Estimating data stream tendencies to adapt clustering parameters

Marcelo Keese Albertini*

School of Computer Science,
Federal University of Uberlandia,
Uberlandia, MG, Brazil
Email: albertini@ufu.br
*Corresponding author

Rodrigo Fernandes de Mello

Institute of Mathematics and Computer Science,
University of Sao Paulo,
Sao Carlos, SP, Brazil
Email: mello@icmc.usp.br

Abstract: A wide-range of applications based on processing of data streams have emerged in the last decade. They require specialised techniques to obtain representative models and extract information. Traditional data clustering algorithms have been adapted to include continuously arriving data by updating the current model. Most of data stream clustering algorithms aggregate new data into models according to parameters usually set by users. Problems arise when choosing the values of given parameters. When the phenomenon under study is stable, an analysis of a sample of the data stream or a priori knowledge can be used. However, when the behaviour changes over collection, parameters become obsolete and, consequently, the performance is degraded. In this paper, we study the problem of how to automatically adapt control parameters of data stream clustering algorithms. In this sense, we introduce a novel approach to estimate and use data tendencies in order to automatically modify control parameters. We present a proof of the convergence of our approach towards an ideal and unknown value of the control parameter. Experimental results confirm the estimation of data tendency improves learning control parameterisation.

Keywords: big data; data clustering; data stream; data sequence; adaptive clustering; data analysis.


Biographical notes: Marcelo Keese Albertini received his PhD in Computer Science from the University of Sao Paulo. He is an Assistant Professor at the School of Computer Science, Federal University of Uberlandia, Brazil. His research interests include issues related to big data, machine learning, data mining, and efficient data structures.

Rodrigo Fernandes de Mello received his MSc from the Federal University of Sao Carlos (UFSCAR), Brazil in 1999 and PhD in 2003 from the University of Sao Paulo (USP), Brazil. He is currently an Associate Professor at the Institute of Mathematics and Computer Sciences, University of Sao Paulo, Sao Carlos, Brazil. His current research interests include machine learning, dynamical systems and data streams.

1 Introduction

Clustering is an unsupervised learning approach to extract structural information from data (Jain et al., 1999; Jain, 2010). For the last decades, this approach has been applied on several application domains, specially on static datasets. Recently, researchers have been paying special attention to obtain structural information from dynamic datasets which are commonly referred to as data streams (Pavlidis et al., 2011). In this latter type of dataset, new data arrive over time and the clustering algorithm needs to be capable of adapting the data representation model. Data streams are produced by different applications ranging from social
networks, search engines, remote sensing and other scientific applications. They are characterised by the continuous production of data at high rates and volumes, which requires data-stream clustering algorithms to quickly process and dispose data while the representation model is updated.

In this manner, the goal of this clustering approach is to model a data stream \( S = \{s_{\infty}, ..., s_k, ..., s_n\} \), representing variations that tend to occur over data collection, in which \( s_k \) is an example at index \( k \) (Sun and Lu, 2006). We consider a data stream is a sequence of examples and no further knowledge is assumed such as its probability distribution function. In order to represent variations, the data model has to be updated according to information extracted from data. This data representation model is organised as a partition in which examples are separated into subsets according to their similarity.

In the context of data-stream clustering, a sequence of partitions is produced as long as examples are received. For instance, consider we have collected data \( s_i \) up to the one at index \( k \) forming stream \( S_k = \{s_{\infty}, ..., s_k\} \), then \( S_k \) is submitted to the clustering algorithm which is responsible for producing a partition \( \mathcal{P}_k \). Consider a partition \( \mathcal{P}_k \) previously obtained, now let the algorithm receive \( S_l \) for \( l > k \), and process the examples at indexes greater than \( k \) to produce \( \mathcal{P}_l \). By taking only the newer data into account, this algorithm saves processing time which makes learning feasible for infinite sequences. This learning modality is referred to as incremental learning.

Incremental learning has the ability to update partitions in order to represent current structural information. The data-stream clustering algorithm is responsible for such partition adaptation, which relies on the algorithm design and on its learning control parameters. Traditionally, learning control parameters are defined by specialists after an initial data analysis using statistical tools or simply by assessing different combinations, which is indeed a good starting point. However, as new examples are collected, data structural information may change and those parameters may not be as effective as before to produce representative partitions, requiring adaptation (Brazdil, 2009; Brazdil et al., 2009).

Once deployed, current data-stream clustering algorithms do not adapt learning control parameters, which eventually become obsolete and, consequently, data structural information is lost. This scenario motivates this paper which proposes and evaluates two automatic approaches to adapt learning control parameters in an attempt to produce more representative partitions.

These approaches are based on computing input data tendencies and consider such information to adapt control parameters. In order to assess our approaches, we selected the following set of clustering algorithms (Daszykowski et al., 2002): growing neural gas (Fritzke, 1995a), neural gas with competitive Hebbian learning (Martinetz and Schulten, 1991), growing grid (Fritzke, 1995b) and self-organising map (SOM) (Kohonen, 1998). The criterion for selection was the algorithm to have learning control parameters.

Other algorithms considered cluster features to update partitions and do not rely on learning control parameters, such as Clustream (Aggarwal et al., 2003) and BIRCH (Zhang et al., 1996).

Experiments were performed to measure the model representativeness for new data. Results confirm our approaches improve or maintain model representativeness for most cases. The variance-based approach outperformed Tukey’s line for scenarios under more uncertainty. We could observe the automatic adaptation of the learning control parameters provides lower sum-of-squared errors (SSEs) for clustering partitions.

The remainder of this paper is organised as follows: Section 2 describes the proposed approaches to automatically adapt learning control parameters; Section 3 introduces the algorithms considered to evaluate the proposed approaches; Section 4 details the process of automatic adaptation of learning control parameters; Section 5 analyses the convergence capability of the proposed approaches; Section 6 presents an experimental evaluation of the proposed approaches; Section 7 discusses experimental results and draws conclusions.

## 2 Automatic approaches for adapting learning control parameters

The learning parameters of data-stream clustering algorithms are usually defined after a first data analysis using statistical tools or simply by assessing different combinations. This approach is useful to define the initial algorithm bias, however as new examples are collected, parameters may need to be adapted to guarantee the quality of further partitions, otherwise those models will not represent data structural information. This issue motivated the proposal of two approaches to assess data tendencies in order to adapt learning control parameters to improve clustering partitions. Both approaches are detailed next.

### 2.1 Tukey-line-based approach

The first approach is based on Tukey’s line, a robust method to obtain the slope of a line (Tukey, 1977). In order to exemplify this method, consider nine examples which form points in a \( \mathbb{R}^2 \) Euclidean space. This approach computes quantiles for the \( x \) dimension to define regions of space which make the estimation of data tendency robust. Figure 1 shows an example of quantiles.

Those examples are processed by Algorithm 1, in which \( x \) and \( y \) correspond to pairs in the two dimensional space and \( n \) is the length of such pairs (nine in this situation). As first step, the algorithm copies data from \( x \) to \( z \) and sorts this last vector.
Figure 1  Tukey’s line: an example of the quantiles for the $x$ dimension and resultant variables

Algorithm 1  Pseudocode for the Tukey-line-based approach: 

getSlope($x, y, n$)

Require: $x, y, n$
Ensure: $slope$
1  $z \leftarrow x$
2  sort($z$)
3  $x_\text{b} \leftarrow \text{quantilePoint}(z, n, 1/6)$
4  $x_1 \leftarrow \text{quantilePoint}(z, n, 2/6)$
5  $x_2 \leftarrow \text{quantilePoint}(z, n, 4/6)$
6  $x_t \leftarrow \text{quantilePoint}(z, n, 5/6)$
7  aux $\leftarrow$ empty list
8  for $i \leftarrow 1$ to $n$
9    if $x[i] \leq x_1$ then
10      add(aux, $y[i]$)  \{Append $y[i]$ into aux.\}
11  end if
12 end for
13 sort(aux)
14 $y_{\leq} \leftarrow \text{quantilePoint}(aux, \text{length}(aux), 0.5)$
15 aux $\leftarrow$ empty list
16 for $i \leftarrow 1$ to $n$
17    if $x[i] \geq x_2$ then
18      add(aux, $y[i]$)  \{Append $y[i]$ into aux.\}
19  end if
20 end for
21 sort(aux)
22 $y_{\geq} \leftarrow \text{quantilePoint}(aux, \text{length}(aux), 0.5)$
23 slope $\leftarrow (\bar{y}_{\leq} - \bar{y}_{\geq}) / (x_t - x_b)$
24 return slope

Algorithm 2 quantilePoint($data, nelements, fraction$)

Require: $data, nelements, fraction$
Ensure: $value$
1  lower $\leftarrow \lfloor nelements \cdot fraction \rfloor$
2  upper $\leftarrow \lfloor nelements \cdot fraction \rfloor$
3  value $\leftarrow 0.5 \cdot (data[lower] + data[upper])$
4  return value

The algorithm then builds an auxiliary vector $y \leq x_1$ with all $y$ values in which the corresponding $x$ is less than or equal to $x_1$ (see left-shaded area in Figure 2). The algorithm computes the median for vector $y \leq x_1$ which is referred to as $\bar{y}_{\leq}$. This step results in pair $(x_b, \bar{y}_{\leq})$ which defines the first point to compose the Tukey’s line.

Next, the algorithm builds another auxiliary vector $y \geq x_2$ having all $y$ values for which the corresponding $x$ is greater than or equal to $x_2$ (see right-shaded area in Figure 2). The algorithm computes the median for $y \geq x_2$ in a similar manner. This step results in pair $(x_t, \bar{y}_{\geq})$ which defines the second point to trace the Tukey’s line. Finally, the slope of the line crossing points $(x_b, \bar{y}_{\leq})$ and $(x_t, \bar{y}_{\geq})$ is an estimator for the tendency of a data stream.

Algorithm 3 presents the next step in which the slope value is used to adapt the learning control parameters. This algorithm periodically computes the slope of Tukey’s line for the last $k$ inputs collected from a data stream. Consequently, it obtains a sequence of slopes as data arrives. Then, it regularises the Tukey’s line slope relatively to the maximum slope obtained over data collection. This regularisation is used to produce a value relative to the maximum slope obtained so far.

Next a hyperbolic tangent rescales the regularised slope in interval $(0, 1)$, since learning control parameters are in such range. Afterwards, the maximum slope is evaluated in terms of the newest slope, which is used in future regularisations. Finally, we apply function $F$ [see equation (1)] to produce $\beta_{k+1} \in (0, 1)$, which results in

Next step is to compute the quantile points (Algorithm 2) considering the division of the $x$ dimension into six parts which results in median $\bar{x}$ plus the following variables: $x_b, x_1, x_2$ and $x_t$, as illustrated in Figure 1. Looking at the $x$ axis, 1/6 of data is in $[0, x_b]$, additional 1/6 of data in $[x_b, x_1]$ and so on.
the necessary adaptation for parameters. Terms in function $F$ correspond to:

1. $\beta_1 \in (0, 1)$ represents the last adaptation applied on control parameters.
2. $r \in (0, 1)$ is the regularised slope.

Function $F$ outputs $\beta_{i+1}$ using $\beta_i$ as reference point and $r$ as an impulse relative to $\beta_i$, consequently this function weighs previous learning tendencies to define the next.

$$F(\beta_i, r) = \frac{1}{2} + \frac{1}{2} \cdot \tanh(\beta_i - r + \text{atanh}(2 \cdot r - 1)) \quad (1)$$

**Algorithm 3** Pseudocode to adapt the learning control parameters: adapt($\beta_i$, maxSlope, $x$, $y$, $n$)

- **Require:** $\beta_i$, maxSlope, $x$, $y$, $n$
- **Ensure:** $\beta_{i+1}$, maxSlope

```
1. slope ← getSlope($x$, $y$, $n$)
2. regularizedSlope ← $\tanh\left(\frac{\text{slope}}{\text{maxSlope}}\right)$
3. maxSlope ← $\max(\text{maxSlope}, \text{slope})$
4. $\beta_{i+1} ← F(\beta_i, \text{regularizedSlope})$ \{Function $F$ is defined in equation (1)\}
5. return $\beta_{i+1}$, maxSlope
```

### 2.2 Variance-based approach

The second approach is based on the statistical variance of every data dimension. It follows the same steps of Algorithm 3 except the replacement of function `getSlope($x$, $y$, $n$)` by Algorithm 4, which computes the statistical variance for one data dimension. The variance for every dimension is then applied in Algorithm 3. As result, we have a set of $\beta_{i+1}$ to define the adaptation for each dimension.

**Algorithm 4** Pseudocode for the variance-based approach: `getVariance($x$, $n$)`

- **Require:** $x$, $n$
- **Ensure:** variance

```
1. sum = 0
2. for $i = 1$ to $n$
3. sum = sum + $x[i]$
4. end for
5. mean = $\frac{\text{sum}}{n}$
6. sum = 0
7. for $i = 1$ to $n$
8. sum = sum + $(x[i] - \text{mean})^2$
9. end for
10. variance = $\frac{\text{sum}}{n}$
11. return variance
```

### 3 Data-stream clustering algorithms

The automatic approaches for adapting learning control parameters, detailed in Section 2, were implemented into four algorithms from literature which are detailed next.

#### 3.1 Self-organising map

The artificial neural network referred to as SOM (Kohonen, 1998) employs competitive learning to train neurons organised in a two-dimensional grid. Every neuron $i$ is represented by a set of weights $w_i \in \mathbb{R}^n$. As input data $s_i \in \mathbb{R}^n$ are received, SOM finds the closest neuron $c$ to $s_i$, which is declared as winner. The position of neuron $c$, i.e., $r_c$, in the two-dimensional grid is used as reference to update nearby neurons. SOM updates neuron weights according to equation (2), in which $h_c(t)$ defines the neighbourhood update.

$$w_i(t + 1) = w_i(t) + h_c(t)[s_i - w_i(t)] \quad (2)$$

The neighbourhood update $h_c(t)$ is defined in equation (3), which considers the grid distance in between the winning neuron $c$, positioned at $r_c$, and any other neuron $i$, at position $r_i$. Term $\sigma(t)$ defines the neighbourhood around $r_c$. As SOM receives more inputs, $\sigma(t)$ may be reduced and so the neighbourhood. According to this equation, neurons closer to the winner are more influenced towards $s_i$. Term $g(t)$ is the learning control parameter used to define the learning strength and it may decrease as new examples are received. Our automatic approaches adapt $g(t)$ in attempt to improve data-stream clustering.

$$h_c(t) = g(t)\frac{1 - \sigma(t)}{\sigma(t)} \quad (3)$$

#### 3.2 Neuron gas with competitive Hebbian learning

Neural gas [Martinetz and Schulten, (1991), p.400] is an online learning artificial neural network based on a soft-competitive strategy (Algorithm 5) which trains the best-winner unit and its closest neurons proportionally to their distances to the input data. This learning modality is commonly referred to as the winner-takes-the-most in reference to the winner-takes-all strategy (Carpenter and Grossberg, 1988). This network considers a constant number of neurons which may be connected by edges, according to their proximity to inputs. This network performs training over data collection. In this case, we adapt the learning control parameter $\theta$ to improve data-stream partitions.

#### 3.3 Growing neuron gas

The growing neuron gas artificial neural network (Fritzke, 1995a) (see Algorithm 6) places neurons over a space to represent input data. The position of every neuron $i$ is defined by its weights $w_i \in \mathbb{R}^n$. Pairs of neurons are connected through edges which correspond to a topological representation of the input space (the space in which
examples are). This approach starts with only two connected neurons and more units are periodically added to reduce the approximation error of the topology created by this network regarding new inputs. For this neural network, we consider $\theta_b$ and $\theta_s$ as learning control parameters to be adapted in attempt to improve data-stream clustering.

### 3.4 Growing grid

The growing grid artificial neural network (Fritzke, 1995b) is similar to SOM, in the sense it also builds a two-dimensional grid to project the $n$-dimensional input space. This grid is represented by a $k \times m$-matrix in which nearby cells correspond to neighbouring neurons. Every neuron $i$ is represented by weights $w_i \in \mathbb{R}^n$, which is in the same space as inputs $s_i \in \mathbb{R}^n$. The criterion used to update neuron weights is the same as SOM. Additionally, this network keeps a resource variable $r_i$ for every neuron $i$ which is incremented by one every time a neuron is given as winner. This variable is considered to add new neurons to the grid.

**Algorithm 5** Pseudocode for the neural gas artificial neural network

```
Require: $n$ neurons, $E_g$ connections, input $s$, and a distance function $f$

Require: Maximum training index $t_{\text{max}}$, initial learning control $\theta_b$, final learning control $\theta_{\text{max}}$, maximum index, sequence learning rate $\lambda_b$, sequence learning rate $\lambda_{\text{max}}$, at maximum position, and maximum age of connections $\text{age}_{\text{max}}$.

Ensure: graph $G = (V, E)$ with vertices representing neurons and edges representing neural connections

1. randomly initialise weights $w_i$ for every neuron $i$
2. initialise $E_{ij} = 0 \forall i, j$
3. for each input $s_i$ do
   4. compute and sort the $k$ lowest distances $f(s_i, w_i)$ for all neurons $i \in \{1 \ldots n\}$ {Update neurons.}
5. for each neuron $i$ among those with the $k$ lowest distances, $f(s_i, w_i)$ do
   6. $\theta = \frac{\theta_f}{\theta_{\text{max}}} \cdot \theta_b$ \{Compute current weighted distance\}
   7. $\vec{F} = -\frac{f}{\lambda_b \theta_{\text{max}} 1 + \frac{1}{\lambda_{\text{max}}}} \cdot \frac{\lambda_b \theta_{\text{max}}}{1 + \frac{1}{\lambda_{\text{max}}}} \{\text{Compute current update}\}$
   8. $w_i^{\text{new}} = w_i^{\text{old}} + \theta_{\text{cur}} \cdot \vec{F} \cdot (s_i - w_i^{\text{old}})$
9. increment the age of existing connections for neuron $i$ by 1
10. remove connections of neuron $i$ when older than $\text{age}_{\text{max}}$
11. if $E_{ij} = 0$, create connection $E_{ij} = 1$
12. end for
13. end for
```

After receiving $k \times m \times \lambda$ examples, a set of new neurons is placed next to the most visited neuron $q$ which has $\tau_q \geq \tau_i \forall i$. In order to proceed with this step, the direct neighbouring neuron $f$ with highest distance in terms of weights is selected. A row (or column) of neurons is added to the grid in order to separate neuron $q$ from $f$, increasing the representation of new inputs in such high-density input area. Neurons in this new row (or column) have weights defined by the interpolation between existing units. After this step, all resource variables are reset, i.e., $\tau_i = 0 \forall i$.

Growing grid has a parameter [Fritzke, (1995a), p.10, equation (4)] which corresponds to parameter $g(t)$ from SOM (Section 3.1). This is the learning control parameter we adapt to improve data-stream clustering.

**Algorithm 6** Pseudocode for the growing neuron gas artificial neural network

```
Require: input $s$, learning control for the nearest neuron $\theta_{\text{cur}}$, learning control for neurons within a neighbourhood $\theta_b$, maximum age $\text{age}_{\text{max}}$, constant to define the periodical addition of neurons $\lambda \in \mathbb{Z}^+$, constant to define the cumulative error for new neurons $\alpha \in (0, 1)$ and constant to decrease the cumulative error of neurons $\beta \in (0, 1)$

Ensure: graph $G = (V, E)$ with vertices representing neurons and edges representing neural connections

1. create two neurons $a$ and $b$ and randomly initialise their weights $w_a$ and $w_b$
2. for each input $s_i$ do
   3. find the nearest neuron $c_1$ and the second-nearest unit $c_2$ from $s_i$
   4. increment the age of all edges connected to neuron $c_1$
   5. compute the cumulative error for the nearest neuron $c_1$
   6. move neuron $c_1$ and every direct topological neighbour $i$ towards $s_i$ as follows:
   7. if $c_1$ and $c_2$ are connected by an edge, set the age of this edge equals to zero. If there is no edge, create it.
   8. remove all edges older than $\text{age}_{\text{max}}$
   9. if the number of examples collected so far is an integer multiple of $\lambda$ then
      10. find neuron $q$ with maximum cumulative error $\epsilon_q(t)$
      11. insert a new neuron $r$ halfway between $q$ and its neighbour $f$ with the highest cumulative error in form: $w_r = 0.5(w_q + w_f)$
      12. insert edges connecting $r$ to neurons $q$ and $f$
      13. remove the edge connecting neuron $q$ to $f$ 14: decrease the cumulative errors $\tilde{\epsilon}_q(t)$ and $\tilde{\epsilon}_f(t)$, multiplying them by constant $\alpha$
      15. initialise $\tilde{\epsilon}_r(t) = \alpha \tilde{\epsilon}_q(t)$
      16. end if
      17. decrease $\epsilon_q(t) \forall i$, multiplying them by constant $\beta$
      18. end for
```

```
4 Applying the approaches on data-stream clustering algorithms

The two proposed approaches to assess data-stream tendencies and adapt learning control parameters were implemented on the clustering algorithms described in the previous sections. All those algorithms rely on equation (4) to adapt the data representation model, in which \( \mathbf{m}(t) \) is a vector representing the current neuron weights, \( \mathbf{i}(t + 1) \) corresponds to the incoming input, \( \psi \in (0, 1] \) defines the learning control parameter, and, finally, \( \mathbf{m}(t + 1) \) is the next weight vector for the same neuron. For each incoming example, equation (4) is applied on every neuron available.

\[
\mathbf{m}(t + 1) = \mathbf{m}(t) + \psi (\mathbf{i}(t + 1) - \mathbf{m}(t)) \tag{4}
\]

In this scenario, the goal is to adapt according to the Tukey’s line and the variance-based approaches (see Section 2) in order to minimise the SSEs, which summarise the model representativeness for new data-stream inputs. Before proceeding with it, we first analysed the influence of the learning control parameter in establishing neuron weights. This parameter produces a memory effect in the equation, in which past values are forgotten at a rate \( \psi \) (here we refer to this parameter as forgetting rate) as long as new inputs arrive.

In order to understand how many inputs are necessary to apply a forgetting rate \( \kappa \) to an input, we consider equation (5), in which \( n \) is the number of iterations that happened after the arrival of the input of interest. Observe that for lower values of \( \psi \in (0, 1) \), more iterations \( n \) will be necessary to forget a given input. When assumes greater values, an incoming input will be quickly forgotten, at an exponential rate.

\[
\kappa \leq \psi \cdot (1 - \psi)^{n-1}, \ n \in \mathbb{Z}, \ n \geq 1
\tag{5}
\]

**Figure 3** Number of iterations necessary to forget an incoming input using forgetting rates \( \kappa \in \{0.0025, 0.01, 0.04\} \)

![Figure 3](image)

From the analysis of dynamics involved in data streams, we observed the learning control parameter should be adapted to improve or keep the quality of the data representation model. This follows from the fact that data-stream clustering algorithms must consider different learning control parameters along data collection. Our hypothesis is clustering algorithms must maintain a given memory level on inputs according to variations in the phenomena represented by the data stream. As an analogy of such concept, let a rare event happen such as someone winning in lottery. In that circumstance, the person would be very impressed and shocked, giving almost no relevance to past events. This is mapped into large values for the forgetting rate \( \kappa \) (see curve with \( \kappa = 0.04 \) in Figure 3), which require few iterations to forget past inputs, given the high importance of this last one. On the other hand, if someone has a slow-paced life, this person will give high importance of this last one. On the other hand, if inputs form a sinusoidal signal, the amplitude and phase are always maintained. In this situation, we expect the clustering algorithm to start at a learning control parameter \( \psi \) [we define \( \psi \in (0, 1) \)] and converge to the ideal value as inputs are received, which must be guaranteed by the adaptation algorithm (Algorithm 3).

Firstly, consider the best learning process for this conservative data stream is performed using the learning control parameter \( \psi^* \). When a clustering algorithm starts having a given \( \psi \), Algorithm 3 must adapt it in attempt to reach the constant learning parameter \( \psi = \psi^* \).

There are two scenarios to analyse convergence of the Algorithm 3 of the \( \psi \) to \( \psi^* \)

1. \( \psi(t + 1) \in \{\psi(t), \psi^*\} \), i.e., must be increased to reach \( \psi^* \) from below
2. \( \psi(t + 1) \in [\psi^*, \psi(t)] \), i.e., approaches \( \psi^* \) from above.

Both scenarios are equivalent, then let us take the first scenario in which \( \psi(t + 1) \in (\psi(t), \psi^*) \). The convergence occurs if at every iteration \( t + 1 \), learning control parameter \( \psi(t + 1) \) is always closer to \( \psi^* \) than \( \psi(t) \). Consequently, we must prove \( \psi(t + 1) > \psi(t) \) \( \forall t \) and also that \( \psi(t + 1) \) does not surpass the boundary \( \psi^* \), i.e., \( \psi(t + 1) \leq \psi^* \).
Equation (6) results in \((t + 1)\), which is another way to express Algorithm 3 in conjunction with equation (1), with \(F(\psi, r) = \psi(t + 1), \psi_t = \psi^*\) and \(r = (t)\).

\[
F(\beta, r) = \frac{1}{2}(1 + \tanh(\beta_t - r + \operatorname{atanh}(2r - 1)))
\]

\[
\psi(t + 1) = \frac{1}{2}(1 + \tanh(\psi^* - \psi(t) + \operatorname{atanh}(2\psi(t) - 1)))
\]

The first part of this proof, i.e., \(\psi(t + 1) > \psi(t)\), is done by verifying the following inequality:

\[
\psi(t + 1) > \psi(t)
\]

\[
\frac{1}{2}(1 + \tanh(\psi^* - \psi(t) + \operatorname{atanh}(2\psi(t) - 1))) > \psi(t) \tag{7}
\]

\[
\tanh(\psi^* - \psi(t) + \operatorname{atanh}(2\psi(t) - 1)) > 2(\psi(t) - 1)
\]

According to this inequality, we can verify that if \(\psi^* = \psi(t)\), then we have:

\[
\tanh(\operatorname{atanh}(2\psi(t) - 1)) > 2(\psi(t) - 1)
\]

\[
2\psi(t) - 1 > 2\psi(t) - 2
\]  \(\tag{8}\)

otherwise, if \(\psi^* > \psi(t)\), then \(K = \psi^* - \psi(t)\) is positive and, observing \(\tanh(\cdot)\) is a monotonic increasing function, we have:

\[
\tanh(\operatorname{atanh}(2\psi(t) - 1)) > 2(\psi(t) - 1)
\]

\[
K' + 2\psi(t) - 1 > 2\psi(t) - 2
\]  \(\tag{9}\)

with \(K' > K\) and, therefore, proving the first part of this theorem, in which we have \(\psi(t + 1) > \psi(t)\).

To demonstrate the second part of this theorem we must prove that \((t + 1) \leq \psi^*\) is always true. This fact can be written as:

\[
f(\psi^* - \psi(t) + g(\psi(t))) \leq \psi^*
\]

where \(f(x) = \frac{1}{2}(1 + \tanh(x))\) and \(g(x) = \operatorname{atanh}(2x - 1)\) are auxiliary functions.

In this situation \(\psi \in (0, \psi^*]\), consequently approximates \(\psi^*\) from below, that is:

\[
f(\psi^* - \psi(t) + \operatorname{atanh}(2\psi^* - 1)) \leq \psi^*
\]

\[
f(\operatorname{atanh}(2\psi^* - 1)) \leq \psi^*
\]

\[
1/2 + 1/2 \tanh(\psi^* - 1) \leq \psi^*
\]

\[
1/2 + 1/2 \cdot (2\psi^* - 1) \leq \psi^*
\]

\[
1/2 + \psi^* - 1/2 \leq \psi^*
\]

\[
\psi^* \leq \psi^*
\]

In case \(\psi \in [\psi^*, 1]\), the situation is similar in which it tends to \(\psi^*\) from above.

In this way we conclude Algorithm 3 converges to a constant learning control value when a function is used to indicate the right direction of data tendency. The two proposed approaches (Section 2) fulfill this requirement by using functions \textit{getSlope} (Algorithm 1) and \textit{getVariance} (Algorithm 4).

6 Experimental results

This section presents experimental results for the two automatic approaches for adapting learning control parameters. We considered three synthetic scenarios. All data-stream clustering algorithms were executed using the standard algorithm (without any adaptation) or using one of the two automatic adaptation approaches activated as detailed next. The proposed approaches modify learning control parameter \(\psi\) defined in equation (4) in attempt to improve the clustering partitions. The experiments used the DemoGNG software, provided by Hartmut S. Loos as a GNU/GPL source code available at http://www.sund.de/netze/applets/gng/full/DemoGNGcode.html, which implements the following data clustering algorithms: SOM, neural gas, growing neural gas, and growing grid.

Figure 4 Data stream considered in the first scenario and weights of best matching neuron of SOM with automatic adaptation or standard approaches (assuming \(\psi = 0.001\))

6.1 Scenario 1: sinusoidal data stream

The first scenario defined for experiments is based on a sequence which represents a conservative data stream, i.e., the stream maintains the same behaviour as data is collected. The 10,000-long data sequence was generated using the R statistical software \((R\ \text{Core}\ \text{Team},\ 2012)\) as follows \(\text{sin(seq(1, 100, length=10000)}\).

We employed all four clustering algorithms, detailed in Section 3, to learn the behaviour of this sequence. Figure 4 illustrates the data stream and how the SOM algorithm follows the data-stream tendencies. The curves graphed for SOM represent the weights of the winning neurons for every incoming data. Three different executions for SOM were performed.
without using our automatic approaches for adapting the learning control parameter \( \psi \), i.e., the standard approach.

2. learning parameter is adapted using the Tukey’s line approach.

3. learning parameter is adapted using the variance-based approach.

The second and third executions modify learning control parameter \( \psi \), defined in equation (4), according to the proposed approaches.

The curves representing SOM using the two proposed approaches are closer to the data stream examples, which implies the reduction of the SSEs, consequently, both automatic adaptations improve learning. In the case adaptation is not used, the weights of the winning neuron have the greatest deviation from data due to the learning control parameter is not adequate.

Next, we evaluate the clustering algorithms in terms of the SSEs considering three situations each: standard approach, using the Tukey’s line and the variance-based approaches. In each situation, the clustering algorithm was executed using an initial value for learning control parameter \( \psi \) and processed all stream inputs, producing a sequence of SSEs. Those errors correspond to the divergence between the weights of the winning neuron and inputs. Then, we computed the average for all errors produced in every execution and present them as points in curves of Figure 5, having \( \psi \), i.e., in the horizontal axis.

Every curve in Figure 5 corresponds to the SSEs of the algorithm considering different initial learning control parameters. For example, Figure 5(a) presents results for the SOM, in which we observe the SSEs are significantly lower when using one of the two automatic approaches for adapting the learning parameter \( \psi \). In fact, whatever initial learning control we take, SOM finds out good parameter values to follow the data-stream behaviour. The same does not happen with the standard approach, which is evident when using \( \psi < 0.1 \), as the error is higher.

Analysing Figure 5 as a whole, we observe the Tukey’s line approach provided the lowest SSEs for the SOM, neuron gas with competitive Hebbian learning and growing grid. The variance-based approach improved in comparison with the standard algorithm in two of those three scenarios, except for neuron gas. When considering the growing neuron gas, the Tukey’s line approach also produced better results than the variance-based one. In this scenario [Figure 5(c)], SSEs are relatively lower with standard approach when starting with \( \psi > 0.15 \), however we remember the greater the value for \( \psi \), the more a new input influences in the neuron weights, consequently there is a risk to cause abruptly changes in weights.

**Figure 5** Scenario 1 – sinusoidal data stream: SSEs curves for clustering algorithms, (a) SOM (b) neuron gas with CHL (c) growing neuron gas (d) growing grid.
6.2 Scenario 2: ARIMA-based data stream

Similarly to the previous case, this scenario aims to evaluate the SSEs of clustering algorithms under the influence of the two proposed approaches: Tukey’s line and the variance-based approach. This second scenario considers a synthetic data stream produced using an ARIMA stochastic model. The data is obtained using the R command `arima.sim(4,000, model = list(1, 1, 0)).`

The variance-based approach presented better results than Tukey’s line or standard approaches. Figure 6(a) shows good results for SOM employing the variance-based approach, while Tukey’s line approach had worse performance than the standard SOM. This pattern of results is also presented for the algorithm neuron gas with CHL in Figure 6(b). Figure 6(c) also shows better results with variance-based approach, although Tukey’s line has also delivered relevant improvements on the low-value range of $\psi$, which is usually preferred by specialists to choose a learning control parameter when little problem knowledge is available. Figure 6(d), regarding algorithm growing grid, did not provide any relevant distinction of results when applying either of the two approaches.

We observed in this scenario that the noisy nature of ARIMA favoured the variance-based over Tukey’s line approach. The variance-based approach computes a statistic to make it more robust to the presence of noise.

6.3 Scenario 3: experiments using the DemoGNG simulator

In the third scenario we employed a test case provided by the DemoGNG simulator. This scenario is based on a non-stationary probability function represented by a moving rectangle which determines the area of a uniform probability distribution function. This scenario represents a noisier situation than the previous ones. At the same time, the moving rectangle represents cases in which is mandatory to update the clustering along data collection in order to preserve information about pattern trends.

Figure 7 shows the SSEs using the proposed approaches (curves Tukey’s line and variance-based) in comparison to the standard way. The four subfigures refer to the SSE obtained for each clustering algorithm.

From results in Figure 7(a), we observe SOM using the variance-based approach presented better or similar results when compared to the standard approach, i.e., without parameter adaptation. Differently, the Tukey’s line was ineffective in this situation and provided degraded results. For the neuron gas with CHL algorithm, Figure 7(b), both approaches improved results similarly and reduced the overall SSE. In Figure 7(c), the results obtained with the variance-based approach show a lower SSE curve than Tukey’s line approach. However, both approaches obtained significant better results in terms of SSE in the low-value range of $\psi \in (0, 0.2)$, which is most often used by specialists. For growing grid, the SSE was not affected.

Figure 6 Scenario 2 – ARIMA-based data stream: SSE curves for clustering algorithms, (a) SOM (b) neuron gas with CHL (c) growing neuron gas (d) growing grid
7 Discussion and conclusions

This paper introduced two automatic approaches to adapt learning control parameters in an attempt to produce more representative clustering partitions for data streams. The main motivation for this work is the fact that streams may change over time, making learning control parameters obsolete which results in the loss of data structural information. Experimental results confirm approaches show some capability of reducing and keeping the SSEs at low levels, independently of the initial setup of learning control parameters.

Analysing both approaches, Tukey’s line is more effective in scenarios under less noise, such as the sinusoidal data-stream (Figure 5), while the variance-based approach provides better results in noisier scenarios. This tendency is expected since as the variance-based approach computes a sample statistic it tends to be more robust in the presence of noise.

We also observed the automatic approaches are less dependent on the initial value of learning control parameters, consequently they are more robust to the setup made by specialists. On the other hand, the standard approach, i.e., without our adaptation approaches, shows a high variation in SSEs when the learning control parameter is modified.

Provided that data streams usually change over time and consequently may present different properties, we recommend the usage of the variance-based approach to support the adaptation of learning control parameters of clustering algorithms.

Acknowledgements

This paper is based upon works supported by FAPESP (Sao Paulo Research Foundation), Brazil, under Grant No. 2014/13323-5; by CNPq (The National Council for Scientific and Technological Development), Brazil, under Grants No. 303051/2014-0 and 441583/2014-8; and by PROPP-UFU (Pro-Reitoria de Pesquisa e Pos-Graduação, Federal University of Uberlândia) under Grant No. 030 of Edital 06/2013. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of FAPESP, CNPq and PROPP-UFU.

References


