A Semi-supervised Incremental Clustering Algorithm for Streaming Data

Maria Halkidi[1], Myra Spiliopoulou[2], and Aikaterini Pavlou[3]

[1] Dept of Digital Systems, University of Piraeus
mhalk@unipi.gr
[2] Faculty of Computer Science, Magdeburg University, Germany
myra@iti.cs.uni-magdeburg.de
[3] Athens University of Economics and Business
aikaterinipavlou@gmail.com

Abstract. Nowadays many applications need to deal with *evolving data streams*. In this work, we propose an incremental clustering approach for the exploitation of user constraints on data streams. Conventional constraints do not make sense on streaming data, so we extend the classic notion of constraint set into a *constraint stream*. We propose methods for using the constraint stream as data items are forgotten or new items arrive. Also we present an on-line clustering approach for the cost-based enforcement of the constraints during cluster adaptation on evolving data streams. Our method introduces the concept of multi-clusters (m-clusters) to capture arbitrarily shaped clusters. An m-cluster consists of multiple dense overlapping regions, named s-clusters, each of which can be efficiently represented by a single point. Also it proposes the definition of outliers clusters in order to handle outliers while it provides methods to observe changes in structure of clusters as data evolves.

Keywords: stream clustering, semi-supervised learning, constraint-based clustering.

1 Introduction

Clustering plays a key role in data analysis, aiming at discovering interesting data distributions and patterns in data. Also it is widely recognized as means to provide an effective way of maintaining data summaries and a useful approach for outlier analysis. Thus clustering is especially important for streaming data management. Streaming data are generated continuously at high rates and due to storage constraints we are not able to maintain in memory the entire data stream. Having a compressed version (synopsis) of data at different time slots, data analysts are able to keep track of the previously arrived data and thus more effectively extract useful patterns from the whole stream of data.

Semi-supervised clustering has received much attention in the last years, because it can enhance clustering quality by exploiting readily available background knowledge, i.e. knowledge on the group membership of some data items or knowledge on some properties of the clusters to be built. The available knowledge is

P.-N. Tan et al. (Eds.): PAKDD 2012, Part I, LNAI 7301, pp. 578–590, 2012.

represented in form of constraints (must-link and/or cannot-link constraints) which are incorporating in the clustering procedure. There is much research on semi-supervised clustering for static data [5,4,9,7,8,15], but less for *streaming data* [13]. However a huge amount of data generated in everyday life can be characterized as data streams (e.g. sensor data, web logs, Internet traffic, RFID) while many of the applications that profit from semi-supervised clustering, are essentially analyzing streaming data. For instance, consider a stream of news in document form and an application that clusters documents in topics. When a piece of news comes in as part of an already existing story, some news agencies also provide links to previous documents of the story. These links can be considered as constraints that indicate which documents are correlated and thus should be classified together.

Semi-supervised stream clustering requires an extension of the successful concept of *instance-level constraints*: Any record in a stream is gradually forgotten; if it is involved in a constraint, then the constraint itself has a limited lifetime. At the same time, some of the arriving items may be labeled, implying new constraints. We deal with both issues in our `Semi-supervised Stream` clustering method (further referred to as `SemiStream`). According to our approach a clustering scheme is adapted to a continuous flow of items, some of which carry labels and thus give raise to instance-level constraints. These labeled data are mapped into Must-Link and Cannot-Link constraints, thus forming a *constraint stream* that accompanies the data stream. Adaptation encompasses the elimination of old items and outdated constraints and the adjustment of the clusters to newly arriving items, taking account of all to-date constraints.

`SemiStream` starts with an initial clustering based on a given set of constraints. Then, at each period of observation t_i, we adapt the clusters to accommodate new data and satisfy new constraints, while old data are gradually forgotten and obsolete constraints (on forgotten data) are eliminated. To this purpose, we propose a *constraint-cost function* that associates constraint violations with penalty values. We use this function to assign each new data item to the cluster with the most proximal representative and incurring minimal cost. We specify an upper boundary to the cost that can be tolerated during cluster adaptation. Then, data items that cannot be placed to clusters without exceeding this boundary are declared as *outliers* and are accommodated to *outlier groups*. Finally, we identify and merge overlapping clusters, thereby checking the value of the constraint-cost function.

To summarize, the contributions of our work are as follows:

1) We propose an incremental approach for clustering evolving data streams based on constraints. The notion of constraint stream is introduced so that we efficiently describe the sequential fashion of labeled data that may emerge as data flows.

2) Our approach adopts the use of i) multiple clusters to represent significant neighboring dense areas in data, capturing thus arbitrarily shaped clusters, and ii) outliers clusters to describe a small set of data whose characteristics seem to deviate significantly from average behavior of the currently processed data.

2 Related Work

Recently, a number of clustering algorithms have been proposed to deal with streaming data. In [1], a framework of a stream clustering approach is proposed, which includes two clustering phases. At the online phase, the proposed approach periodically maintain statistical information about local data in terms of *micro-clusters*, while at the off-line phase, the decision maker uses these statistics to provide a description of clusters in the data stream. The main drawback of this algorithm is that the number of micro-clusters needs to be predefined. HPStream [2] incorporates a fading cluster structure and a projection-based clustering methodology to deal with the problem of high-dimensionality in data streams. Another stream clustering algorithm is Denstream [6]. It is based on the idea of DBSCAN [10] and it forms local clusters progressively by detecting and connecting dense data item neighborhoods.

A version of the AP algorithm that is closer to handling streaming data is presented in [16]. According to this approach, as data flows, data items are compared one-by-one to the exemplars and they are assigned to their nearest one if a distance threshold condition is satisfied. Otherwise, data are considered outliers and they are put in a reservoir. A cluster redefinition is triggered if the number of outliers exceeds a heuristic (user-defined) reservoir size or if a change in data distribution is detected.

Though there is a lot of work on constraint-based clustering methods for static data [3,8,15], the related work on clustering streaming data based on constraints is limited. Ruiz et al. [14] have presented a conceptual model for constraints on streams and extended the constraint-based K-means of [15] for streaming data. In `SemiStream`, we refine this model by proposing a *constraint stream* of instance-level constraints and we adapt the clusters incrementally instead of rebuilding them from scratch. Moreover an extension of Denstream so that constraints are taken into account during the clustering process is proposed in [13]. C-Denstream ensures that all given constraints are satisfied. However this is achieved at the cost of creating many small clusters. Moreover, in case of conflicts among constraints, C-Denstream is unable to conclude in a clustering.

3 A Model for Constraint-Based Clustering on Streaming Data

We study a stream of items, for some of which we have background knowledge in the form of instance-level constraints. We model constraints and clusters over a stream and then model the cost of assigning an item to a cluster, thereby possibly violating some constraints.

3.1 Modeling a Stream of Constraints

Let $t_1, \ldots, t_n$ be a series of time points. The stream is partitioned in data snapshots $D_1, \ldots, D_n$, where D_i consists of the items arriving during $(t_{i-1}, t_i]$. The

Algorithm 1. DYNAMIC CONSTRAINT UPDATER$(D, CS, t, \widehat{CS}_{old})$

Data: Snapshot D at t, new constraint-set CS, set of active constraints $\widehat{CS}_{old}$.
begin
  $\widehat{CS} = CS \cup \widehat{CS}_{old}$. ;
  for $cs \in \widehat{CS}$ **do**
  ;
  Let $cs =\prec x, y \succ$;
  if $teenage(x,t) == 0$ *or* $teenage(y,t) == 0$ **then** $\widehat{CS}_i = \widehat{CS} \setminus \{cs\}$. ;
  else $weight(cs) = \min\{teenage(x,t), teenage(y,t)\}$. ;
  return $\widehat{CS}$.;
end

dataset remembered at each t_i, $\widehat{D}_i \subseteq \cup_{j=1}^{i} D_j$ is determined by a decay age function. Presently, we assume a sliding window of size $w \in [0, n]$, so that $\widehat{D}_i = \cup_{j=u}^{i} D_j$ for $u = \max\{1, i - w + 1\}$. We further assign weights to the items inside the window, so that more recent items acquire higher weights. So, at time point t_i, the weight of an item x that arrived at time point $t_j \leq t_i$ is $teenage(x, t_i) = 1 - \frac{t_i - t_j}{w}$.

We consider Must-Link and Cannot-Link instance-level constraints [15]. Must-link constraints indicate data items that should belong to the same cluster while Cannot-link constraints refer to data items that should not be assigned to the same cluster. Such constraints involve two items x, y and are denoted as $\prec x, y \succ$. Since constraints refer to items that are associated with an age-dependent weight, they also have weights. So, at t_i, the weight of a constraint $cs =\prec x, y \succ$ is

$$weight(cs, t_i) = \min\{teenage(x, t_i), teenage(y, t_i)\} \quad (1)$$

At each time point t_i, some items of the snapshot D_i are involved in instance-level constraints, together with items of earlier time points. These new constraints constitute the *new* constraint-set CS_i. From them, we derive the set of *active constraints* at t_i, $\widehat{CS}_i = \cup_{j=u}^{i} CS_j, u = \max\{1, i - w\}$. In Alg. 1, we depict the DYNAMIC CONSTRAINT UPDATER, which builds $\widehat{CS}_i$. At each $t = t_2, t_3, \ldots$, the DYNAMIC CONSTRAINT UPDATER reads the set of old active constraints $\widehat{CS}_{old}$ and the current constraint-set CS. It combines them to produce the new set of active constraints $\widehat{CS}$ by (a) marking constraints on outdated items as *obsolete*, (b) recomputing the weights of non-obsolete, i.e. *active* constraints and (c) taking the union of the resulting set of constraints with CS to form $\widehat{CS}$.

3.2 Cost of Assigning Data Items to a Cluster

The assignment of an item x to a cluster c at time point t incurs a *cost*, which reflects constraint violations caused by the assignment, as well as the distance of x to the center of c. We first model the cost of constraint violations. Let $\widehat{CS}$ be the active set of constraints. From this set, we derive $ML(x)$, the set of items

that are involved in Must-Link constraints together with x, and $CL(x)$, the corresponding set of items for Cannot-Link constraints. Then, the constraint-violation cost for assigning x to c is given by:

$$costCV(x, c, \widehat{CS}, t) = \sum_{y \in ML(x), y \notin c} weight(\prec x, y \succ, t) \times f_{ML}(x, y) + \sum_{y \in CL(x), y \in c} weight(\prec x, y \succ, t) \times f_{CL}(x, y) \quad (2)$$

In this formula, we consider the items that must be linked with x and are not members of cluster c, as well as the items in c that should not be linked to x. For such an item y, the weight of the corresponding constraint is $weight(\prec x, y \succ, t)$, according to Eq. 1.

The functions $f_{ML}(\cdot)$, $f_{CL}(\cdot)$ denote the cost of violating a Must-Link, resp. Cannot-Link constraint. We consider that the cost of violating a must-link constraint between two close points should be higher than the cost of violating a must-link constraint between two points that are far apart. Thus we implement the cost function $f_{ML}(\cdot)$ as $f_{ML}(x, y) = d_{max} - d(x, y)$, where d_{max} is the maximum distance encountered between two items in the dataset. Similarly, the cost of violating a cannot-link constraint between two distant points should be higher than the cost of violating a cannot-link constraint between points that are close. Then we define the $f_{CL}(\cdot)$ function as the distance between the two items involved in a cannot-link constraint and we set $f_{CL}(x, y) = d(x, y)$.

Then, the cost of assigning an item x to a cluster c for a given set of active constraints $\widehat{CS}$ at time point t consists of the cost of effected constraint violations and the overhead of placing this item to this cluster. The latter is represented by the distance of the item to the cluster center:

$$cost(x, c, \widehat{CS}, t) = costCV(x, c, \widehat{CS}, t) + d(x, rep(c)) \quad (3)$$

3.3 A Clustering Model for Streaming Data

Our constraint-based clustering approach is based on the incremental adjustment of the clusters to the arrival of new data and constraints and to the decay of old data and constraints. In this approach, we distinguish between items that can be assigned to a cluster and those that cannot, either because they violate some constraints or because they are far away from any cluster. We call the latter *outlier items*. As the stream of data and the stream of constraints proceed, it may become possible to merge earlier clusters together and even place outlier items to some cluster. We present here a model that captures those cases, by distinguishing between *s-clusters* which correspond to core clusters of data, *o-clusters* that are groups of outlier items and *m-clusters* that are the result of merging core data clusters, i.e. dense areas in the dataset.

An *s-cluster* is a conventional cluster c, built at time point t_i by a constraint-based clustering algorithm like MPCK-Means. For such a cluster, we define concepts that describe its structure and surroundings.

Definition 1 (s-cluster Nucleus). *Let c be an s-cluster. We denote as $rep(c)$ the representative or conceptual center of c, consisting of the mean values of the*

items in c across all dimensions of the feature space. The within-cluster distance of c is the average distance of a data item from the cluster representative:

$$withinClusterDistance(c) = \frac{\sum_{x \in c} d(x, center(c))}{|c|} \tag{4}$$

With some abuse of conventions, we also use the term "radius" for the within-cluster-distance, i.e. we set $radius(c) \equiv withinClusterDistance(c)$. *Then, the items within the radius of c constitute its "nucleus":*

$$nucleus(c) = \{x \in c | d(x, center(c) \leq radius(c)\} \tag{5}$$

By Def. 1, we distinguish between items close to the s-cluster's representative (center) and remote ones. Pictorially, the former constitute the core or nucleus of the s-cluster, while the latter form the rim of the s-cluster. Also data items that are distant from the rest data are perceived as outliers.

Definition 2 (s-cluster Rim). *Let c be a s-cluster. Assume that ξ is the current clustering of streaming data. Then, the* rim *of c consists of the data items that are outside the nucleus of c but their distance from the representative of c is lower than some multiple* $\tau > 1$ *of its radius:*

$$rim(c) = \{x \in c | radius(c) < d(x, rep(c)) \leq \tau \times radius(c)\}$$

whereby depending on the specification of τ, a cluster may have an empty rim.

Next to items that are too far from the center of their s-cluster, we have also items whose assignment violates an instance-level constraint. For these data items, we use the term *outlier (items).*

Definition 3 (Outlier Item). *Assume that ξ is a clustering of streaming data that has been defined at time point t based on a given set of active constraints* $\widehat{CS}$. *We say that a data item x is an "outlier" for a s-cluster* $c \in \xi$ *or, equivalently, that x belongs to the set* $outliers(c, CS, t)$, *if x is closest to c than any other cluster in ξ, but*
1) x does not belong to the rim of c (see Def. 2), and
2) the cost of assigning x to c with respect to $\widehat{CS}$ *exceeds a threshold* τ_{CS}, *i.e.* $costCV(x, c, \widehat{CS}, t) > \tau_{CS}$

The threshold τ_{CS} *is user-defined and indicates our tolerance to constraint violation. Thus if* $\tau_{CS} = 0$ *then none of the given constraints are allowed to be violated.*

At time point t, the items of a data batch are assigned to the existing s-clusters. After assigning all items, the clusters' nuclei, rims and outliers are recomputed. It may then happen that two clusters have moved closer to each other so that their rims overlap. If their nuclei also overlap, then they are candidates for merging.

Definition 4 (Overlap between s-clusters). *Let c_1, c_2 be two s-clusters built at time point t. The* overlap *of c_2 with respect to c_1 is the number of items in the nucleus of c_2 that are within the nucleus of c_1, normalized to the cardinality of c_1:*

$$overlap(c_1, c_2) = \frac{1}{|c_1|} \times |\{x \in nucleus(c_2) | d(x, rep(c_1)) \leq radius(c_1)\}| \quad (6)$$

The $overlap(c_2, c_1)$ is defined accordingly. We say that the s-clusters c_1, c_2 "overlap" if either $overlap(c_1, c_2)$ or $overlap(c_2, c_1)$ is larger than some threshold $\tau_{overlap}$.

When two nuclei overlap, the s-clusters are so close that they can be merged into one cluster, which we term as *m-cluster*.

Definition 5 (m-cluster). *Let ξ be the set of s-clusters built at time point t. A "multi-cluster" or "m-cluster" $C \subseteq \xi$ is a group of s-clusters, such that:*
1)For each s-cluster $c \in C$ there is a s-cluster $c' \in C$ such that c, c' overlap according to Def. 4.
2) For each s-cluster $c \in \xi$ such that there is a s-cluster $c' \in C$ that overlaps with c it holds that $c \in C$.

Then we define the representatives of a m-cluster C as the set of its s-clusters' representatives, i.e. $rep(C) = \{rep(c) | c \in C\}$.

By this definition, a m-cluster is a maximal set of overlapping s-clusters within the clustering built at a certain time point. The reader should notice the similarity to the definition of "cluster" in DBSCAN [10] as a maximal group of overlapping neighborhoods. Our notion of "overlap" is the counterpart of density-connectivity as proposed in [10]. The advantage of defining m-clusters for constraint-based stream mining is that an algorithm like DBSCAN lends itself elegantly to constraint enforcement, as has been shown in [12].

Definition 6 (o-cluster). *Let t_i be a time point and ξ be the set of s-clusters built at this time point. A group of (constraint-based) outlier items c_o constitutes an "outlier cluster" or "o-cluster" if and only if:*
1) For each $x, y \in c_o$ and for each $c \in \xi$ it holds that $d(x, y) < d(x, rep(c))$ and $d(x, y) < d(y, rep(c))$.
2)For each $x, y \in c_o$ there is no Cannot-Link constraint $< x, y >$.

Similarly to a s-cluster, an o-cluster c_o has a representative $rep(c_o)$ composed of the mean of the elements of c_o. However, we define neither nucleus nor rim for an o-cluster, since the assignment of items to it is not driven by proximity to the o-cluster's center but by remoteness to neighboring s-clusters.

The reader may have noticed that the above definition contains no specification of maximality. In the next section, we explain how o-clusters are built progressively and compensate for the absence of a maximality constraint here.

4 Incremental Clustering towards a Constraint-Stream

Our incremental clusterer `SemiStream` takes as input a stream of data items arriving in snapshots/batches $D_1, \ldots, D_n$, an accompanying stream of arriving constraints $CS_1, \ldots, CS_n$ and the cost function of Eq. 3 for the assignment of items to clusters, subject to Must-Link and Cannot-Link constraints.

At the beginning of the observation time t_0, we assume an initial constraint-based clustering with a conventional semi-supervised algorithm like MPCK-Means [4]. At each subsequent time point t_i, we assign all items of the batch D_i into s-clusters, re-compute the s-clusters' nuclei, their rims and outlier items. Then, we study the proximity of outliers to each other, eventually merging them into o-clusters. Similarly, we check the proximity of s-clusters to their surroundings and merge them into m-clusters - or detach them accordingly.

Processing an Item of the Stream. Let D_i be the set of data items arriving at $(t_{i-1}, t_i]$ and let ξ be the set of s-level clusters at this time point, where some s-clusters may be part of an earlier formed m-cluster. For each item $x \in D_i$ we consider the following cases:

1. There is a s-cluster $c \in \xi$ so that $x \in \mathit{nucleus}(c)$ *and* $\mathit{cost}(x, c, CS, t_i) < \tau_{CS}$: x is assigned to c.
2. Otherwise, x is termed as outlier.

Building Outlier Clusters. If an item is defined to be an outlier, we check whether there is another item that is proximal to it and with which it can be grouped without violating any Cannot-Link constraints. The two items become part of an *outlier cluster*.

When computing the proximity of an item to an s-cluster, we do so on the basis of the cluster's nucleus and radius. When considering the proximity of an outlier to another outlier, we do not have such a basis. We therefore set a constant ε and specify that an item x can be assigned to an outlier cluster c_o if and only if $d(x, \mathit{rep}(c_o)) < \varepsilon$ and there is no violation constraint. The center of an outlier cluster is defined as the mean of the points that belong to this cluster. This allows us to group items together into an outlier cluster without allowing the *withinClusterDistance* of the outlier cluster to grow exuberantly (cf. Def. 1).

The constant ε can be set to a small value, e.g. equal to the smallest nucleus encountered at t_i, thus allowing only groups of very proximal outliers. Alternatively, ε may be set to a large value, e.g. equal to the largest encountered nucleus, thus allowing rather distant outliers to form a cluster. In any case, when specifying ε at each time point t_i, it must be checked whether the constant will allow an overlap between the outlier cluster and one of the s-clusters. In that case, the clusters can be merged.

Moreover, we note that *an outlier cluster can grow into a s-cluster*. We consider only o-clusters that are adequately large and homogeneous:

- An o-cluster is *adequately large* if its cardinality is comparable to the cardinality of the smallest s-cluster.

– An o-cluster is *adequately homogeneous* if its variance is comparable to the average variance of s-clusters.

Candidates for Cluster Merging. After incorporating all items of D_i into the original clustering (i.e. previously defined clustering ξ), some clusters may have grown or moved closer to each other. If their nuclei have come to overlap, we can merge them into an m-cluster. At the same time, some clusters may have moved apart from each other; if they were previously part of an m-cluster, then we must detach them. In general, we consider the following types of candidate clusters:

1. c, c' are *s-clusters*, such that $overlap(c, c') > \tau_{overlap}$ or $overlap(c', c) > \tau_{overlap}$.
2. c, c' are *o-clusters*, such that $d((rep(c), rep(c')) < \varepsilon$; an o-cluster may consist of a single item.

For each pair of candidates we check whether there is a Cannot-Link constraint $< x, y >$ such that $x \in c, y \in c'$. If $cost(x, c', CS) < \tau_{CS}$ (or $cost(y, c, CS) < \tau_{CS}$), then the clusters can be merged. We merge s-clusters into an m-cluster, so that nuclei and rims become part of the m-cluster.

As a third case, we may consider the *merging of an s-cluster with an o-cluster* into an m-cluster, if the center of the o-cluster is within the nucleus of the s-cluster and no Cannot-Link constraints are violated by the merge. However, we consider only o-clusters that can grow into s-clusters, i.e. are adequately large and homogeneous.

In the cases discussed above, the s-clusters may be already members of distinct m-clusters. If this holds true, then the test for constraint violation is extended to all members of each m-cluster involved.

The new clustering ξ', defined after applying any of the above merging cases to ξ, is evaluated based on clustering quality criteria (i.e. cluster compactness and separation). The new clustering is acceptable if its quality does not significantly changes with respect to the quality of ξ.

Candidates for M-Cluster Split. After considering the merging of clusters, we also consider cases where m-clusters need to be split. This may happen because of new Cannot-Link constraints and because of changes in the clusters' nuclei:

1. c is member of an m-cluster C and there is no $c' \in C$ such that $overlap(c, c') + overlap(c', c) > \tau_{overlap}$.
2. c is an o-cluster and member of an m-cluster C and there is no $c' \in C$ such that $rep(c) \in nucleus(c')$.
3. c is member of an m-cluster C and there is a Cannot-Link constraint such that $x \in c$ and $y \in C \setminus c$.

In all cases, the cluster is removed from the m-cluster, possibly causing further cluster detachments. Similarly to the merging case, the clustering defined after splitting is evaluated based widely known cluster quality criteria. Specifically, a

cluster validity method is adopted to evaluate the results of the proposed incremental clustering approach at specific time slots in terms of constraint accuracy as well as clustering compactness and separation [11]. Then if the quality of the newly emerged clustering ξ' is comparable to the old clustering, ξ' is an accepted clustering. The cluster validity method may trigger the re-clustering of data when significant changes are observed in the quality of currently defined clusters.

5 Experimental Evaluation

In this section we present experimental evaluation of our approach using different datasets. We implemented `SemiStream` in JAVA. All experiments were conducted on a 2.53GHz Intel(R) Core 2 Duo PC with 6GB memory.

Data Sets and Constraints Generation. We generate some synthetic datasets with different numbers of clusters and dimensions. For the sake of visualization, we chose here to present the performance of our approach on two-dimensional datasets. Specifically, to evaluate the sensitivity of our algorithm and its clustering quality in case of arbitrarily shaped clusters, we consider the synthetic datasets depicted in Figure 1, which have also been used in [10]. We also generated an evolving data stream, ESD, by choosing one of the three datasets (denoted SD1, SD2, SD3 in Figure 1) 10 times. Each of the datasets contains $10,000$ points and thus the total length of ESD is $100,000$.

To generate constraints for our experimental study, we consider a set of labeled data. Our algorithm is fed with constraints each time a new batch of data arrive. Following a strategy similar to this used in the static semi-supervised approaches, the constraints are generated from labeled data so that they correspond to a percentage of data in the considered window.

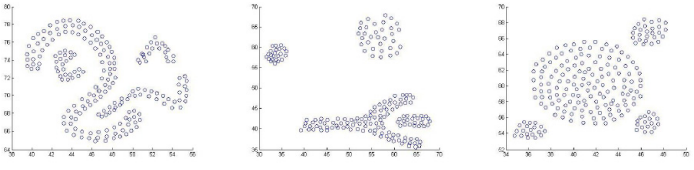

Fig. 1. Visualization of synthetic datasets

Evaluation Criterion. Evaluation of quality of clustering amounts to measuring the purity of defined clusters with respect to the true class labels. For experimental purposes, we assume that we know the label of data items. We run our approach on the set of unlabeled data and we assess the purity of the defined clusters. Then we can define the purity measure as follows: $Purity = \frac{1}{k} \cdot \sum_{i=1}^{k} \frac{|C_i^d|}{|C_i|}$,

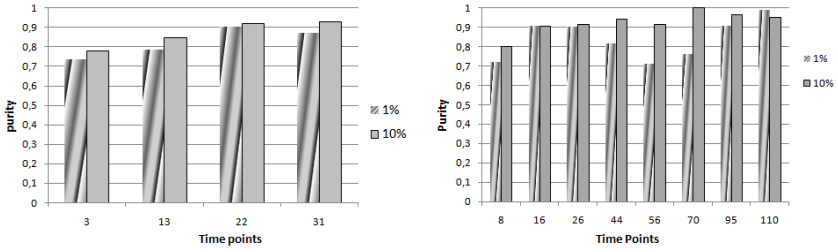

Fig. 2. Clustering purity vs Time points: a) SD1 dataset, horizon = 2, stream speed = 250, b) ESD dataset, horizon = 2, stream speed = 1,000

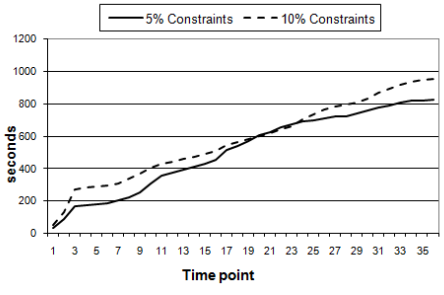

Fig. 3. Execution time vs length of stream

where $|C_i^d|$ denotes the number of majority class items in cluster i, $|C_i|$ is the size of cluster i and k is the number of clusters. The results of clustering purity presented in the following section are an average of results over 5 runs of our algorithm.

Clustering Quality Evaluation. First, we test the clustering quality of `SemiStream` using the SD1 dataset. We consider that the stream speed is 250 data items per time point, the window size is set to 4 time points. Also a new set of stream constraints is received as data arrives which corresponds to a percentage of data in the window. We can observe that the clusters are arbitrarily shaped and thus the majority of clustering algorithms are not able to identify them. Our study shows that the use of constraints can assist with the clustering procedure. Since the points fades out as time passes, we compute the purity of clustering results in a pre-defined horizon (h) from current time. Figure 2(a) presents the purity of clustering defined by `SemiStream` in a small horizon ($h = 2$) when the constraints are 1% and 10% of the arrived data. It can be seen that `SemiStream` gives very good clustering quality. The clustering purity is always higher than 75%. Also Figure 2 (a) shows that increasing the number of constraints, the purity of identified clusters is increased.

Then we evaluate the performance of our algorithm using the evolving data stream ESD at different time units. We set the stream speed at 1,000 points

per time unit and horizon equals to 2. Figure 2 (b) depicts the clustering purity results of our algorithm when the constraints correspond to 1% and 10% of the arrived data. It can be seen that `SemiStream` achieves to identify the evolution of the clusters as new data arrives, resulting in clusters with purity higher than 80%. Also we can observe the advantage that the use of constraints provides. Using 10% of data as constraints, our approach can achieve a clustering model with purity 99%.

Time Complexity. We evaluate the efficiency of `SemiStream` measuring the execution time. The algorithm periodically stores the current clustering results. Thus the execution time refers to the time that our algorithm needs to store clustering results, read data from previous time slots and redefine clusters. Figure 3 shows execution time for synthetic dataset using different number of constraints. We can observe that the execution time grows almost linearly as data stream proceeds.

6 Conclusions

We present `SemiStream`, an algorithm that incrementally adapts a clustering scheme to streaming data which are also accompanied by a set of constraints. Modeling constraints as a stream and associating constraint violation with a penalty function allowed us to design a cost-based strategy for cluster adaptation to snapshots of arriving data and constraints. We introduce the use of i) *s-clusters* to describe dense areas in the data set and ii) *multiple clusters (m-clusters)* to represent overlapping dense areas in order to capture arbitrarily shaped clusters. Moreover we use the structure of outliers clusters to describe a small set of data whose characteristics seem to deviate significantly from average behavior of the currently processed data. Based on a set of adaptation criteria `SemiStream` achieve to observe changes in structure of clusters as data evolve.

References

1. Aggarwal, C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proc. of VLDB (2003)
2. Aggarwal, C., Han, J., Wang, J., Yu, P.: A framework for projected clustering of high dimensional data streams. In: Proc. of VLDB (2004)
3. Basu, S., Banerjee, A., Mooney, R.J.: Semi-supervised Clustering by Seeding. In: Proc. of ICML (2002)
4. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Proc. of ICML (2004)
5. Bilenko, M., Basu, S., Mooney, R.J.: A probabilistic framework for semi-supervised clustering. In: Proc. of KDD, p. 8 (2004)
6. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: Proc. of SDM (2006)
7. Davidson, I., Ravi, S.S.: Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 59–70. Springer, Heidelberg (2005)

8. Davidson, I., Ravi, S.: Clustering with constraints: Feasibility issues and the k-means algorithm. In: Proc. of SDM, Newport Beach, CA (April 2005)
9. Davidson, I., Wagstaff, K.L., Basu, S.: Measuring Constraint-Set Utility for Partitional Clustering Algorithms. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 115–126. Springer, Heidelberg (2006)
10. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-Based Algortihm for Discovering Clusters in Large Spatial Database with Noise. In: Proc. of KDD (1996)
11. Halkidi, M., Gunopulos, D., Kumar, N., Vazirgiannis, M., Domeniconi, C.: A Framework for Semi-Supervised Learning Based on Subjective and Objective Clustering Criteria. In: Proc. of ICDM (2005)
12. Ruiz, C., Menasalvas, E., Spiliopoulou, M.: Constraint-based Clustering. In: Proc. of AWIC. SCI. Springer (2007)
13. Ruiz, C., Menasalvas, E., Spiliopoulou, M.: C-DenStream: Using Domain Knowledge on a Data Stream. In: Gama, J., Costa, V.S., Jorge, A.M., Brazdil, P.B. (eds.) DS 2009. LNCS, vol. 5808, pp. 287–301. Springer, Heidelberg (2009)
14. Ruiz, C., Spiliopoulou, M., Menasalvas, E.: User Constraints Over Data Streams. In: IWKDDS (2006)
15. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-means Clustering with Background Knowledge. In: Proc. of ICML (2001)
16. Zhang, X., Furtlehner, C., Sebag, M.: Data Streaming with Affinity Propagation. In: Daelemans, W., Goethals, B., Morik, K. (eds.) ECML PKDD 2008, Part II. LNCS (LNAI), vol. 5212, pp. 628–643. Springer, Heidelberg (2008)