Neural Networks 78 (2016) 36–50

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2016 Special Issue

# A new Growing Neural Gas for clustering data streams

Mohammed Ghesmoune*, Mustapha Lebbah, Hanene Azzag

*University of Paris 13, Sorbonne Paris City LIPN-UMR 7030 - CNRS, 99, av. J-B Clément – F-93430 Villetaneuse, France*

ARTICLE INFO

*Article history:*
Available online 26 February 2016

*Keywords:*
Data stream clustering
Topological structure
GNG

ABSTRACT

Clustering data streams is becoming the most efficient way to cluster a massive dataset. This task 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 shapes 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 datasets.

## 1. Introduction

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 datasets; for example, customer click streams, financial transactions, search queries, Twitter updates, telephone records, and observational science data are better modeled as data streams (Guha, Meyerson, Mishra, Motwani, & O'Callaghan, 2003). Mining data streams can be defined as the process of finding a complex structure in these large data. While 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), clustering data streams requires, additionally, a process capable of partitioning observations continuously with restrictions of memory and time. In the literature, many data stream clustering algorithms have been adapted from clustering algorithms, e.g., the partitioning method $k$-means (Ackermann et al., 2016), the density-based method DBSCAN (Cao, Ester, Qian, & Zhou, 2006; Isaksson, Dunham, & Hahsler, 2012), or the message passing-based method Affinity Propagation (AP) (Zhang, Furtlehner, & Sebag, 2008). In this work, we provide a one-pass streaming clustering algorithm titled G-Stream (Growing Neural Gas over Data Streams). We modify Growing Neural Gas (GNG) to obtain a new algorithm, whose main features and advantages are described as below:

- 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.
- 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.
- 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 algorithm (Fritzke, 1994).

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 datasets. 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++* (Ackermann et al., 2016), *CluStream* (Aggarwal et al., 2003), *DenStream* (Cao et al., 2006), or *ClusTree* (Kranen, Assent, Baldauf, & Seidl, 2011)) divide the clustering

* Corresponding author.
*E-mail addresses:* Mohammed.Ghesmoune@lipn.univ-paris13.fr (M. Ghesmoune), Mustapha.Lebbah@lipn.univ-paris13.fr (M. Lebbah), Hanene.Azzag@lipn.univ-paris13.fr (H. Azzag).

http://dx.doi.org/10.1016/j.neunet.2016.02.003
0893-6080/© 2016 Elsevier Ltd. All rights reserved.

process in two phases: (a) *Online*, the data will be summarized; (b) *Offline*, the final clusters will be generated. Both *CluStream* (Aggarwal et al., 2003) and *DenStream* (Cao et al., 2006) use a temporal extension of the *Clustering Feature* (CF) vector (Zhang, Ramakrishnan, & Livny, 1996) (called *micro-clusters*) to maintain statistical summaries about data locality and timestamps during the online phase. *CluStream* uses the concepts of a *pyramidal time frame* in conjunction with a *micro-clustering* approach. The online phase stores $q$ micro-clusters in (secondary) memory, where $q$ is an input parameter. Each micro-cluster has a maximum boundary, which is computed as the standard deviation of the mean distance of the data points to their centroids multiplied by a factor $f$. Each new point is assigned to its closest micro-cluster (according to the Euclidean distance) if the distance between the new point and the closest centroid falls within the maximum boundary. If so, the point is absorbed by the cluster and its summary statistics are updated. If none of the micro-clusters can absorb the point, a new micro-cluster is created. This is accomplished by either deleting the oldest micro-cluster or by merging two micro-clusters. The oldest micro-cluster is deleted if its timestamp is below a given threshold $\delta$ (input parameter). The $q$ micro-clusters are stored in a secondary storage device in particular time intervals that decrease exponentially, which are referred as *snapshots*. These snapshots allow the user to search for clusters in different time horizons through a *pyramidal time window* concept. The use of the pyramidal pattern provides an effective trade-off between the storage requirements and the ability to recall summary statistics from different time horizons. This summary information in the micro-clusters is used by an offline component which is dependent upon a wide variety of user inputs such as the time horizon or the granularity of clustering (Aggarwal et al., 2003).

*DenStream* (Cao et al., 2006) is a density-based data stream clustering algorithm that also uses a feature vector based on the *CF* vector. By creating two kinds of micro-clusters (*potential* and *outlier micro-clusters*), in its online phase, *DenStream* overcomes one of the drawbacks of *CluStream*, its sensitivity to noise. Each *potential-micro-cluster* structure has an associated weight $w$ that indicates its importance based on temporality (micro-clusters with no recent points tend to lose importance, i.e. their respective weights continuously decrease over time in *outdated-micro-clusters*). When a new data point arrives, the algorithm tries to insert it into its nearest *potential-micro-cluster* based on its updated radius. If the insertion is not successful, the algorithm tries to insert the data point into its closest *outlier micro-cluster*. If the insertion is successful, the cluster summary statistics will be updated accordingly. Otherwise, a new *outlier micro-cluster* is created to absorb this point. However, the non-release of the allocated memory when either deleting a micro-cluster or merging two old micro-clusters is considered as a limitation of the *DenStream* algorithm as well as the time-consuming pruning phase for removing outliers (Amini, Teh, & Saboohi, 2014). 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++* (Ackermann et al., 2016) maintains a small outline of the input data using the *merge-and-reduce* technique. The merge step is performed by a means of data structure, named the *bucket set*, which is a set of $L$ buckets (also named buffers), where $L$ is an input parameter. The reduce step is performed by a significantly different summary data structure, the *coreset tree*, when we consider that it reduces $2m$ points to $m$ points. When a new data point arrives, it is stored in the first bucket. If the first bucket is full, all of its data are moved to the second bucket. If the second bucket is full, the two buckets are merged resulting in $2m$ points, which is then reduced to $m$ points, by the construction of a *coreset tree*, stored in the third bucket. In its offline phase, a variant of $k$-means, called $k$-means++ (Arthur & Vassilvitskii, 2007), is used for finding the final clusters.

*ClusTree* (Kranen et al., 2011) 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. All algorithms based on the $k$-means approach (e.g. *StreamKM++* (Ackermann et al., 2016), *CluStream* (Aggarwal et al., 2003), or *ClusTree* (Kranen et al., 2011)) inherit the drawbacks of the latter, which are the non-ability to find clusters of arbitrary shapes (only spherical clusters may be found); and the need to know in advance the number of clusters.

*SOStream* (Isaksson et al., 2012) is a density-based clustering algorithm inspired by both the principle of the DBSCAN algorithm and self-organizing maps (SOM) (Kohonen, Schroeder, & Huang, 2001), in the sense that a winner influences its immediate neighborhood. When a new point arrives, the nearest cluster is selected, based on the Euclidean distance to existing micro-clusters, and then absorbs this point if the calculated distance is less than a dynamically defined threshold. It also assigns the micro-clusters' neighbors to the nearest cluster. If the new point is not absorbed by any micro-cluster, a new micro-cluster is created for it. In the *SOStream* algorithm, merging, updating and adapting dynamically the threshold value for each cluster are performed in an online manner. However, no split feature is proposed in the algorithm. *SOStream* also uses an exponential fading function to reduce the impact of old data whose relevance diminishes over time.

*E-Stream* (Udommanetanakit, Rakthanmanon, & Waiyamai, 2007) classifies the evolution of data into five categories: appearance, disappearance, self evolution, merge, and split. It uses another data structure for saving summary statistics, named the $\alpha$-bin histogram. *E-Stream* starts empty, and every new point either is mapped into one of the existing clusters (based on a radius threshold) or a new cluster is created around it. Any cluster not meeting a predefined density level is considered inactive and remains isolated until achieving a desired weight. The algorithm employs an exponential decay function to weigh down the influence of older data, thus focusing on keeping an up-to-date view of the data distribution. Clusters which are not active for a certain time period may be deleted from the data space.

*StrAP* (Zhang et al., 2008), an extension of the Affinity Propagation (AP) algorithm (Frey & Dueck, 2007) for data stream, uses a reservoir for saving potential outliers. At first, the stream model (the first prototypes) is initialized by using the AP algorithm. The distance between the new data point and the nearest prototype is calculated and the decision whether this point is considered as an outlier or added to the stream model is made. A statistical test is employed for change point detection. The stream model is rebuilt if triggered by the change detection test or if the number of outliers exceeds the reservoir size. In *SVStream* (Wang, Lai, Huang, & Zheng, 2013), 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). The curious reader can refer to Amini et al. (2014) and de Andrade Silva et al. (2013) for more details on data stream clustering algorithms.

A number of authors have proposed variations on the Growing Neural Gas (GNG) approach (this network is described in more detail in Section 3.1). The GNG algorithm creates a new node every $\lambda$ iterations ($\lambda$ is fixed by the user as an input parameter). Hence, it is not adapted for data streams, or non-stationary datasets, or to novelty detection. In order to deal with non-stationary datasets,