreover, HDDStream takes into account the fact that some points might correspond to noise and do not necessarily fit into an existing microcluster. In particular, the notion of o–microClusters allows for the true noisy points to be discarded (since they will not receive further points from the stream), and for the false noisy points to grow into actual microclusters (since they will be enriched with more points from the stream). On the contrary, HPStream creates a new cluster whenever a point does not fit into the existing clusters and it deletes the oldest from the previous clusters. So, if the new point corresponds to noise, a new cluster would be created and an old (possibly still valid) cluster would be deleted.

The memory usage is measured by the number of (micro)clusters maintained by each algorithm. In Figure 5, the number of (micro)clusters is depicted, for window size $w = 1000$ points per time. HPStream utilizes a constant number of $k = 23$ clusters over time (straight blue line), whereas HDDStream adjusts the number of microclusters to the incoming stream data. There are timepoints where HDDStream utilizes only a single microcluster to describe the incoming stream. By an inspection of the original raw data at these timepoints, we found that they correspond to attacks of a single type, namely "smurf" attacks and actually they all are described by the same values. So, it is reasonable to be summarized by a single microcluster. In the above results, note that the window size w determines the number of points received per each timepoint. Due to the history of the stream though, the actual weight of the stream at each timepoint is much higher (Recall the relevant discussion in Section 4.4).

As we can see from this figure, by monitoring the number of microclusters over time one can get useful insights on the network status. In particular, we might observe different peaks in the network activity which might correspond to changes in the underlying connections, e.g. some new kind of attack. Such a peak might act as an alert for the end user and calls for closer inspection. This is extremely useful for intrusion detection systems where the types of attacks are not known in advance and also, the intruders might test new attacks and abandon old, already known (and blocked) intrusion types.

Due to the limitation on the fixed number of clusters, HPStream tries to assign the new instances to some of the existing clusters or if this is not possible, to create a new cluster in place of some old one. For example, at timepoint $t = 215$, 86 "normal", 8 "smurf", 2 "ftp_write" and 104 "nmap" connections arrive (in the scenario of window size $w = 200$ points per time). HPStream achieves a purity of 86%, because it mixes different attack types and normal connections into the same clusters. For example, in one of the clusters HPStream assigns 8 "smurf", 1 "ftp_write", 17 "nor-

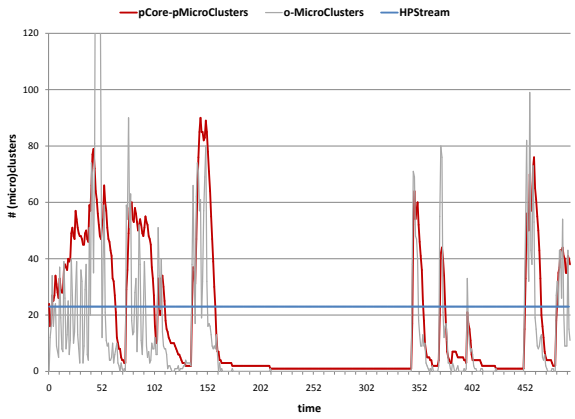

Figure 5: Number of (micro)clusters (Network Intrusion dataset, window size w=1000)

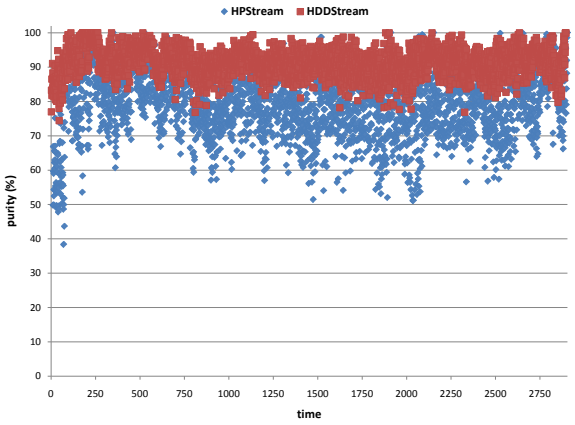

Figure 6: Clustering quality (Forest Cover Type dataset, window size $w = 200$)

mal" and 104 "nmap" connections. On the contrary, HDDStream achieves a purity of 100% w.r.t. the true class labels of the incoming points.

Forest Cover Type Dataset We also tested the clustering quality of HPStream and HDDStream on the Forest Cover Type Dataset. The parameters for HPStream were set according to [2].The parameters for HDDStream were set as follows: density threshold for projected core microclusters $\mu = 10$, density factor for potential core projected microclusters $\beta = 0.5$, radius threshold $\varepsilon = 0.2$, maximal projected dimensionality $\pi = 8$, variance threshold $\delta = 0.01$.

In Figure 6, we display the cluster purity during the whole period of stream execution for window size, $w = 200$ points per time. Again, HDDStream achieves better cluster purity compared to HPStream. In contrast to the Network Intrusion Dataset, where there exist timepoints for which both algorithms achieve a purity of 100%, in the Forest Cover Type dataset, no algorithm achieves 100% purity for many time points. This is due to the characteristics of the datasets. The Network Intrusion Dataset is a rapidly evolving dataset for which there is usually one dominant class in the stream over time (either the normal type connections or some attack type). Contrary to this, in the Forest Cover Type dataset, instances are arriving from more than one class at each time point. To this end, for the Forest Cover Type dataset we measure the average purity achieved by both algorithms. In Figure 7, we display the average purity achieved by both algorithms under different window sizes, varying from $w = 200$ to $w = 3,000$ points per each timepoint. As we can see, HDDStream outperforms HPStream for all window sizes.

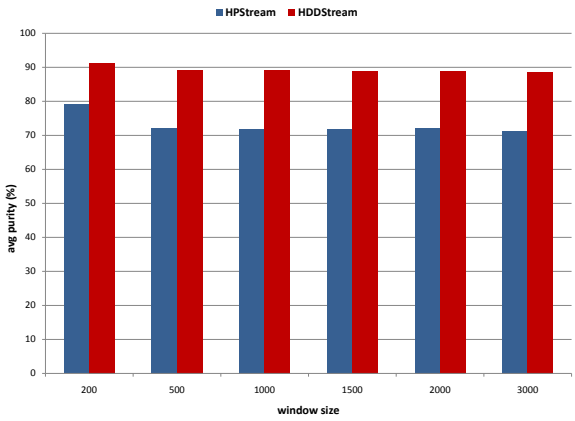

Figure 7: Clustering quality for different window sizes (Forest Cover Type dataset)

Regarding the memory usage, the number of (micro)clusters is depicted in Figure 8 for window size, $w = 200$ points per time. HPStream utilizes a constant number of $k = 7$ clusters over time (straight blue line), whereas HDDStream adjusts the number of microclusters to the incoming stream data. At different time points, the data characteristics are captured by a different number of microclusters. The increased number of microclusters in the beginning of the stream execution is due to the fact that initially all seven classes are represented in the incoming data. However, as the stream proceeds, the number of classes represented by the incoming instances is reduced. It is difficult to derive such kind of insights from HPStream, since the number of clusters to be discovered is required as an input and also, it remains constant over time.

6 Conclusions

While the problem of clustering in subspaces of high dimensional static data and the problem of clustering

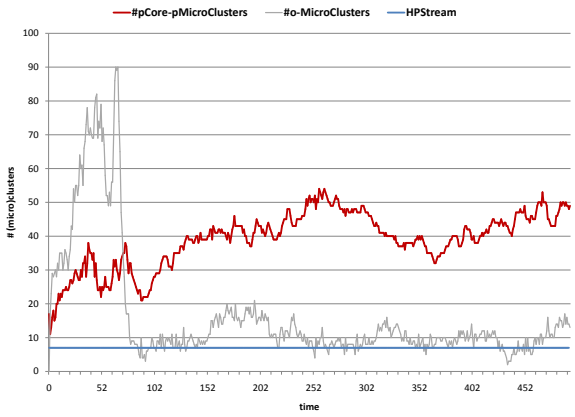

Figure 8: Number of (micro)clusters (Forest Cover Type dataset, window size w=200)

stream data using all dimensions both attracted a wealth of approaches, the combination of both tasks still poses a hardly tackled problem.

Here, we proposed a new algorithm, HDDStream, for projected clustering over high dimensional stream data. As opposed to existing work, HDDStream follows the density-based clustering paradigm, hence overcoming certain drawbacks of partitioning approaches. The important points in our contribution, contrary to existing work, are: (i) HDDStream allows noise handling, (ii) HDDStream does not rely on any assumptions regarding the number of (micro)clusters and (iii) HDDStream features interesting possibilities for monitoring the behavior of the data stream in order to detect drastic changes in the population. To this end, we demonstrated the detection of an attack on network monitoring data by means of monitoring the number of microclusters over time. Overall, in our experiments the clustering quality of HDDStream was superior to the clustering quality of the canonical competitor.

Acknowledgement

I. Ntoutsi is supported by an Alexander von Humboldt Foundation fellowship for postdocs (`http://www.humboldt-foundation.de/`).

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. VLDB*, 2003.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. VLDB*, 2004.

[3] C. C. Aggarwal, C. M. Procopiuc, J. L. Wolf, P. S. Yu, and J. S. Park. Fast algorithms for projected clustering. In *Proc. SIGMOD*, 1999.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. SIGMOD*, 1998.

[5] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, 2010.

[6] C. Böhm, K. Kailing, H.-P. Kriegel, and P. Kröger. Density connected clustering with local subspace preferences. In *Proc. ICDM*, 2004.

[7] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proc. SDM*, 2006.

[8] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proc. KDD*, 2007.

[9] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proc. VLDB*, 1998.

[10] 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. KDD*, 1996.

[11] J. Gao, J. Li, Z. Zhang, and P.-N. Tan. An incremental data stream clustering algorithm based on dense units detection. In *Proc. PAKDD*, 2005.

[12] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: you only get one look. A tutorial. In *Proc. SIGMOD*, 2002.

[13] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[14] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. Self-adaptive anytime stream clustering. In *Proc. ICDM*, 2009.

[15] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek. Density-based clustering. *WIREs DMKD*, 1(3):231–240, 2011.

[16] H.-P. Kriegel, P. Kröger, I. Ntoutsi, and A. Zimek. Density based subspace clustering over dynamic data. In *Proc. SSDBM*, 2011.

[17] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM TKDD*, 3(1):1–58, 2009.

[18] P. Kröger, H.-P. Kriegel, and K. Kailing. Density-connected subspace clustering for high-dimensional data. In *Proc. SDM*, 2004.

[19] G. Moise, J. Sander, and M. Ester. Robust projected clustering. *KAIS*, 14(3):273–298, 2008.

[20] M. L. Yiu and N. Mamoulis. Iterative projected clustering by subspace mining. *IEEE TKDE*, 17(2):176–189, 2005.

[21] Q. Zhang, J. Liu, and W. Wang. Incremental subspace clustering over multiple data streams. In *Proc. ICDM*, 2007.

[22] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. SIGMOD*, 1996.