Density Based Subspace Clustering Over Dynamic Data

Hans-Peter Kriegel, Peer Kröger, Irene Ntoutsi, Arthur Zimek

Institute for Informatics, Ludwig-Maximilians-Universität München
http://www.dbs.ifi.lmu.de
{kriegel,kroeger,ntoutsi,zimek}@dbs.ifi.lmu.de

Abstract. Modern data are often high dimensional and dynamic. Subspace clustering aims at finding the clusters and the dimensions of the high dimensional feature space where these clusters exist. So far, the subspace clustering methods are mainly static and cannot address the dynamic nature of modern data. In this paper, we propose a dynamic subspace clustering method, which extends the density based projected clustering algorithm PreDeCon for dynamic data. The proposed method efficiently examines only those clusters that might be affected due to the population update. Both single and batch updates are considered.

1 Introduction

Clustering is the unsupervised classification of data into natural groups (called clusters) so that data points within a cluster are more similar to each other than to data points in other clusters. Due to its broad application areas, the clustering problem has been studied extensively in many contexts and disciplines, including Data Mining. As a result, a large number of clustering algorithms exists in the literature (see [16] for a thorough survey). However, modern data impose new challenges and requirements for the clustering algorithms due to their special characteristics. First of all, a *huge amount* of data is collected nowadays as a result of the wide spread usage of computer devices. This possibility of cheaply recording massive data sets may also be the reason for another new characteristic of modern data; the *high dimensionality* of objects. While years ago, data recording was more expensive and, thus, the relevance of features was carefully evaluated before recording, nowadays, people tend to measure as much as they can. As a consequence, an object might be described by a large number of attributes. Many of these attributes may be irrelevant for a given application like cluster analysis and there might be correlations or overlaps between these attributes. In addition to their quantity and high dimensionality, today's data is often highly *dynamic*, i.e., new data records might be inserted and existing data records might be deleted, as time goes by.

As an example, consider the data derived from the Bavarian newborn screening program [20]. For each newborn in Bavaria, Germany, the blood concentrations of 43 metabolites are measured in the first 48 hours after birth producing

a vast amount of high dimensional data that is highly dynamic (new individuals are added usually in a batch on a daily or weekly basis). The analysis of these data shall help doctors in the diagnosis the exploration of known and new metabolic diseases. Clustering the data is a crucial step in this process. However, for different diseases, it is very likely that different metabolites are relevant. Thus, clusters representing groups of newborns with a homogeneous phenotype, e.g. suffering from a similar disease, can usually only be found in subspaces of the data. As batches of new individuals are coming in every day or week, the detected clustering structure needs to be updated as well. Due to the huge amount of data, the update of the clustering structure should be done incrementally only for the changing part of the structure, rather than re-computing the complete structure from scratch. Let us note that such screening projects are implemented in a large number of states/countries so that there are many data sets having similar characteristics that need to be analyzed.

The scenario described above represents a general *data warehouse environment*. With this term, we do not associate a certain architecture, but describe an environment in which changes in the transactional database are collected over some period (e.g. daily) and the data warehouse is updated using batch operations. Beside data originating from scientific experiments, also many companies store terabytes of corporate data in such an environment. Applications like scientific data analysis or industrial decision support systems in such environments require not only high accuracy from data analysis methods but also fast availability of up-to-date knowledge — a prohibitive demand for many data mining algorithms which are able to gain knowledge only from scratch using highly complex operations. Rather, to cope with the problem of updating mined patterns in a data warehouse environment, algorithms preferably should permanently store the acquired knowledge in suitable data structures and facilitate an efficient adaptation of this stored knowledge whenever the raw data changes.

Lately, a lot of work has been carried out on adapting traditional clustering algorithms in order to meet the requirements of modern systems or on proposing new algorithms that are specialized on handling data with the above features. In particular, several methods have been proposed for each of the aforementioned problems separately, like for clustering of large amounts of data, e.g. [22, 11, 5], for clustering over data streams, e.g. [15, 2], for change detection and monitoring over evolving data, e.g. [1, 21], as well as for clustering high dimensional data (see [19] for a survey). Less work though has been done to tackle the complete list of challenges in a single, unified approach.

We propose a new algorithm, based on the density based subspace clustering algorithm PreDeCon [6] providing a solution to the problem of high dimensionality by finding both clusters and subspaces of the original feature space where these clusters exist. The original PreDeCon works upon static datasets. In this work, we propose an incremental version of PreDeCon, which also deals with the issue of dynamic data.[1] The new algorithm can also serve as a framework for

[1] A preliminary version of this paper has been discussed at the StreamKDD 2010 workshop [18].

monitoring clusters in a dynamic environment. We choose the algorithm PreDeCon [6] because it already addresses the problem of high dimensional data (for static scenarios) and it relies on a density-based clustering model such that updates usually do not affect the entire clustering structure but rather cause only limited local changes. This is important to explore update procedures for dynamic data.

The rest of the paper is organized as follows. In Section 2, we discuss the related work and our contribution. In Section 3, we present the basic notions of PreDeCon which are necessary for the understanding of the incremental method. In Section 4, we present the incremental algorithm, incPreDeCon. We distinguish between a single update scenario and a batch update scenario (Section 5 and 6, respectively). Experimental results are reported in Section 7. Section 8 concludes our work.

2 Related Work and Contributions

2.1 Subspace clustering

The area of subspace clustering has lately emerged as a solution to the problem of the high dimensionality of the data. Its goal is to simultaneously detect both clusters (i.e.,, sets of objects) and subspaces of the original feature space where these clusters exist. This is in contrast to the traditional clustering that searches for groups of objects in the full dimensional space [16]. Also this is in contrast to global dimensionality reduction techniques like Principal Component Analysis (PCA) that search for clusters in the reduced (though full) dimensional space. In subspace clustering different features might be relevant for different clusters and the goal is to find both the clusters and the features that form these clusters.

Recent work on subspace clustering (see e.g. [19] for a review) so far focus on finding clusters in different subspaces of the original feature space in static data. None of these methods are suitable to efficiently keep track of changes of the clustering structure over time. Rather, the clustering structure can only be updated by computing the entire clustering from scratch.

2.2 Incremental clustering

Traditional incremental clustering methods rely on the old clustering at time point $t-1$ (based on dataset $\mathcal{D}_{t-1}$) and on the update operations at time point t in order to derive the new clustering at t. In this category belong methods like incDBSCAN [10] which is the incremental version of the density based algorithm DBSCAN [11] and incOPTICS [17] which is the incremental version of the density based hierarchical clustering algorithm OPTICS [5]. Both incDBSCAN and incOPTICS methods exploit the fact that, due to the density based nature of the corresponding static algorithms, an update operation affects only some part of the old clustering instead of the whole clustering. The update process works directly upon raw data. Both methods produce the same results with the

corresponding static methods when the latest are applied over the accumulative dataset $\mathcal{D}_t$. Charikar et al.[8] present an incremental K–Means method which maintains a collection of k clusters as the dataset evolves. When a new point is presented, it is either assigned to one of the current k clusters, or it starts a new cluster while two existing clusters are merged into one, so as the total number of clusters does not exceed the threshold k. Chen et al. [9] propose the incremental hierarchical clustering algorithm GRIN which is based on gravity theory in physics. In the *first phase*, GRIN constructs the initial clustering dendrogram, which is then flattened and its bottom levels are pruned in order to derive the so called tentative dendrogram. For each cluster, the tentative histogram keeps the centroid, the radius and the mass of the cluster. In the *second phase*, new data instances are inserted one by one and it is decided whether they belong to leaf nodes of the tentative dendrogram or are outliers. If the tentative outlier buffer exceeds some threshold, a new tentative dendrogram is reconstructed. Both [8] and [9] are approximate methods, by means that the resulting clustering after the update is not assured to be identical to the one we would obtain if we applied from scratch the static versions of the algorithms over the accumulative dataset $\mathcal{D}_t$. This is due to the fact, that the update process works upon cluster summaries rather than upon raw data; the new data at t are actually "mapped" to the closer cluster of the existing clustering (from timepoint $t-1$).

2.3 Stream clustering

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" [14]. As a result, several methods have been proposed that first summarize the data through some summary structure and then apply clustering over these summaries instead of the original raw data. With respect to the clustering quality, these summaries might be either lossy (that is, they correspond to some approximation of the raw data) or lossless (that, is they exactly maintain the information contained in the original raw data). Agrawal et al. [2] propose the CluStream framework for clustering of evolving data streams. The clustering process is split into an online and an offline part: The online component periodically stores summary statistics (the so called, micro–clusters), whereas the offline component uses these micro–clusters for the formation of the actual clusters (the so called, macro–clusters) over a user–defined time horizon. No access to raw data is required in this method, since the clustering takes place over the microclusters, which correspond to a lossy representation of the original data. The incremental part in this case is the online component which updates the micro–clusters, whereas the clustering process is applied from scratch over these updated summaries. DenStream [7] follows the online–offline rationale of CluStream [2] but in contrast to CluStream that is specialized to spherical clusters, it can detect clusters of arbitrary shapes. In the context of their DEMON framework, Ganti et al. [12] present BIRCH+, an incremental extension of BIRCH [22]. The original BIRCH [22] first summarizes the data into subclusters and then it clusters those subclusters using some traditional clustering

algorithm. The subclusters are represented very concisely through cluster features. In BIRCH+, the cluster features are maintained incrementally as updates occur, and then the clustering step takes place as in BIRCH over those (now updated) summaries. So, the incremental part is that of summary structure update, whereas clustering is then applied from scratch over the updated summary structure. The incremental version produces the same results as the static version when applied on the accumulative dataset $\mathcal{D}_t$.

2.4 High dimensional stream clustering

Gao et al. [13], propose DUCStream, an incremental data stream clustering algorithm that applies the idea of dense units introduced in CLIQUE [4] to stream data. As in CLIQUE [4], the data space is split into units and a cluster is defined as a maximal set of connected dense units. Their method relies on incrementally updating, according to the update operation, the density of these units and on detecting units that change from dense to non-dense and the inverse. After the grid update phase, they identify the clusters using the original procedure of CLIQUE. DUCStream does not require access to the raw data of the past time points, but only over the summary grid structure. The incremental version produces the same results as the static version when applied to the accumulative dataset $\mathcal{D}_t$. Note that although CLIQUE is a subspace clustering algorithm, the proposed method [13] updates incrementally only the grid summary structure, whereas the clusters are discovered from scratch over the (now updated) grid. This is a clear difference to our work, where the goal is to incrementally update the existing clustering (at $t-1$) based on the dataset updates at t, so as to finally derive the new clustering at t. Agrawal et al. [3] extend the idea of CluStream [2] to high dimensional data streams by proposing HPStream, a method for projected data stream clustering. A summary structure, the so called fading cluster structure, is proposed which comprises a condensed representation of the statistics of the points inside a cluster and can be updated effectively as the data stream proceeds. The input to the algorithm includes the current cluster structure and the relevant set of dimensions associated with each cluster. When a new point arrives, it is assigned to the closest cluster structure or if this violates the limiting radius criteria, a new cluster is created and thus some old cluster should be deleted in order for the total number of clusters to not exceed the maximum number k. In each case, the cluster structure and the relevant dimensions for each cluster are dynamically updated. Although HPStream is a subspace clustering method and we propose an incremental subspace clustering method in this work, there are core differences between the two approaches and their scopes. In particular, HPStream is targeted to stream data and thus works upon summaries and provides an approximation solution to the clustering problem. On the other hand, our incPreDeCon method works upon dynamic data, requires access to raw data (although this access is restricted to only a subset of the original dataset) and provides exact solution to the clustering problem (i.e., we obtain the same results with those obtained by applying the static PreDeCon over the acumulated dataset $\mathcal{D}_t$).

2.5 Contributions

None of the existing methods can be applied to the scenario of massive, high dimensional databases that are updated over time like in a data warehouse environment. In this work, we propose an incremental version of the density based subspace preference clustering algorithm PreDeCon [6] which comprises a first step towards an integrated approach to the above listed challenges. We choose the algorithm PreDeCon because it already addresses the problem of high dimensional data (for static scenarios) and it relies on a well-known and established clustering model. Let us note that we do not discuss nor evaluate benefits and limitations of different cluster models in this paper but solely propose concepts to adapt an existing model to the various challenges of today's data.

The methods for finding subspace clusters in data streams mentioned above are to some degree related to the incremental subspace clustering in data warehouses. Both methodologies aim at providing the user with up-to-date information on subspace clusters very quickly in a dynamic, high dimensional environment. However, data streams impose different requirements on clustering algorithms and the entire data mining process. In particular, in a data warehouse, the clustering algorithm has access to all points currently in the database and not necessarily only to the most recently inserted points or to summaries of the raw data as for stream data. In addition, when clustering stream data, the algorithm for updating the summaries is restricted to sequential access to newly inserted objects and the clustering is then re-computed on the summary information only. This restriction does not apply to algorithms for incremental clustering in a data warehouse environment. Our solutions are therefore different from the data stream clustering context in these two aspects.

3 The Algorithm PreDeCon

PreDeCon [6] adapts the concept of density based clusters, introduced in DBSCAN [11], to the context of subspace clustering. The notion of *subspace preferences* for each point defines which dimensions are relevant to cluster the point. Roughly speaking, a dimension is relevant to cluster a point if its neighborhood along this dimension has a small variance. Intuitively, a *subspace preference cluster* is a density connected set of points associated with a similar subspace preference vector.

Let $\mathcal{D}$ be a database of d-dimensional points ($\mathcal{D} \subseteq R^d$), where the set of attributes is denoted by $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$, and $dist : R^d \times R^d \rightarrow R$ is a metric distance function between points in $\mathcal{D}$. Let $\mathcal{N}_\varepsilon(p)$ be the ε-neighborhood of $p \in \mathcal{D}$, i.e., $\mathcal{N}_\varepsilon(p)$ contains all points $q \in \mathcal{D}$ with $dist(p,q) \leq \varepsilon$. The variance of $\mathcal{N}_\varepsilon(p)$ along an attribute $A_i \in \mathcal{A}$ is denoted by $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(p))$. Attribute A_i is considered a *preferable (relevant) dimension* for p if the variance with respect to A_i in its neighborhood is smaller than a user-defined threshold δ, i.e., $\text{VAR}_{A_i} \leq \delta$. All preferable attributes of p are accumulated in the so-called *subspace preference vector*. This d-dimensional vector $\bar{\mathbf{w}}_p = (w_1, w_2, \ldots, w_d)$ is defined such that $w_i = 1$ if attribute A_i is irrelevant, i.e., $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(p)) > \delta$ and

$w_i = \kappa$ ($\kappa \gg 1$) if A_i is relevant, i.e., $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(p)) \leq \delta$. The subspace preference vector of points defines the *preference weighted similarity* function associated with a point p, $dist_p(p,q) = \sqrt{\sum_{i=1}^{d} w_i \cdot (\pi_{A_i}(p) - \pi_{A_i}(q))^2}$, where w_i is the i-th component of $\bar{\mathbf{w}}_p$. Using the preference weighted similarity, the preferable attributes are weighted considerably lower than the irrelevant ones. This distance is not symmetric. A symmetric distance is defined by the *general preference similarity*, $dist_{pref}(p,q) = \max\{dist_p(p,q), dist_q(q,p)\}$. The *preference weighted $\varepsilon-$neighborhood* of a point p contains all points of $\mathcal{D}$ that are within a preference weighted distance ε from p: $\mathcal{N}_\varepsilon^{\bar{\mathbf{w}}_o}(o) = \{x \in \mathcal{D} \mid dist_{pref}(o,x) \leq \varepsilon\}$.

Based on these concepts, the classical definitions of density-based clustering have been derived:

Definition 1 (preference weighted core points [6]). *A point $o \in \mathcal{D}$ is called* preference weighted core point *w.r.t. ε, μ, δ, and λ (denoted by $\text{CORE}_{\text{den}}^{\text{pref}}(o)$), if i) the preference dimensionality of its ε-neighborhood is at most λ and ii) its preference weighted ε-neighborhood contains at least μ points.*

Definition 2 (direct preference reachability [6]). *A point $p \in \mathcal{D}$ is* directly preference reachable *from a point $q \in \mathcal{D}$ w.r.t. ε, μ, δ, and λ (denoted by $\text{DIRREACH}_{\text{den}}^{\text{pref}}(q,p)$), if q is a preference weighted core point, the subspace preference dimensionality of $\mathcal{N}_\varepsilon(p)$ is at most λ, and $p \in \mathcal{N}_\varepsilon^{\bar{\mathbf{w}}_q}(q)$.*

Definition 3 (preference reachability [6]). *A point $p \in \mathcal{D}$ is* preference reachable *from a point $q \in \mathcal{D}$ w.r.t. ε, μ, δ, and λ (denoted by $\text{REACH}_{\text{den}}^{\text{pref}}(q,p)$), if there is a chain of points $p_1, \ldots, p_n$ such that $p_1 = q, p_n = p$ and p_{i+1} is directly preference reachable from p_i.*

Definition 4 (preference connectivity [6]). *A point $p \in \mathcal{D}$ is* preference connected *to a point $q \in \mathcal{D}$, if there is a point $o \in \mathcal{D}$ such that both p and q are preference reachable from o.*

Definition 5 (subspace preference cluster [6]). *A non-empty subset $\mathcal{C} \subseteq \mathcal{D}$ is called a* subspace preference cluster *w.r.t. ε, μ, δ, and λ, if all points in $\mathcal{C}$ are preference connected and $\mathcal{C}$ is maximal w.r.t. preference reachability.*

As DBSCAN, PREDECON determines a cluster uniquely by any of its preference weighted core points. As far as such a point is detected, the associated cluster is defined as the set of all points that are preference reachable from it.

4 Incremental PreDeCon

Let $\mathcal{D}$ be the accumulated data set until the time point $t-1$ and let ζ be the corresponding clustering at $t-1$ (built upon data set $\mathcal{D}$). Let $\mathcal{U}$ be a set of *update operations* (insertions of new points). Let $\mathcal{D}^*$ be the newly accumulated data set at time slot t, which is the result of applying $\mathcal{U}$ over $\mathcal{D}$, i.e., $\mathcal{D}^* = \mathcal{D} \cup \mathcal{U}$. The goal of incremental PREDECON is to update the so far built clustering ζ

(at timepoint $t-1$) based on the update set $\mathcal{U}$ (at timepoint t) and thus, to derive the valid clustering ζ^* for time point t. The key observation is that the preference weighted core member property of an object might change due to the update. As a result, the existing clustering might change too, e.g., new clusters might arise, old clusters might be abolished or merged into a new cluster and so on. The challenge is to exploit the old clustering ζ at $t-1$ (both clusters and subspaces where these clusters exist) and to adjust only that part of it which is affected by the update set $\mathcal{U}$ at time point t. Due to the density based nature of the algorithm, such an adjustment is expected (although not ensured in general) to be restricted to some (local) part of the clustering instead of the whole clustering.

We consider a dynamic environment where data are coming sequentially either as: (i) *single updates* ($|\mathcal{U}|$=1), e.g., in streams, or as (ii) *batch updates* ($|\mathcal{U}| = m > 1$), e.g., in data warehouses where updates are collected and periodically propagated. In case of *single updates*, each update is treated independently. In case of *batch updates*, the idea is to treat the effects of all these updates together instead of treating each update independently. The rationale is that the batch might contain updates that are related to each other (e.g., one update might correspond to an object that belongs to the neighborhood of another object which is also updated). This is common in many applications, e.g., news data: when a story arises usually within a small time interval there exists a burst of news articles all referring to this story.

5 Dealing with single updates

Due to the density based nature of PreDeCon, a preference weighted cluster is uniquely defined by one of its preference weighted core points. The key idea for the incremental version is to check whether the update operation affects the preference weighted core member property of some point. If a non-core point becomes core, new density connections might be *established*. On the other hand, if a core point becomes non-core, some density connections might be *abolished*. There is also another case in PreDeCon, when a core point remains core but under different preferences. Such a change might cause either the establishment of new connections or the abolishment of existing ones.

5.1 Effect on the core member property

The insertion of a point p *directly affects* the points that are in the ε-neighborhood of p, i.e., all those points $q \in \mathcal{D} : dist(p,q) \leq \varepsilon$. In particular, the neighborhood of q, $\mathcal{N}_\varepsilon(q)$, might be affected, since the newly inserted object p is now a member of this neighborhood. Since $\mathcal{N}_\varepsilon(q)$ might change, the variance of $\mathcal{N}_\varepsilon(q)$ along some dimension $A_i \in A$ might also change causing A_i to turn into a preferable or non-preferable dimension. This might change the subspace preference dimensionality of q, $\text{PDIM}(\mathcal{N}_\varepsilon(q))$. Also, the subspace preference vector of q, $\bar{\mathbf{w}}_q$, might change; this in turn, might result in changes in the preference ε-neighborhood

of q, $\mathcal{N}_\varepsilon^{\bar{\mathbf{w}}}(q)_q$. As a result, the core member property of q might be affected. According to Def. 1, two conditions should be fulfilled in order for a point q to be core: In terms of condition 1, the preference dimensionality of q must contain at most λ dimensions (i.e., $\text{PDIM}(\mathcal{N}_\varepsilon(q)) \leq \lambda$). In terms of condition 2, the preference weighted ε-neighborhood of q should contain at least μ points.

Let p be the new point, and let $\mathcal{D}^* = \mathcal{D} \cup \{p\}$ be the new data set after the insertion of p. The addition of p might affect the core member property of any object $q \in \mathcal{N}_\varepsilon(p)$. In particular, since $\mathcal{N}_\varepsilon(q)$ changes, the variance along some attribute $A_i \in A$, i.e., $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(q))$ might also change. (i) If A_i was a non-preferable dimension (that is, $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(q)) > \delta$), it might either remain non-preferable (if still $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(q)) > \delta$) or it might become preferable (if now $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(q)) \leq \delta$). (ii) If A_i was a preferable dimension, it might either remain preferable (if still $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(q)) \leq \delta$) or it might become non-preferable (if now $\text{VAR}_{A_i}(\mathcal{N}_\varepsilon(q)) > \delta$). A change in the preference of A_i might result in changes in the subspace preference vector of q, $\bar{\mathbf{w}}_q$, since some dimension might swap from preferable to non preferable and vice versa. Thus, we can have more or less preferable dimensions comparing to the previous state (*quantitative differences*) or we can have the same dimensionality but under different preferred dimensions (*qualitative differences*). A change in $\bar{\mathbf{w}}_q$, might cause changes in both $\text{PDIM}(\mathcal{N}_\varepsilon(q))$ and in $\mathcal{N}_\varepsilon^{\bar{\mathbf{w}}_q}(q)$.

If the subspace preference dimensionality of q, $\text{PDIM}(\mathcal{N}_\varepsilon(q))$, changes, the first condition of Definition 1 (referring to dimensionality) might be violated. In particular, if $|\text{PDIM}(\mathcal{N}_\varepsilon(q))| > \lambda$, the point q cannot be core. So, if q was a core point, it now looses this property (*core* $\rightarrow$ *noncore*), whereas if it was non-core it still remains non-core. This is the first condition to be checked, and it is quantitative since it is based on the number of preferred dimensions (whether they exceed δ or not). If after the insertion of p, this condition holds (that is, $|\text{PDIM}(\mathcal{N}_\varepsilon(q))| \leq \lambda$), the second condition of Definition 1 (preferred neighborhood size) is to check assessing whether q is core after the update. (i) If q was a core point, and now $|\mathcal{N}_\varepsilon^{\bar{\mathbf{w}}}(q)_q| < \mu$, then q loses its core member property (*core* $\rightarrow$ *noncore*). Otherwise, it remains core. (ii) If q was not a core point, and now $|\mathcal{N}_\varepsilon^{\bar{\mathbf{w}}}(q)_q| \geq \mu$ then q becomes core (*noncore* $\rightarrow$ *core*). Otherwise, it remains non core. (iii) There is also another case of change for q, where it still remains core (*core* $\rightarrow$ *core*) but under different preferences (this might happen e.g., when there are qualitative changes in $\bar{\mathbf{w}}_q$). Note that, although q might remain core its neighborhood might change due to different preferred dimensions.

Note again that the objects with a changed core member property are all located in $\mathcal{N}_\varepsilon(p)$, since such a change is due to the insertion of p.

5.2 Affected objects

So far, we referred to the objects in $\mathcal{N}_\varepsilon(p)$ that are *directly affected* by the insertion of p and we discussed when and how their core member property might change. Note however, that a change in the core member property of an object q might cause changes in the objects that are preference reachable from q (*indirectly affected*). If q was a core point before the insertion and it becomes

non-core after the insertion, then any density connectivity that relied on q is destroyed. On the other hand, if q was a non-core point before the insertion and it turns into core after the insertion, then some new density connectivity based on q might arise.

We denote by $\textsc{Affected}_{\mathcal{D}}(p)$ the set of points in $\mathcal{D}$ that might be affected after the insertion of p. This set contains both directly affected points (those located in $\mathcal{N}_\varepsilon(p)$, which might change their core member property after the update) and indirectly affected objects (those that are density reachable by some point in $\mathcal{N}_\varepsilon(p)$, which might change their cluster membership after the update).

Definition 6 (Affected objects). *Let $\mathcal{D}$ be a data set and let $\mathcal{D}^* = \mathcal{D} \cup \{p\}$ be the new data set after the insertion of object p. We define the set of objects in $\mathcal{D}$ affected by the insertion of p as:*

$\textsc{Affected}_{\mathcal{D}}(p) = \mathcal{N}_\varepsilon(p) \cup \{q | \exists o \in \mathcal{N}_\varepsilon(p) : \textsc{Reach}^{\text{pref}}_{\text{den}}(o, q)$ *in* $\mathcal{D}^*\}$

The update of p might cause changes in the cluster membership of only some objects $q \in \textsc{Affected}_{\mathcal{D}}(p)$. A naive solution would be to reapply the static PreDeCon over this set in order to obtain the new clustering for the set of affected data. This way however, although one would restrict reclustering over only this subset of the data, one actually ignores any old clustering information for this set and build it from scratch. Our solution is based on the observation that any changes in $\textsc{Affected}_{\mathcal{D}}(p)$, are exclusively initiated by objects that change their core member property, i.e., those in $\mathcal{N}_\varepsilon(p)$. So, instead of examining all objects in $\textsc{Affected}_{\mathcal{D}}(p)$, we can start searching from objects in $\mathcal{N}_\varepsilon(p)$ and discover the rest of the affected objects on the road (those objects would belong to $\textsc{Affected}_{\mathcal{D}}(p)$ though). Note also that there is no need to examine each $q \in \mathcal{N}_\varepsilon(p)$ since some objects might have not changed their core member property so related density connections from the previous clustering would be still valid. So, we need to examine only those objects in $\mathcal{N}_\varepsilon(p)$ that change their core member property after the insertion of p, instead of all objects in $\mathcal{N}_\varepsilon(p)$, so as to avoid rediscovering density connections. As already described, a possible change in the core member property of an object after the insertion of p falls into one of the following cases: (i) core $\rightarrow$ non-core, (ii) non-core $\rightarrow$ core and, (iii) core $\rightarrow$ core but under different preferences.

When the core member property of a point $q \in \mathcal{N}_\varepsilon(p)$ changes, we should consider as seed points for the update any core point $q' \in \mathcal{N}_\varepsilon(q)$. That is, the update process starts from core points in the neighborhood of the objects with changed core member property (which, in turn are all located in $\mathcal{N}_\varepsilon(p)$).

Definition 7 (Seed objects for the update). *Let $\mathcal{D}$ be a data set and let $\mathcal{D}^* = \mathcal{D} \cup \{p\}$ be the new data set after the insertion of p. We define the seed objects for the update as:*

$\textsc{UpdSeed} = \{q$ *is core in* $\mathcal{D}^* | \exists q' : q \in \mathcal{N}_\varepsilon(q')$ *and* q' *changes its core property*$\}$

5.3 Updating the clustering

After the insertion of a new object p, new density connections might be established whereas existing connections might be abolished or modified. We can

```
algorithm incPreDeCon(𝒟, 𝒰, ε, μ, λ, δ)
  for each p ∈ 𝒰 do
    1. 𝒟* = 𝒟 ∪ p;
    2. compute the subspace preference vector w̄_p;
    // update preferred dimensionality and check core member property in 𝒩_ε(p)
    3. for each q ∈ 𝒩_ε(p) do
    4.    update w̄_q;
    5.    check changes in the core member property of q and if change exists, add q to Affected;
    6. compute UpdSeed based on Affected
    7. for each q ∈ UpdSeed do
    8.    expandCluster(𝒟*, UpdSeed, q, ε, μ, λ);
  end;
```

Fig. 1. Pseudo code of the algorithm incPreDeCon.

detect these changes starting with the seed objects in UpdSeed. As in PreDeCon, the cluster is expanded starting from objects in UpdSeed and considering the results of the so far built clustering. The pseudo code of the algorithm is displayed in Figure 1. The existing database $\mathcal{D}$, the update set $\mathcal{U}$ and the PreDeCon parameters (namely, the distance threshold ε, the neighborhood size threshold μ and the dimensionality threshold λ) are the input to the algorithm. The updated clustering ζ^* is the output of the algorithm.

The algorithm works as follows: After the insertion of a point p (line 1), its subspace preference vector is computed (line 2), and its neighborhood $\mathcal{N}_\varepsilon(p)$ is updated (lines 3–6). In particular, for each object $q \in \mathcal{N}_\varepsilon(p)$ (line 3), we first update its subspace preference vector (line 4) and then check for any changes in the core member property of q (line 5). If the core member property of q is found to be affected, q is added to the Affected set. After the Affected set is computed, we derive the seed objects for the update (line 6). Based on these objects, the reorganization of the old clustering starts, which involves some call to the *expandCluster*() function of PreDeCon. This is a generic solution that works on every effect caused by the update of p. Of course, there are simpler cases where we can deal with the update without invoking the *expandCluster*() procedure of PreDeCon. For example, if the update of p does not affect the core member property of its neighborhood and its neighborhood belongs to exactly one cluster before the update, then p is also added to this cluster (absorption). However there are many such special cases, since, as already stated, the update of p might both destroy old density connections and create new density connections depending on the changes in the core member property of its neighborhood. The proposed method is lossless, that is the incrementally updated model ζ^* at t (which is based on the clustering model ζ at $t-1$ and on the update set $\mathcal{U}$ at t) is identical to the one we would obtain if we applied from scratch the traditional PreDeCon over the accumulated data set $\mathcal{D}^*$ at time point t.

6 Dealing with batch updates

We now consider the case of batch updates where m ($m > 1$) points are inserted at each time point. The rationale behind this alternative is that it is possible for

the batch data to be related to each other instead of independent (Consider for example an earthquake in some place and Twitter response to such an event; a flow of tweets would appear referring to that event.). Hence, instead of updating the clustering for each operation independently (as in the single update case, c.f. Section 5.3), one should consider the accumulative effect of all these batch operations on the clustering and treat them together. This way, the objects that might be affected by more than one single update operations are examined only once. Otherwise, such objects should be examined after each single operation (multiple checks).

The algorithm is similar to the algorithm for the single update case (c.f. Figure 1). The key difference is that instead of inserting objects one by one, examining the side effects of each single insert and updating the clustering based on each single insert, we now insert the whole batch (all m objects), we examine how the database is affected by the whole batch and we update the clustering model based on the whole batch. In more detail, we first insert the m objects of the batch in the existing database. Then, we continue the same rationale as with the single update case: First, we update the subspace preference vector of each of the inserted objects in the batch. Next, we check for objects with affected core member property. Recall (Section 5.1) that any objects with affected core member property lie in the neighborhood of some inserted point. Note also, that the points of the batch are all inserted into the database, so these operations also consider the newly inserted points. The AFFECTED set now contains the objects that might be affected due to the whole batch of points. Based on the AFFECTED set, the UPDSEED set is constructed which also refers to the whole batch. The points in *UpdSeed* serve as the starting points for the cluster reorganization.

The benefit of the batch method is that some computations might take place only once, instead of after each single update as is the case for the single update method. For example, let p be an object in the database which is part of the neighborhood of both points p_1, p_2 in the batch. According to the single update case, this object should be examined twice, i.e., after each single insert, for any changes in its core member property. According to the batch update case though, this object should be examined only once.

7 Experiments

Since INCPREDECON computes the same results as PREDECON, we compare their performances in terms of efficiency. For massive data sets, the bottleneck of PREDECON and INCPREDECON is the number of range queries in arbitrary (possibly 2^d different) subspaces that cannot be supported by index structures. We report the speed-up factor defined as the ratio of the cost of PREDECON (applied to the accumulative data set $\mathcal{D}^*$) and the cost of INCPREDECON (applied to the initial data set $\mathcal{D}$ plus the updates $\mathcal{U}$).

We used a synthetic data generated according to a cluster template that describes the population of the corresponding cluster, the generating distribution, and range of dimension values for each dimension. In addition, we report a case

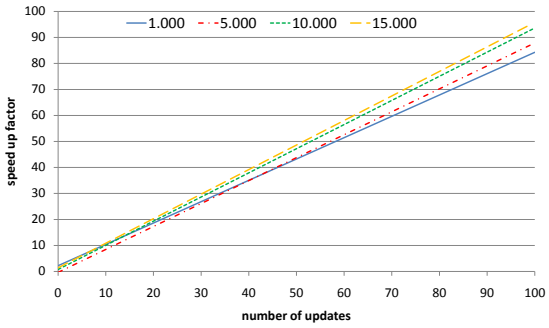

Fig. 2. Speed-up factors w.r.t. data set size.

study of incrementally keeping track of clusters in real-world data. Let us again note that we do not compare different clustering models here, since the main focus of our paper is to provide a solution for incremental density-based subspace clustering in a high dimensional, dynamic environment.

7.1 Experiments on single updates

Varying the data set population. Four synthetic data sets of varying size between 1.000 and 15.000 objects were generated. From each data set, 100 objects were randomly extracted and used as the update set. The number of required range queries was computed after each insertion.

Figure 2 displays the speed-up factors w.r.t. the different data set sizes. IncPreDeCon outperforms PreDeCon with the speed-up factors of 2–100. As expected, with increasing number of updates the gain for IncPreDeCon is higher. Analogously, the bigger the data set population is, the greater are the benefits of using IncPreDeCon instead of PreDeCon.

Varying the number of generated clusters. Five data sets with varying number of clusters but constant dimensionality and fixed population of each cluster were used next. From each data set, 100 objects were randomly extracted and used as the update set. Figure 3 displays the speed-up factors for all data sets. Again, IncPreDeCon outperforms PreDeCon for all datasets with the speed-up factors increasing with the number of updates and lying in the range [1–100]. Comparing the different data sets, however, we cannot draw some clear conclusion regarding whether more generated clusters result in greater gainings for IncPreDeCon or no. This is intuitive since we generate random updates that do not necessarily correlate with the cluster structure. Thus, the number of clusters does not have a significant impact on the performance.

Varying the number of dimensions. Five data sets with varying dimensions but constant number of clusters and fixed population of each cluster were used next. From each data set, 100 objects were randomly extracted and used as the update set. Figure 4 displays the speed-up factors for all data sets. Again, In-

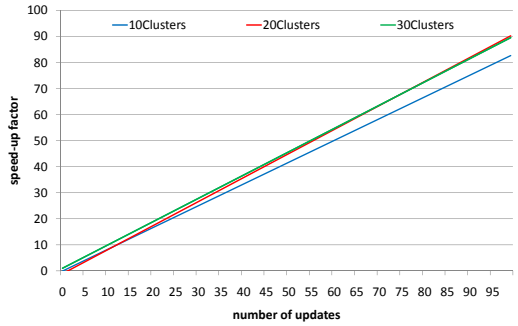

Fig. 3. Speed-up factors w.r.t. the number of generated clusters.

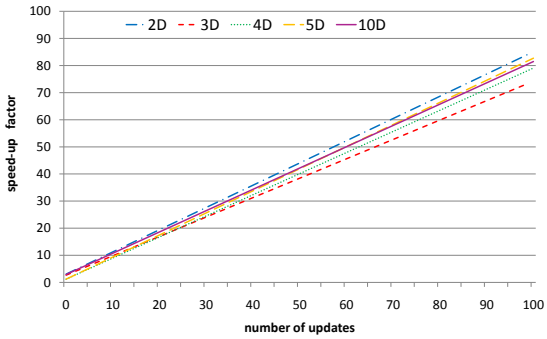

Fig. 4. Speed-up factors w.r.t. data dimensionality.

cPreDeCon outperforms PreDeCon with the speed up factors lying in the range 2–100. A comparison of the different data sets remains inconclusive w.r.t. whether or not more dimensions result in greater gain for incPreDeCon. This was expected since the dimensionality of the data should not have a different impact on the performances (in terms of required range queries) of PreDeCon and incPreDeCon.

7.2 Experiments on batch updates

100 *random updates* were performed in a batch way (with batch sizes of 5, 10, 15, 20 updates) on a data set of 1.000 objects. Figure 5 displays the speed-up factors for the different batch sizes (the single update case is also partially depicted). incPreDeCon outperforms PreDeCon for all different batch sizes. The highest gain exists for the single update case. As the batch size increases, the gain decreases.

The gain is expected to be even higher when the updates are not random but reflect the clustering structure. To verify this, we used an update set of objects

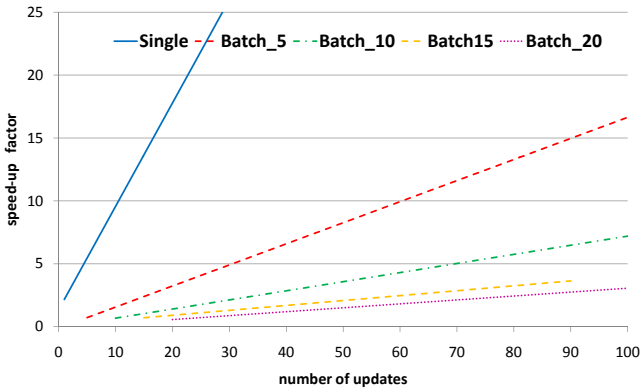

Fig. 5. Speed-up factors for random batch updates.

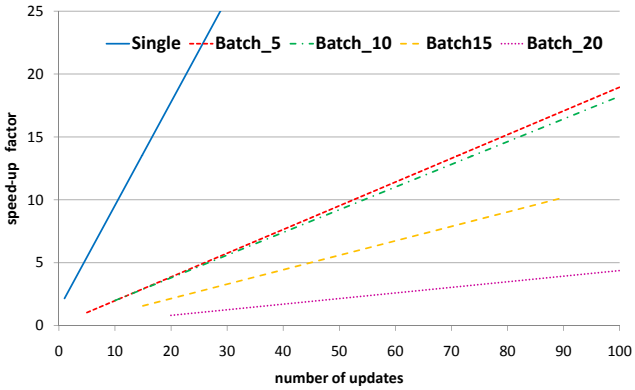

Fig. 6. Speed-up factors for "local" batch updates.

extracted from 2 clusters in the generated data set (50 objects per cluster). As expected, the speed-up factors (cf. Figure 6) are higher compared to the random update case (cf. Figure 5).

The experiments showed the benefit of incPreDeCon versus PreDeCon. The gain was very high for the single update case, whereas for the batch case the larger the batch size was, the lower the gain was. For example, in Figures 7, and 8 we can see the actual number of range queries required by PreDeCon and incPreDeCon for two synthetic data sets. It can be observed that in the single update case (denoted by batch size = 1 in these figures), incPreDeCon (right bar) requires considerably less number of range queries comparing to PreDeCon (left bar). As the batch size increases however, the gainings for incPreDeCon are decreased. Note that we run random updates in these experiments (Figure 7 and Figure 8). Greater savings are expected for "local updates", i.e., updates that correspond to a specific subcluster/ area of the population, since

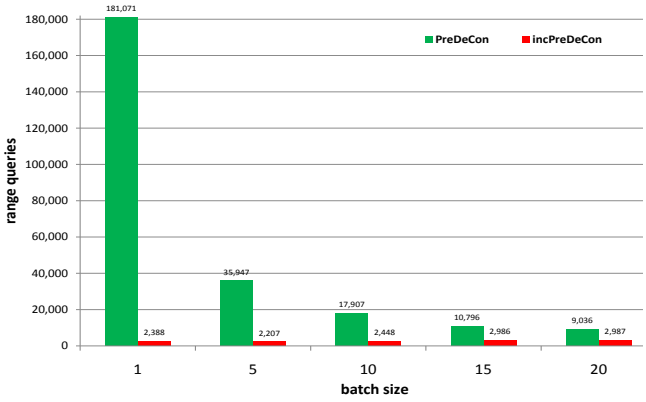

Fig. 7. Range queries for PreDeCon and incPreDeCon (data set size 5.000).

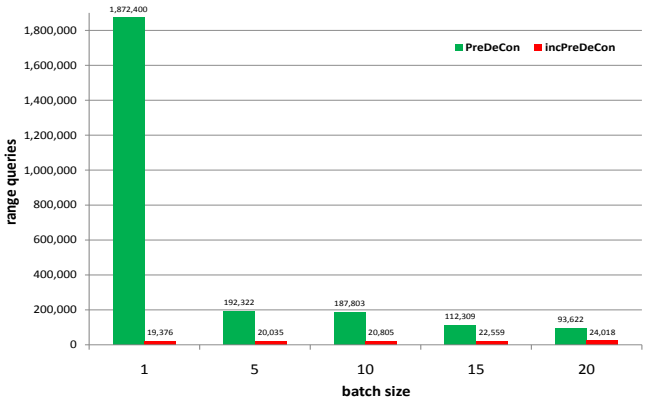

Fig. 8. Range queries for PreDeCon and incPreDeCon (data set size 10.000).

the batch method performs better when the update set contains related updates (recall our previous discussion on Figure 5 and Figure 6).

7.3 A case study on real-world data

We applied the original PreDeCon on a small sample of 1,000 objects of the Bavarian newborn screening data, added batches of 200 objects and updated the cluster structure using incPreDeCon. Sample results from different time slots are sketched in Figure 9. Cluster 1 representing newborns suffering PKU refines the set of relevant attributes over time. This indicates that the attributes that have been evaluated as relevant for that cluster in the beginning might be false positives or might be relevant only for a subset of cluster members. The latter might indicate an interesting subtype of the PKU disease. In any case, these results might trigger the medical doctors to initiate further investigations on this

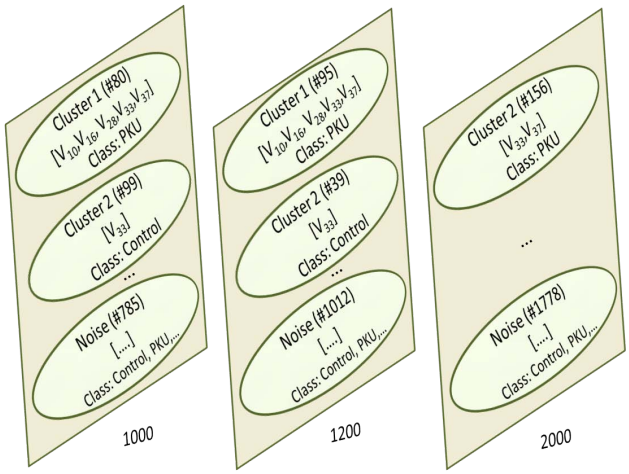

Fig. 9. Clusters on Bavarian newborn screening data evolving over time.

issue. Cluster 2 representing a subset of the control group (healthy newborns) disappears over time since the clear subspace structure is absorbed by full dimensional noise. In fact, the attribute that is preferred by members of Cluster 2 at the beginning turns out to be not discriminative later on. Let us note that this phenomenon could not have been found by a full dimensional clustering method (e.g. by DBSCAN [11]) because the disappearance of that cluster is only possible when considering relevant projections of the feature space.

8 Conclusions

In this paper, we presented the incremental density based subspace clustering algorithm incPreDeCon. The algorithm can handle several prevalent challenges posed by today's data, including massive, dynamic, and high dimensional data sets. The update strategy, exploits the density based nature of clusters and, thus. manages to restructure only that part of the old clustering that is affected by the update. Both a single and a batch update method have been proposed. Our experimental results demonstrate the efficiency of the proposed method against the static application of PreDeCon. As future work, we plan to examine subspace clustering methods over data streams where access to raw data is usually not provided for efficiency reasons.

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

References

1. Aggarwal, C.C.: On change diagnosis in evolving data streams. IEEE TKDE 17(5), 587–600 (2005)
2. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proc. VLDB (2003)
3. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for projected clustering of high dimensional data streams. In: Proc. VLDB (2004)
4. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. SIGMOD (1998)
5. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: OPTICS: Ordering points to identify the clustering structure. In: Proc. SIGMOD (1999)
6. Böhm, C., Kailing, K., Kriegel, H.P., Kröger, P.: Density connected clustering with local subspace preferences. In: Proc. ICDM (2004)
7. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: Proc. SDM (2006)
8. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic information retrieval. SICOMP 33(6), 1417–1440 (2004)
9. Chen, C., Hwang, S., Oyang, Y.: An incremental hierarchical data clustering algorithm based on gravity theory. In: Proc. PAKDD (2002)
10. Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X.: Incremental clustering for mining in a data warehousing environment. In: Proc. VLDB (1998)
11. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. KDD (1996)
12. Ganti, V., Gehrke, J., Ramakrishnan, R.: DEMON: Mining and monitoring evolving data. IEEE TKDE 13(1), 50–63 (2001)
13. Gao, J., Li, J., Zhang, Z., Tan, P.N.: An incremental data stream clustering algorithm based on dense units detection. In: Proc. PAKDD (2005)
14. Garofalakis, M., Gehrke, J., Rastogi, R.: Querying and mining data streams: you only get one look. A tutorial. In: Proc. SIGMOD (2002)
15. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE TKDE 15(3), 515–528 (2003)
16. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM CSUR 31(3), 264–323 (1999)
17. Kriegel, H.P., Kröger, P., Gotlibovich, I.: Incremental OPTICS: efficient computation of updates in a hierarchical cluster ordering. In: Proc. DaWaK (2003)
18. Kriegel, H.P., Kröger, P., Ntoutsi, I., Zimek, A.: Towards subspace clustering on dynamic data: an incremental version of PreDeCon. In: StreamKDD'10 (2010)
19. Kriegel, H.P., Kröger, P., Zimek, A.: Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. IEEE TKDD 3(1), 1–58 (2009)
20. Liebl, B., Nennstiel-Ratzel, U., von Kries, R., Fingerhut, R., Olgemöller, B., Zapf, A., Roscher, A.A.: Very high compliance in an expanded MS-MS-based newborn screening program despite written parental consent. Preventive Medicine 34(2), 127–131 (2002)
21. Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., Schult, R.: MONIC: modeling and monitoring cluster transitions. In: Proc. KDD (2006)
22. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering method for very large databases. In: Proc. SIGMOD. pp. 103–114 (1996)