Neural Networks 110 (2019) 141–158

Contents lists available at ScienceDirect

Neural Networks

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

A density-based competitive data stream clustering network with self-adaptive distance metric

Baile Xu, Furao Shen *, Jinxi Zhao

National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing University, China

ARTICLE INFO

Article history:
Received 31 October 2017
Received in revised form 17 November 2018
Accepted 20 November 2018
Available online 27 November 2018

Keywords:
Unsupervised learning
Clustering methods
Competitive neural networks
Stream learning

ABSTRACT

Data stream clustering is a branch of clustering where patterns are processed as an ordered sequence. In this paper, we propose an unsupervised learning neural network named Density Based Self Organizing Incremental Neural Network(DenSOINN) for data stream clustering tasks. DenSOINN is a self organizing competitive network that grows incrementally to learn suitable nodes to fit the distribution of learning data, combining online unsupervised learning and topology learning by means of competitive Hebbian learning rule. By adopting a density-based clustering mechanism, DenSOINN discovers arbitrarily shaped clusters and diminishes the negative effect of noise. In addition, we adopt a self-adaptive distance framework to obtain good performance for learning unnormalized input data. Experiments show that the DenSOINN can achieve high standard performance comparing to state-of-the-art methods.

1. Introduction

Clustering is an important machine learning technique that has attracted a lot of research interest in past decades (Jain & Dubes, 1988; Jain, Murty, & Flynn, 1999). It has been widely used in many applications of data mining, natural language processing, computer vision etc (Lee, 1981; Ma, Zhong, & Zhang, 2015; Srivastava & Sahami, 2009; Zhong, Ma, Ong, Zhu, & Zhang, 2018). The main task of clustering is grouping similar patterns in a given dataset into meaningful clusters. Data stream clustering is a special type of clustering that draws much attention in recent years, motivated by applications involving large datasets and real time learning tasks. A data stream is a set of patterns organized as an ordered sequence. Stream clustering models have many differences compared to traditional off-line algorithms (Silva, Faria, Barros, & Hruschka, 2013).

First, input patterns arrive continuously and the learning algorithms have no control over the input sequence. Stream clustering models cannot access patterns randomly or iteratively learn the whole dataset. Only a small amount of patterns can be stored in memory and need to be discarded after they are processed.

Second, underlying data distribution in the stream might be non-stationary. Stream clustering models should have the ability to learn new data distribution without forgetting previously learned knowledge.

Third, incoming data stream can be very large while time and storage space available to an algorithm is limited. Stream clustering models should use concise representation of clusters, and be highly efficient in time.

Depending on demand of clustering task, some other problems may be taken into consideration. Input data may contain errors and noises in some tasks, which would favor algorithms that can detect outliers and act accordingly (Cao, Ester, Qian, & Zhou, 2006; Shen & Hasegawa, 2006). Some learning tasks may consider recent data more important than old data, where time window models are adopted to meet this demand (Cao et al., 2006; Ghesmoune, Lebbah, & Azzag, 2016).

However, a problem remains unsolved in all existing models. As a measure of dissimilarity between patterns, the distance metric used in a distance based algorithm plays an important role. For many algorithms like k-means (Macqueen, 1967) and k-nearest neighbor classifier (Cover & Hart, 1967), choosing a proper distance metric could greatly improve their performances. In many real world datasets, the range of different features varies widely. Some widely used distance metrics like standard Euclidean distance do not work well in this situation, because features with relatively wider range of value will be decisive in the final distance. To make sure each feature contributes equally in the distance metric, data normalization (Aksoy & Haralick, 2001) (i.e. feature scaling) is used as an important preprocess technique to normalize the ranges of the features. However, most data normalization algorithms are difficult to be implemented under streaming constraints because they need to compute the statistical characteristics of the whole dataset, which is unfeasible when data access is sequential.

* Corresponding author.
E-mail addresses: dg1633021@smail.nju.edu.cn (B. Xu), frshen@nju.edu.cn (F. Shen), jxzhao@nju.edu.cn (J. Zhao).

https://doi.org/10.1016/j.neunet.2018.11.008
0893-6080/

In this paper, we propose an innovate data stream clustering method named Density Based Self Organizing Incremental Network (DenSOINN). The main contributions of our work can be summarized as following:

(1) We adopt a self-adaptive distance metric to approximate the effect of data normalization, making the algorithm works on raw data as well as on normalized data.

(2) We provide a novel density based method to solve the problem of finding clusters in competitive neural networks.

Unlike many existing methods that deal with evolving data stream, we consider data stream clustering as an incremental learning problem and designed DenSOINN to learn new knowledge while keeping old knowledge unforgotten, therefore we do not use the time window to emphasize new data over old data.

This paper is an extension of a prior work (Xu, Shen, & Zhao, 2016). In this paper, we improved the algorithm design based on the formerly proposed method to facilitate analysis. We provide more theoretical analysis and discussions, as well as more extensive experiments in this paper. The rest of the paper is organized as follows: Section 2 reviews related works, Section 3 describes details of DenSOINN, Section 4 presents analyses and discussions, Section 5 presents experimental results on both artificial and real world datasets, Section 6 concludes the paper.

2. Related works

2.1. Data stream clustering algorithms

In the past decades, many stream clustering algorithms have been proposed based on adaption of off-line clustering algorithms, such as BIRCH (Zhang, Ramakrishnan, & Livny, 1997), CluStream (Aggarwal, Han, Yu, & Yu, 2003), DenStream (Cao et al., 2006), StreamKM++ (Ackermann, Raupach, Swierkot, Lammersen, & Sohler, 2012), StrAP (Zhang, Furtlehner, Germain-Renaud, & Sebag, 2014), etc. These algorithms can be generally summarized into two steps: an online learning step that derives a data abstraction from input data stream, and an offline clustering step that generates the final result.

During the online learning step, input patterns are summarized into specific data structures in order that the algorithm can preserve data distribution without storing these patterns. The use of such data structure was first introduced in the BIRCH algorithm. The data structure named Clustering Feature vector (CF) has three components: the number of patterns N, the linear sum of patterns LS, and the sum of squared patterns SS. Local distribution information is computed from these components, such as cluster mean $\mu = \frac{LS}{N}$ and standard deviation $\delta = \sqrt{\frac{SS}{N} - (\frac{LS}{N})^2}$. Clustering feature vectors can be maintained incrementally, and can be simply merged by adding two vectors. CluStream extends the clustering feature vector to a data structure named micro-cluster by adding temporal features into the vector, maintaining statistical summaries of pattern timestamps in the micro-cluster. DenStream also names its data summarization structure micro-cluster, which contains three components similar with clustering feature vector of BIRCH. DenStream can detect outliers in input data, therefore it is less sensitive to noise. DenStream employs a damped window model to assign higher weight for recent patterns than older patterns, using a user-defined parameter to allow control over the degree of emphasis on new data. StreamKM++ maintains its data summarization by the merge-and-reduce technique. StrAP records input patterns that do not match with existing micro-clusters as outliers, and uses a reservoir to store these outliers.

The offline clustering step is mostly a variation of standard clustering algorithms. The data summarization produced by the online learning step is viewed as a set of weighted pseudo-points to which clustering algorithms are applied. K-means (Lloyd, 1982; Macqueen, 1967) and DBSCAN (Ester, Kriegel, & Xu, 1996) are most widely used standard clustering algorithms. BIRCH, Clustream and StreamKm++ adopt K-Means in their offline clustering step. StreamKm++ also uses K-means++(Arthur & Vassilvitskii, 2007) seeding technique to aid cluster center initialization. DenStream uses DBSCAN to cluster micro-clusters. However, both methods have their disadvantages. K-Means needs to decide the number of clusters k, and tends to generate similar sized spherical clusters. DBSCAN can detect a suitable number of arbitrary shaped clusters, but it has two parameters that are critical to clustering performance but sometimes difficult to find appropriate values, especially on high dimensional data. Moreover, DBSCAN has difficulty in separating overlapped clusters. StrAP is extended from Affinity Propagation (Frey & Dueck, 2007), a message passing-based clustering method that attracts many research interests after it is proposed. A review of early data stream clustering algorithms can be found in Ding, Wu, Qian, Jia, and Jin (2015). Several new data stream clustering algorithms have been proposed in recent years, including Str-FSFDP (Khan, Huang, & Ivanov, 2016), IDEStream (Chen & He, 2016) and CEDAS (Hyde, Angelov, & Mackenzie, 2017).

Competitive neural network is another type of models that can be used in stream clustering, like Self Organizing Maps (Kohonen, 1982), Growing Neural Gas (Martinetz, Berkovich, & Schulten, 1993) and Self Organizing Incremental Neural Network (SOINN) (Shen & Hasegawa, 2006). During online learning step, each input pattern would trigger a competition among neural nodes, and the winner will be given a reward. As learning procedure goes on, the nodes are trained to become a summarization of historical input data. However, how to get final clusters from the network remains a problem. Some neural network based stream clustering algorithms, like AING (Bouguelia, Belaïd, & Belaïd, 2013) and G-Stream (Ghesmoune et al., 2016), were proposed in recent years. Both AING and G-Stream appoint each node as a cluster, but the numbers of nodes in both networks grow approximately linearly with the number of input patterns until reaches a predefined limit, which usually far exceeds the number of underlying classes. SOINN appoints each connected component of the topological graph as a cluster: starting from an unclassified node, mark all the nodes with a path to it with a specific label. However, this method cannot separate overlapped clusters perfectly, and is quite sensitive to parameter setting. There are several improved models based on SOINN, including E-SOINN (Furao, Ogura, & Hasegawa, 2007), Adjusted-SOINN (Shen & Hasegawa, 2008) and LB-SOINN (Zhang, Xiao, & Hasegawa, 2014). These models made some improvements to detect overlapped areas in the network.

2.2. Density based clustering

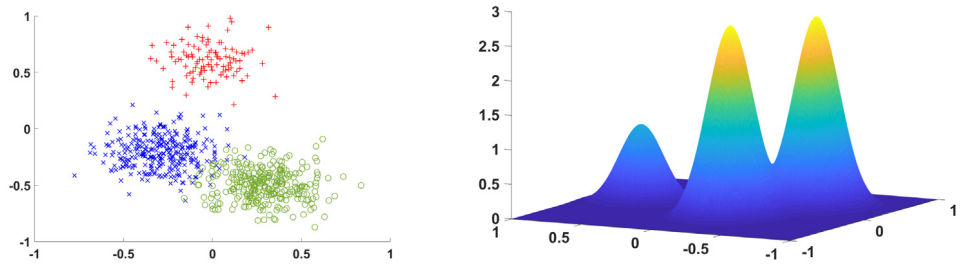
In density based clustering methods, a cluster is defined as a set of input patterns that form a connected high density region, which is separated from other clusters by connected low density regions. The pairwise dissimilarity between patterns in a density based cluster are not necessarily low when measured by any dissimilarity function. As a consequence, it is difficult to define a loss function for optimization. Density based methods perform well in some specific applications because they have the following advantages:

(1) Do not need to assume the number of clusters.

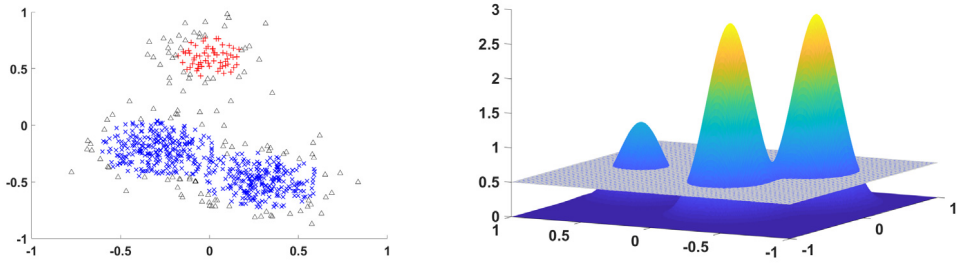
(2) Capable of generating arbitrarily shaped clusters.

(3) Can identify outliers in input patterns. Therefore density based methods are usually robust to noise.

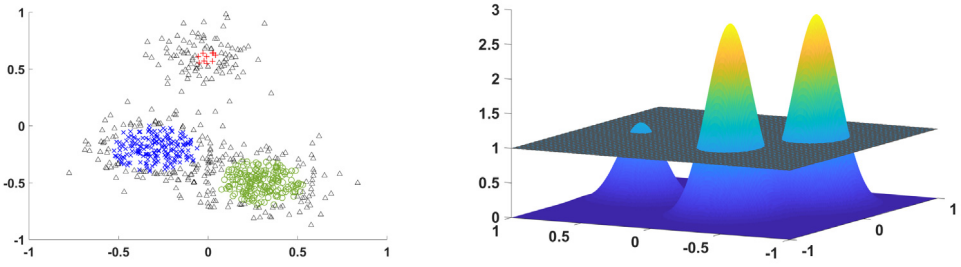
Density-based methods need to solve two problems, specifically, how to estimate the density distribution and how to define density connectivity. In many density based algorithms such as DBSCAN (Ester et al., 1996) and OPTICS (Ankerst, Breunig, Kriegel,



(a) A toy dataset and its probability density distribution



(b) Clustering result with low density threshold: failed to separate overlapped clusters



(c) Clustering result with high density threshold: too many patterns are marked as noise

Fig. 1. A toy dataset and density based clustering results with different density thresholds. The left column illustrates 2-D data points and clustering results. The right column illustrates the probability density distribution and the density threshold for each clustering. The dataset is comprised of patterns sampled from three Gaussian distributions. Patterns in different clusters are shown as markers with different shapes and colors. Patterns marked as noise are shown as black triangles.

& Sander, 1999), the density of a pattern x is computed by counting the number of patterns within a predefined radius ϵ. DENCLUE (Hinneburg & Keim, 1998) uses the kernel density estimation to estimate probability density distribution. Density connectivity is usually defined based on distances between patterns, for example, two patterns are defined as directly connected if the distance between them does not exceed the threshold ϵ in DBSCAN. Density connectivity based algorithms can be viewed as defining a threshold in the probability density function, and clusters are found in connected regions where the probability density is higher than the threshold.

However, there are some challenging issues in these algorithms, as illustrated in Fig. 1. If the density threshold is set too low, the algorithm could not separate overlapped clusters. If the density threshold is set too high, too many patterns would be considered as noise. In datasets where the local densities of clusters are largely different, it is especially difficult to choose a proper density level. Moreover, the choice of density level is sensitive to parameters, but sometimes the parameters are hard to set. Take DBSCAN as an example, both the maximum radius of the neighborhood(ϵ) and the minimum number of neighbors required to be a core point(*minPts*) are hard to set in high dimensional datasets.

In recent years, a novel density based algorithm was proposed in Rodriguez and Laio (2014), where clustering is performed by finding density peaks in patterns. The "density peak" method can be described as:"cluster centers are surrounded by neighbors with lower local density and they are at a relatively large distance from any points with a higher local density". This algorithm avoids the challenging problem of finding a proper density level to separate clusters. However, the density estimation method is the same with that in DBSCAN, which still needs to preset a distance threshold.

3. Density Based Self Organizing Incremental Neural Network

Density Based Self Organizing Incremental Network (DenSOINN) adopts a single-layer network structure, which is similar with traditional competitive neural networks like Self Organizing Map, Neural Gas and E-SOINN. The network structure can

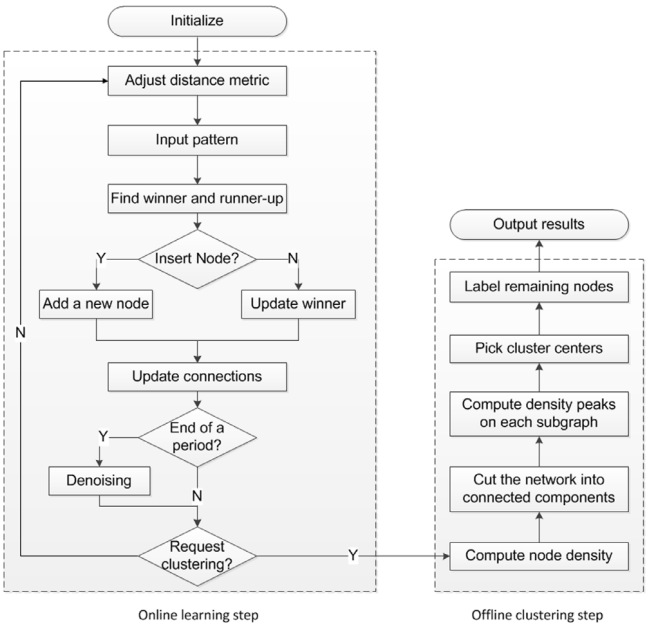


Fig. 2. Flowchart of DenSOINN.

be represented by a directed graph $G = \langle V, E \rangle$, where V is the set of neural nodes and E is the set of connections. Neighboring patterns on the feature space are mapped to a same node or two adjacent nodes in the network. Each node $i \in V$ is a prototype of the micro cluster formed by input patterns mapped to it. Edges in E are directed edges, we denote the edge from node i to node j as (i, j). As a clustering algorithm, V is separated into many subsets each of which represents a cluster. The clustering result is denoted as a label vector $\boldsymbol{l} = (l_1, l_2, \ldots, l_{|V|})^T$. Two nodes i and j are in the same cluster if $l_i = l_j$. DenSOINN outputs V, E and $\boldsymbol{l}$ when receiving a clustering request.

A flowchart of DenSOINN is shown in Fig. 2. Like most stream clustering algorithms, DenSOINN can be summarized into two steps: an online training of network structure and an offline density based clustering of neural nodes. During the online learning step, the distance metric is adjusted at the beginning of each learning round. When an input pattern comes, a competition among nodes is triggered and the winner and the runner-up are found. If both the winner and the runner-up are activated by the input pattern, a connection is built between them and the winner is updated. Otherwise the input patterns is added to the network as a new node. A denoising procedure is implemented after a number of learning rounds. DenSOINN starts the offline clustering step when receiving a clustering request. First the network is cut into connected components, then density peaks on each subgraph are assigned as cluster centers and other nodes are grouped accordingly. We will describe the detail of each step in following parts of this section.

3.1. Self-adaptive distance metric

We discussed in the introduction that data normalization is difficult to implement in stream clustering due to limited access of training data. A self-adaptive distance metric is adopted to solve the problem. The metric function is a Mahalanobis distance, the formula of which is written as:

$$d_A(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T A (\boldsymbol{x} - \boldsymbol{y})} \tag{1}$$

Here A is a positive semi-definite parameter matrix. The distance metric is "self-adaptive" because A is updated incrementally in each learning round based on recorded statistical characteristics of input data.

The Mahalanobis distance can be viewed as the Euclidean distance in arbitrary linear scalings and rotations of the original input space. We use this distance metric on raw data to approximate the Euclidean distance on normalized data. Specifically, the parameter matrix A adapts to varying value ranges of features, making the features contribute approximately proportionately to the final distance.

In the algorithms and experiments described in this paper, we adjust A according to the method of min–max normalization. Suppose the range of a feature in ith-dimension is $[min_i, max_i]$, and the target range of min–max normalization is $[0, 1]$, then for a pattern $\boldsymbol{x}$ the formula of min–max normalization is:

$$x'_i = \frac{x_i - min_i}{max_i - min_i} \tag{2}$$

Denote $\boldsymbol{x}'$ and $\boldsymbol{y}'$ as normalized versions of patterns $\boldsymbol{x}$ and $\boldsymbol{y}$, then we can get the following theorem to compute the parameter matrix A.

Theorem 1. *Let A be a diagonal parameter matrix of the Mahalanobis distance where $A_{ii} = \frac{1}{(max_i - min_i)^2}$, then $d_A(\boldsymbol{x}, \boldsymbol{y}) = d_{euc}(\boldsymbol{x}', \boldsymbol{y}')$.*

Proof. According to Eq. (1):

$$d_A(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} A_{ii}(x_i - y_i)^2} = \sqrt{\sum_{i=1}^{n} (\frac{x_i - y_i}{max_i - min_i})^2}$$

And the Euclidean distance between $\boldsymbol{x}'$ and $\boldsymbol{y}'$ is

$$d_{euc}(\boldsymbol{x}', \boldsymbol{y}') = \sqrt{\sum_{i=1}^{n} (\frac{x_i - min_i}{max_i - min_i} - \frac{y_i - min_i}{max_i - min_i})^2} = \sqrt{\sum_{i=1}^{n} (\frac{x_i - y_i}{max_i - min_i})^2}$$

Hence we can get $d_A(\boldsymbol{x}, \boldsymbol{y}) = d_{euc}(\boldsymbol{x}', \boldsymbol{y}')$. □

In data stream clustering, the maximum and minimum values of features cannot be computed directly. In a straightforward way, we only need to record the maximum and minimum values of historical input patterns during the online learning step, and adjust A accordingly when an input pattern renews the records. However, a major problem of min–max normalization is that the maximum and minimum values of input patterns are easily affected by outliers, therefore we use the maximum and minimum positions of network nodes instead, which are less easily affected. For a node j, denote its position in n-dimensional feature space as: $\boldsymbol{w_j} = (w_{j,1}, w_{j,2}, \ldots, w_{j,n})^T$. Because positions of nodes change in each learning round, A needs to be updated accordingly. At the beginning of each learning round, the parameter matrix A is adjusted by Algorithm 1.

In the following parts of this paper, we will omit the parameter matrix A and use $d(\boldsymbol{x}, \boldsymbol{y})$ to denote the distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ computed by this metric unless A needs to be specifically defined.

3.2. Online network training

The nodes and connections in DenSOINN are iteratively trained during the online learning step. Each node in the network represents a micro-cluster of input patterns. For each node i, three properties are preserved to describe the micro-cluster: (1) a prototype vector $\boldsymbol{w_i} = (w_{i,1}, w_{i,2}, \ldots, w_{i,n})^T$ that represents the centroid of the micro-cluster, which is also the "position" of node i in the

Algorithm 1 Self-adaptive Distance Metric

Input: V: set of nodes;
Output: A: $n \times n$ parameter matrix of Mahalanobis distance;
Initialize A as a zero matrix;
for each dimension of input space i **do**
  $max_i = -\infty$, $min_i = \infty$;
  for each node $j \in V$ **do**
    if $w_{j,i} > max_i$ **then**
      $max_i = w_{j,i}$;
    else
      if $w_{j,i} < min_i$ **then**
        $min_i = w_{j,i}$;
      end if
    end if
  end for
  if $max_i = min_i$ **then**
    $A_{ii} = 0$;
  else
    $A_{ii} = \frac{1}{(max_i - min_i)^2}$;
  end if
end for
return A

feature space; (2) a weight m_i that counts the accumulated number of input patterns in the micro-cluster; (3) an activation threshold t_i that describes its relationship with other nodes.

If node i has direct topological neighbors, then t_i is computed as the maximum distance between i and its neighbors:

$$t_i = \max_{j \in N_i} d(\boldsymbol{w_j}, \boldsymbol{w_i}) \tag{3}$$

In Eq. (3), N_i denotes the set of neighbors of node i, i.e. $N_i = \{j \in V | (i,j) \in E \text{ or } (j,i) \in E\}$. If N_i is empty, then t_i is computed as the minimum distance between i and other nodes:

$$t_i = \min_{j \in V} d(\boldsymbol{w_j}, \boldsymbol{w_i}) \tag{4}$$

The edge from node i to its neighbor j, denoted by (i,j), has a weight $m_{i,j}$ that decreases when the position of i changes and increases when i and j are both activated.

At the beginning of the algorithm, the network is initialized as an empty graph $G = \langle V, E \rangle$, where V and E are empty sets. Before processing each incoming pattern, the distance metric is adjusted by Algorithm 1. The learning procedure in each iteration is illustrated in Fig. 3. When an input pattern $\boldsymbol{x}$ arrived, if $|V| < 2$, then $\boldsymbol{x}$ forms a new micro-cluster, i.e. a new node i. The centroid of the micro-cluster $\boldsymbol{w_i} = \boldsymbol{x}$ and its weight, i.e. the accumulated count of patterns, $m_i = 1$. If $|V| \geq 2$, trigger a competition among the nodes in V. The winner s_1 and runner-up s_2 are found by the following equations:

$$s_1 = arg \min_{i \in V} d(\boldsymbol{x}, \boldsymbol{w_i}) \tag{5}$$

$$s_2 = arg \min_{i \in V - \{s_1\}} d(\boldsymbol{x}, \boldsymbol{w_i}) \tag{6}$$

If $d(\boldsymbol{x}, \boldsymbol{w_{s_1}}) \leq t_{s_1}$ and $d(\boldsymbol{x}, \boldsymbol{w_{s_2}}) \leq t_{s_2}$, then $\boldsymbol{x}$ can activate both s_1 and s_2. In this case $\boldsymbol{x}$ is judged as belonging to the micro-cluster represented by s_1, and then m_{s_1} is increased by 1 and $\boldsymbol{w_{s_1}}$ is updated as following:

$$\boldsymbol{w_{s_1}} = \boldsymbol{w_{s_1}} + \frac{1}{m_{s_1}}(\boldsymbol{x} - \boldsymbol{w_{s_1}}) \tag{7}$$

As the position of s_1 changes, the old connections starting from s_1 might be outdated. The weights of connections are decreased each time the position of s_1 changes, and a connection is deleted when its weight drops below a threshold. Here we define the threshold as 10% of the summed weights of connections starting from s_1. The weights of connections are updated as following, for each edge $(s_1, i) \in E$:

$$m_{s_1,i} = 2^{-\gamma} m_{s_1,i} \tag{8}$$

Here γ is a user defined parameter that controls the decay of edges. If no edge from s_1 to s_2 exists, add a new edge (s_1, s_2) to E, and set $m_{s_1,s_2} = 1$, otherwise $m_{s_1,s_2} = m_{s_1,s_2} + 1$. Then for each edge $(s_1, i) \in E$, if $m_{s_1,i} < \frac{\sum_{(s_1,j) \in E} m_{s_1,j}}{10}$, the edge (s_1, i) is removed from E. The edge management scheme will be further discussed in Section 4.2.

If $\boldsymbol{x}$ cannot activate s_1 or s_2, then $\boldsymbol{x}$ does not belong to any existing micro-cluster. In this case a micro-cluster is added, represented by a new node r in the network. The centroid $\boldsymbol{w_r} = \boldsymbol{x}$ and its weight $m_r = 1$. A new edge (r, s_1) connecting the new node and its nearest neighbor is added to E. The procedure of network building is illustrated in Fig. 3.

The maximum volume of the network can be controlled by a user-defined parameter *MaxNodes*, in case that the number of nodes becomes too high and the computation time is unaffordable. If $|V| \geq MaxNodes$, the network will not generate new nodes and new input patterns always belong to its nearest micro-cluster.

At the end of each time period, the algorithm enters a denoising procedure. The denoising intensity is controlled by a parameter ϵ. A node i is marked as noise if it has few neighbors and its weight is lower than a proportion of average weight of all nodes. If the weight of two nodes are equal, then the nodes with more neighbors are less likely to be noise. Namely, node i is considered to be noise if the following condition is met:

$$m_i < \frac{10^{1-|N_i|}\epsilon}{|V|} \sum_{j \in V} m_j \tag{9}$$

Noise nodes and related edges are removed from the network.

The online training step repeats until all patterns in the stream have been learned or a clustering request is received. The algorithm of online training step is given in Algorithm 2.

3.3. Offline clustering of nodes

A clustering of nodes starts when a clustering request is received. Due to the effect of the distance metric, the micro clusters represented by nodes are elliptical shaped. However, the shape of underlying clusters in data is uncertain. The target of node clustering is to combine these micro clusters into macro clusters, which could possibly be clusters of any shape. Therefore we adopt a density based offline clustering method.

We first describe how node density is computed. The density of node i, denoted by p_i, is defined as the mean probability density of its Voronoi region R_i. Denote the volume of R_i as $vol(R_i)$, then the probability that a pattern $\boldsymbol{x}$ locates in R_i is $P(\boldsymbol{x} \in R_i) = p_i vol(R_i)$, and can be estimated by $P(\boldsymbol{x} \in R_i) \approx \frac{m_i}{\sum_{j \in V} m_j}$. Hence we can get:

$$p_i \approx \frac{m_i}{vol(R_i) \sum_{j \in V} m_j} \tag{10}$$

However, the volume of Voronoi region is difficult to compute in high dimensional feature spaces. A naive estimation is $vol(R_i) \propto d_i^n$, where n is the dimensionality and d_i is the mean distance from node i to its neighbors:

$$d_i = \frac{1}{|N_i|} \sum_{j \in N_i} d(\boldsymbol{w_i}, \boldsymbol{w_j}) \tag{11}$$

However, this estimation does not work well in real world datasets. In high dimensional datasets, patterns usually distribute

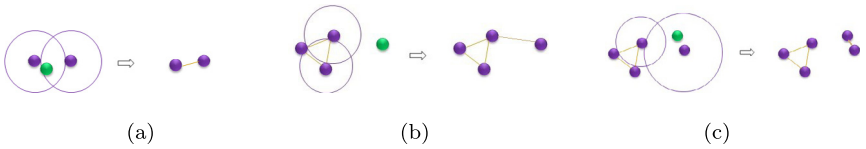


Fig. 3. Online network training. In the figures, green nodes represent the input examples, other nodes represent the nodes in the network. In Fig.(a), the nearest node is pulled to the input example and an edge is established between two anchor points. In Fig.(b) and (c), new nodes are inserted adaptively and new edges are established between the new node and its nearest neighbor.

Algorithm 2 Online Network Training of DenSOINN

Input: The input dataset, X; The speed of edge weight decay, γ; The number of input patterns in a time period, *StreamSpeed*; The intensity of denoising, ϵ; The maximum number of nodes, *MaxNodes*;
Output: The set of nodes, V; The set of edges, E;
1: initialize the network with $V = \phi, E = \phi$;
2: **for** each pattern $\boldsymbol{x} \in X$ **do**
3: **if** $|V| < 2$ **then**
4: create a new node r that $\boldsymbol{w_r} = \boldsymbol{x}$ and add it to V, goto step 2;
5: **end if**
6: adjust the distance metric by Algorithm 1;
7: find winner s_1 and runner-up s_2 using Eqs. (5) and (6);
8: compute activation threshold t_{s_1} and t_{s_2} with Eqs. (3) and (4);
9: **if** $|V| <$ *MaxNodes* and ($d(\boldsymbol{x}, \boldsymbol{w_{s_1}}) > t_{s_1}$ or $d(\boldsymbol{x}, \boldsymbol{w_{s_2}}) > t_{s_2}$) **then**
10: create node r that $\boldsymbol{w_r} = \boldsymbol{x}$ and add it to V;
11: build edge (r, s_1) and add it to E;
12: **else**
13: increase m_{s_1} by 1 and update $\boldsymbol{w_{s_1}}$ using Eq. (7);
14: decrease the weight of edges starting from s_1 using Eq. (8);
15: search and delete edges with small weights;
16: **if** (s_1, s_2) does not exist **then**
17: build edge (s_1, s_2) and add it to E;
18: **else**
19: increase the weight m_{s_1,s_2} by 1;
20: **end if**
21: **end if**
22: **if** the number of inputs is an integer multiple of *StreamSpeed* **then**
23: compute the average weight of nodes;
24: search and delete nosing nodes and their edges using Eq. (9);
25: **end if**
26: **end for**
27: **return** V, E

on a manifold with a lower intrinsic dimensionality. The intrinsic dimensionality, denoted as n', is usually much smaller than n. Therefore we estimate $vol(R_i) \propto d_i^{n'}$. n' cannot be known as priority, but in our experiments we found that setting it as a small value like 2 could usually get acceptable results. Combining the equations above, the value of p_i is estimated as:

$$p_i \propto \frac{m_i}{1 + d_i^2} \tag{12}$$

Here a regularizer is added in order to prevent the density of nodes with very small neighborhood radius being too high.

The offline clustering of nodes in DenSOINN can be considered as a combination of "density connectivity" and "density peak" methods. Density connected regions are defined as weakly connected components in the network graph. We first cut the network into weakly connected subgraphs, then find density peaks on each subgraph and appoint them as cluster centers. A simple example that illustrates the procedure is shown in Fig. 4.

Take a weakly connected component $SG = \langle SV, SE \rangle$ as an example, for each node $i \in SV$, find the nearest node in SV that has a higher density than i, denote the node as q_i:

$$q_i = arg \min_{j \in SV \text{ and } p_i < p_j} d(\boldsymbol{w_i}, \boldsymbol{w_j}) \tag{13}$$

Then we assign node i a "peak score" denoted as h_i:

$$h_i = p_i d(\boldsymbol{w_i}, \boldsymbol{w_{q_i}}) \tag{14}$$

Node i is chosen as a cluster center if it satisfies the following condition:

$$h_i \geq \alpha h_{mean} \tag{15}$$

In Eq. (15) α is predefined parameter, and h_{mean} is the average peak score of nodes in SV.

It is noticeable that if the density of node i is the highest in SV, then q_i cannot be found and hence h_i cannot be computed. In this case we set $h_i = \infty$, which means node i is always chosen as a cluster center, and h_{mean} is computed without h_i.

If node i is not chosen as a cluster center, it is grouped into the same cluster with q_i. To assure q_i is already assigned to a cluster when i is processed, SV is sorted in descend order of node density, thus q_i is always processed before i.

The method is described in Algorithm 3.

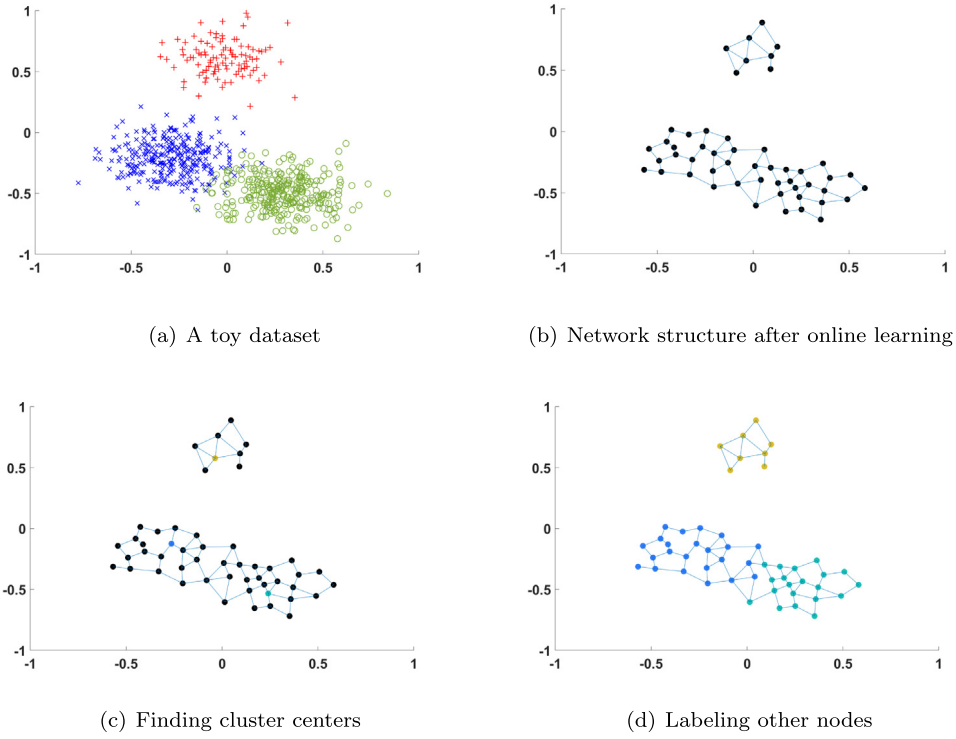
4. Discussion and analysis

4.1. Discussion of self-adaptive distance metric

We described the self-adaptive distance metric in Section 3.1, which approximates min–max normalization in an incremental approach. The effect of self-adaptive distance metric is similar to a simple metric learning algorithm. The target of metric learning is choosing a proper distance metric that could improve the performance of a specific metric based learning algorithm, like k-means and k-nearest neighbor classifier.

In existing metric learning frameworks (Davis, Kulis, Jain, Sra, & Dhillon, 2007; Weinberger & Saul, 2009; Xing, Ng, Jordan, & Russell, 2002), the metric is learned before used in a metric based algorithm. However, the self-adaptive distance metric in DenSOINN is learned simultaneously while it is used in clustering, which distinguishes it from metric learning algorithms.

Other effects besides min–max normalization can also be achieved by incrementally updating the parameter matrix A of the metric. For example, whitening transformation is another widely used preprocessing technique that could eliminate the correlations between features. When the parameter matrix A is assigned as the inverse of the covariance matrix Σ of the input dataset, the Mahalanobis distance in the input space is equivalent to the Euclidean



(a) A toy dataset (b) Network structure after online learning

(c) Finding cluster centers (d) Labeling other nodes

Fig. 4. Clustering procedure on the same dataset shown in Fig. 1. The network graph is separated into weakly connected subgraphs. Then density peaks on each subgraph are appointed as cluster centers. Remaining nodes are grouped into the same cluster with the nearest node that has higher density.

Algorithm 3 Clustering by node density

Input: V: set of nodes, E: set of edges, α: parameter for cluster center selection;
Output: $\boldsymbol{l}$: cluster labels of nodes;

for each node $i \in V$ **do**
  compute node density p_i by Eqs. (12) and (11);
end for
separate the graph $G =< V, E >$ into connected components $\{SG_1, SG_2, ...\}$;
$y = 1$;
for each connected component $SG_t =< SV_t, SE_t >$ **do**
  sort nodes in SV_t in descend order of node density;
  for each node $i \in SV_t$ **do**
    find the nearest node with a higher density q_i by Eq. (13);
    compute peak score h_i by Eq. (14);
  end for
  compute mean peak score h_{mean};
  for each node $i \in SV_t$ **do**
    if $h_i \geq \alpha h_{mean}$ **then**
      i is chosen as a cluster center: $l_i = y, y = y + 1$;
    else
      group i into the same cluster with q_i: $l_i = l_{q_i}$;
    end if
  end for
end for
$\boldsymbol{l} = (l_1, ..., l_{|V|})^T$;
return $\boldsymbol{l}$

distance in the transformed space. In data stream clustering, the covariance matrix Σ as well as its inverse cannot be directly computed due to limited accessibility of training data. However, we are able to incrementally learn a matrix A that approximates C^{-1} in each learning round using the following theorem.

Theorem 2. *Denote the mean value and the covariance matrix of the first n input patterns* $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ *as* $\bar{\boldsymbol{x}}_n$ *and* C_n *respectively, then* C_n^{-1} *can incrementally computed as:*

$$C_n^{-1} = \frac{n}{(n-1)} C_{n-1}^{-1} - \frac{[C_{n-1}^{-1}(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})][(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})^T C_{n-1}^{-1}]}{(n-1)[1 + \frac{1}{n}(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})^T C_{n-1}^{-1}(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})]} \tag{16}$$

Proof. It is obvious that $\bar{\boldsymbol{x}}_n$ can be incrementally computed as:

$$\bar{\boldsymbol{x}}_n = \bar{\boldsymbol{x}}_{n-1} + \frac{(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})}{n}$$

Then we can update C_n as following:

$$\begin{aligned} C_n &= \sum_{i=1}^{n} \frac{(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_n)(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_n)^T}{n} \\ &= \sum_{i=1}^{n} [\frac{(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_{n-1})(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_{n-1})^T}{n} \\ &\quad - \frac{(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})(\boldsymbol{x}_i - \bar{\boldsymbol{x}}_{n-1})^T + (\boldsymbol{x}_i - \bar{\boldsymbol{x}}_{n-1})(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})^T}{n^2} \\ &\quad + \frac{(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})^T}{n^3}] \\ &= \frac{(n-1)C_{n-1}}{n} + \frac{(n-1)(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})^T}{n^2} \end{aligned}$$

Then C_n^{-1} can be incrementally compute using the Sherman–Morrison formula, which can be written as

$$(M + \boldsymbol{b}\boldsymbol{c}^T)^{-1} = M^{-1} + \frac{M^{-1}\boldsymbol{b}\boldsymbol{c}^T M^{-1}}{1 + \boldsymbol{c}^T M^{-1}\boldsymbol{b}}$$

where $M = \frac{(n-1)C_{n-1}}{n}$, $\boldsymbol{b} = (n-1)(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1})$ and $\boldsymbol{c} = \frac{\boldsymbol{x}_n - \bar{\boldsymbol{x}}_{n-1}}{n^2}$ specifically. Hence we can get Eq. (16) by simplifying the formula above. □

If the distribution of training data is stationary, then C_n^{-1} would approximate C^{-1} after learning a part of training data.

The self-adaptive distance metric introduced in Section 3.1 simplifies the parameter matrix into a diagonal matrix. In this case, the metric applies linear scaling to input feature space, thus realizing the effect of feature scaling, but it allows correlation between features.

As described in Section 3.1, the diagonal matrix A is computed with respect to the algorithm of min–max normalization in DenSOINN. It is worth mentioning that there are many other data normalization algorithms besides min–max normalization, and no single algorithm works best on all datasets. Other data normalization algorithms whose relevant parameters have an incremental way of computation can also fit in our self-adaptive distance framework.

For example, we can define a self-adaptive distance metric that approximate the effect of standardization. For each feature i, the formula of standardization is $x'_i = \frac{x_i - \bar{x}_i}{\delta_i}$, where $\bar{x}_i$ and δ_i are the mean and standard deviation of the ith feature. It is easy to infer that $d_{euc}(\boldsymbol{x}', \boldsymbol{y}') = \sqrt{\sum_{i=1}^{n}(\frac{x_i - y_i}{\delta_i})^2}$.

Combining this metric with the Mahalanobis distance, we can get that the ith diagonal element of A is $A_{ii} = \frac{1}{\delta_i^2}$. A_{ii} can be computed by updating δ_i^2 when receiving each pattern. Denote mean and variance of the ith feature after learning t input pattern $\bar{x}_{i,t}$ and $\delta_{i,t}^2$ respectively, the formulas are:

$$\mu_{i,t} = \mu_{i,t-1} + \frac{x_i - \mu_{i,t-1}}{t} \tag{17}$$

$$\delta_{i,t}^2 = \frac{(t-1)\delta_{i,t-1}^2 + (x_i - \bar{x}_{i,t})(x_i - \bar{x}_{i,t-1})}{t} \tag{18}$$

4.2. Density connectivity in DenSOINN

We described in Section 3.3 that density connected regions are defined as weakly connected components in the network. In this subsection we will discuss the relationship between the network graph and density connectivity.

The edges of DenSOINN is built following competitive Hebbian rule, which is highly related with Voronoi tessellation. Suppose the feature space is n-dimensional real space, and the probability distribution function of input data is an unknown function $P(x)$, then the 2nd-order Voronoi region R_{ij} associated with node i and j is defined as:

$$R_{ij} = \{\boldsymbol{x} | \forall k \notin \{i, j\}, d(\boldsymbol{x}, \boldsymbol{w_i}) \leq d(\boldsymbol{x}, \boldsymbol{w_j}) \leq d(\boldsymbol{x}, \boldsymbol{w_k})\} \tag{19}$$

R_{ij} is not empty only when R_i and R_j are adjacent.

Recall that every time DenSOINN receives an input pattern $\boldsymbol{x}$, it finds the closest and second closest nodes to $\boldsymbol{x}$, denoted by s_1 and s_2 respectively. Obviously $\boldsymbol{x} \in R_{s_1 s_2}$. If $\boldsymbol{x}$ can activate both s_1 and s_2, then the edge (s_1, s_2) is created or m_{s_1,s_2} is increased. Otherwise a new node r is created whose feature vector $\boldsymbol{w_r} = \boldsymbol{x}$. In this case $d(\boldsymbol{x}, \boldsymbol{w_r}) = 0 \leq d(\boldsymbol{x}, \boldsymbol{w_{s_1}}) \leq d(\boldsymbol{x}, \boldsymbol{w_{s_2}})$, therefore $\boldsymbol{x} \in R_{rs_1}$ and the edge (r, s_1) is created. Therefore we can conclude that the edge (i, j) represents the 2nd-order Voronoi region R_{ij} where $p(\boldsymbol{x} \in R_{ij}) = \int_{R_{ij}} P(x) > 0$.

The network built by this method is usually highly connected because it does not set any density thresholds. Although we could separate clusters in a totally connected network by finding density peaks, the clustering quality would still benefit from cutting the network into subgraphs before this step. Therefore we remove edges that represent relatively low density regions in order to make connected components of the network graph represent connected high density regions. A 2nd-order Voronoi region R_{ij} is considered as low density if $p(\boldsymbol{x} \in R_{ij} | \boldsymbol{x} \in R_i)$ is small. Another reason of edge deletion is that the positions of nodes changes all the time during the online learning step, and Voronoi regions would change accordingly. In this case connected nodes might not be neighboring after some learning rounds, and the edges connecting them become outdated.

Low density regions are represented by edges whose weights are relatively small. If node i has k edges starting from it, and a pattern $\boldsymbol{x}$ locates in R_i, then which 2nd-order Voronoi region $\boldsymbol{x}$ belongs to follows a multinomial distribution with parameter $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$. We want to find regions whose probabilities of containing $\boldsymbol{x}$ are low, therefore we consider the distribution of parameter $\boldsymbol{\mu}$, which is a Dirichlet distribution with hyperparameter $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_k)$. The probability distribution and expected value of $\boldsymbol{\mu}$ are computed as following:

$$p(\boldsymbol{\mu}|\boldsymbol{\phi}) = \frac{\Gamma(\sum_{j=1}^{k}\phi_j)}{\prod_{j=1}^{k}\Gamma(\phi_j)} \prod_{j=1}^{k} \mu_j^{\phi_j - 1} \tag{20}$$

$$E(\boldsymbol{\mu}) = \frac{\boldsymbol{\phi}}{\sum_{j=1}^{k}\phi_j} \tag{21}$$

The hyperparameter $\boldsymbol{\phi}$ is computed by counting the effective number of patterns in each 2nd-order Voronoi region, which is recorded in the weights of edges starting from node i.

Consider the simple situation that all positions of nodes are stable, which means no new nodes will be added and positions of existing nodes will not change, and the Voronoi regions will not change too. In this case ϕ_j is simply the count of patterns in R_{ij}.

However, the positions of nodes and the Voronoi regions change from time to time. Each time the position of i changes, the counts of historical patterns in its associated 2nd-order Voronoi regions become less effective. In order to record the effective number of patterns, we adopt a damped window model that assign higher effectiveness for more recent patterns. The effectiveness of each pattern $\boldsymbol{x} \in R_i$ decreases exponentially with the times of position change of node i. The decreasing function is $e(\boldsymbol{x}) = 2^{-\gamma t}$, where $\gamma > 0$ and t is the times of position change of i after $\boldsymbol{x}$ is input. The weight of edge (i, j) is computed as $m_{i,j} = \sum_{\boldsymbol{x} \in R_{ij}} e(\boldsymbol{x})$.

The computation of edge weights can be implemented incrementally, as described in Section 3.2. Each time a pattern $x \in R_{ij}$ is input, the position of node i is updated, then the weights of edges starting from i are decreased by multiplying $2^{-\gamma}$ and $m_{i,j}$ is increased by one. The expected value of $p(\boldsymbol{x} \in R_{ij} | \boldsymbol{x} \in R_i)$ is computed as:

$$E(p(\boldsymbol{x} \in R_{ij} | \boldsymbol{x} \in R_i)) = \frac{m_{i,j}}{\sum_{(i,r)\in E} m_{i,r}} \tag{22}$$

If $E(p(\boldsymbol{x} \in R_{ij} | \boldsymbol{x} \in R_i)) < 0.1$, R_{ij} is considered as low density region and (i, j) is removed from E. By this means low density 2nd-order Voronoi regions are found and edges representing these regions are removed from the network. The effect of the parameter γ is similar to the parameters in density based clustering algorithms that set the density threshold, such as the radius parameter ϵ and minimum number of neighbors *minPts* in DBSCAN. However, γ is much easier to tune because it is not affected by the dimensionality and value range of features.

Normally, overlapped clusters cannot be completely separated by cutting the network into weakly connected components. Therefore we still need to find cluster centers on each subgraph, as described in Section 3.3.

4.3. Complexity analysis

Suppose the input data steam is a stream of $|X|$ patterns, where each pattern is a n-dimensional feature vector. The most time consuming computation in each online learning round is finding the nearest and second nearest node to the input pattern, which is executed in $O(n|V|)$. Here $|V|$ is the number of neural nodes in the

Table 1
Datasets used in experiments.

Dataset	Classes	Instances	Attributes
Artificial I	2	10000	2
Swiss Roll	4	4000	3
London Air Quality	–	2976	2
Segment	7	2310	19
Pendigits	10	10992	16
HAR	6	10299	561
Usps	10	9298	256
Yale Face	10	5850	1200
MNIST	10	70000	780
Cover Type	7	581012	54
KDD99	23	4898431	72
URL Reputation	2	2396130	64

network. However, $|V|$ changes from time to time during the online learning step. Denote the maximum of $|V|$ as k, the total execution time of online learning step is $O(nk|X|)$.

$|V|$ is usually approximate to k when the offline clustering step starts. The most time consuming computation in the offline clustering step is finding the nearest node with a higher density for each node, which needs to compute distances between each pair of nodes in the same subgraph. In the worst situation when the network is totally connected, the complexity of this computation is $O(nk^2)$.

Normally, k is far smaller than $|X|$, therefore the time complexity of algorithm DenSOINN is $O(nk|X|)$

The data structures that DenSOINN needs most memory space to store are the properties of nodes and a weighted graph data structure. The properties of each node takes $n + 2$ memory units, therefore the total memory space for storing nodes is $O(nk)$. The graph is saved as an oriented incidence matrix in our implementation, which requires at most $O(k^2)$ memory units to store. Therefore the total space complexity of DenSOINN is $O(nk) + O(k^2)$.

5. Experimental validation

5.1. Datasets, methods and validation criteria

We performed our experiments on both artificial and real datasets to evaluate the clustering quality of our algorithm. Properties of the datasets used in our experiments are given in Table 1. Note that the datasets are not normalized before given to the clustering algorithms. The artificial datasets are low dimensional in order that the results can be presented visually. First one is a 2-dimensional dataset created by randomly sampling from a Gaussian Mixture Model with different scales on X-axis and Y-axis, named as Artificial I. The second dataset is a Swiss Roll dataset, which is a 3-dimensional dataset converted from a 2-dimensional Gaussian Mixture Model with swiss roll mapping $(x, y) -> (x\cos x, y, x\sin x)$. The figures of artificial datasets are illustrated in Fig. 5.

In our experiments, DenSOINN is compared with four stream clustering algorithms: G-Stream (Ghesmoune et al., 2016), StrAP (Zhang, Xiangliang et al., 2014), online K-Means (Lloyd, 1982) and StreamKM++(Ackermann et al., 2012). All comparative algorithms are implemented in Matlab except StreamKM++, which is implemented with a C++ code provided by the authors of Ackermann et al. (2012) (we removed most console output in this code to accurately record computation time of the algorithm).

In order to compare these algorithms from different aspects, we use five clustering validation measures to evaluate the experimental results, which can be categorized into two groups. External measures use class labels in datasets as "ground truth" and compare them with the partitions produced by clustering. The external measures used in our experiments are accuracy, Normalized Mutual Information (NMI) (Fred & Jain, 2003) and Adjusted Rand index (ARI) (Hubert & Arabie, 1985). Internal measures only rely on input data and clustering partitions. Silhouette coefficient (SC) and the quantization error are adopted as internal measures in our experiments. Due to the high computational cost of SC, this measure is not used on datasets with more than 50,000 examples. In this paper, we use Root Mean Squared Error (RMSE) to measure quantization error. RMSE is the square root of the average squared Euclidean distance between each data point and its nearest exemplar in the model (neural nodes of DenSOINN/GStream and cluster centers of online K-Means/StrAP/StreamKM++). The value ranges of accuracy, NMI, SC are [0,1], while the value range of ARI is $[-1, 1]$ and the value range of RMSE is $[0, +\infty)$. Higher values of accuracy, NMI, ARI and SC imply better clustering performance, but RMSE is the smaller the better.

We also report the number of clusters found by each algorithm because we consider that for an algorithm that could determine the number of clusters by itself, whether the number is reasonable can be used to judge its performance. Execution times of algorithms are reported to compare time efficiency.

5.2. Experiments on artificial datasets

In this subsection, we show visual results of our experiments on 2-Dimensional artificial datasets. In these figures, nodes and connections of neural networks are illustrated by colored points and lines between them. Nodes of DenSOINN that belong to a same cluster are denoted as points with same color. The first experiment on dataset Artificial I is aimed at validating the effect of the self-adaptive distance metric described in Section 3.1. The Artificial I dataset consists of 10,000 patterns sampled equally from two Gaussian distributions as shown in Fig. 5(a). In order to simulate unnormalized real world dataset, value ranges are different on X axis and Y axis.

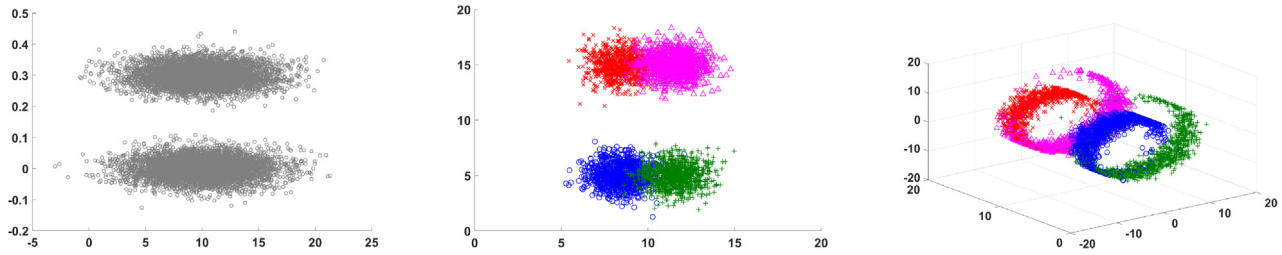
We compare DenSOINN with G-Stream (Ghesmoune et al., 2016) on both raw data and data normalized by min–max normalization. Parameters of DenSOINN are set as $\gamma = 0.25$, $StreamSpeed = 100, \epsilon = 1, \alpha = 5$; G-Stream uses the default parameters suggested in Ghesmoune et al. (2016). The result is shown in Fig. 6. We can see DenSOINN works fine in both environments and the results are approximately the same, while G-Stream generates a connected graph on the raw data. It indicates that the self-adaptive distance metric in DenSOINN can work on raw data just like Euclidean distance works on normalized data.

The second experiment is performed on the SwissRoll dataset with different level of noises in order to test the robustness to noise. The parameters are set as $\gamma = 0.5$, $StreamSpeed = 100$, $\epsilon = 0.5$, $\alpha = 6$. Fig. 7 depicts the results of this experiment.

It can be seen from Fig. 7 that the network contains more noisy nodes and connections as the amount of noise gets higher, but the clusters are still well separated in general and the clustering quality is not affected. The reason of this phenomenon is that the clustering result of DenSOINN mainly depends on the choice of cluster centers. Although the network might contain nodes and edges that represent noisy data, noisy nodes are not likely to be chosen as cluster centers, therefore its clustering quality does not suffer much loss.

5.3. Experiments on real world datasets

In this subsection, we present experimental results on real-world datasets. The 10% sampling set of KDD99 is used in this experiment instead of the full dataset. In each experiment, each algorithm is repeatedly run 10 times on a given dataset. Datasets are organized into random input sequences, each pattern can be

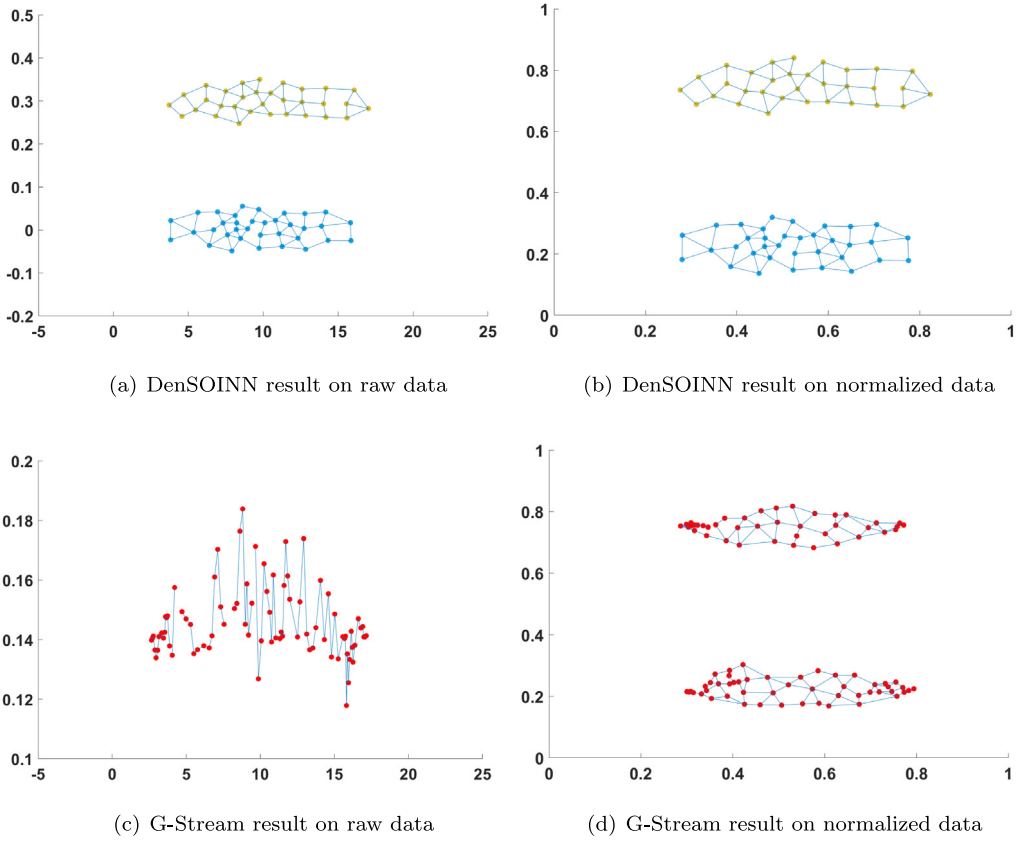


(a) Artificial I: mixture of 2 unnormalized Gaussian distributions

(b) Swiss Roll: mixture of four 2-dimensional Gaussian distributions

(c) Swiss Roll: the 3-dimensional Swiss Roll data converted from the Gaussian Mixture Model

Fig. 5. Artificial datasets. The Artificial I dataset consists of 10 000 patterns sampled equally from two Gaussian distributions. The Swiss Roll dataset is a 3-dimensional dataset converted with swiss roll mapping from a 2-dimensional Gaussian Mixture Model that consists of 4000 patterns sampled from four Gaussian distributions.



(a) DenSOINN result on raw data

(b) DenSOINN result on normalized data

(c) G-Stream result on raw data

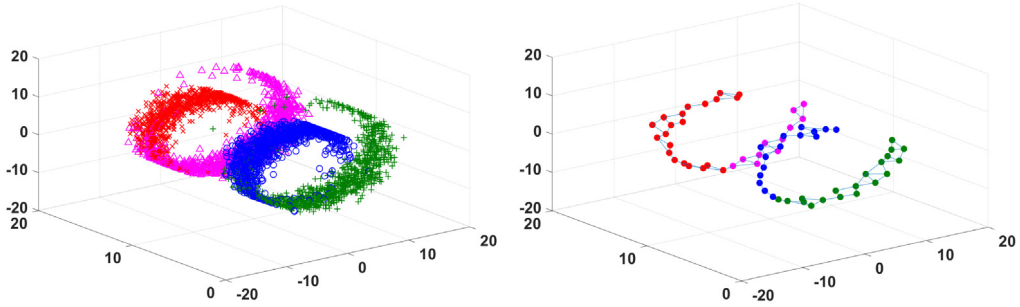
(d) G-Stream result on normalized data

Fig. 6. Results on Artificial I. DenSOINN is compared with G-Stream on both raw data and data normalized by min–max normalization. Nodes and connections of neural networks are illustrated by colored points and lines between them. Each node in G-Stream represents a cluster, and same color nodes of DenSOINN belongs to a same cluster.

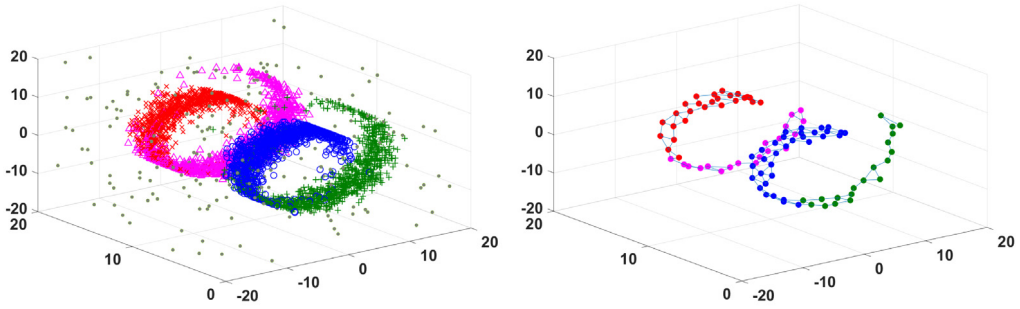
learned only once. Clustering requests are made after the input sequences are completely learned.

The parameters of comparative algorithms are the default parameters in the codes provided by the authors of relevant papers. The numbers of clusters in K-Means based algorithms are set as the numbers of classes in input data stream. The parameters of DenSOINN are set as: $\gamma = 0.25$, $StreamSpeed = 200$, $\epsilon = 0.5$, $\alpha = 5$ except for MNIST, where we set $\alpha = 3.5$. The maximum sizes of DenSOINN and G-Stream are not limited. The average value and standard deviation of clustering quality are reported in Table 2, as well as the number of clusters and running time(seconds).

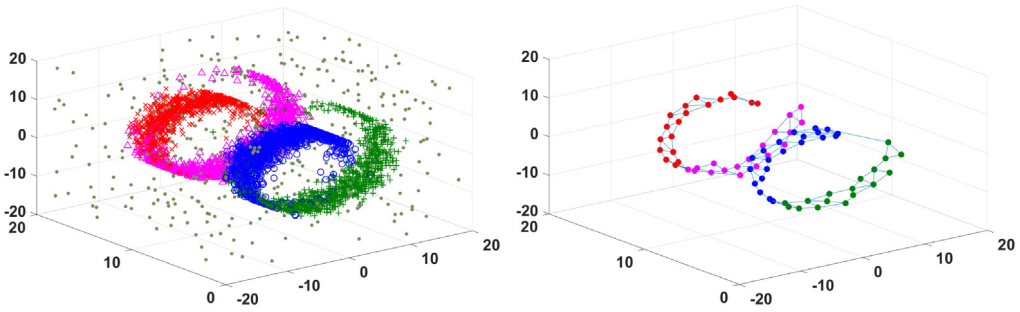
Judging by external measures, DenSOINN achieves the highest average NMI and ARI in most experiments. Its accuracy is lower than those of G-Stream and StrAP, but partly because DenSOINN generates much less clusters. High accuracy is easy to achieve by increasing the number of clusters, for example, the accuracy would be 1 in the extreme condition where each cluster only contains one pattern. Generally speaking, the partitions learned by DenSOINN is the most similar with the "ground-truth" in labeled datasets. However, the clustering quality of DenSOINN is more easily affected by the input sequence of patterns. It is noticeable that DenSOINN performs significantly better than other algorithms on KDD99 and Segment. The reason is that these datasets have



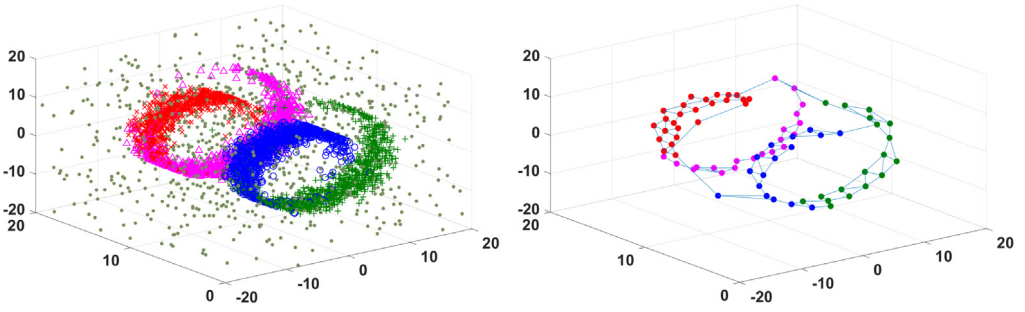
(a) 0% noise, resulting NMI=0.7299



(b) 5% noise, resulting NMI=0.7474



(c) 10% noise, resulting NMI=0.7555



(d) 20% noise, resulting NMI=0.7657

Fig. 7. Results on SwissRoll with noise. The left column shows training data with different level of noises, and the right column illustrates the learning results of DenSOINN.

some attributes whose range of value far exceed other attributes, and the distance metric in DenSOINN manages to cope with this problem.

Considering internal measures, the silhouette coefficient of DenSOINN is not as good as K-Means based algorithms in all experiments, while the RMSE of DenSOINN is the smallest in most experiments. This phenomenon implies that the nodes of DenSOINN could form a good summarization of input data, but the clusters are less compact than the clusters learned by K-Means algorithms because the macro clusters are formed by clustering

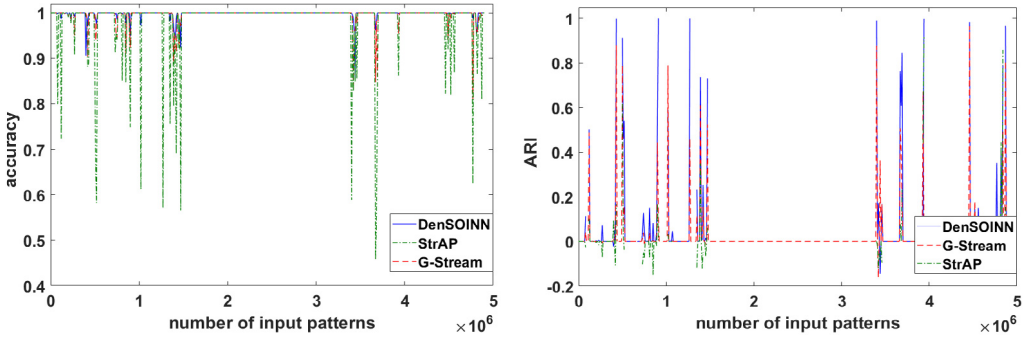
Table 2
Results on real-world datasets. The best result is shown in bold symbol.

Dataset		DenSOINN	G-Stream	StrAP	online K-Means	StreamKM++
Segment	Clusters	6.6 ± 1.1	23 ± 2.2	42.8 ± 3.8	7*	7*
	Acc	0.6150 ± 0.0648	0.6442 ± 0.0300	**0.7801 ± 0.0051**	0.5815 ± 0.0327	0.5441 ± 0.0350
	ARI	**0.4214 ± 0.0724**	0.2802 ± 0.0378	0.1906 ± 0.0066	0.3652 ± 0.0398	0.3602 ± 0.0549
	NMI	**0.6176 ± 0.0330**	0.4713 ± 0.0291	0.5301 ± 0.0056	0.5207 ± 0.0379	0.5292 ± 0.0456
	SC	0.0415 ± 0.0366	0.1990 ± 0.0435	0.4402 ± 0.0088	0.4102 ± 0.0310	**0.4698 ± 0.0086**
	RMSE	60.19 ± 10.95	91.24 ± 2.25	**39.8 ± 6.06**	90.45 ± 1.70	77.8 ± 0.4834
	Time	0.3358 ± 0.0187	0.0546 ± 0.0077	3.4540 ± 0.6214	0.0201 ± 0.0022	0.0893 ± 0.0055
Pendigits	Clusters	9.1 ± 0.9944	72.6 ± 4.326	40 ± 0	10*	10*
	Acc	0.6974 ± 0.0596	0.8896 ± 0.0095	**0.8908 ± 0.0010**	0.6852 ± 0.0282	0.7188 ± 0.0307
	ARI	**0.5642 ± 0.0568**	0.2801 ± 0.0124	0.3702 ± 0.0129	0.5121 ± 0.0329	0.5437 ± 0.0312
	NMI	**0.7387 ± 0.0347**	0.6413 ± 0.0073	0.6760 ± 0.0073	0.6516 ± 0.0146	0.6800 ± 0.0126
	SC	0.3916 ± 0.0390	0.2001 ± 0.0256	0.3377 ± 0.0045	0.4279 ± 0.0174	**0.4667 ± 0.0198**
	RMSE	**35.32 ± 0.70**	50.65 ± 0.64	51.65 ± 0.74	69.6 ± 1.90	67.5 ± 0.44
	Time	2.1440 ± 0.0769	0.6545 ± 0.0204	2.8410 ± 0.1311	0.0861 ± 0.0075	0.5515 ± 0.0028
Usps	Clusters	5.7 ± 0.94	77.5 ± 4.478	69.3 ± 2.263	10*	10*
	Acc	0.5378 ± 0.0570	0.8464 ± 0.0090	**0.8531 ± 0.0013**	0.6633 ± 0.0309	0.6975 ± 0.0275
	ARI	0.4623 ± 0.0981	0.2941 ± 0.0435	0.2687 ± 0.0022	0.4796 ± 0.0362	**0.5046 ± 0.0239**
	NMI	**0.6283 ± 0.05019**	0.5724 ± 0.0096	0.5712 ± 0.0020	0.5764 ± 0.0211	0.5943 ± 0.0162
	SC	0.1121 ± 0.0686	0.1080 ± 0.0212	0.1614 ± 0.0133	0.2318 ± 0.0170	**0.2413 ± 0.0106**
	RMSE	**4.53 ± 0.04**	5.38 ± 0.02	5.62 ± 0.02	6.14 ± 0.02	6.09 ± 0.01
	Time	6.261 ± 0.290	0.683 ± 0.028	4.759 ± 0.443	0.2267 ± 0.0039	8.899 ± 0.596
HAR	Clusters	4.8 ± 0.9189	79.7 ± 4.322	40.7 ± 2.45	6*	6*
	Acc	0.6038 ± 0.0767	**0.8331 ± 0.0096**	0.7873 ± 0.0158	0.5849 ± 0.0599	0.6064 ± 0.0128
	ARI	**0.4798 ± 0.0873**	0.1169 ± 0.0063	0.2075 ± 0.0110	0.4283 ± 0.0703	0.4505 ± 0.0254
	NMI	**0.6252 ± 0.0599**	0.4743 ± 0.0054	0.4955 ± 0.0091	0.5781 ± 0.0530	0.5996 ± 0.0059
	SC	0.2082 ± 0.0935	0.0321 ± 0.0074	0.0429 ± 0.0069	**0.2712 ± 0.0800**	0.2414 ± 0.0237
	RMSE	**3.63 ± 0.06**	3.71 ± 0.01	4.63 ± 0.02	4.30 ± 0.09	4.22 ± 0.01
	Time	9.573 ± 1.565	1.071 ± 0.045	17.26 ± 0.657	0.408 ± 0.006	19.326 ± 0.714
Yale Face B	Clusters	8 ± 1.63	49.1 ± 3.14	82.4 ± 3.13	10*	10*
	Acc	0.6328 ± 0.1091	0.9128 ± 0.0234	**0.9412 ± 0.0026**	0.6431 ± 0.0659	0.7672 ± 0.0542
	ARI	0.5907 ± 0.1428	0.4331 ± 0.0273	0.2504 ± 0.0177	0.4821 ± 0.0901	**0.6272 ± 0.0732**
	NMI	0.7746 ± 0.0671	0.7317 ± 0.0151	0.6770 ± 0.0045	0.6979 ± 0.0459	**0.7931 ± 0.0383**
	SC	0.1660 ± 0.0696	0.1265 ± 0.0242	0.2189 ± 0.0103	0.2282 ± 0.0379	**0.2656 ± 0.0084**
	RMSE	**781.9 ± 20.65**	1077 ± 14.41	1009 ± 6.92	1243 ± 34.2	1204 ± 3.39
	Time	30.022 ± 3.553	1.562 ± 0.042	9.365 ± 3.317	0.854 ± 0.018	26.052 ± 1.538
MNIST	Clusters	11.4 ± 1.83	91.4 ± 6.45	94 ± 6.55	10*	10*
	Acc	0.6156 ± 0.0742	**0.7907 ± 0.0070**	0.7684 ± 0.0266	0.5490 ± 0.0329	0.5992 ± 0.0081
	ARI	**0.3996 ± 0.0891**	0.1554 ± 0.0099	0.1428 ± 0.0033	0.3419 ± 0.0296	0.3853 ± 0.0060
	NMI	**0.5763 ± 0.0428**	0.5004 ± 0.0038	0.4601 ± 0.0107	0.4665 ± 0.0234	0.4998 ± 0.0114
	RMSE	**1337 ± 9.5**	1436 ± 4.16	1695 ± 4.83	1607 ± 7.50	1608 ± 1.62
	Time	191.9 ± 22.06	14.14 ± 0.24	2215 ± 808.3	5.49 ± 0.076	146.622 ± 1.570
CoverType	Clusters	14.1 ± 1.72	84.6 ± 4.74	34.3 ± 1.15	7*	7*
	Acc	**0.5637 ± 0.0145**	0.5626 ± 0.0042	0.5449 ± 0.0051	0.4917 ± 0.0037	0.4919 ± 0.0024
	ARI	**0.0284 ± 0.0306**	0.0050 ± 0.0003	0.0093 ± 0.0013	0.0035 ± 0.0029	−0.0048 ± 0.0018
	NMI	**0.1461 ± 0.0296**	0.0988 ± 0.0015	0.0969 ± 0.0046	0.0740 ± 0.0041	0.0731 ± 0.0012
	RMSE	**459.5 ± 15.55**	479.4 ± 27.39	535.4 ± 5.50	894 ± 9.95	880.6 ± 0.72
	Time	258 ± 18.76	71.34 ± 3.69	8176 ± 368.8	5.888 ± 0.16	77.364 ± 0.367
KDD99	Clusters	4.6 ± 1.14	36.6 ± 6.18	33 ± 3.46	23*	23*
	Acc	0.9641 ± 0.0189	**0.9747 ± 0.0186**	0.9603 ± 0.0017	0.9538 ± 0.0264	0.7361 ± 0.1126
	ARI	**0.8685 ± 0.1065**	0.6503 ± 0.0052	0.6955 ± 0.0022	0.5280 ± 0.0446	0.4967 ± 0.3807
	NMI	**0.8075 ± 0.0707**	0.6572 ± 0.0036	0.6798 ± 0.0041	0.6144 ± 0.0075	0.4760 ± 0.2991
	RMSE	$9.888 \times 10^5 \pm 11.49$	$9.879 \times 10^5 \pm 625.5$	$9.887 \times 10^5 \pm 10.35$	$9.731 \times 10^5 \pm 5109$	**1880 ± 47.43**
	Time	153.3 ± 10.97	68.77 ± 4.156	7670 ± 258	16.17 ± 0.097	72.307 ± 0.730

nodes in a density based way. However, natural clusters in datasets are not necessarily compact, and density based clustering methods are designed to find such clusters. Silhouette coefficient, as well as most internal validation measures, emphasize intra-class similarity and inter-class difference, therefore these measures naturally favor clustering algorithms that learn compact clusters, which are usually classified as "centroid-based clustering". In contrast, density based clustering algorithms emphasize density-connectivity in clusters and density-separation between clusters. The shape of clusters is usually arbitrary, therefore these methods tend to present bad values for silhouette coefficient.

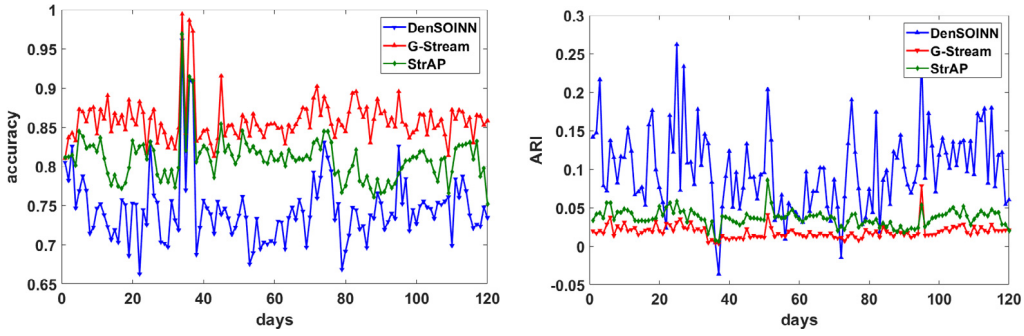
By viewing the numbers of clusters reported by each algorithm in Table 2, we can see that the numbers of clusters DenSOINN are close to the numbers of underlying classes in most experiments, but it is difficult to approximate the number of underlying classes in unbalanced datasets like KDD99 and CoverType. In contrast, the number of clusters discovered by G-Stream and StrAP are too large. Although the number of classes is not necessarily equal to underlying clusters, it can still be used as a reference.

Comparing the execution time of stream clustering algorithms, online K-Means is the most efficient because of its low time complexity. DenSOINN and G-Stream are less efficient than online K-Means, because the numbers of nodes in both networks are higher than the number of clusters k of K-Means. G-Stream is much more efficient than DenSOINN on small datasets, but this advantage is not obvious on large datasets like KDD99 and CoverType. Because the size of DenSOINN grows fast at the beginning but remains stable afterwards, while the size of G-Stream grows approximately linearly with the number of input patterns. Based on current parameter setting, the number of neural nodes of G-Stream is smaller than that of DenSOINN. StrAP is slower than other methods, especially on large and high-dimensional datasets. StreamKm++ is also



(a) Accuracy (b) ARI

Fig. 8. Periodical results of DenSOINN, G-Stream and StrAP on KDD99 data stream. The left figure compares the clustering accuracy, and the right figure compares the ARI.



(a) Accuracy (b) ARI

Fig. 9. Daily results of DenSOINN, G-Stream and StrAP on URL Reputation data stream. The left figure compares the clustering accuracy, and the right figure compares the ARI.

faster than DenSOINN in most datasets, but it is implemented in another platform. Considering the characteristics of the algorithm, C++ is much more efficient than Matlab in this situation.

5.4. Experiments on evolving data streams

In this subsection, we introduce three experiments on evolving data streams and demonstrate the periodical results during the clustering procedures. DenSOINN was compared with G-Stream and StrAP in these experiments.

The first experiment was performed on the KDD99 dataset with its original input sequence. The data stream is separated into 500 periods with 10,000 patterns in each period. After each period, clustering requests are given and the clustering qualities are evaluated using the data learned in the last period. Both DenSOINN and G-Stream have a parameter to control the maximum number of nodes, which was set as 1000 in this experiment. We use accuracy and ARI to evaluate clustering quality in this experiment. The comparison of accuracy and ARI over time is shown in Fig. 8.

It can be seen that the accuracy and ARI of DenSOINN are better than those of G-Stream and StrAP. The KDD99 data stream only contains data from a single class in most periods. Therefore the accuracies in all results are 1, and the ARIs in all results are 0 in these periods. In the periods when the data stream has multiple classes, the accuracy would drop and the ARI would raise. It is observable that both the accuracy and ARI of DenSOINN are better than comparative methods most of the time.

The second experiment was performed on the URL Reputation dataset (Ma, Saul, Savage, & Voelker, 2009). The dataset is a 120-day subset of the ICML-09 URL data containing 2.4 million examples and 3.2 million features. We used the 64 numeric features in this experiment. Clustering requests are given after learning the

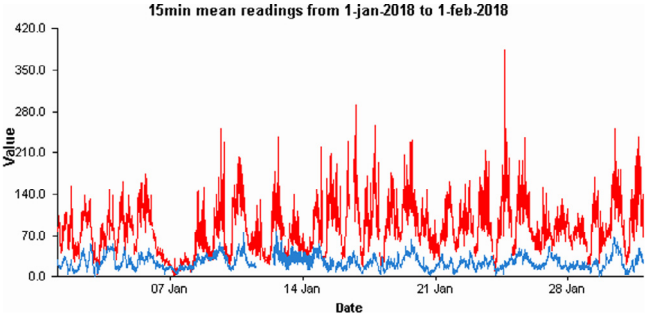
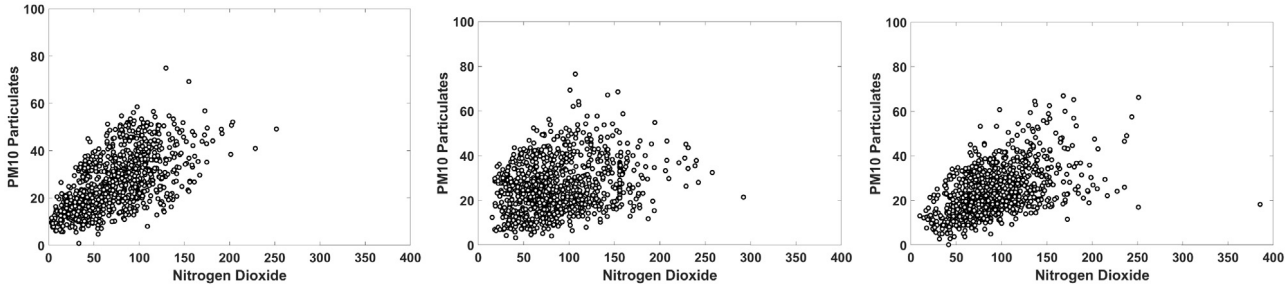


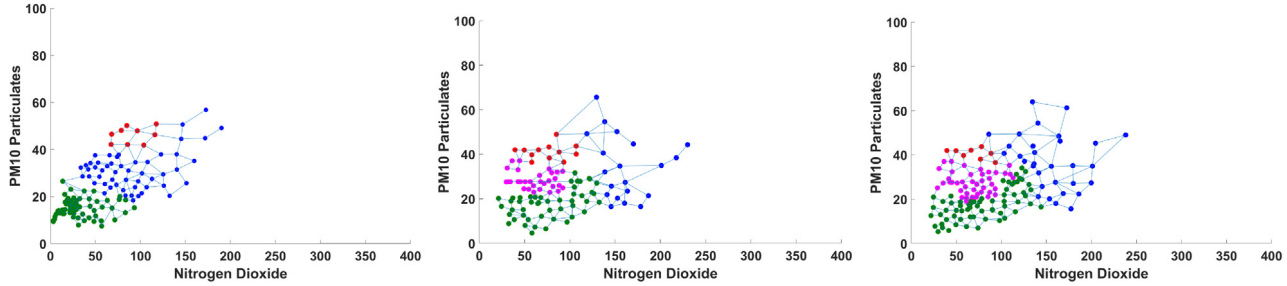
Fig. 10. London Air Quality monitoring data. The red line illustrates the amount of Nitrogen Dioxide, and the blue line illustrates the amount of PM10 Particulate. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

data of each day, and then the clustering qualities are evaluated. Same as the previous experiment, the maximum capacities of DenSOINN and G-Stream are also limited to 1000, and we still use accuracy and ARI to evaluate clustering quality. The comparison of accuracy and ARI of each day is shown in Fig. 9. In this experiment, DenSOINN has a worse accuracy comparing to G-Stream and StrAP, while its ARI is the best. The reason for this phenomenon is that the dataset only contains two classes, but both G-Stream and StrAP report too many clusters. In contrast, DenSOINN only reports several clusters most of the time, therefore its accuracy is not high, but reported data partition is more similar with the underlying classes.

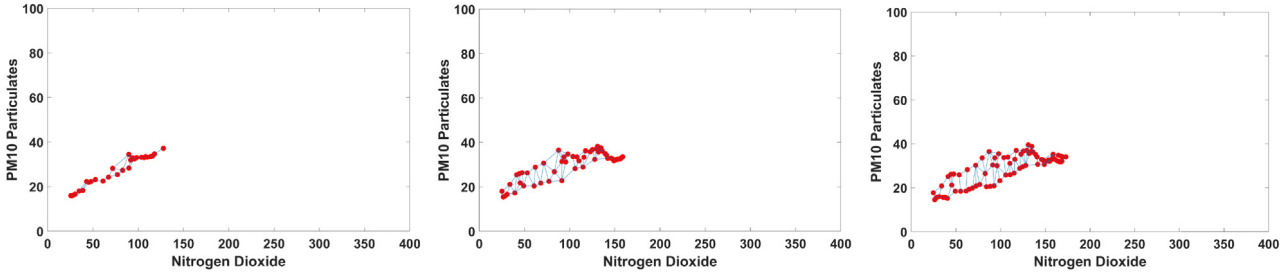
The third experiment was carried out using the London Air Quality data (K.C.L. Environmental Research Group, 2015). We



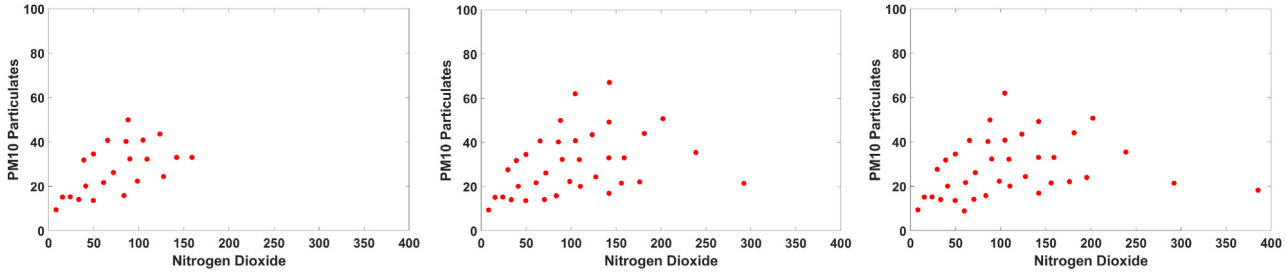
(a) Input data points



(b) DenSOINN results, $SC = (0.1537, 0.0882, -0.1389)$, $RMSE = (4.34, 5.09, 6.07)$



(c) G-Stream results, $SC = (-0.0207, 0.1372, 0.1432)$, $RMSE = (13.44, 13.59, 11.77)$



(d) StrAP results, $SC = (0.4915, 0.4494, 0.4370)$, $RMSE = (7.37, 9.53, 6.28)$

Fig. 11. Training data and clustering results of London Air Quality. The three columns show data and learning results of each period respectively.

obtained the monitoring data of Westminster - Marylebone Road site from the web, starting from 1-Jan-2018 to 1-Feb-2018. The air quality was captured once per 15 min, hence the dataset contains 2976 samples. The amounts of Nitrogen Dioxide ($\mu g/m^3$) and PM10 Particulates ($\mu g/m^3$) are used as data features, as plotted in Fig. 10. We separate the dataset into three subsets, each containing data collected in about 10 days. In this experiment the parameters of G-Stream and StrAP are changed because the dataset is small and low-dimensional. The interval of node creation in G-Stream is set as 100, and the restarting criteria in StrAP is set as *MaxCache* = 300. The data points and results are illustrated in Fig. 11. The dataset is unlabeled, therefore we use SC and RMSE to evaluate clustering quality.

From the figures, it is observable that the network structure of DenSOINN forms a good summarization of input data. DenSOINN reports three clusters in the first period, and four clusters in the other periods. In contrast, the nodes of G-Stream distribute in a small area near the center of input data. The exemplars of StrAP also form a good data summarization, but the number of exemplars is quite large. The silhouette coefficient of DenSOINN is still worse than those of G-Stream and StrAP, but its RMSE is the smallest.

Table 3
DenSOINN results with different parameter settings.

Dataset		*StreamSpeed* $= 200, \epsilon = 0.25$			*StreamSpeed* $= 200, \epsilon = 0.5$			*StreamSpeed* $= 100, \epsilon = 1$		
		$\gamma = 0.25$	$\gamma = 0.5$	$\gamma = 1$	$\gamma = 0.25$	$\gamma = 0.5$	$\gamma = 1$	$\gamma = 0.25$	$\gamma = 0.5$	$\gamma = 1$
Segment	Clusters	27	27.8	31.6	6.6	7.2	9	3.2	4.4	5.4
	Acc	0.8377	0.8390	0.8449	0.6518	0.5930	0.6115	0.4319	0.5374	0.5377
	ARI	0.3464	0.3254	0.3084	0.4731	0.3900	0.3982	0.2648	0.3663	0.3561
	NMI	0.6142	0.6083	0.6031	0.6341	0.5993	0.5996	0.5024	0.6049	0.5762
	Neural nodes	770.2	786.4	818	149	162	162.6	61.6	75.6	73.2
Pendigits	Clusters	17.5	19	18.8	9.3	12.6	12	8	7.6	8.3
	Acc	0.8302	0.8214	0.8531	0.7618	0.8184	0.8398	0.684	0.6655	0.6709
	ARI	0.5995	0.5728	0.6149	0.6375	0.6532	0.6832	0.5012	0.4903	0.5484
	NMI	0.7444	0.7413	0.7649	0.7697	0.7877	0.7906	0.7301	0.7157	0.7156
	Neural nodes	801.7	808.3	825.3	244.3	256	278.7	81.67	77.33	69.33
Usps	Clusters	17.6	17	16.8	6.4	6	6.8	2.8	3.2	3.8
	Acc	0.7785	0.7727	0.7620	0.5661	0.5662	0.5673	0.3681	0.4052	0.4336
	ARI	0.4876	0.4965	0.4968	0.4426	0.4390	0.3871	0.2309	0.2918	0.2776
	NMI	0.6923	0.6838	0.6914	0.6636	0.6467	0.6358	0.4571	0.5110	0.4882
	Neural nodes	743.5	729.1	732.6	223.5	232	238	85	82.6	77.2
HAR	Clusters	16.6	16.2	18.2	6.4	6.2	4.4	2.2	2.4	2
	Acc	0.7717	0.7649	0.7721	0.6579	0.6566	0.5734	0.3600	0.4259	0.3545
	ARI	0.3892	0.3879	0.3975	0.4878	0.4782	0.4497	0.3279	0.3941	0.3299
	NMI	0.5819	0.5832	0.5895	0.6116	0.6072	0.6030	0.5389	0.6057	0.5468
	Neural nodes	430.3	424.6	400.2	93	104.7	69.3	29.4	25	19.2
Yale Face B	Clusters	19.8	20.4	23.2	8.4	9.6	10.8	6.2	6.2	7
	Acc	0.8750	0.8486	0.8753	0.6670	0.7333	0.7393	0.4549	0.4643	0.5416
	ARI	0.6021	0.5732	0.5925	0.5797	0.6459	0.6351	0.2935	0.2681	0.3762
	NMI	0.8011	0.7905	0.7971	0.7992	0.8347	0.8356	0.6152	0.6415	0.7056
	Neural nodes	891.1	893.6	889.9	217.5	216.6	222.8	90.1	97.4	86.4
MNIST	Clusters	13.8	15.6	13.8	5.4	6.4	4.4	1.2	2.2	2.6
	Acc	0.5329	0.5974	0.6134	0.3612	0.3776	0.3631	0.1321	0.2096	0.2494
	ARI	0.3248	0.3741	0.4048	0.2063	0.2111	0.1844	0.0100	0.0619	0.0817
	NMI	0.5572	0.5910	0.5994	0.4173	0.4321	0.4115	0.04439	0.1743	0.2704
	Neural nodes	615.4	593.2	580.2	237.8	229.6	186.6	64.4	58.6	40.8
CoverType	Clusters	37.6	39.2	44.6	13.4	15.2	18.2	2	2.4	3.6
	Acc	0.6284	0.6320	0.6378	0.5621	0.5766	0.5946	0.4898	0.501	0.5036
	ARI	0.0404	0.0398	0.0422	0.0178	0.0300	0.0452	0.0121	0.0114	0.0003
	NMI	0.1787	0.1794	0.1832	0.1448	0.1488	0.1573	0.0381	0.0438	0.0446
	Neural nodes	725.2	827.6	784	218.2	214.2	196	47.2	39.4	20.8
KDD99	Clusters	19.8	20.4	23.2	8.4	9.6	10.8	6.2	6.2	7
	Acc	0.9751	0.9666	0.9701	0.9716	0.9691	0.8333	0.8206	0.7391	0.7244
	ARI	0.8223	0.7515	0.7145	0.7705	0.8160	0.6456	0.7669	0.5720	0.3965
	NMI	0.7420	0.7134	0.7110	0.7637	0.7812	0.6385	0.7441	0.5370	0.3869
	Neural nodes	93	85	58.2	51.6	35	8.9	15.3	9	4

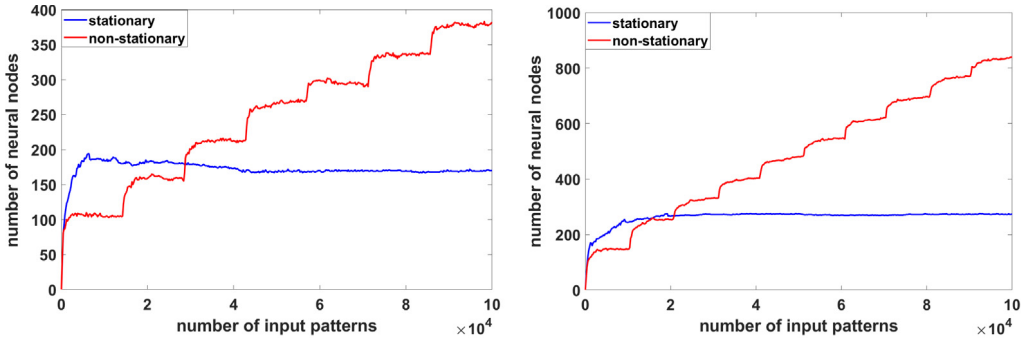
5.5. Parameter setting and sensitivity analysis

We will experimentally analyse the effect of parameters in this subsection. There are totally five parameters in DenSOINN, which can be grouped into three groups according to their effects.

The first group consists of γ, *StreamSpeed* and ϵ. *StreamSpeed* and ϵ control the frequency and intensity of denoising respectively. γ affects survival time of the edges that are not recently updated, which further affects the deletion of nodes during denoising. These parameters together affect the number of nodes which is decisive to execution time and memory usage as discussed in Section 4.3. The recommended ranges of these parameters are $\gamma \in [0.1, 1]$, *StreamSpeed* $\in [50, 200]$ and $\epsilon \in [0.2, 1]$ respectively. Setting a small γ would provide better topology preserving ability for DenSOINN, which works fine on datasets with well separated clusters and little noise, like the artificial datasets used in Section 5.2. However, real world datasets usually contain overlapped clusters and noisy data, therefore it is necessary to remove outdated edges in time. Setting a larger *StreamSpeed* would lower the frequency of denoising, but if *StreamSpeed* is too large or ϵ is too small, the network would contain more nodes than necessary, therefore higher *StreamSpeed* does not lead to faster processing. Meanwhile, if *StreamSpeed* is too small or ϵ is too large, then too many nodes would be deleted and the network structure would become unstable. In Section 5.3 we set *StreamSpeed* $= 200, \epsilon = 0.5$ as default parameters because they work well on most real world datasets, yet adjusting these parameters with respect to characteristic of input data could improve clustering quality or time efficiency. We recommend setting *StreamSpeed* $= 200$ and $\epsilon = 0.25$ when clustering small datasets with hundreds of patterns. If the input dataset is very large and the running time based on default parameters is unacceptable, setting *StreamSpeed* $= 100$ and $\epsilon_1 = 1$ could greatly reduce the number of nodes and accelerate training speed.

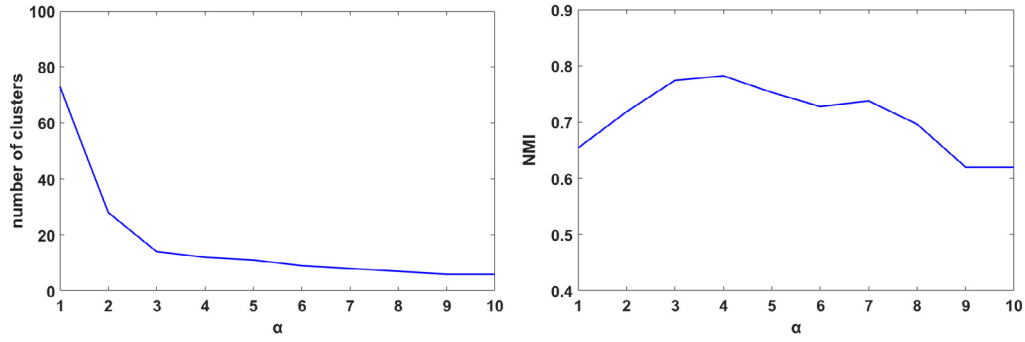
We performed experiments on real world datasets under different settings of γ, *StreamSpeed* and ϵ. We chose three groups of *StreamSpeed* and ϵ, corresponding to three levels of denoising. In each group of *StreamSpeed* and ϵ, we tried selecting γ from 0.25, 0.5 and 1 respectively. We set *MaxNode* $= 1000$ and $\alpha = 5$ in all these experiments. In each experiment, DenSOINN was repeatedly run 10 times using each parameter setting on a given dataset, and the average results are reported in Table 3.

It can be observed that setting *StreamSpeed* $= 200, \epsilon = 0.5$ led to best results in most experiments, except on MNIST where the data is not sufficiently separated. Setting $\epsilon = 0.25$ could result in too many nodes and a much longer training time, while its results are significantly worse than the default parameters. Setting *StreamSpeed* $= 100$ and $\epsilon = 1$ could greatly reduce the number of nodes and training time, at the cost of clustering quality. Changing γ only slightly affects the result, but in most comparisons setting $\gamma = 0.25$ results in less neural nodes and better clustering quality.

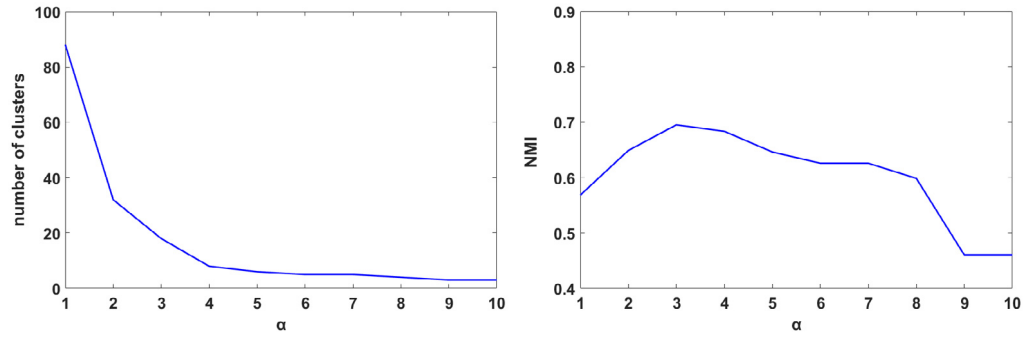


(a) Network growth on Segment (b) Network growth on Pendigits

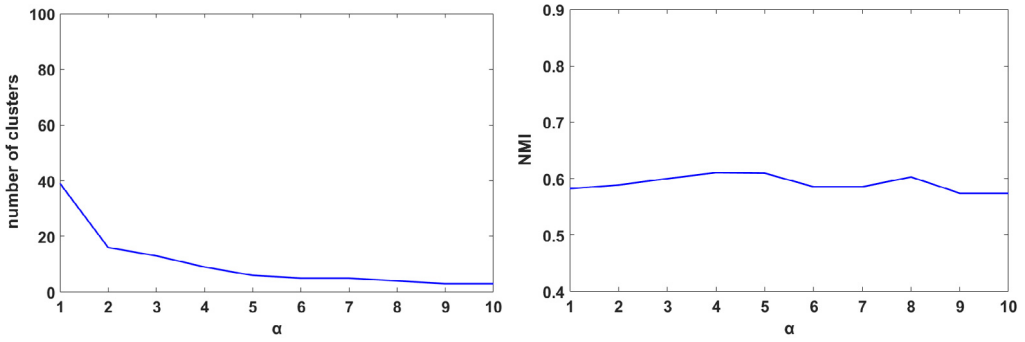
Fig. 12. Number of neural nodes versus number of input patterns on Segment and Pendigits. Blue lines illustrate network growth in stationary environments. Red lines illustrate network growth in non-stationary environments. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



(a) Pendigits



(b) Usps



(c) Segment

Fig. 13. Results under different settings of α. The left column illustrates resulting number of clusters versus α, and the right column illustrates NMI versus α.

Another observable phenomenon in these experiments is that the number of nodes in DenSOINN does not depend on the size of training dataset. For example, the numbers of nodes resulting from learning KDD99 is much smaller than any other datasets. In fact, DenSOINN is able to generate a suitable number of nodes to represent data distribution in a short time, and the network size remains stable unless data distribution changes. However, it is sensitive to change of data distribution. We ran another experiment to illustrate the procedure of network growth on Pendigits and Segment in stationary and non-stationary environments. In this experiment, we randomly sample 100,000 patterns from the original dataset. For non-stationary environments, the input sequence of patterns is sorted according to class label. In this experiment the parameters were set as $\gamma = 0.25$, $StreamSpeed = 200$, $\epsilon = 0.5$, and $\alpha = 5$, and the network capacity is not limited. The growth of network size is illustrated in Fig. 12.

It can be seen in Fig. 12 that the number of neural nodes grows fast at the beginning and remains stable afterwards in stationary environments. In non-stationary environments, the network grows larger each time a new class of patterns is input. The proposed method aims to solve the data stream clustering problem as an incremental learning problem, where old data and new data are considered equally important, therefore it is designed to learn new knowledge while keeping old structures. However, in extreme conditions where data distribution keeps changing all the time, the network would keep adding new nodes and its size would become very large. Therefore we use a parameter *MaxNodes* to control the maximum capacity of the network, in case that the input data stream is extremely non-stationary and the computational cost is unacceptable. But even if we do not limit the network capacity, the number of nodes would not grow out of control unless in such extreme conditions.

The last parameter α controls the number of cluster centers in each connected subgraph of the network. The reasonable range of α is $\alpha \in [2, 10]$. α has a critical effect on the clustering result, but its value is not difficult to set because setting $\alpha \in [3, 6]$ works fine on most real world datasets. We fix α at $\alpha = 5$ when clustering real world datasets, and the resulting number of clusters is usually close to the number of classes. We performed experiments on Pendigit, Usps and Segment under different settings of α. We recorded the number of clusters in each experiment, and evaluated the clustering quality by NMI. The results are illustrated in Fig. 13.

The number of clusters decreases as α goes up until reaches a minimum, which is the number of connected components in the network. It can be seen in the right column that setting $\alpha \in [3, 6]$ could usually result in acceptable clustering quality.

6. Conclusion

In this paper, we proposed a new competitive neural network named Density Based Self Organizing Incremental Neural Network(DenSOINN). This method aims at finding underlying clusters in data when the data is organized in the form of a stream. DenSOINN also applied Competitive Hebbian Learning in order to learn a topological expression of input data. By adopting a self-adaptive distance metric, DenSOINN can be applied on a raw dataset as if it has been normalized, which solves the data normalization problem in data stream clustering tasks. By combining traditional density connectivity based clustering and finding density peaks, DenSOINN can separate highly overlapped clusters while finding a suitable number of arbitrary shaped clusters.

However, there are still further works left to be done. The quality of clustering lacks a theoretical analysis, mainly because the density based clustering algorithm is not designed based on optimizing a certain loss function like K-Means based algorithms. The experimental results on real world datasets indicate that the clustering quality of DenSOINN is more sensitive to the input sequence of patterns than K-Means based algorithms. Moreover, there are many parameters in the model and the optimal values of parameters are not easy to set. We will make efforts to improve the stability of DenSOINN in future, and we try to provide an automatic method to find the optimal parameters in the model.

Acknowledgments

This work is supported in part by the National Science Foundation of China under Grant No. (61876076), and Jiangsu NSF, China grant (BK20141319).

References

Ackermann, M. R., Raupach, C., Swierkot, K., Lammersen, C., & Sohler, C. (2012). StreamKM++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics*, *17*, 2.4.

Aggarwal, C. C., Han, J., Yu, P. S., & Yu, P. S. (2003). A framework for clustering evolving data streams. In *VLDB* (pp. 81–92).

Aksoy, S., & Haralick, R. M. (2001). Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern Recognition Letters*, *22*(5), 563–582.

Ankerst, M., Breunig, M. M., Kriegel, H. P., & Sander, J. (1999). OPTICS: ordering points to identify the clustering structure. *Acm Sigmod Record*, *28*(2), 49–60.

Arthur, D., & Vassilvitskii, S. (2007). k-means++:the advantages of careful seeding. In *Eighteenth Acm-Siam symposium on discrete algorithms* (pp. 1027–1035).

Bouguelia, M. R., Belaïd, Y., & Belaïd, A. (2013). An adaptive incremental clustering method based on the growing neural gas algorithm. In *International conference on pattern recognition applications and methods* (pp. 42–49).

Cao, F., Ester, M., Qian, W., & Zhou, A. (2006). Density-Based clustering over an evolving data stream with noise. In *Siam international conference on data mining* (pp. 328–339).

Chen, J. Y., & He, H. H. (2016). *A fast density-based data stream clustering algorithm with cluster centers self-determined for mixed data* (pp. 271–293). Elsevier Science Inc.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*(1), 21–27.

Davis, J. V., Kulis, B., Jain, P., Sra, S., & Dhillon, I. S. (2007). Information-theoretic metric learning. In *International conference on machine learning* (pp. 209–216).

Ding, S., Wu, F., Qian, J., Jia, H., & Jin, F. (2015). Research on data stream clustering algorithms. *Artificial Intelligence Review*, *43*(4), 593–600.

Ester, M., Kriegel, H. P., & Xu, X. (1996). A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *International conference knowledge discovery and data mining* (pp. 226–231).

Fred, A. L. N., & Jain, A. K. (2003). Robust data clustering. In Computer vision and pattern recognition, 2003. Proceedings. 2003 IEEE computer society conference on, vol. 2, pp. II–128–II–133.

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(5814), 972–976.

Furao, S., Ogura, T., & Hasegawa, O. (2007). An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks the Official Journal of the International Neural Network Society*, *20*(8), 893.

Ghesmoune, M., Lebbah, M., & Azzag, H. (2016). A new growing neural gas for clustering data streams. *Neural Networks the Official Journal of the International Neural Network Society*, *78*(3–4), 36–50.

Hinneburg, A., & Keim, D. A. (1998). An efficient approach to clustering in large multimedia databases with noise. In *International conference on knowledge discovery and data mining* (pp. 58–65).

Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, *2*(1), 193–218.

Hyde, R., Angelov, P., & Mackenzie, A. R. (2017). Fully online clustering of evolving data streams into arbitrarily shaped clusters. *Information Sciences*, *382-383*, 96–114.

Jain, A. K., & Dubes, R. C. (1988). Algorithms for clustering data. *Technometrics*, *32*(2), 227–229.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). *Data clustering: A review* (pp. 264–323). ACM.

K.C.L. Environmental Research Group London air quality network :: Welcome to the london air quality network data. (2015). http://www.londonair.org.uk/london/asp/datadownload.asp.

Khan, I., Huang, J. Z., & Ivanov, K. (2016). Incremental density-based ensemble clustering over evolving data streams. *Neurocomputing*, *191*, 34–43.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43*(1), 59–69.

Lee, R. C. T. (1981). *Clustering analysis and its applications* (pp. 169–292). Springer US.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*(2), 129–137.

Ma, J., Saul, L. K., Savage, S., & Voelker, G. M. (2009). Identifying suspicious URLs: an application of large-scale online learning. In *International Conference on Machine Learning* (pp. 681–688).

Ma, A., Zhong, Y., & Zhang, L. (2015). Adaptive multiobjective memetic fuzzy clustering algorithm for remote sensing imagery. *IEEE Transactions on Geoscience & Remote Sensing*, *53*(8), 4202–4217.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. of berkeley symposium on mathematical statistics and probability* (pp. 281–297).

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, *4*(4), 558–569.

Rodriguez, A., & Laio, A. (2014). Machine Learning. Clustering by fast search and find of density peaks. *Science*, *344*(6191), 1492.

Shen, F., & Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Netw*, *19*(1), 90–106.

Shen, F., & Hasegawa, O. (2008). A fast nearest neighbor classifier based on self-organizing incremental neural network. *Neural Networks the Official Journal of the International Neural Network Society*, *21*(10), 1537.

Silva, J. A., Faria, E. R., Barros, R. C., & Hruschka, E. R. (2013). Data stream clustering: A survey. *Acm Computing Surveys*, *46*(1), 13.

Srivastava, A., & Sahami, M. (2009). *Text mining: classification, clustering, and applications*.

Weinberger, K. Q., & Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research (JMLR)*, *10*(1), 207–244.

Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2002). Distance metric learning, with application to clustering with side-information. In *International conference on neural information processing systems* (pp. 521–528).

Xu, B., Shen, F., & Zhao, J. (2016). Density based self organizing incremental neural network for data stream clustering. In *International joint conference on neural networks* (pp. 2654–2661).

Zhang, X., Furtlehner, C., Germain-Renaud, C., & Sebag, M. (2014). Data stream clustering with affinity propagation. *IEEE Transactions on Knowledge & Data Engineering*, *26*(7), 1644–1656.

Zhang, T., Ramakrishnan, R., & Livny, M. (1997). BIRCH: A new data clustering algorithm and its applications. *Data Mining & Knowledge Discovery*, *1*(2), 141–182.

Zhang, H., Xiao, X., & Hasegawa, O. (2014). A load-balancing self-organizing incremental neural network. *IEEE Transactions on Neural Networks & Learning Systems*, *25*(6), 1096–1105.

Zhong, Y., Ma, A., Ong, Y. S., Zhu, Z., & Zhang, L. (2018). Computational intelligence in optical remote sensing image processing. *Applied Soft Computing*, *64*, 75–93.