Clustering Over Data Streams Based on Growing Neural Gas

Mohammed Ghesmoune(✉), Mustapha Lebbah, and Hanene Azzag

University of Paris 13, Sorbonne Paris City LIPN-UMR 7030 - CNRS,
99, av. J-B Clément, 93430 Villetaneuse, France
{mohammed.ghesmoune,mustapha.lebbah,hanene.azzag}@lipn.univ-paris13.fr

Abstract. Clustering data streams requires a process capable of partitioning observations continuously with restrictions of memory and time. In this paper we present a new algorithm, called G-Stream, for clustering data streams by making one pass over the data. G-Stream is based on growing neural gas, that allows us to discover clusters of arbitrary shape without any assumptions on the number of clusters. By using a reservoir, and applying a fading function, the quality of clustering is improved. The performance of the proposed algorithm is evaluated on public data sets.

Keywords: Data stream clustering · Topological structure · GNG

1 Introduction

Clustering is the problem of partitioning a set of observations into clusters such that observations assigned in the same cluster are similar (or close) and the inter-cluster observations are dissimilar (or distant). The other objective of clustering is to quantify the data by replacing a group of observations (cluster) with one representative observation (or prototype). A data stream is a sequence of potentially infinite, non-stationary (i.e., the probability distribution of the unknown data generation process may change over time) data arriving continuously (which requires a single pass through the data) where random access to data is not feasible and storing all arriving data is impractical. The stream model is motivated by emerging applications involving massive data sets; for example, customer click streams, financial transactions, search queries, Twitter updates, telephone records, and observational science data are better modeled as data streams [9]. Mining data streams can be defined as the process of finding complex structures in large data. Clustering data streams requires a process capable of partitioning observations continuously with restrictions of memory and time. In the literature, many data stream algorithms have been adapted from clustering algorithms, e.g., the density-based method DBScan [7,10], the partitioning method k-means [1], or the message passing-based method AP [18]. In this paper, we propose G-Stream, a novel algorithm for discovering clusters of

T. Cao et al. (Eds.): PAKDD 2015, Part II, LNAI 9078, pp. 134–145, 2015.
DOI: 10.1007/978-3-319-18032-8_11

arbitrary shape in an evolving data stream, whose main features and advantages are described as follows: (*a*) The topological structure is represented by a graph wherein each node represents a cluster, which is a set of "close" data points and neighboring nodes (clusters) are connected by edges. The graph size is not fixed but may evolve; (*b*) We use an exponential fading function to reduce the impact of old data whose relevance diminishes over time. For the same reason, links between nodes are also weighted by an exponential function; (*c*) Unlike many other data stream algorithms that start by taking a significant number of data points for initializing the model (these data points can be seen several times), G-Stream starts with only two nodes. Several nodes (clusters) are created in each iteration, unlike the traditional Growing Neural Gas (GNG) [8] algorithm; (*d*) All aspects of G-Stream (including creation, deletion and fading of nodes, edges management, and reservoir management) are performed online; (*e*) A reservoir is used to hold, temporarily, the very distant data points, compared to the current prototypes. The remainder of this paper is organized as follows: Section 2 is dedicated to related works. Section 3 describes the G-Stream algorithm. Section 4 reports the experimental evaluation on both synthetic and real-world data sets. Section 5 concludes this paper.

2 Related Works

This section discusses previous works on data stream clustering problems, and highlights the most relevant algorithms proposed in the literature to deal with this problem. Most of the existing algorithms (e.g. *StreamKM++* [1], *CluStream* [2], *DenStream* [7], or *ClusTree* [12]) divide the clustering process in two phases: (*a*) *Online*, the data will be summarized; (*b*) *Offline*, the final clusters will be generated. Both *CluStream* [2] and *DenStream* [7] use a temporal extension of the *Clustering Feature vector* [17] (called *micro-clusters*) to maintain statistical summaries about data locality and timestamps during the online phase. By creating two kinds of micro-clusters (*potential* and *outlier micro-clusters*), *DenStream* overcomes one of the drawbacks of *CluStream*, its sensitivity to noise. In the offline phase, the micro-clusters found during the online phase are considered as *pseudo-points* and will be passed to a variant of k-means in the *CluStream* algorithm (resp. to a variant of DBScan in the *DenStream* algorithm) in order to determine the final clusters. *StreamKM++* [1] maintains a small outline of the input data using the *merge-and-reduce* technique. The merge step is performed by a means of a data structure, named the *bucket set*. The reduce step is performed by a significantly different summary data structure, the *coreset tree*. *ClusTree* [12] is an anytime algorithm that organizes micro-clusters in a tree structure for faster access and automatically adapts micro-cluster sizes based on the variance of the assigned data points. Any clustering algorithm, e.g. k-means or DBScan, can be used in its offline phase. *SOStream* [10] is a density-based clustering algorithm inspired by both the principle of the DBScan algorithm and that of self-organizing maps (SOM) [11]. *E-Stream* [15] classifies the evolution of data into five categories: appearance, disappearance, self evolution, merge, and

Table 1. Comparison between algorithms (WL: weighted links, 2 phases : online+offline)

Algorithms	based on	topology	WL	phases	remove	merge	split	fade
G-Stream	NGas	✓	✓	online	✓	✗	✗	✓
AING	NGas	✓	✗	online	✗	✓	✗	✗
CluStream	k-means	✗	✗	2 phases	✓	offline	✗	✗
DenStream	DBScan	✗	✗	2 phases	✓	offline	✗	✓
SOStream	DBScan, SOM	✗	✗	online	✓	✓	✗	✓
E-Stream	k-means	✗	✗	2 phases	✓	✓	✓	✓
StreamKM++	k-means	✗	✗	2 phases	✓	✓	✓	✓
StrAP	AP	✗	✗	2 phases	✓	✗	✗	✓
SVStream	SVC, SVDD	✗	✗	online	✓	✓	✓	✓

split. It uses another data structure for saving summary statistics, named α-bin histogram. *StrAP* [18], an extension of the Affinity Propagation algorithm for data streams, uses a reservoir for saving potential outliers. In *SVStream* [16], the data elements of a stream are mapped into a kernel space, and the support vectors are used as the summary information of the historical elements to construct cluster boundaries of arbitrary shape. *SVStream* is based on support vector clustering (SVC) and support vector domain description (SVDD) [16]. *AING* [6], an incremental GNG that learns automatically the distance thresholds of nodes based on its neighbors and data points assigned to the node of interest. It merges nodes when their number reaches a given *upper-bound*. Table 1 summarizes the main features offered by each algorithm in terms of: the basic clustering algorithm, whether the algorithm identifies a topological structure or not, whether the links (if they exist) between clusters (nodes) are weighted, how many phases it adopts (online and offline), the types of operations for updating clusters (remove, merge, and split cluster), and whether a *fading* function is used.

3 Growing Neural Gas Over Data Stream

In this section we introduce Growing Neural Gas over Data Stream (G-Stream) and highlight some of its novel features. G-Stream is based on Growing Neural Gas (GNG), which is an incremental self-organizing approach that belongs to the family of topological maps such as Self-Organizing Maps (SOM) [11] or Neural Gas (NG) [13]. It is an unsupervised algorithm capable of representing a high dimensional input space in a low dimensional feature map. Typically, it is used for finding topological structures that closely reflect the structure of the input distribution. We assume that the data stream consists of a sequence $\mathcal{DS} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n}\}$ of n (potentially infinite) elements of a data stream arriving at times $T_1, T_2, ..., T_n$, where $\mathbf{x_i} = (x_i^1, x_i^2, ..., x_i^d)$ is a vector in $\Re^d$. The notations used in this paper are presented in Table 2. At each time, G-Stream

Table 2. Notations used in the algorithm

Notation	Description
$\mathcal{DS} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$	set of n (potentially infinite) data streams
$\mathbf{x}_i = (x_i^1, x_i^2, ..., x_i^d)$	d-dimensional data point
t_i	time-stamp of data point $\mathbf{x}_i$
$\mathbf{w}_c$	prototype $\mathbf{w}_c = (w_c^1, w_c^2, ..., w_c^d)$ of node c
δ_c	threshold distance of node c
$error(c)$	local accumulated error variable
$weight(c)$	local weight variable
bmu	best matching unit (the nearest node)
α_1	winning node (the nearest node) adaptation factor
α_2	winning node, neighbor adaptation factor
β	cycle interval between node insertions
age_{max}	oldest age allowed for an edge
λ_1	decay factor in the fading function
λ_2	strength factor in weighting edges

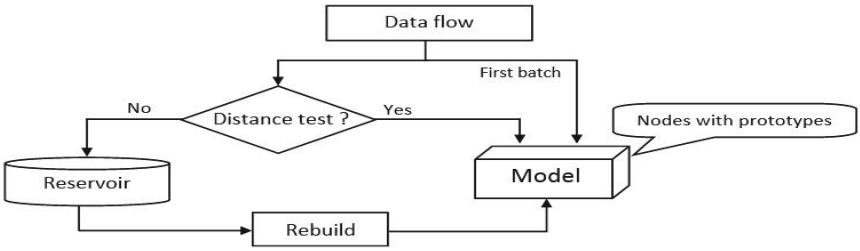


Fig. 1. Diagram of G-Stream algorithm

is represented by a graph $\mathcal{C}$ where each node represents a cluster. Each node $c \in \mathcal{C}$ has a prototype $\mathbf{w}_c = (w_c^1, w_c^2, ..., w_c^d)$ (resp. a distance threshold δ_c) representing its position (resp. the distance from the node to the farthest data point assigned to it). Starting with two nodes, and as a new data point is reached, the nearest and the second-nearest nodes are identified, linked by an edge, and the nearest node with its topological neighbors are moved toward the data point. Each node has an accumulated error variable and a weight, which varies over time using fading function. Using edge management, one, two or three nodes are inserted into the graph between the nodes with the largest error values. Nodes can also be removed if they are identified as being superfluous. Figure 1 represents a schematic diagram of the algorithm.

Fading function: In most data stream scenarios, more recent data can reflect the emergence of new trends or changes in the data distribution [3]. There are

three window models commonly studied in data streams: landmark, sliding and damped. We consider, like many others, the damped window model, in which the weight of each data point decreases exponentially with time t via a fading function $f(t) = 2^{-\lambda_1(t-t_0)}$, where $\lambda_1 > 0$, defines the rate of decay of the weight over time, t denotes the current time and t_0 is the timestamp of the data point. The weight of a node is based on data points associated therewith: $weight(c) = \sum_{i=1}^{m} 2^{-\lambda_1(t-t_{i_0})}$, where m is the number of points assigned to the node c at the current time t. If the weight of a node is less than a threshold value then this node is considered as outdated and then deleted (with its links).

Edge management: The edge management procedure performs operations related to updating graph edges, as illustrated in steps 13-16 of Algorithm 1. The way to increase the age of edges is inspired by the fading function in the sense that the creation time of a link is taken into account. Contrary to the *fading* function, the age of the links will be strengthened by the exponential function $2^{\lambda_2(t-t_0)}$, where $\lambda_2 > 0$, defines the rate of growth of the age over time, t denotes the current time and t_0 is the creation time of the edge. The next step is to add a new edge that connects the two closest nodes. The last step is to remove each link exceeding a maximum age, since these links are no longer useful because they were replaced by younger and shorter edges that were created during the graph refinement in steps 18-22.

Reservoir management: The aim of using the reservoir is to hold, temporarily, the distant data points. As mentioned before, each node has a threshold distance. The first batch of data is assigned to nearest nodes without comparing distance thresholds. The distance threshold of each node is learned by taking the maximum distance of the node to the farthest point that it has been assigned. When the reservoir is full, its data is re-passed for learning. They are placed in the heap of the data stream, $\mathcal{DS}$, to be dealt with first and the distance thresholds of nodes are updated accordingly.

Computational complexity: It is obvious that the most consuming operations, in Algorithm 1, are steps 4, 18-22, 23, and 24 with $O(k)$ time complexity each, where k is the number of nodes in the graph. The node insertion phase (step 22) is repeated $\frac{3.n}{\beta}$ times. Seeking the nearest node (step 4), fading function (step 22), and adjusting the error variable (step 24) phases are repeated whenever a new data point is available, i.e. n times. The other steps have a constant time complexity. Therefore, G-Stream has a complexity given by $n.(3.O(k)) + \frac{3.n}{\beta}.O(k) = n.(3 + \frac{3}{\beta}).O(k) = O(nk)$.

4 Experimental Evaluations

In this section, we present an experimental evaluation of the G-Stream algorithm. We compared our algorithm with the GNG algorithm and several well-known and relevant data stream clustering algorithms, including StreamKM++, DenStream, and ClusTree. Our experiments were performed on MATLAB platform using real-world and synthetic data sets. All the experiments are conducted on a

Algorithm 1. G-Stream

Data: $\mathcal{DS} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
Result: set of nodes $\mathcal{C} = \{c_1, c_2, ...\}$ and their prototypes $\mathbf{W} = \{\mathbf{w}_{c_1}, \mathbf{w}_{c_2}, ...\}$
Initialize the node set $\mathcal{C}$ to contain two nodes, c_1 and c_2: $\mathcal{C} = \{c_1, c_2\}$;
while *there is a data point to proceed* **do**
  Get the next data point in the data stream, $\mathbf{x}_i$;
  Find the nearest node $bmu_1 \in \mathcal{C}$ and the second nearest node $bmu_2 \in \mathcal{C}$;
  if $\|\mathbf{x}_i - \mathbf{w}_{bmu_1}\| > \delta_{bmu_1}$ **then**
    put $\mathbf{x}_i$ in the reservoir;
    if *the reservoir is full* **then** Reservoir management ;
  else
    Increment the number of points assigned to bmu_1;
    $error(bmu_1) = error(bmu_1) + \|\mathbf{x}_i - \mathbf{w}_{bmu_1}\|^2$;
    Move bmu_1 and its topological neighbors towards $\mathbf{x}_i$:
    $\mathbf{w}_{bmu_1} = \mathbf{w}_{bmu_1} + \alpha_1.(\mathbf{x}_i - \mathbf{w}_{bmu_1})$;
    $\mathbf{w}_c = \mathbf{w}_c + \alpha_2.(\mathbf{x}_i - \mathbf{w}_c)$ for all direct neighbors c of node bmu_1;
    Increment the age of all edges emanating from bmu_1 and weight them;
    if bmu_1 *and* bmu_2 *are connected by an edge* **then** set the age of this edge to zero ;
    else create an edge between bmu_1 and bmu_2, and mark its time stamp;
    Remove the edges whose age is greater than age_{max};
    if *the number of points passed is a multiple of a parameter* β **then**
      for *i=1 to 3* **do**
        Find the node q with the maximum accumulated error;
        Find the neighbor f of q with the largest accumulated error;
        Add the new node, r, half-way between nodes q and f;
        Insert the edges connecting the new node r with nodes q and f, and Remove the original edge between q and f;
    Apply *fading*, delete outdated and isolated nodes;
    Finally, decrease the error of all units;

PC with Core(TM)i7-4800MQ with two 2.70 GHz processors, and 8GB of RAM, which runs Windows 7 professional operating system.

4.1 Data Sets and Quality Criteria

To evaluate the clustering quality and scalability of the G-Stream algorithm both real and synthetic data sets are used. The two synthetic data sets used are DS1 and letter4. All the others are real-world publicly available data sets. Table 3 overviews all the data sets used. DS1 is generated by http://impca.curtin.edu.au/local/software/synthetic-data-sets.tar.bz2. The letter4 data set is generated by a Java code https://github.com/feldob/Token-Cluster-Generator. The Sea data set was taken from http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift. The HyperPlan data set was taken from [19]. The real-world databases were taken from the UCI repository [4], which are the

Table 3. Overview of all data sets

Datasets	#records	#features	#classes
DS1	9,153	2	14
letter4	9,344	2	7
Sea	60,000	3	2
HyperPlan	100,000	10	5
KddCup99	494,021	41	23
CoverType	581,012	54	7

KDD-CUP'99 Network Intrusion Detection stream data set (KddCup99) and the Forest CoverType data set (CoverType) respectively.

The algorithms are evaluated using three performance measures: Accuracy (Purity), Normalized Mutual Information (NMI) and Rand index [14]. The value of each measure lies between 0 and 1. A higher value indicates better clustering results. The Accuracy (Purity) averages the fraction of items belonging to the majority class of in each cluster. $Acc = \frac{\sum_{i=1}^{K} \frac{|N_i^d|}{|N_i|}}{K} \times 100\%$, where K denotes the number of clusters, N_i^d denotes the number of points with the dominant class label in cluster i, and N_i denotes the number of points in cluster i. Intuitively, the accuracy (purity) measures the purity of the clusters with respect to the true cluster (class) labels that are known for our data sets [7]. Normalized mutual information provides a measure that is independent of the number of clusters as compared to purity. It reaches its maximum value of 1 only when the two sets of labels have a perfect one-to-one correspondence [14]. The Rand index measures how accurately a clusterer can classify data elements by comparing cluster labels with the underlying class labels. Given N data points, there are a total of $\binom{N}{2}$ distinct pairs of data points which can be categorized into four categories: (a) pairs having the same cluster label and the same class label (their number denoted as N^{11}); (b) pairs having different cluster labels and different class labels (their number denoted as N^{00}); (c) pairs having the same cluster label but different class labels (their number denoted as N^{10}); (d) pairs having different cluster labels but the same class label (their number denoted as N^{01}). The Rand index is defined as: $Rand = (N^{11} + N^{00})/\binom{N}{2}$.

4.2 Evaluation and Performance Comparison

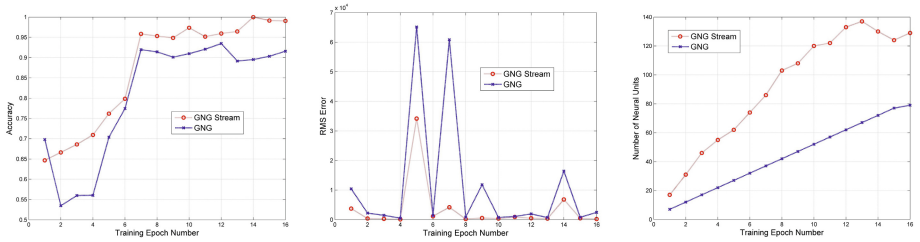
This section aims to evaluate the clustering quality of the G-Stream and compare it to well-known data stream clustering algorithms, as well as the GNG algorithm. As explained in section 3, the GNG and G-Stream algorithms start with two nodes. We used an online version of GNG but without the parameters that we added expressly to show the interest and contribution of these parameters in G-Stream. Therefore, we carried out experiments by initializing two nodes randomly among the first 20 points and we repeated this 10 times. For comparison purposes, we used DenStream [7] and ClusTree [12] from the **stream** R package [5]. Comparison is also performed with StreamKM++ [1] (this latter algorithm was coded in the C language). StreamKM++ was evaluated by

choosing randomly the seed node (please refer to [1] for details) among the first 20 points. DenStream was evaluated by performing a variant of the DBScan algorithm in the offline step. ClusTree was evaluated by performing the k-means algorithm in the offline step by setting the k parameter to 10. All experiments were repeated 10 times and the results (the average value with its standard deviation) are reported in Table 4. In this Table, it is noticeable that G-Stream's Accuracies (Acc) are higher for all data sets as compared to StreamKM++, DenStream and CluStree, except for DenStream for the HyperPlan data set. Its NMI values are higher than the other algorithms except for DenStream for the Sea and HyperPlan data sets. Its Rand index values are higher than the other algorithms except for StreamKM++ for the Sea data set. We recall that G-Stream proceeds in one single phase whereas StreamKM++, DenStream and ClusTree proceed in two phases (online and offline phase).

Figure 2a (resp. Figure 2b) compares G-Stream (red line with circle) with GNG (blue line with cross) with respect to accuracy (resp. RMS error, number of nodes) for the letter4 data set. For almost all cases, the accuracy value (resp. RMS error) of G-Stream is higher (resp. is less) than the one of GNG. Figure 2c compares the two algorithms in terms of the number of nodes creating the graph. Despite that we create several nodes at each iteration (against a single node for GNG), the number of nodes created by G-Stream becomes steady (against a continuous increase for GNG) due to the application of the fading function. The same result can be seen for the remaining data sets.

Table 4. Comparing G-Stream with different algorithms

Datasets		**G-Stream**	StreamKM++	DenStream	ClusTree
DS1	Acc	**0.9809±0.0061**	0.6754±0.0183	0.7740±0.0000	0.6864±0.0275
	NMI	**0.7289±0.0113**	0.7021±0.0209	0.6973±0.0000	0.7064±0.0168
	Rand	**0.8530±0.0024**	0.8443±0.0048	0.8491±0.0000	0.8442±0.0066
letter4	Acc	**0.9832±0.0050**	0.6871±0.0263	0.8110±0.0000	0.8110±0.0000
	NMI	**0.6265±0.0064**	0.5532±0.0219	0.1637±0.0000	0.2425±0.0000
	Rand	**0.8156±0.0015**	0.7941±0.0145	0.5019±0.0000	0.5514±0.0000
Sea	Acc	**0.8386± 0.0021**	0.7886±0.0091	0.8240±0.0001	0.8224±0.0065
	NMI	0.1380±0.0009	0.1463±0.0042	**0.1646±0.0000**	0.1583±0.0095
	Rand	0.4707±0.0001	**0.5072±0.0016**	0.4700±0.006	0.4917±0.0034
HyperPlan	Acc	0.4238±0.0021	0.3966±0.0055	0.4250±0.0000	**0.4380±0.0089**
	NMI	0.0186±0.0009	0.0103±0.0023	**0.0208±0.0000**	0.0170±0.0042
	Rand	**0.7042±0.0008**	0.6674±0.0004	0.6038±0.0000	0.6529±0.0016
KddCup99	Acc	**0.9805±0.0050**	0.6922±0.1140	0.9544±0.0031	0.8182±0.1304
	NMI	**0.6670±0.0089**	0.3926±0.2815	0.6290±0.0300	0.5724±0.2974
	Rand	**0.8380±0.0036**	0.6339±0.2316	0.8164±0.0106	0.8289±0.1798
CoverType	Acc	**0.6085±0.0087**	0.5266±0.0074	0.5850±0.0011	0.5850±0.0000
	NMI	**0.1403±0.0029**	0.0874±0.0086	0.0475±0.0201	0.0362±0.0042
	Rand	**0.6231±0.0008**	0.6106±0.0018	0.4604±0.0070	0.5080±0.0005



(a) accuracy **(b)** RMS error **(c)** Nb nodes

Fig. 2. Accuracy, RMS error, and number of nodes for G-Stream and GNG on letter4

4.3 Visual Validation

Figure 3 shows the evolution of the node creation by applying G-Stream on the letter4 data set (green points represent data points of the data stream and blue points are nodes of the graph with edges in blue lines). It illustrates that G-Stream manages to recognize the structures of the data stream and can separate these structures with the best visualization. Figure 4 compares G-Stream with GNG-online on 2-dimensional data sets (DS1 and letter4), in terms of visual results i.e., the final graph found by GNG-online/G-Stream for each data set. As illustrated on these figures, the G-Stream algorithm is superior to the GNG-online with respect to visual structures found.

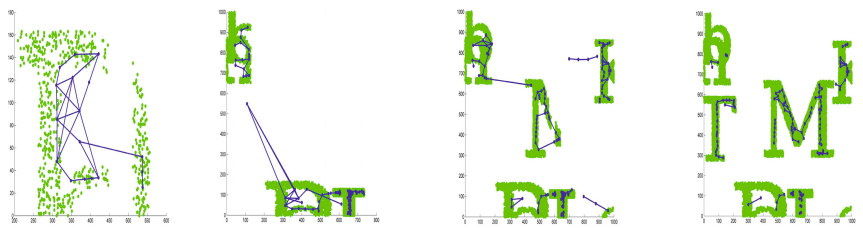
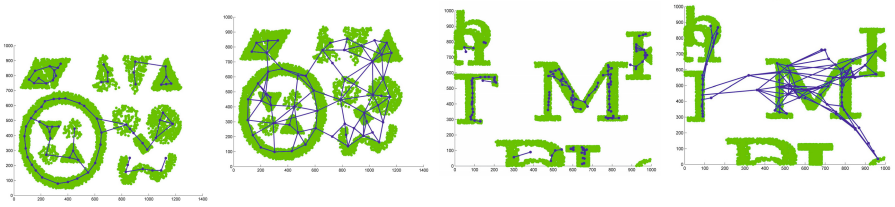


Fig. 3. Evolution of graph creation of G-Stream on letter4 (data set and topological result)

4.4 Evolving Data Streams

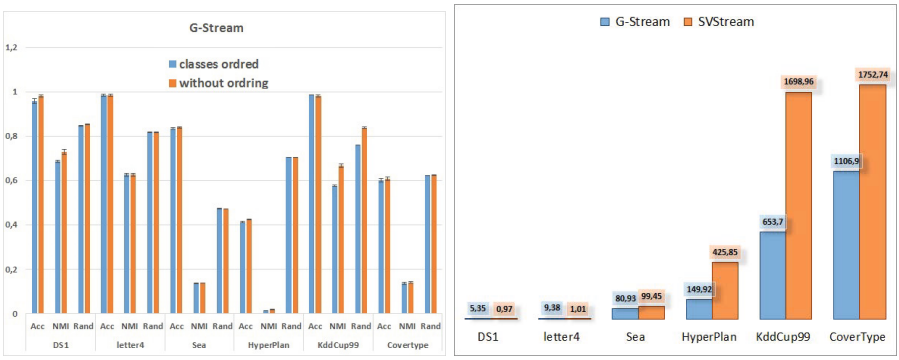
In this subsection, we perform G-Stream on different data streams ordered by class labels to demonstrate its effectiveness in clustering evolving data streams (i.e., data points of the first class arrive in first, then the ones of the second, third, etc. class). In this case, old concepts (class labels) disappear due to the use of fading function. In the same time, new concepts (class labels) appear as



(a) G-Stream on DS1 **(b)** GNG-online on DS1 **(c)** G-Stream on letter4 **(d)** GNG-online on letter4

Fig. 4. Visual result comparison of G-Stream with GNG-online (dataset and topological result)

new data points arrive. We use the same experimental protocol as described in section 4.2, i.e., we did experiments by initializing two nodes randomly among the first 20 points, we repeated this 10 times, and we report the average value with its standard deviation in Figure 5a. Figure 5a shows that G-Stream can find clusters with performance measures as comparable to those without ordering classes.



(a) G-Stream with and without ordering of classes **(b)** Execution time (in seconds)

4.5 Execution Time

The efficiency of algorithms is measured by the execution time. Referring to the computational complexity we calculated in Section 3, the execution time strongly depends to the number of nodes creating the graph and the size of the data stream. We recall that G-Stream is implemented in MATLAB and SVStream is the only MATLAB program that we have (the other algorithms are implemented in Java, R, or C languages). Figure 5b shows the execution time of G-Stream and that of SVStream. We can see that both the execution

time of G-Stream and SVStream grow as the size of the data stream grows, and G-Stream is more efficient than SVStream.

5 Conclusion

In this paper, we have proposed G-Stream, an efficient method for topological clustering an evolving data stream in an online manner. In G-Stream, the nodes are weighted by a fading function and the edges by an exponential function. Starting with two nodes, G-Stream confronts the arriving data points to the current prototypes, storing the very distant ones in a reservoir, learns the threshold distances automatically, and many nodes are created in each iteration. Experimental evaluation over a number of real and synthetic data sets demonstrates the effectiveness and efficiency of G-Stream in discovering clusters of arbitrary shape. Our experiments show that G-Stream outperformed the GNG algorithm in terms of visual results and quantitative criteria such as accuracy, the Rand index and NMI. Its performance, in terms of clustering quality as compared to three relevant data stream algorithms are promising. We plan in the future to implement adaptive windows, make our algorithm as autonomous as possible and develop it in Spark Streaming.

Acknowledgments. This research has been supported by the French Foundation FSN, PIA Grant Big data-Investissements d'Avenir. The project is titled "Square Predict" (http://square-predict.net/). We thank anonymous reviewers for their insightful remarks.

References

1. Ackermann, M.R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., Sohler, C.: StreamKM++: A clustering algorithm for data streams. ACM Journal of Experimental Algorithmics, **17**(1) (2012)
2. Aggarwal, C.C., Watson, T.J., Ctr, R., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB, pp. 81–92 (2003)
3. de Andrade Silva, J., Faria, E.R., Barros, R.C., Hruschka, E.R., de Carvalho, A.C.P.L.F., Gama, J.: Data stream clustering: A survey. ACM Comput. Surv. **46**(1), 13 (2013)
4. Bache, K., Lichman, M.: UCI machine learning repository (2013). http://archive.ics.uci.edu/ml
5. Bolanos, M., Forrest, J., Hahsler, M.: Stream: Infrastructure for Data Stream Mining (2014). http://CRAN.R-project.org/package=stream, r package version 0.2-0
6. Bouguelia, M.R., Belaïd, Y., Belaïd, A.: An adaptive incremental clustering method based on the growing neural gas algorithm. In: ICPRAM, pp. 42–49 (2013)
7. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SDM, pp. 328–339 (2006)
8. Fritzke, B.: A growing neural gas network learns topologies. In: NIPS, pp. 625–632 (1994)

9. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE Transactions on Knowledge and Data Engineering **15**(3), 515–528 (2003)
10. Isaksson, C., Dunham, M.H., Hahsler, M.: SOStream: Self Organizing Density-Based Clustering over Data Stream. In: Perner, P. (ed.) MLDM 2012. LNCS, vol. 7376, pp. 264–278. Springer, Heidelberg (2012)
11. Kohonen, T., Schroeder, M.R., Huang, T.S. (eds.): Self-Organizing Maps, 3rd edn. Springer, Secaucus (2001)
12. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: The ClusTree: indexing micro-clusters for anytime stream mining. Knowledge and Information Systems **29**(2), 249–272 (2011)
13. Martinetz, T., Schulten, K.: A "Neural-Gas" Network Learns Topologies. Artificial Neural Networks **I**, 397–402 (1991)
14. Strehl, A., Ghosh, J.: Cluster ensembles — a knowledge reuse framework for combining multiple partitions. Journal of Machine Learning Research **3**, 583–617 (2002)
15. Udommanetanakit, K., Rakthanmanon, T., Waiyamai, K.: E-Stream: Evolution-Based Technique for Stream Clustering. In: Alhajj, R., Gao, H., Li, X., Li, J., Zaïane, O.R. (eds.) ADMA 2007. LNCS (LNAI), vol. 4632, pp. 605–615. Springer, Heidelberg (2007)
16. Wang, C., Lai, J., Huang, D., Zheng, W.: SVStream: A support vector-based algorithm for clustering data streams. IEEE Trans. Knowl. Data Eng. **25**(6), 1410–1424 (2013). http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.263
17. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: An efficient data clustering method for very large databases. In: SIGMOD Conference, pp. 103–114 (1996)
18. 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)
19. Zhu, X.H.: Stream data mining repository (web site) (2010). http://www.cse.fau.edu/xqzhu/stream.html