Information Systems 62 (2016) 60–73

Contents lists available at ScienceDirect

Information Systems

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

SNCStream$^+$: Extending a high quality true anytime data stream clustering algorithm

Jean Paul Barddal [a], Heitor Murilo Gomes [a], Fabrício Enembreck [a], Jean-Paul Barthès [b]

[a] Programa de Pós-Graduação em Informática – Pontifícia Universidade Católica do Paraná, Brazil
[b] UTC - Université de Technologie de Compiègne, France

ARTICLE INFO

Article history:
Received 27 May 2015
Received in revised form
29 February 2016
Accepted 17 June 2016
Recommended by Laks Lakshmanan
Available online 25 June 2016

Keywords:
Data stream clustering
Unsupervised learning
Social networks theory

ABSTRACT

Data Stream Clustering is an active area of research which requires efficient algorithms capable of finding and updating clusters incrementally as data arrives. On top of that, due to the inherent evolving nature of data streams, it is expected that algorithms undergo both concept drifts and evolutions, which must be taken into account by the clustering algorithm, allowing incremental clustering updates. In this paper we present the Social Network Clusterer Stream$^+$ (SNCStream$^+$). SNCStream$^+$ tackles the data stream clustering problem as a network formation and evolution problem, where instances and micro-clusters form clusters based on homophily. Our proposal has its parameters analyzed and it is evaluated in a broad set of problems against literature baselines. Results show that SNCStream$^+$ achieves superior clustering quality (CMM), and feasible processing time and memory space usage when compared to the original SNCStream and other proposals of the literature.

1. Introduction

In the latest years, the interest in mining massive, potentially unbounded, data instances which arrive at rapidly rates, namely *data streams*, has grown substantially. In this context, a variety of inductive learning techniques were developed to extract useful knowledge from data streams and achieved concrete results in both supervised [1,2] and unsupervised learning [3–6] approaches.

Even though a lot of effort has been devoted to the development of supervised machine learning, specially classification, many real world problems do not include labelled data. Thus, for many applications it is only suitable to apply unsupervised techniques, such as clustering. In these scenarios, processing data in high velocity and extracting useful knowledge from these sequences of data is a current research challenge achievable solely with unsupervised approaches. Examples of data stream clustering applications are: consumer click streams, telephone usage flows, multimedia data mining [7,8], computer networks intrusion detection [3], XML and HTML structures mining [9] and sensor network data clustering [10].

Data stream clustering can be described as the act of grouping streaming data in meaningful clusters [8]. As any other task in data streams, clustering must be performed within limited resources (time and memory) and deal with data stream peculiarities, i.e. changes in data distribution, namely concept drift [11]; and clusters appearance and disappearance, namely concept evolution [12]. Apart from the time and memory constraints, two requirements are long-awaited for data stream clustering algorithms:

E-mail addresses: jean.barddal@ppgia.pucpr.br (J.P. Barddal), hmgomes@ppgia.pucpr.br (H.M. Gomes), fabricio@ppgia.pucpr.br (F. Enembreck), jean-paul.barthes@utc.fr (J.-P. Barthès).

http://dx.doi.org/10.1016/j.is.2016.06.007
0306-4379/

(i) there must be no assumptions about the number of clusters to be found and (ii) algorithms must be able to discover clusters with arbitrary shapes, since most of real data streams are irregular, i.e. do not follow a Gaussian distribution [4,5].

In [4] we presented the Social Network Clusterer Stream (SNCStream), a density and social network-based algorithm. SNCStream differs from other algorithms presented in the literature since it does not perform an offline step, i.e. it does not need batch processing for finding final clusters. Furthermore, SNCStream does not need the information of the amount of clusters to be found and is able to discover non hyper-spherical clusters.

In this paper we present SNCStream$^+$, an extension to our prior proposal SNCStream. SNCStream$^+$ inherits the density and social network characteristics of the original SNCStream. Although the algorithm asymptotical computational complexity remains the same when compared to the original SNCStream, our proposal is more efficient since it executes in decreased complexity in the average case. We present empirical results comparing SNCStream$^+$ to SNCStream and other literature baselines. These experiments evaluate three important aspects of data stream clustering: memory usage, processing time and clustering quality (CMM).

Additionally, in this paper we also contribute with several studies over important aspects of SNCStream$^+$:

- Vertices (instances and micro-clusters) are added to SNCStream's network performing ω connections (edges). Intuitively, if ω is assigned with a large value (e.g. 15), the network may become so dense that only one cluster will be found. Conversely, if ω is assigned with a small value (e.g. 1), the network will become sparse and a high amount of clusters will be found. In this paper we investigate the impact of parameter ω in terms of clustering quality, and suggest a default value.
- SNCStream verifies micro-clusters' weights accordingly to a clean-up window size T_p, originally used in [5]. If they take too long to perform such verifications, micro-clusters that do not represent the current stream concept will join the network and jeopardize the quality of final clusters. Conversely, performing this verification too often may jeopardize algorithms' processing time. In this paper, we verify the impact of T_p in terms of clustering quality and processing time.
- The original SNCStream uses Euclidian distance to compute dissimilarities between instances and micro-clusters. In [4], we observed that SNCStream's clustering quality decays in high dimensional streams [13]. In this paper, we empirically verify that the original SNCStream falls in the vastitude of high dimensional streams, and show how it is possible to extend it to decrease the impact of the curse of dimensionality [3] by adopting specific distance metrics.

This paper is divided as follows: Section 2 presents the problem of data stream clustering and surveys related work. Section 3 presents our proposal, the Social Network Clusterer Stream$^+$, detailing its phases and presenting a time and space complexity analysis. Later, Section 4 presents the empirical evaluation of SNCStream$^+$, discussing parameter sensitivity and highlighting the empirical evaluation against literature baselines, where our proposal shows high clustering quality and feasible processing time and memory space usage. Finally, Section 5 concludes this paper.

2. Data stream clustering

Let $\mathcal{S}$ denote a data stream providing instances $\vec{x}_i$ rapidly and intermittently, where $\vec{x}_i$ is a d-dimensional data object which arrives at a timestamp t_i. Extracting useful knowledge from data streams is a challenge. Most traditional data mining techniques assume that data is generated by a stationary probability distribution and stored in a finite database. Thus, it is feasible to iteratively process data using a batch algorithm [14]. This same approach cannot be used for data stream mining since a stream S is assumed to provide a large, potentially infinite, amount of instances. More important, the stream's data distribution may change over time. Therefore, it is neither practical nor possible to store all instances in memory before processing. As a consequence, instances should be processed right after their arrival (single-pass processing) or in limited size chunks [8].

Developing data stream clustering algorithms which are fast use a limited amount of memory and present high acuity is an effervescent research area [8]. The task of data stream clustering can be described as the act of grouping streaming data in a set of meaningful clusters $\mathcal{K} = \{k_1, k_2, \ldots, k_K\}$ [15]. The principle behind most of clustering techniques is that instances within a cluster are more similar to each other when compared to instances in other clusters [16].

Data stream clustering algorithms must be capable of dealing with concept drifts and evolutions. Concept drifts occur whenever the data distribution changes [11], while concept evolution refers to the appearance or disappearance of clusters [12]. Ideally, clustering algorithms must: (i) detect concept drifts and adapt its clusters accordingly; (ii) detect concept evolutions and create/delete clusters automatically; (iii) discern between seeds of new clusters and noisy data; and finally (iv) not rely on a multitude of parameters.

This last characteristic is very important in a data stream processing context, since optimal values for parameters very often depend on the incoming data. Thus, if a drift or evolution happens the parameters' values become outdated. A canonical example is the parameter that defines the ground-truth number of clusters to be found K. Any algorithm that demands a predefined value of K is unable to cope with concept evolutions.

A variety of data stream clustering algorithms were developed in the last years. Most of these algorithms [3,5,17,6] process incoming instances intercalating online and offline steps.

During the online step, algorithms incrementally update specific data structures aiming at dealing with the

evolving nature of data streams, and time/space constraints. To represent instances and comply to time and memory space restrictions, the feature vector data structure if often used. A feature vector is a triple vector $CF = \langle LS, SS, N \rangle$, where LS stands for the sum of the objects summarized, SS is the squared sum of these objects and N is the amount of objects [8]. Feature vectors are able to represent hyper-spherical clusters incrementally due to its incremental and additive properties. Basically, an instance $\vec{x}_i$ can increment a feature vector CF_j as follows: $LS_j \leftarrow LS_j + \vec{x}_i$, $SS_j \leftarrow SS_j + \vec{x}_i^2$ and $N_j \leftarrow N_j + 1$. As for the additive property, two feature vectors CF_i and CF_j can be merged in a third CF_l as follows: $LS_l \leftarrow LS_i + LS_j$, $SS_l \leftarrow SS_i + SS_j$ and $N_l \leftarrow N_i + N_j$.

In order to assign more importance to recently retrieved instances, various models featuring sliding, damped and landmark windows were developed [8].

During the offline step, traditional batch clustering algorithms, such as k-means [18] and DBSCAN [19] are applied. These algorithms must be adapted to work with data structures that summarizes many instances, for example, feature vectors.

In the next sections we briefly survey the state-of-the-art algorithms for data stream clustering.

2.1. CluStream

During the online step, CluStream uses a landmark windowing technique, whose size is determined by a parameter Horizon ($\mathcal{H}$), determining disjoint chunks that keep statistical summaries in different granularity levels for both spatial and temporal aspects of the stream [3]. CluStream assumes a number q of feature vectors that should be maintained at any instant of the stream. These q feature vectors are initially computed with a amount of instances $\mathcal{N}$, given by the user. Instances obtained from the stream should be merged within existing CFs or should initiate new ones according to user-given thresholds.

During the offline step, the original CluStream uses an adaptation of the k-means algorithm [18] in order to obtain clusters based on the q feature vectors computed during the online step. Hence, one of the major limitations of CluStream is its inability to find non-hyper-spherical clusters and the fact that it depends on a parameter of clusters to be found K.

2.2. ClusTree

ClusTree [6] maintains feature vectors in a R-Tree [20]. ClusTree hypothesis is the creation of a hierarchy of feature vectors at different granularity levels. Accordingly to user-given thresholds, it is determined whether an instance should be merged with an existing feature vector. In the negative case, a new feature vector is created and added to the R-Tree. ClusTree copes with noisy data by using outlier-buffers.

During the offline step, algorithms such as k-means [18] and DBSCAN [19] are used to find final clusters, where feature vectors' centers are treated as centroids.

2.3. DenStream

DenStream [5] extends the DBSCAN [19] algorithm and presents the definition of core micro-cluster. A core object is an object which ϵ-neighborhood has at least ψ neighbors and a dense area is the union of all ϵ-neighborhoods of all core objects. Therefore, a core micro-cluster is a CF in a time instant t is defined as $CMC(w, c, r, t_c, t_u)$ to a group of near instances $\vec{x}_i, \vec{x}_{i+1}, \ldots, \vec{x}_n$ where w stands for its weight, c its center, r its radius, t_c the timestamp of its creation and the timestamp of its last update (increment or addition) t_u; where $w \geq \psi$, $r \leq \epsilon$, $f(\cdot)$ is a exponential decay function and $d(\cdot,\cdot)$ is an Euclidean distance.

Core micro-clusters are classified based on their weights w: if $w \geq \beta\psi$, it is a potential micro-cluster (PMC), otherwise, it is considered an outlier micro-cluster (OMC).

DenStream's online step aims on maintaining a group of potential micro-clusters stored in main memory since it assumes that most part of the instances $\vec{x}_i$ retrieved from $\mathcal{S}$ belong to clusters; and outlier micro-clusters, which are stored in secondary memory. Arriving instances $\vec{x}_i$ are merged within an existing micro-cluster if the resulting micro-cluster's radius $r(PMC_p + \vec{x}_i)$ is below ϵ. If this merge occurs with an outlier micro-cluster, and its weight w is above $\beta\psi$, it is removed from secondary memory and promoted to a potential micro-cluster.

All micro-clusters weights' decay exponentially. As stated earlier, in case the weight w of a micro-cluster is below $\beta\psi$ it should be treated as an outlier micro-cluster. Verifying all potential micro-clusters weights according to the arrival of each instance $\vec{x}_i$ can be too computationally costly, therefore, DenStream encompasses a periodic verification performed based on a clean-up window size T_p. Eq. (1) presents the computation of the clean-up window size T_p presented in [5]:

$$T_p = \left\lceil \frac{1}{\lambda} \log \frac{\beta\psi}{\beta\psi - 1} \right\rceil \tag{1}$$

Finally, the offline step of DenStream applies a variation of DBSCAN to find final cluster upon the potential micro-clusters maintained during the online step, thus ignoring outlier micro-clusters.

2.4. HAStream

To automatically detect clusters of different densities, HAStream [17] performs a hierarchical density-based clustering that automatically and independently adapts its density thresholds accordingly to the arriving streaming data.

During the online step of HAStream all retrieved instances are processed using any feature vector model such as presented in CluStream [3], ClusTree [6] or DenStream [5].

On the offline step, HAStream generates the final clusters using a hierarchical density-based clustering. Since returning every possible hierarchical cluster forces the evaluator to define the correct amount of clusters, HAStream uses the concept of cluster stability [17]. In order to obtain the final clusters, HAStream computes a flat clustering by maximizing cluster stabilities in branches of the dendrogram.

2.5. Social network clusterer stream

The Social Network Clusterer Stream (SNCStream) [4] is a one-step data stream clustering algorithm capable of finding non-hyper-spherical clusters. SNCStream models the stream clustering problem as the evolution of a social network. SNCStream's hypothesis is that vertices (instances or micro-clusters), by constantly performing local rewirings of its edges, split the network in subgroups, the so desired clusters. The network $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{W})$ starts empty, and an instance $\overrightarrow{x}_i$ retrieved from the stream is inserted in the network as a vertex and edges are created connecting $\overrightarrow{x}_i$ to its ω closest neighbors. However, the core of SNCStream is the rewiring procedure. After $\overrightarrow{x}_i$'s addition to the network, all other nodes $v_i \in \mathcal{V}$ perform rewirings based on homophily, thus, v_i replaces edges with higher dissimilarities with new edges with closest neighbors, which account for lower dissimilarity. Due to the rewiring process, communities of instances appear naturally since the amount of intra-clusters edges between similar instances grows and of dissimilar instances (between clusters), shrinks.

After a initial window $\mathcal{N}$, all existing instances in $\mathcal{V}$ are converted into potential micro-clusters, enabling SNCStream to summarize great amounts of data with decreased memory space by using potential and outlier micro-clusters data structures. SNCStream adapts to both concept drifts and evolutions by using a damped window model, where micro-clusters' weights decay as the stream progresses.

We refrain from detailing SNCStream's functioning since the overall method is analogous to the one presented in Section 3. Overall, SNCStream's asymptotic computational complexity is of $\mathcal{O}(V\omega^2)$ (where V is the amount of vertices currently in the network), a relatively high computational cost when compared to the latter surveyed algorithms, which are linear or logarithmic, however, not prohibitive since ω is usually a small value (e.g. 4) and $V \approx 50$ in a variety of domains [4].

3. The social network clusterer stream$^+$

One of the drawbacks of existing data stream clustering algorithms is the existence of batch processing during the offline step, which is responsible for finding clusters based on statistical summaries computed during the online step. Ideally, a data stream clustering algorithm must be able to incrementally update not only its statistical summaries but also its clusters, thus, eliminating the necessity of batch processing upon clustering requests on offline steps. If no batch processing is needed to find final clusters, then the clustering algorithm is said to be anytime.

During the offline step, conventional batch clustering techniques such as k-means [18] and DBSCAN [19] are commonly used and rely on experts' knowledge, which must set parameters accordingly to the stream's data domain. In the k-means approach, besides being limited to find hyper-spherical clusters, it is unfeasible to assume that the user is an expert in the data domain, thus might not be able to define correctly the amount of clusters to be found upon every clustering request. Additionally, clusters may appear and disappear as the stream progresses, therefore, a very common implementation approach is to inform k-means with the correct amount of clusters to be found K at every offline step, a solution that departs from most real-world scenarios.

Conversely, the major limitation of using DBSCAN on the offline step is its amount of parameters and their influence in final clusters. Small variations in density parameters may jeopardize final clusters, finding a very smaller (even none) or greater amount of clusters when compared to the real ones. Again, two possibilities are stated: (i) assume that the user is a domain expert and will be able to define its parameters correctly or (ii) use optimization algorithms to define suboptimal parameters for each domain.

All the cited parametrization alternatives are undesirable and are impracticable in real data streams environments: assuming that the user knows the data domain is unrealistic; informing k-means about the correct amount of clusters distorts the clustering task goal; and the usage of optimization algorithms may find optimal values for certain chunks of the stream, yet, we cannot assume that such values imply in good results for future chunks. Therefore, it is long awaited that data stream clustering algorithms become capable of finding clusters in entirely unsupervised fashion, with (i) a smaller amount of parameters, while (ii) dealing with time and memory space constraints, (iii) finding non-hyper-spherical clusters, detecting and adapting to concept drifts and evolutions; and (iv) discerning between the seeds of new clusters and noisy data [3,15,5,14,6,8].

The rationale behind the Social Network Clusterer Stream$^+$ (SNCStream$^+$) is that, micro-clusters, besides being incrementally updated accordingly to the arrival of instances from $\mathcal{S}$, are also able to form clusters incrementally, using a formation and evolution network model. This network model is based on homophily, where micro-clusters form clusters naturally during the online step, eliminating the necessity of batch techniques such as k-means and DBSCAN. Whenever a clustering request occurs, SNCStream$^+$ returns its subgroups, therefore, performs data stream clustering in true real-time fashion.

The Social Network Clusterer Stream$^+$ is based on the hypothesis that data points within clusters are related (connected) due to high similarity and points inter-clusters are not, due to low similarity. SNCStream$^+$ models this problem as a social network [21] represented as a graph $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{W})$, where $\mathcal{V}$ represents the set of vertices (instances or micro-clusters), $\mathcal{E}$ is the set of edges which relate vertices in $\mathcal{V}$; and $\mathcal{W}$ is the set is edges weights, it is, distances between neighbor vertices. Additionally, groups of connected vertices $S \subseteq \mathcal{V}$ of the network represent clusters which form the final clustering $\mathcal{K}$.

SNCStream$^+$ is divided in three major phases: Initial Network Formation, Network Transformation and Network

Evolution. These phases are responsible for, respectively, creating an initial network structure based on instances retrieved from the data stream; transform this initial network of instances in a network of micro-clusters; and evolve the network with the arrival of further instances. We emphasize that all the above phases are located at the online step of the algorithm.

The following sections detail each of the phases encompassed by SNCStream$^+$.

3.1. Initial network construction

This phase aims on developing a network based on the beginning of the stream. SNCStream$^+$ starts with an empty network $\mathcal{G} = (\mathcal{V} = \emptyset, \mathcal{E} = \emptyset, \mathcal{W} = \emptyset)$. Based on an initial window size parameter $\mathcal{N}$, instances $\vec{x}_i$ are retrieved from the data stream $\mathcal{S}$ and inserted in the network as vertices.

On the arrival of $\vec{x}_i$, it is simply added to the vertex set $\mathcal{V}$ and edges are built connecting $\vec{x}_i$ to the ω closest neighbors (accordingly to a distance metric), where ω is a user-given parameter. Newly created edges e_i are then add to the edge set $\mathcal{E}$ and its corresponding distances (weights) are stored in $\mathcal{W}$.

The insertion procedure guarantees that the last inserted instance is connected to its ω closest instances, however, the same may not be true for the other instances. To exemplify such problem, we refer to Fig. 1, which depicts the insertion of 4 vertices in a network, where one can see that the vertex v_1 is not connected to the closest ω possible vertices (Fig. 1(d)) when assuming an Euclidian distance.

Therefore, SNCStream$^+$ applies a rewiring procedure based on homophily. Homophily occurs in real social networks where nodes replace its connections with time based on similarity and affinity [21]. The rewiring procedure adopted by SNCStream$^+$ promotes vertices to seek the best connections possible, it is, maintaining edges with the closest vertices possible while preserving its current degree ($deg(\cdot)$), a trait which segments the network in subnetworks (clusters). Rewiring is based on the hypothesis that neighbors of a vertex v_j tend to be neighbors of v_i, if v_i is adjacent to v_j. In another words, 2-hop neighbors (neighbors of current neighbors) of a vertex v_i are likely to be neighbors of v_i and therefore, are accounted for distance comparison.

To exemplify the rewiring procedure, Fig. 2 continues the example from Fig. 1. SNCStream determines all of the 2-hop neighbors of v_1, thus determining the closest vertices when compared to the current neighbors (1-hop neighbors). In Fig. 2(b) it occurs that $d(v_1, v_4) < d(v_1, v_3)$, therefore, v_1 replaces the connection with v_3 by an edge with v_4, as seen in Fig. 2(c).

Finally, the rewiring procedure has an important feature: it segments the network in subnetworks (clusters). Fig. 3 presents a complete example of network initial construction, where two clusters emerge in Fig. 3(r) due to the rewiring process.

3.2. Network transformation

The insertion and rewiring procedures are capable of finding clusters incrementally. However, they have a strong limitation: the network grows indefinitely. This prevents the algorithm to be scalable to data streams, since it is impossible to store the whole data stream and due to the growing amount of distance computations needed at the arrival of each instance $\vec{x}_i$ retrieved from $\mathcal{S}$.

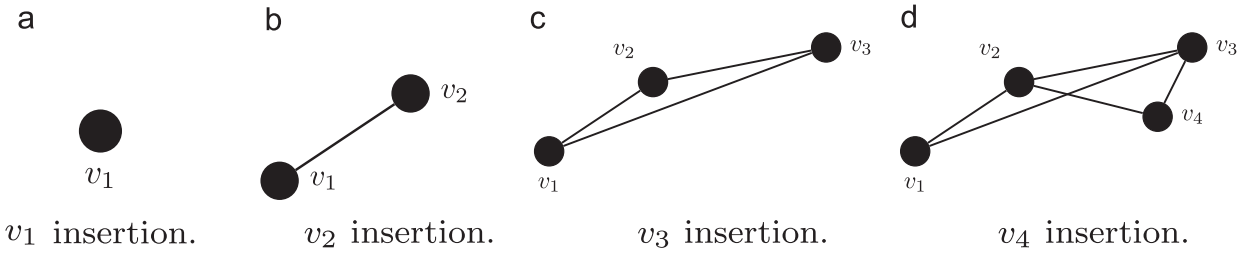


Fig. 1. Insertion process example assuming $\omega = 2$.

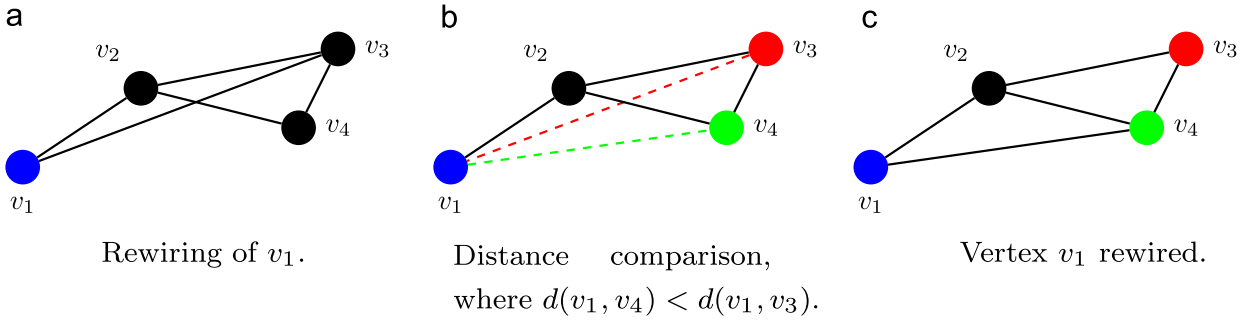


Fig. 2. v_1 rewiring procedure example, where $deg(v_1) = 2$.

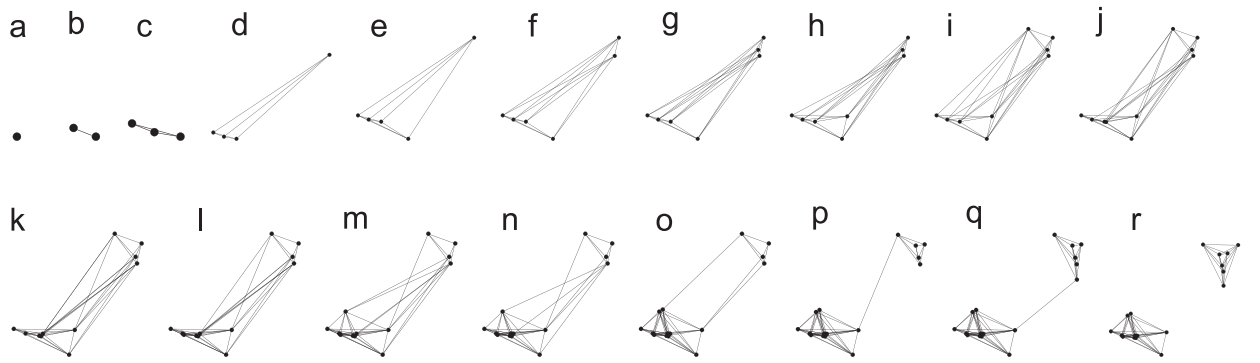


Fig. 3. Example of network initial construction where two clusters emerge due the rewiring process. Adapted from [4].

Therefore, SNCStream$^+$ adopts statistical summary structures to summarize a great amount of instances in decreased memory space, and perform "forget" older data to adapt to new concepts rapidly.

SNCStream$^+$ uses an initial window of size $\mathcal{N}$ to retrieve the first instances from the stream. When the amount of vertices in the network $|\mathcal{V}|$ reaches $\mathcal{N}$, all vertices $v_i \in \mathcal{V}$, which until this moment represented instances, are converted to potential micro-clusters with $LS = \vec{x}_i$, $SS = \vec{x}_i^2$, $N=1$ and $t_c = t_u = t_i$.

Finally, an outlier micro-cluster buffer $\mathcal{B}$ is instantiated. As in DenStream [5], this buffer is used to store micro-clusters with weights below $\beta\psi$, which do not participate of the network nor affect final clustering.

3.3. Network evolution

After the transformation of the network into a micro-clusters network, our proposal incrementally updates its statistical summaries based on DenStream [5].

At the arrival of an instance $\vec{x}_i$, SNCStream$^+$ determines the closest potential micro-cluster to $\vec{x}_i$: PMC_i. Later, it verifies if the increment of $\vec{x}_i$ in PMC_i results in a micro-cluster which radius is below ϵ. If this occurs, $\vec{x}_i$ is added to PMC_i. Otherwise, this process is repeated for outlier micro-clusters: SNCStream$^+$ finds the closest outlier micro-cluster OMC_i to $\vec{x}_i$ and they are merged if the resulting radius of its increment is below ϵ.

If $\vec{x}_i$ incremented an outlier micro-cluster, SNCStream$^+$ verifies whether if this *OMC* has become a *PMC*, i.e. its weight is now above $\beta\psi$. In the positive case, it is removed from $\mathcal{B}$ and inserted in the network as in the first phase to its ω closest neighbors and the homophily-based rewiring procedure is performed.

As in DenStream, both potential and outlier micro-clusters' weights decay over time accordingly to an exponential decay function presented in Eq. (2), where $\Delta t = t_i - t_u$ is the difference between the current timestamp and the instant of the last update of each core-micro-cluster, N is the amount of instances summarized by such micro-cluster and λ is the parameter for the exponential function:

$$w(CMC_i) = 2N^{-\lambda\Delta t} \tag{2}$$

The last procedure in SNCStream$^+$ is the removal of both potential and outlier micro-clusters. Firstly, SNCStream$^+$ verifies whether the weights of each potential micro-cluster in $\mathcal{V}$ is below the minimum density threshold $\beta\psi$ every T_p instances (Eq. (1)).

If this occurs with PMC_i, all of its neighbors PMC_j are allowed to rewire with the restriction to ignore PMC_i since it will be removed from the network. After the rewiring of all PMC_i's neighbors, PMC_i is removed from the network.

After the verification occurs for potential micro-clusters, the same procedure is repeated for outlier micro-clusters located in $\mathcal{B}$. In this last case, the process is simpler, where all outlier micro-clusters with weight below an threshold ξ (Eq. (3)) are removed from $\mathcal{B}$. For details on the computation of ξ, the reader is referred to [5]:

$$\xi = \frac{2^{t_i - t_c + T_p} - 1}{2^{-\lambda T_p} - 1} \tag{3}$$

3.4. Algorithm speedup, time and space complexity analysis

SNCStream and SNCStream$^+$ core resides in the rewiring procedure. However, this procedure is quite computationally complex since a big amount of distance computations occur on the arrival of each instance or insertion of a new potential micro-cluster in the network. In order to speedup SNCStream's performance, two assumptions are made: (i) rewiring is likely to compute distances between vertices that were already computed earlier, and (ii) after the rewiring of an arbitrary vertex v_i, SNCStream performs a linear access to the remaining vertices, however, vertices that do not participate of v_i's subgroup will perform rewiring unnecessarily. Additionally, vertices with a large distance in terms of hops are doubtfully affected by rewirings, thus, performing linear access is a naive approach that must be refined.

3.4.1. Distances memoization

Computing distances between data points can be problematic, specially in high dimensional spaces. Due to the rewiring procedure, distances between vertices are constantly re-computed unnecessarily. To overcome this drawback, SNCStream$^+$ encompasses a memoization technique [22] to store and reuse previously computed distances.

This memoization process consists in using a hash table in the $\langle key, value \rangle$ to efficiently store and retrieve distances between a pair of vertices v_i and v_j. The hash table key is an unique representation of a distance between v_i and v_j, such that their unique identifiers, i.e. i and j, are combined into an unique integer using the Cantor Pairing function [23]. In practice, the usage of Cantor pairing is optional since most of current mainstream programming techniques allow the usage of pair integers as key. In order to use Cantor's function, we assume that $i < j$, since $\pi(i,j) \neq \pi(j,i)$:

$$\pi(i,j) = \langle i,j \rangle = \frac{1}{2} \times (i+j) \times (i+j+1) + j \tag{4}$$

Cantor's pairing function is reversible, therefore, based on a natural number $\langle i,j \rangle$ it is possible to recover the original values of i and j. In order to perform such operation, one must follow:

$$z = \langle i,j \rangle$$
$$w = \left\lfloor \frac{\sqrt{8z+1}-1}{2} \right\rfloor$$
$$t = \frac{w^2+w}{2}$$
$$j = z - t$$
$$i = w - j$$

With this function, SNCStream$^+$ is capable of storing and reusing distances between pairs of vertices of the network, since these are stored in a hashtable in the $\langle \pi(i,j), d(v_i,v_j) \rangle$ form, thus allowing quick distance recovery without the need of recomputations.

Since every value present in this hash represents a distance between a pair of vertices in the network, its size is equal to the amount of edges in the network, therefore impacting in an extra memory consumption in the $\mathcal{O}(|\mathcal{V}|^2)$ order. Therefore, we highlight that whenever an edge removal occurs in the network, its corresponding value is removed from the memoization hash in order to free up memory space.

3.4.2. Rewiring through dissipation

The original rewiring procedure performs linear access to all existing vertices in the network, most of times, unnecessarily. This occurs since most of the vertices in the network are likely to be "far" from the location of the last inserted vertex or if it was inserted in a different subnetwork. In Algorithm 1 we present an iterative version for our faster rewiring procedure. This algorithm receives as input two parameters: the last inserted vertex v_i and the entire network $\mathcal{G}$.

Initially, two structures are initialized: *rewired* (a list of already rewired vertices) and *toRewire* (a queue of vertices that still must perform rewirings). While $toRewire \neq \emptyset$, a slight modification to the earlier stated rewiring procedure is performed with the head of the queue: v_j. Firstly, SNCStream$^+$ verifies if the newly chosen neighbors for v_j differ from its current neighbors. Newly chosen neighbors are defined as the k_j closest neighbors to v_j when assuming a 2-hop distance, which occurs in $\mathcal{O}(\omega^2)$. If the current neighbors differ from the new neighbors, all of v_j's neighbors, with the exception of v_i and other already rewired vertices are enqueued for rewiring.

Algorithm 1. Iterative optimized rewiring procedure.

```
Input the last added vertex v_i and the network G = (V, E, W).
1  rewired ← ∅;
2  toRewire ← adj(v_i);
3  while toRewire ≠ ∅ do
4  |  v_j ← dequeueFirst(toRewire);
5  |  currentNeighbors ← adj(v_j);
6  |  k_j ← deg(v_j);
7  |  newNeighbors ← the k_j closest neighbors to v_j assuming a
   |  2 − hop distance;
8  |  if currentNeighbors ≠ newNeighbors then
9  |  |  Remove all edges connecting v_j in E and its corresponding
   |  |  weights in W;
10 |  |  foreach v_k ∈ newNeighbors do
11 |  |  |  newEdge ← new_edge(v_k, v_j);
12 |  |  |  weight ← d(v_k, v_j);
13 |  |  |  E ← E ∪ {newEdge};
14 |  |  |  W ← W ∪ {weight};
15 |  |  toRewire ← toRewire ∪ (adj(v_j) − {v_i} − rewired);
16 |  rewired ← rewired ∪ {v_j};
```

If one assumes an unlikely condition that all nodes always must perform rewirings, i.e. the condition $currentNeighbors \neq newNeighors$ is always satisfied, the optimized rewiring procedure runs in $\mathcal{O}(V\omega)$. Since the difference between SNCStream and SNCStream$^+$ resides in this optimized rewiring algorithm, we state that SNCStream depends on $\mathcal{O}(V+\omega+V\omega^2)$, thus, its overall complexity is $\mathcal{O}(V\omega^2)$, as the original SNCStream. As stated earlier, this latter condition is unlikely to occur, therefore, in the average case, this optimization does provide a decreased computational time when compared to the original rewiring procedure adopted by SNCStream [4].

Fig. 4 presents an example of network where, after v_{13}'s insertion, two subgroups would not be verified by SNCStream$^+$ unnecessarily, therefore, decreasing the amount of distance computations and undesired rewirings.

4. Empirical evaluation

In this section we empirically analyze SNCStream$^+$'s clustering performance by comparing its results against CluStream [3], ClusTree [6], DenStream [5] and HAStream [17]. Clustering quality results from the original SNCStream are omitted since these differ only in computational cost, i.e. processing time and memory space usage.

First, we present the data generators and real datasets used for this evaluation and the experimental protocol for the following experiments. Later, we present a parameter sensitivity analysis, focusing on three of the major parameters: (i) the amount ω of edges created during each node insertion, (ii) the size of the clean-up window T_p; and (iii) the adopted distance metric for dissimilarity computation. Finally, we present a performance study of SNCStream$^+$ against literature baselines, highlighting SNCStream$^+$'s high clustering quality.

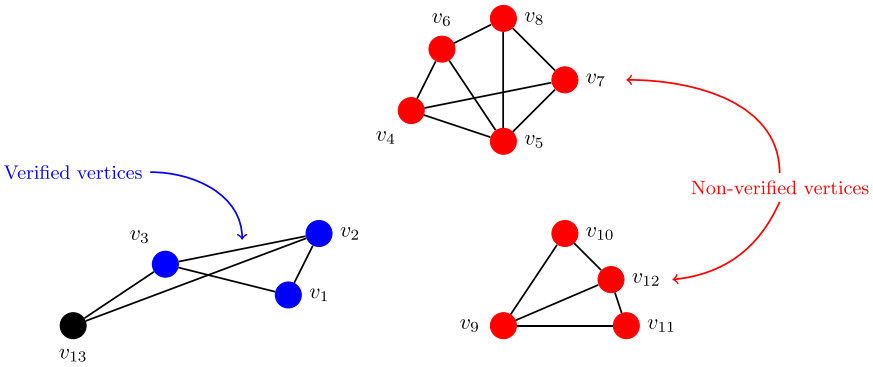


Fig. 4. Example of rewiring after the insertion of vertex v_{13}, where verified vertices are displayed in blue and non-verified in red. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

4.1. Data generators and real datasets

Synthetic data streams were generated using the Radial Basis Function (RBF) generator available at MOA framework [24]. The RBF generator creates a user-given amount of drifting centroids, which are defined by a label, center, weight and standard deviation given by a Gaussian distribution. Another possibility of the RBF generator is the appearance or disappearance of ground-truth clusters (concept evolution). We adopt the RBF_d notation to refer to an experiment that uses the RBF generator with a dimensionality d. In all RBF-based experiments, streams were created with 50,000 instances where the ground-truth amount of ground-truth clusters vary in the $[2;8]$ interval and an appearance/disappearance of a centroid occurs randomly every 3000 instances.

Besides the usage of data generators, SNCStream$^+$'s performance is also evaluated along publicly available real datasets. The following paragraphs briefly describe these datasets.

Airlines: This dataset contains 539,383 instances and 8 attributes and represents all flight arrivals and departures from USA airports, from October 1987 until April 2008 [25].

Electricity: This dataset was created by the Australian New South Wales Electricity Market and stores energy prices obtained every 5 min [26]. In this problem, energy prices are not fixed, since they vary accordingly to market supply and demand. The Electricity dataset consists of 45,312 instances and 8 attributes.

Forest Covertype: This dataset models the forest covertype prediction problem based on cartographic variables [27]. This dataset consists of 900 m^2 cells obtained from US Forest Service Region 2 Resource Information System and contains 581,012 instances with 54 attributes.

KDD'99: This dataset is composed by raw TCP dump data for a LAN. Each connection has 42 attributes and most of the 4,898,431 instances are labeled as normal connections. In addition, this dataset is known to present appearances of clusters over time, representing previously unknown types of cyber-attacks [3].

Body Posture and Movements (*BPaM*): This dataset consists of 165,632 instances collected on 8 hours of activities of 4 healthy subjects. The original goal is to classify whether the subject is sitting down, standing up, standing, walking or sitting based on 18 attributes [28].

4.2. Experimental protocol

All experiments presented in this paper were performed on a Intel Xeon CPU E5649 @ 2.53 GHz $\times$ 8 based computer running CentOS with 16GB of memory using Massive Online Analysis (MOA) framework [24]. All algorithms parameters were set accordingly to their original papers. CluStream parameters are: Horizon $\mathcal{H}=1000$ and $q=1000$ [3]. ClusTree parameters are: a Horizon $\mathcal{H}=1000$ and a maximum tree height $=8$ [6]. DenStream parameters are: $\psi=1$, $\mathcal{N}=1000$, $\lambda=0.25$, $\epsilon=0.02$ and $\beta=0.2$ [5]. All of the above cited algorithms run a DBSCAN at the offline step, using the following parameters: $\psi=1$, $\mathcal{N}=1000$, $\lambda=0.25$, $\epsilon=0.02$, $\beta=0.2$ and a offline multiplier $\eta=2$. HAStream uses a DenStream-like online step, therefore, adopts the same density parameters [17]. Finally, SNCStream and SNCStream$^+$'s parameters are: $\psi=1$, $\mathcal{N}=100$, $\lambda=0.25$, $\epsilon=0.02$, $\beta=0.2$ and $\omega=4$ [4]. The SNCStream$^+$ implementation used during these experiments is available for download from https://sites.google.com/site/moasocialbasedalgorithms/home as a plugin for the MOA framework.

In the following experiments, clustering quality is calculated using the Cluster Mapping Measure (CMM), a metric developed aiming the characteristics of data streams [29]. Differently from batch clustering evaluation metrics (e.g. Purity, Precision, Recall), CMM is an external clustering evaluation metric that accounts for non-associated, mis-associated instances and noisy data inclusion. It is also important to emphasize that CMM, in opposition to conventional clustering quality measures such as purity, considers recently retrieved instances with more weight than older ones by using an exponential decay function inside evaluation windows. CMM is bounded in the $[0;1]$ interval, being 1 the representation of a perfect clustering. For more details on the computation on CMM the reader is referred to [29].

Processing time is computed as the time that each algorithm used of CPU in seconds and memory usage is calculated in RAM-Hours, where each RAM-Hour equals to 1 GB of RAM dispensed for an hour of processing.

In order to determine whether there is significant statistical difference between algorithms, a combination of Friedman's [30] and Nemenyi's [31] non-parametric hypothesis tests is used. Finally, we report significant statistical differences with critical differences (*CD*) graphics.

4.3. Parameter sensitivity analysis

Although SNCStream has shown superior clustering quality when compared to other algorithms [4], a few questions arise on its parametrization and its power when performing in high dimensional data streams, such as:

- To what extent the amount of connections ω, which each instance and micro-cluster establish during their insertion in the network, impacts clustering quality?
- As in DenStream [5], SNCStream encompasses a clean-up window size T_p which determines when micro-clusters' weights should be verified for removal. What is the impact of T_p in both clustering quality and processing time?
- Although the original SNCStream presented higher clustering quality [4], its acuity decays with the increase of the dimensionality d. Is SNCStream jinxed by the curse of dimensionality? How can we decrease the impact of this phenomenon?

To address these questions, we present three testbeds. In Section 4.3.1 we examine the impact of the amount of connections ω that each instance or micro-cluster establishes and its impact in clustering quality. In Section 4.3.2 we debate the impact of the clean-up window size T_p aiming clustering quality and processing time. In Section 4.3.3, we discuss about the limitation of the original SNCStream in high dimensional spaces and how SNCStream can be boosted to decrease it.

4.3.1. The impact of parameter ω

Contrarily to density-based algorithms such as DenStream [5], which rely on a variety of parameters due to DBSCAN execution during the offline step, SNCStream relies on only one parameter to find and maintain up-to-date clusters: the amount of connections ω that each instance and micro-cluster establishes when added to the network.

Therefore, one of the questions around SNCStream is the impact of the parameter ω in the final clusters. Intuitively, if ω is small, the network would be very sparse, and accordingly to the rewiring process, a higher amount of clusters would emerge with the addition of new instances or micro-clusters. Intuitively, if ω is large, it may lead us to think that the network would be dense due to a higher amount of edges. In this case, the rewiring process would not be able to split the network in subgroups and just one cluster would be found.

As any other parameter, finding the optimal value of ω reckons on the data stream domain, however, in the original paper we discussed about possible values of ω and its impact in clustering quality, showing that $\omega=4$ is a good choice for a variety of domains [4]. In this testbed we perform a deeper analysis of such values of ω and its impact in clustering quality.

In Table 1 we present the CMM results obtained by SNCStream when varying ω in the $[1;10]$ interval, where no single value outperforms all others and there is not much variation among datasets, with the exception of $\omega=1$, which is the worst ranked configuration; and $\omega=4$, which presents the best average results. To determine whether there is significant statistical difference between any of the tested values, we used Friedman and Nemenyi tests. The results of such tests are presented in Fig. 5, where $\omega \in [2;10] \succ (\omega = 1)$.

The diminished clustering quality obtained when $\omega=1$ occurs due to a network characteristic: low clustering coefficient. In Graph Theory, the clustering coefficient is a metric which measures the tendency of vertices forming concise subgroups (clusters) [21]. Clustering coefficients can be measured both locally (measures how embedded in the network a given vertex is) and globally (how dense and concise are the clusters in the network). Eq. (5) presents the computation of the cluster coefficient for an arbitrary vertex v_i, while Eq. (6) presents the computation of the average cluster coefficient for a network $\mathcal{G}$, where $deg(v_i)$ stands for the degree of a vertex v_i, $e_i = \sum_{v_k}^{adj(v_i)} deg(v_k)$ and $adj(v_i)$ is the set of neighbors of v_i in $\mathcal{G}$. The values for clustering coefficients are bounded in [0;1], where 1 represents the maximum value, implying that the network is an entirely connected component:

$$\mathcal{CC}(v_i) = \frac{2e_i}{deg(v_i) \cdot (deg(v_i) - 1)} \tag{5}$$

Table 1
CMM obtained by varying ω for SNCStream$^+$ algorithm.

Experiment	CMM									
	$\omega=1$	$\omega=2$	$\omega=3$	$\omega=4$	$\omega=5$	$\omega=6$	$\omega=7$	$\omega=8$	$\omega=9$	$\omega=10$
RBF_2	0.82	0.97	**0.99**	**0.99**	0.90	0.89	0.88	0.88	0.87	0.87
RBF_5	0.80	0.97	**0.99**	**0.99**	0.91	0.91	0.90	0.87	0.85	0.88
RBF_{20}	0.71	**0.91**	0.89	0.88	0.80	0.78	0.81	0.76	0.75	0.75
RBF_{50}	0.65	**0.86**	0.82	0.84	0.75	0.77	0.74	0.69	0.73	0.78
Airlines	0.84	0.89	0.94	0.96	0.97	0.97	0.97	**0.98**	**0.98**	**0.98**
Electricity	0.71	0.75	0.84	0.89	0.91	0.93	0.93	0.94	**0.95**	0.87
Forest Covertype	0.80	0.85	0.93	0.94	0.97	0.93	0.97	**0.99**	**0.99**	**0.99**
KDD'99	0.76	0.81	0.86	0.90	0.92	0.93	0.94	0.94	0.94	**0.95**
BPaM	0.88	0.93	0.96	0.98	0.98	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**

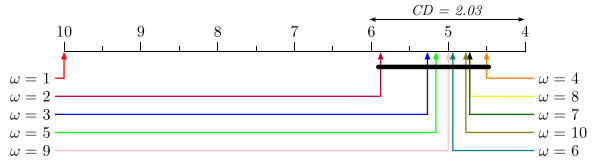


Fig. 5. Nemenyi's critical differences for CMM comparison when varying ω.

$$\mathcal{CC}(\mathcal{G}) = \frac{1}{V}\sum_{v_i \in \mathcal{V}} \mathcal{CC}(v_i) \tag{6}$$

Recent works show that, in most part of real-world social networks with apparent communities, vertices tend to participate of clusters with a high density of edges and present average clustering coefficients around 0.6 [32,21]. In Fig. 6 we present the cluster coefficient obtained in the networks when varying the parameter ω in all experiments where $\omega=1$ presents low clustering coefficient in all cases since the networks obtained are not dense enough to form clusters (communities).

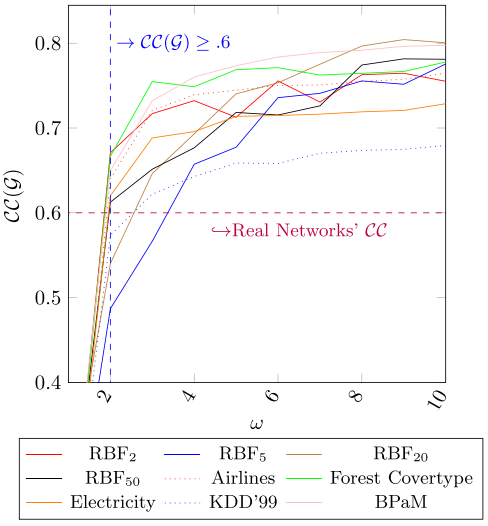


Fig. 6. Clustering coefficient versus ω in the performed experiments.

4.3.2. The impact of the clean-up window size

Although SNCStream is not bounded to DBSCAN runs, it still employs the notion of density. As in DenStream, micro-clustersapos weights decay with time accordingly to an exponential function and such weights are verified periodically accordingly to a clean-up window size T_p.

Determining a window size to verify micro-clusters' weights can be problematic. If the parameters λ, β and ψ originally suggested by authors in [5] are employed, this verification will be performed quite often, i.e. $T_p \leq 4$, theoretically inducing a high computational cost due to excessive linear accesses to micro-clusters.

Although verifying micro-clusters' weights at the arrival of each incoming instance, i.e. $T_p = 1$, provides prompt removal of those with inadequate weights from the network, it may jeopardize processing time due to unnecessary extra linear access to micro-clusters weights [5].

In this section we evaluate the impact of different values of T_p varying in the [1;50] interval in clustering quality and processing time for the SNCStream.

Fig. 7(a) presents the average results of CMM obtained during the RBF_{20}, RBF_{50}, Airlines, Electricity, Forest Covertype and BPaM experiments. In all cases, the increase of T_p impacts in a *CMM* decrease. This corroborates to the fact that higher values of T_p allow micro-clusters to join the network while they do not represent the actual state of the stream, therefore lessening cluster quality.

In Fig. 7(b) and (c) we present the overall CPU Time used by SNCStream on the RBF_2, RBF_5 and RBF_{20} and Electricity and KDD'99 experiments, where, in opposition to what is expected, CPU time increases with higher values of T_p.

Higher values of T_p cause an increase on the amount of micro-clusters stored in memory, and consequently, more distance computations must be performed for each arriving instance $\vec{x}_i$. Fig. 7(d) presents a graphic which relates values of T_p and the amount of distance computations performed during experiments. We highlight that with the increase of T_p, the amount of distance computations exponentially augments due to the higher amount of micro-clusters maintained by SNCStream. Therefore, we concluded that $T_p = 1$ is the best choice in terms of both clustering quality and processing time since it allows SNCStream to eliminate old micro-clusters promptly and decrease the amount of distance computations.

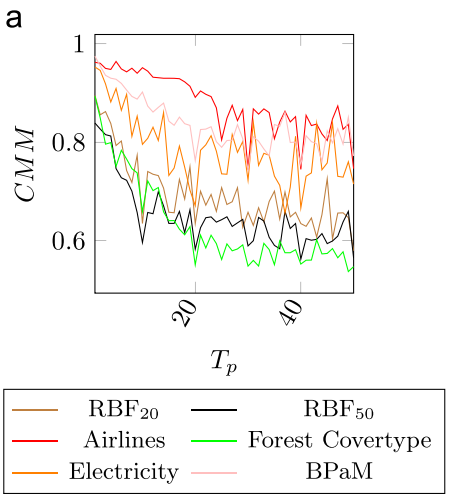
4.3.3. The impact of distance metrics

As any other clustering algorithm that uses the notion of distance, SNCStream may fall in the curse of dimensionality. The original SNCStream adopts the Euclidian distance to compute dissimilarity between instances and micro-clusters [4]. As shown in seminal works [5,8], Euclidian distances fail to represent significantly the dissimilarity between points in high dimensional spaces, enforcing algorithms to fall in their vastitude.

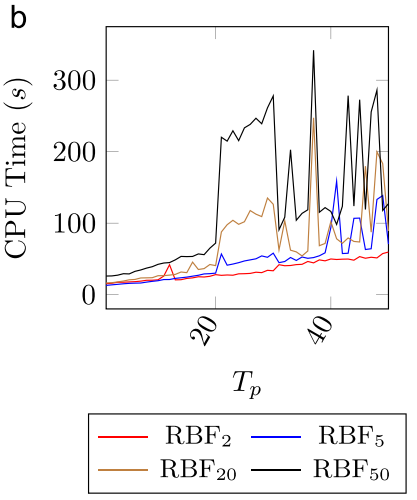
In most high dimensional applications the choice of the distance metric is concealed and the computation of dissimilarities is rather heuristical [13]. There is very little work in the literature providing guidance on choosing the correct distance metric to calculate dissimilarity between two instances. Most part of algorithms adopt the L_p norm, which is also known to be susceptible to the curse of dimensionality in such spaces [3,5,6]. Generally, the L_p distance between two instances $\vec{x}_i$ and $\vec{x}_j$ can be computed according to the following equation:

$$d_{L_p}\left(\vec{x}_i, \vec{x}_j\right) = \left[\sum_{v=1}^{d} |\vec{x}_i[v] - \vec{x}_j[v]|^p\right]^{\frac{1}{p}} \tag{7}$$

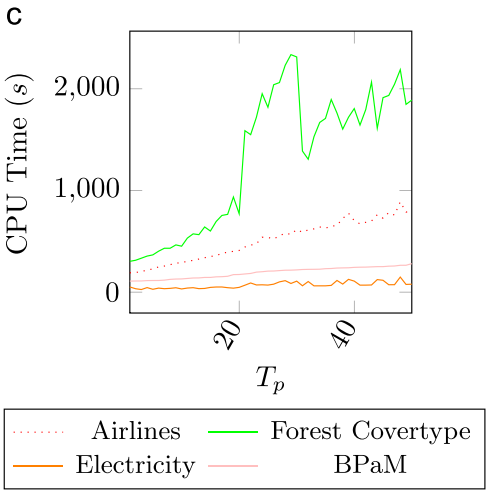
In [13], authors discuss about different values of p, enlightening that $p=1$ (Manhattan distance) and $p=2$ (Euclidian distance) are theoretically more efficient than $p \geq 3$. In addition, they prove that the meaningfulness of the L_p form decays fast with the increase of the dimensionality d [13,33]. Encouraged by this trend, authors in [13] examined the behavior of fractional distance metrics, where $0 \leq p \leq 1$, and pointed $p=0.3$ as an interesting value for many domains after applying k-means batch clustering in a series of datasets.



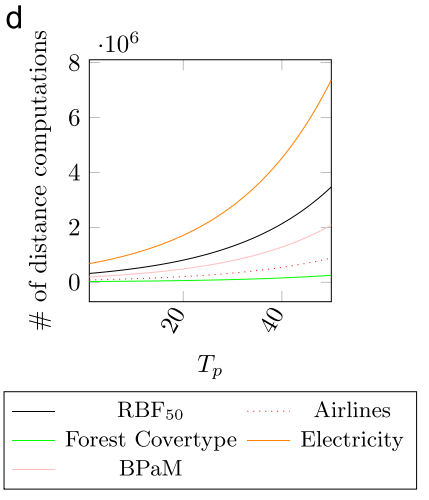
T_p versus CMM.



T_p versus CPU Time (s) for synthetic experiments.



T_p versus CPU Time (s) for real dataset experiments.



T_p versus amount of distance computations.

Fig. 7. Results obtained varying T_p.

Another popular distance metric commonly used in clustering techniques is Cosine distance. Cosine distance assumes that the dissimilarity between two vectors (instances) $\vec{x}_i$ and $\vec{x}_j$ can be computed as the angle between these vectors [13]. Eq. (8) presents the cosine distance computation adapted to the problem of determining the dissimilarity of two instances:

$$d_{Cosine}\left(\vec{x}_i, \vec{x}_j\right) = 1 - \frac{\sum_{v=1}^{d} \vec{x}_i[v] \times \vec{x}_j[v]}{\sqrt{\sum_{v=1}^{d} (\vec{x}_i[v])^2} \sqrt{\sum_{v=1}^{d} (\vec{x}_j[v])^2}} \tag{8}$$

Assuming the conventional Euclidian distance (L_2) it follows immediately that all instances $\vec{x}_i$ with the same distance to the origin satisfy the equation of a spheroid $\sum_{i=1}^{n} (\vec{x}_i)^2 = r^2$, where r is its radius. This means that all components of a dataset contribute equally to the Euclidian distance from the center. Nevertheless, in statistics, it is preferable a distance metric that accounts for the variability of each dimension. In Mahalanobis distance [34], dimensions with high variability receive less weight than components with low variability. This is done by rescaling the dimensions using their standard deviation σ as stated in the following equation:

$$d_{Mahalanobis}\left(\vec{x}_i, \vec{x}_j\right) = \sqrt{\sum_{v=1}^{d} \frac{(\vec{x}_i[v] - \vec{x}_j[v])^2}{\sigma_v}} \tag{9}$$

Based on the previously discussed distance metrics we hypothesize that SNCStream is able to decrease the impact of the curse of dimensionality, therefore augmenting its clustering quality in high dimensional data streams. In order to verify this hypothesis, we extended SNCStream to

compute the latter distance metrics which are known to perform well on high dimensional problems. In this testbed we evaluate the power provided by such metrics in SNCStream and finally, compare all algorithms adopting the best ranked metric.

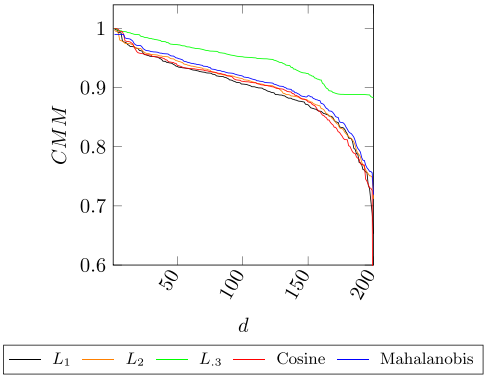


Fig. 8. CMM obtained by SNCStream adopting different distance metrics when varying the dimensionality d.

Table 2
CMM obtained by varying SNCStream$^+$'s distance metric.

Experiment	CMM – SNCStream$^+$				
	L_1	L_2	$L_{.3}$	Cosine	Mahalanobis
RBF2	0.98	0.98	**0.99**	0.92	0.98
RBF5	0.98	0.98	**0.99**	0.95	0.97
RBF20	0.89	0.88	**0.99**	0.85	0.88
RBF50	0.84	0.84	**0.96**	0.76	0.86
Airlines	0.97	0.96	**0.98**	0.96	0.96
Electricity	0.90	0.96	**0.98**	0.89	0.96
Forest Covertype	0.89	0.89	**0.97**	0.95	0.95
KDD99	0.90	0.94	**0.95**	0.90	0.94
BPaM	0.98	0.98	**0.99**	0.98	0.98

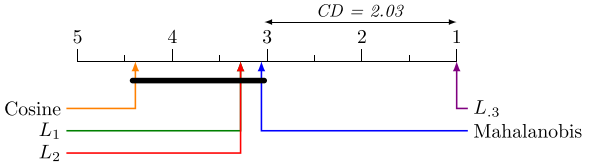


Fig. 9. Nemenyi's critical differences for CMM comparison on different distance metrics.

In Fig. 8 we present the CMM obtained in RBF experiments varying the dimensionality d in the [2;200] interval, where one can see that with the increase of d, the clustering quality decays, specially in the L_1 (Manhattan), L_2 (Euclidian) and Cosine distance metrics. In addition, we highlight the superior results obtained by the fractional distance metric $L_{.3}$ specially with the increase of d, therefore corroborating its power as stated in [13] (Table 2).

In order to determine whether there is statistical difference between SNCStream's clustering quality when varying the adopted distance metric, we executed Friedman followed by Nemenyi's post-hoc test. Fig. 9 presents the graphical results obtained, where $\{L_{.3}\} \succ \{Mahalanobis, L_2, L_1, Cosine\}$ with a 95% confidence level since it differs at least by one Critical Difference (CD).

4.4. Performance against literature baselines

In this testbed we evaluate CluStream [3], ClusTree [6], DenStream [5], HAStream [17] and SNCStream [4] using both synthetic and real data earlier presented in Section 4.1.

Firstly, we compare SNCStream adopting the fractional distance metric ($L_{0.3}$) against other algorithms. Table 3 presents the results for clustering quality (CMM), where SNCStream outperforms others in all experiments.

Significant differences among the algorithms were found after performing Friedman and Nemenyi tests, in which the ranks differ by at least a Critical Difference (CD). Fig. 10(a) shows the graphical representation of results obtained where $\{SNCStream^+\} \succ \{HAStream, DenStream, ClusTree, CluStream\}$ enlightening the high clustering quality obtained by SNCStream.

Additionally, we compared results for CPU Time and RAM-Hours. Fig. 10(b) and (c) presents the results obtained for these metrics, respectively, where $\{ClusTree, ^{SNCStream+}, HAStream\} \succ \{CluStream, DenStream, SNCStream\}$ in terms of processing time. Although SNCStream$^+$'s asymptotic complexity is the same as the original SNCStream's, these results show that there is a huge gain due to the optimized rewiring procedure. As for RAM-Hours, we obtained $\{CluStream\} \succ \{SNCStream, SNCStream^+, HAStream, DenStream\} \succ \{ClusTree\}$. We highlight the fact that SNCStream$^+$ requires extra memory space, yet, still appears as the third best ranked algorithm in this property.

Table 3
Average CMM obtained in experiments.

Experiment	CMM				
	CluStream	ClusTree	DenStream	HAStream	SNCStream$^+$ ($L_{0.3}$)
RBF_2	0.83	0.88	0.94	0.87	**0.99**
RBF_5	0.67	0.80	0.84	0.86	**0.99**
RBF_{20}	0.47	0.88	0.85	0.91	**0.99**
RBF_{50}	0.48	0.44	0.85	0.81	**0.96**
Airlines	0.67	0.76	0.50	0.70	**0.98**
Electricity	0.41	0.44	0.61	0.43	**0.98**
Forest Covertype	0.41	0.43	0.38	0.37	**0.97**
KDD'99	0.50	0.49	0.69	0.77	**0.95**
BPaM	0.61	0.69	0.74	0.74	**0.99**

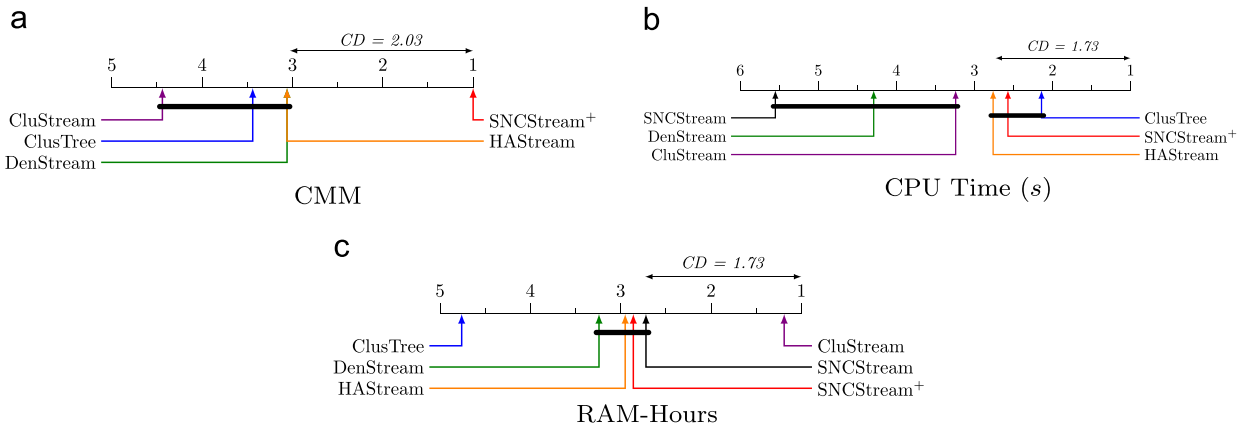


Fig. 10. Results obtained by Friedman and Nemenyi hypothesis tests.

Table 4
Average CMM obtained in experiments.

Experiment	$L_{0.3}$ – CMM				
	CluStream	ClusTree	DenStream	HAStream	SNCStream$^+$
RBF_2	0.87	0.92	0.97	0.91	**0.99**
RBF_5	0.70	0.84	0.88	0.90	**0.99**
RBF_{20}	0.49	0.92	0.89	0.95	**0.96**
RBF_{50}	0.50	0.46	0.89	0.85	**0.99**
Airlines	0.70	0.80	0.52	0.74	**0.98**
Electricity	0.43	0.46	0.64	0.45	**0.98**
Forest Covertype	0.44	0.45	0.40	0.39	**0.97**
KDD'99	0.53	0.51	0.72	0.81	**0.95**
BPaM	0.64	0.72	0.77	0.77	**0.99**

Finally, to determine if other algorithms are capable of reaching SNCStream$^+$'s high clustering quality, we repeat the same experiments adopting a $L_{0.3}$ distance metric in all algorithms and the results obtained are stated in Table 4. Furthermore, SNCStream$^+$ still outperforms others in clustering quality, something corroborated by the hypothesis test presented in Fig. 11 with a 95% confidence level. Nevertheless, we emphasize that all algorithms presented better results when compared to its earlier results (shown in Table 3), showing that $L_{0.3}$ is beneficial regardless or the clustering algorithm.

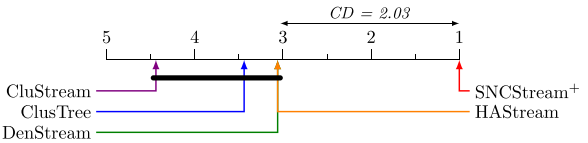


Fig. 11. Nemenyi test results for CMM comparison of algorithms adopting the $L_{0.3}$ distance metric.

5. Conclusion

Clustering streaming data is of extreme importance in many real applications, such as consumer click streams, telephone usage flows, multimedia data mining [7,8], computer networks intrusion detection [3], XML and HTML structures mining [9] and sensor network data clustering [10]. In this paper we presented SNCStream$^+$, an extension to the SNCStream [4], a high quality true real-time data stream clustering algorithm. We analyzed the main parameter of SNCStream$^+$, ω, and its impact in clustering quality. Additionally, we unveiled the oddity regarding the value of the clean-up window, which optimally must be set to $T_p = 1$ to maximize cluster quality and diminish processing time. Finally, a comparison between a variety of distance metrics showed that the clustering quality of SNCStream$^+$ on high dimensional data streams can be boosted, therefore decreasing the impact of the curse of dimensionality. Based on this extensive empirical comparison with recent approaches, we have shown that SNCStream$^+$ outperforms a variety of data stream clustering algorithms in terms of clustering quality and acts within limited processing time and memory space for both synthetic and real data problems. We emphasize that SNCStream$^+$ presents decreased average computation complexity with an extra memory consumption bounded to $\mathcal{O}(|\mathcal{V}|^2)$ when compared to the original SNCStream.

Acknowledgments

Authors would like to thank the anonymous reviewers for their constructive comments that highly improved this paper. This research was partially funded by CAPES

(Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and Fundação Araucária.

References

[1] A. Bifet, J. Read, I. Zliobaite, B. Pfahringer, G. Holmes, Pitfalls in benchmarking data stream classification and how to avoid them, in: ECML/PKDD (1), 2013, pp. 465–479.

[2] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, ACM, New York, NY, USA, 2000, pp. 71–80, http://dx.doi.org/10.1145/347090.347107.

[3] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB '03, VLDB Endowment, vol. 29, 2003, pp. 81–92.

[4] J.P. Barddal, H.M. Gomes, F. Enembreck, SNCStream: a social network-based data stream clustering algorithm, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC), New York, NY, USA, 2015, pp. 935–940.

[5] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: SDM, 2006, pp. 328–339.

[6] P. Kranen, I. Assent, C. Baldauf, T. Seidl, The clustree: indexing micro-clusters for anytime stream mining, Knowl. Inf. Syst. 29 (2) (2011) 249–272.

[7] S. Guha, Tight results for clustering and summarizing data streams, in: Proceedings of the 12th International Conference on Database Theory, ICDT '09, ACM, New York, NY, USA, 2009, pp. 268–275, http://dx.doi.org/10.1145/1514894.1514926.

[8] J.A. Silva, E.R. Faria, R.C. Barros, E.R. Hruschka, A.C.P.L.F.d. Carvalho J.a. Gama, Data stream clustering: a survey, ACM Comput. Surv. 46 (1) (2013) http://dx.doi.org/10.1145/2522968.2522981. 13:1–13:31.

[9] C.C. Aggarwal, P. Yu, On clustering techniques for change diagnosis in data streams, in: O. Nasraoui, O. Zaiane, M. Spiliopoulou, B. Mobasher, B. Masand, P.S. Yu (Eds.), Advances in Web Mining and Web Usage Analysis, Lecture Notes in Computer Science, vol. 4198, Springer, Berlin, Heidelberg, 2006, pp. 139–157.

[10] P. Rodrigues, J. Gama, J. Pedroso, Hierarchical clustering of time-series data streams, IEEE Trans. Knowl. Data Eng. 20 (5) (2008) 615–627, http://dx.doi.org/10.1109/TKDE.2007.190727.

[11] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) http://dx.doi.org/10.1145/2523813. 44:1–44:37.

[12] M.M. Masud, J. Gao, L. Khan, J. Han, B.M. Thuraisingham, Classification and novel class detection in concept-drifting data streams under time constraints, IEEE Trans. Knowl. Data Eng. 23 (6) (2011) 859–874.

[13] C.C. Aggarwal, A. Hinneburg, D.A. Keim, On the surprising behavior of distance metrics in high dimensional space, in: Database Theory — ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 420–434, http://dx.doi.org/10.1007/3-540-44503-X_27.

[14] J. Gama, Knowledge Discovery from Data Streams, 1st Edition, Chapman & Hall/CRC, 2010.

[15] A. Amini, T.Y. Wah, On density-based data streams clustering algorithms: a survey, J. Comput. Sci. Technol. 29 (1) (2014) 116–141, http://dx.doi.org/10.1007/s11390-014-1416-y.

[16] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.

[17] M. Hassani, P. Spaus, T. Seidl, Adaptive multiple-resolution stream clustering, in: P. Perner (Ed.), Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Computer Science, vol. 8556, Springer International Publishing, Cham, 2014, pp. 134–148.

[18] S. Lloyd, Least squares quantization in pcm, IEEE Trans. Inf. Theor. 28 (2) (1982) 129–137.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: E. Simoudis, J. Han, U.M. Fayyad (Eds.), KDD, AAAI Press, Portland, Oregon, USA, 1996, pp. 226–231.

[20] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD '84, ACM, New York, NY, USA, 1984, pp. 47–57, http://dx.doi.org/10.1145/602259.602266.

[21] M. Van Steen, Graph Theory and Complex Networks: An Introduction, Maarten Van Steen, Amsterdam, The Netherlands, 2010.

[22] D. Michie, 'Memo' functions and machine learning, Nature 218 (1968) 19–22, http://dx.doi.org/10.1038/218019a0.

[23] A.L. Rosenberg, Efficient pairing functions—and why you should care, in: Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS '02, IEEE Computer Society, Washington, DC, USA, 2002.

[24] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, Moa: massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604.

[25] E. Ikonomovska, J. Gama, B. Zenko, S. Dzeroski, Speeding-up Hoeffding-based regression trees with options, in: ICML, 2011, pp. 537–544.

[26] M. Harries, N.S. Wales, Splice-2 comparative evaluation: Electricity Pricing, 1999.

[27] P. Kosina, J.a. Gama, Very fast decision rules for multi-class problems, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, ACM, New York, NY, USA, 2012, pp. 795–800, http://dx.doi.org/10.1145/2245276.2245431.

[28] Wallace Ugulino, Débora Cardador, Katia Vega, Eduardo Velloso, Ruy Milidiú, Hugo Fuks, Wearable computing: accelerometers' data classification of body postures and movements, in: Barros, Leliane N and Finger, Marcelo and Pozo, Aurora T. and Gimenénez-Lugo, Gustavo A. and Castilho, Marcos (eds), Advances in Artificial Intelligence—SBIA 2012, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 52–61.

[29] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, B. Pfahringer, An effective evaluation measure for clustering on evolving data streams, in: Proceedings of the 17th ACM Conference on Knowledge Discovery and Data Mining (SIGKDD 2011), San Diego, CA, USA, ACM, New York, NY, USA, 2011, pp. 868–876.

[30] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Am. Stat. Assoc. 32 (200) (1937) 675–701, http://dx.doi.org/10.2307/2279372.

[31] P. Nemenyi, Distribution-free multiple comparisons (Ph.D. thesis), New Jersey, USA, 1963.

[32] R. Albert, A.-L. Barab´asi, Statistical mechanics of complex networks, Rev. Mod. Phys. 74 (2002) 47–97.

[33] K.S. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is "nearest neighbor" meaningful? in: Proceedings of the 7th International Conference on Database Theory, ICDT '99, Springer-Verlag, London, UK, UK, 1999, pp. 217–235.

[34] M.R. Ackermann, J. Blömer, C. Sohler, Clustering for metric and nonmetric distance measures, ACM Trans. Algorithms 6 (4) (2010) 59:1–59:26.