Silva and Marques *Journal of Big Data* (2015) 2:27
DOI 10.1186/s40537-015-0033-0

Journal of Big Data
a SpringerOpen Journal

METHODOLOGY Open Access

The ubiquitous self-organizing map for non-stationary data streams

Bruno Silva[1*] and Nuno Cavalheiro Marques[2]

*Correspondence:
bruno.silva@estsetubal.ips.pt
[1] DSI/ESTSetúbal, Instituto Politécnico de Setúbal, Campus do IPS, Estefanilha, 2914-761 Setúbal, Portugal
Full list of author information is available at the end of the article

Abstract

The Internet of things promises a continuous flow of data where traditional database and data-mining methods cannot be applied. This paper presents improvements on the Ubiquitous Self-Organized Map (UbiSOM), a novel variant of the well-known Self-Organized Map (SOM), tailored for streaming environments. This approach allows ambient intelligence solutions using multidimensional clustering over a continuous data stream to provide continuous exploratory data analysis. The average quantization error and average neuron utility over time are proposed and used to estimating the learning parameters, allowing the model to retain an indefinite plasticity and to cope with changes within a multidimensional data stream. We perform parameter sensitivity analysis and our experiments show that UbiSOM outperforms existing proposals in continuously modeling possibly non-stationary data streams, converging faster to stable models when the underlying distribution is stationary and reacting accordingly to the nature of the change in continuous real world data streams.

Keywords: Self-organizing maps, Data streams, Non-stationary data, Clustering, Exploratory analysis, Sensor data

Introduction

At present, all kinds of stream data processing based on instantaneous data have become critical issues of Internet, Internet of Things (ubiquitous computing), social networking and other technologies. The massive amounts of data being generated in all these environments push the need for algorithms that can extract knowledge in a readily manner.

Within this increasingly important field of research the application of artificial neural networks to such task remains a fairly unexplored path. The self-organizing map (SOM) [1] is an unsupervised neural-network algorithm with topology preservation. The SOM has been applied extensively within fields ranging from engineering sciences to medicine, biology, and economics [2] over the years. The powerful visualization techniques for SOM models result from the useful and unique feature of SOM for detection of emergent complex cluster structures and non-linear relationships in the feature space [3]. The SOM can be visualized as a sheet-like neural network array, whose neurons become specifically tuned to various input vectors (examples) in an orderly fashion. For instance, the SOM and $\mathcal{K}$-means both represent data in a similar way through prototypes of data, i.e., centroids in $\mathcal{K}$-means and neuron weights in SOM, and their relation

and different usages has already been studied [4]. However, it is the topological ordering of these prototypes in large SOM networks that allows the application of exploratory visualization techniques.

This paper is an extended version of work published in [5], introducing a novel variant of SOM, called the ubiquitous self-organizing map (UbiSOM), specially tailored for streaming and big data. We extend our previous work by improving the overall algorithm with the use of a drift function to estimate learning parameters, that weighs the previous *average quantization error* and a new introduced metric: the *average neuron utility*. Also, the UbiSOM algorithm now implements a finite-state machine, which allows it to cope with drastic changes in the underlying stream. We also performed parameter sensitivity analysis on new parameters imposed by the algorithm.

Our experiments, with artificial data and a real-world electric consumption sensor data stream, show that UbiSOM can be applied to data processing systems that want to use the SOM method to provide a fast response and timely mine valuable information from the data. Indeed our approach, albeit being a single-pass algorithm, outperforms current online SOM proposals in continuously modeling non-stationary data streams, converging faster to stable models when the underlying distribution is stationary and reacting accordingly to the nature of the change.

Background and literature review

In this section we introduce data streams and review current SOM algorithms that can, in theory, be used for streaming data, highlighting their problems in this setting.

Data streams

Nowadays, data streams [6, 7] are generated naturally within several applications as opposed to simple datasets. Such applications include network monitoring, web mining, sensor networks, telecommunications, and financial applications. All have vast amounts of data arriving continuously. Being able to produce clustering models in real-time assumes great importance within these applications. Hence, learning from streams not only is required in ubiquitous environments, but also is of relevance to other current hot topics, namely Big Data. The rationale behind the requirement of learning from streams is that the amount of information being generated is to big to be stored in devices, where traditional mining techniques could be applied. Data streams arrive continuously and are potentially unbounded. Therefore, it is impossible to keep the entire stream in memory.

Data streams require fast and real time processing to keep up with the high rate of data arrival and mining results are expected to be available within short response time. Data streams also imply non-stationarity of data, i.e., the underlying distribution may change. This may involve appearance/disappearance of clusters, changes in mean and/or variance and also correlations between variables. Consequently, algorithms performing over data streams are presented with additional challenges not previously tackled in traditional data mining. One thing that is agreed is that these algorithms can only return approximate models since data cannot be revisited to fine-tune the models [7], hence the need for incremental learning.

More formally, a data stream $\mathcal{S}$ is a massive sequence of examples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, i.e., $\mathcal{S} = \{\mathbf{x}_i\}_{i=1}^{N}$, which is potentially unbounded ($N \to \infty$). Each example is described by an

d-dimensional feature vector $\mathbf{x} = [x_i^j]_{j=1}^d$ belonging to a feature space Ω that can be continuous, categorical or mixed. In out work we only consider continuous spaces.

The Self-Organizing Map

The SOM establishes a projection from the manifold Ω onto a set $\mathcal{K}$ of neurons (or units), formally written as $\Omega \rightarrow \mathcal{K}$, hence performing both vector quantization and projection. Each unit $\mathcal{K}$ is associated with a prototype $\mathbf{w}_k \in \mathbb{R}^d$, all of which establish the set $\mathcal{K}$ that is referred as the codebook. Consequently, the SOM can be interpreted as a topology preserving mapping from an high-dimensional input space onto the 2D grid of map units. The number of prototypes K is defined by the dimensions of the grid (lattice size), i.e, $width \times height$.

The classical *Online* SOM algorithm performs iteratively over time. An example $\mathbf{x}$ is presented at each iteration t and distances between $\mathbf{x}_t$ and all prototypes are computed. Usually the Euclidean distance is used and previous normalization of $\mathbf{x}$ is suggested to equate the dynamic ranges along each dimension; this ensures that no feature dominates the distance computations, improving the numerical accuracy. The best matching unit (BMU), which we denote by c, is the map unit with prototype closest to $\mathbf{x}_t$:

$$\mathbf{w}_c(t) = \min_k \|\mathbf{x} - \mathbf{w}_k\|. \tag{1}$$

It is important to highlight that $E(t) = \| \mathbf{x}_t - \mathbf{w}_c(t) \|$ is the map error at time t, and is referred to as the *quantization error*.

Next, the prototype vectors are updated: the BMU and its topological neighbors are moved towards the example in the input space by the *Kohonen* learning rule [1, 8]:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \eta(t)\, h_{ck}(t)[\mathbf{x}_t - \mathbf{w}_k(t)] \tag{2}$$

where:

t is the time;
$\eta(t)$ is the learning rate;
$h_{ck}(t)$ is the neighborhood kernel centered on the BMU:

$$h_{ck}(t) = e^{-\left(\frac{\|r_c - r_k\|}{\sigma(t)}\right)^2} \tag{3}$$

where r_c and r_k are positions of units c and k on the SOM grid. Both $\sigma(t)$ and $\eta(t)$ decrease monotonically with time, a critical condition for the network to converge steadily towards a topological ordered state and to map the input space density. The following decreasing functions are common:

$$\sigma(t) = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{t/t_f}, \quad \eta(t) = \eta_i \left(\frac{\eta_f}{\eta_i}\right)^{t/t_f}, \tag{4}$$

where σ_i and σ_f are respectively the initial and final neighborhood width and η_i and η_f the initial and final learning rate. From [8] it is suggested that: the width of the

neighborhood should be decreased from a width approximately the width of the lattice, for an initial global ordering of the prototypes, down to only encompassing the adjacent map units. This iterative scheme endures until t_f is reached and is typically defined so the dataset is presented several times.

SOM models for streaming data

In a real-world streaming environment t_f is unknown or not defined, so the classical algorithm cannot be used. Even with a bounded stream the *Online* SOM loses plasticity over time (due to the decrease of the learning parameters) and cannot cope easily with changes in the underlying distribution.

Despite the huge amount of SOM literature around SOM and SOM-like networks, there is surprisingly and comparatively very little work dealing with incremental learning. Furthermore, most of these works are based on incremental models, that is, networks that create and/or delete nodes as necessary. For example, the modified GNG model [9] is able to follow non-stationary distributions by creating nodes like in a regular GNG and deleting them when they have a too small utility parameter. Similarly, the evolving self-organizing map (ESOM) [10, 11] is based on an incremental network quite similar to GNG that creates dynamically based on the measure of the distance of the BMU to the example (but the new node is created at exact data point instead of the midpoint as in GNG). Self-organizing incremental neural network (SOINN) [12] and its enhanced version (ESOINN) [13] are also based on an incremental structure where the first version is using a two layers network while the enhanced version proposed a single layer network. These proposals, however, do not guarantee a compact model, given that the number of nodes can increase unbounded in a non-stationary environment if not parameterized correctly.

On the other hand, our proposal keeps the size of the map fixed. Some SOM time-independent variants, obeying to this restriction, have been proposed. The two most recent examples are: the Parameterless SOM (PLSOM) [14], which evaluates the local error $E(t)$ and calculates the learning parameters depending on the local quadratic fitting error of the map to the input space, and; the Dynamic SOM (DSOM) [15] which follows a similar reasoning by adjusting the magnitude of the learning parameters to the local error, but fails to converge from a totally unordered state. Moreover, authors of both proposals admit that their algorithms are unable to map the input space density onto the SOM, which has a severe impact on the application of common visualization techniques for exploratory analysis. Also, these variants are very sensitive to outliers, i.e., noisy data, by using instantaneous $E(t)$ values.

On the other hand, the proposed UbiSOM algorithm in this paper estimates learning parameters based on the performance of the map over streaming data by monitoring the average quantization error, being more tolerant to noise and aware of real changes in the underlying distribution.

The ubiquitous self-organizing map

The proposed UbiSOM algorithm relies on two learning assessment metrics, namely the *average quantization error* and the *average neuron utility*, computed over a sliding window. While the first assesses the trend of the vector quantization process towards the

underlying distribution, the later is able to detect regions of the map that may become "unused" given some changes in the distribution, e.g., disappearance of clusters. Both metrics are weighed in a drift function that gives an overall indication of the performance of the map over the data stream, used to estimate learning parameters.

The UbiSOM implements a finite state-machine consisting in two states, namely ordering and learning. The ordering state allows the map to initially unfold over the underlying distribution with monotonically decreasing learning parameters; it is also used to obtain the first values of the assessment metrics, transitioning afterwards to the learning state. Here, the learning parameters, i.e., learning rate and neighborhood radius, are decreased or increased based on the drift function. This allows the UbiSOM to retain an indefinite plasticity, while maintaining the original SOM properties, over non-stationary data streams. These states also coincide with the two typical training phases suggested by Kohonen. It is possible, however, that unrecoverable situations from abrupt changes in the underlying distribution are detected, which leads the algorithm to transition back to the ordering state.

Notation

Each UbiSOM neuron k is a tuple $\mathcal{W}_k = \langle \mathbf{w}_k, t_k^{update} \rangle$, where $\mathbf{w}_k \in \mathbb{R}^d$ is the prototype and t_k^{update} stores the time stamp of the last time its prototype was updated. For each incoming observation $\mathbf{x}_t$, presented at time t, two metrics are computed, within a sliding window of length T, namely the average quantization error $\overline{qe}(t)$ and the average neuron utility $\overline{\lambda}(t)$. We assume that all features of the data stream are equally normalized between $[d_{min}, d_{max}]$. The local quantization error $E(t)$ is normalized by $|\Omega| = (d_{max} - d_{min})\sqrt{d}$, so that $\overline{qe}(t) \in [0, 1]$. The $\overline{\lambda}(t)$ metric averages neuron utility ($\lambda(t)$) values that are computed as a ratio of updated neurons during the last T observations. Both metrics are used in a drift function $d(t)$, where the parameter $\beta \in [0, 1]$ weighs both metrics.

The UbiSOM switches between the ordering and learning states, both using the classical SOM update rule, but with different mechanisms for estimating learning parameters σ and η. The ordering state endures for T examples, until the first values of $\overline{qe}(t)$ and $\overline{\lambda}(t)$ are available, establishing an interval $[t_i, t_f]$, during which monotonically decreasing functions $\sigma(t)$ and $\eta(t)$ are used to decrease values between $\{\sigma_i, \sigma_f\}$ and $\{\eta_i, \eta_f\}$, respectively. The *learning* state estimates learning parameters as a function of the drift function. UbiSOM neighborhood function is defined in a way that uses $\sigma \in [0, 1]$ as opposed to existing variants, where the domain of the values is problem-dependent.

Online assessment metrics

The purpose of these metrics is to assess the "fit" of the map to the underlying distribution. Both proposed metrics are computed over a sliding window of length T.

Average quantization error

The widely used global quantization error (QE) metric is the standard measure of fit of a SOM model to a particular distribution. It is typically used to compare SOM models obtained for different runs and/or parameterizations and used in a batch setting. The

rationale is that the model which exhibits a lower QE value is better at summarizing the input space.

Regarding data streams this metric, as it stands, is not applicable because data is potentially infinite. Competing approaches to the proposed UbiSOM use only the local quantization error $E(t)$. Kohonen stated that both $\eta(t)$ and $\sigma(t)$ should decrease monotonically with time, a critical condition to achieve convergence [8]. However, the local error is very unstable because $\Omega \rightarrow \mathcal{K}$ is a many-to-few mapping, where some observations are better represented than others. As an example, with stationary data the local error does not decrease monotonically over time. We argue this is the reason why other existing approaches, e.g., PLSOM and DSOM, fail to model the input space density correctly.

In the proposed algorithm, the quantization error was modified to a running mean in the form of the average quantization error $\overline{qe}(t)$, based on the premise that the error of a learner will decrease over time for an increasing number of examples if the underlying distribution is stationary; otherwise, if the distribution changes, the error increases. For each observation $\mathbf{x}_t$ the $E'(t)$ local quantization error is obtained during the BMU search, as the normalized Euclidean distance

$$E'(t) = \frac{\| \mathbf{x}_t - \mathbf{w}_c \|}{|\Omega|}. \tag{5}$$

These values are averaged over a window of length $T \gg 0$ to obtain $\overline{qe}(t)$, defined in Eq. (5). Consequently, the value of T establishes a short, medium or long-term trend of the model adaptation.

$$\overline{qe}(t) = \frac{1}{T} \sum_{t}^{t-T+1} E'(t) \tag{6}$$

Figure 1 depicts the typical behavior of $E'(t)$ values obtained during a run of the classical SOM algorithm, together with the computed $\overline{qe}(t)$ values for $T = 2000$, over a data stream where the underlying distribution suffers an abrupt change at $t = 50\,000$. We can observe that $E'(t)$ values exhibit a large variance throughout time, as opposed to $\overline{qe}(t)$ which is smoother and indicates the trend of the convergence. Therefore, it is implicitly

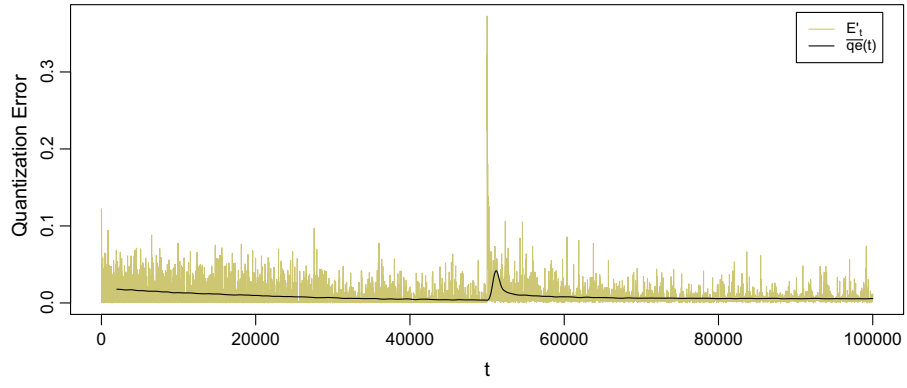

Fig. 1 Behavior of local $E'(t)$ vs. average quantization error $\overline{qe}(t)$

assumed that if $\overline{qe}(t)$ is decreasing, then the underlying distribution is stationary; otherwise, it is changing.

Average neuron utility

The average quantization error $\overline{qe}(t)$ may be a good overall indicator of the fit of the model. Despite that, it may be unable to detect the abrupt disappearance of clusters. Figure 2 illustrates such a scenario, depicting the "unused" area of the map after the inner cluster disappears. Here $\overline{qe}(t)$ does not increase, however in this situation, the learning parameters should increase and allow the map to recover from this situation.As a consequence, the average neuron utility was proposed as a means to detect these cases.

To compute this assessment metric each UbiSOM neuron $\mathcal{K}$ is extended with a time stamp t_k^{update} which stores the last time the corresponding prototype was updated, functioning as an *aging* mechanism. A prototype is updated if it is the BMU or if it falls in the influence region of the BMU, limited by the neighborhood function. Initially, $t_k^{update} = 0$. The neuron utility $\lambda(t)$ is given by Eq. (7). It measures the ratio of neurons that were updated within the last T observations, over the total number of neurons. Consequently, if all neurons have been recently updated, then $\lambda(t) = 1$. The values are then averaged by Eq. (8) to obtain $\overline{\lambda}(t)$.

$$\lambda(t) = \frac{\sum_{k=1}^{K} 1_{\{t - t_k^{update} \leq T\}}}{K} \tag{7}$$

$$\overline{\lambda}(t) = \frac{1}{T} \sum_{t}^{t-T+1} \lambda(t). \tag{8}$$

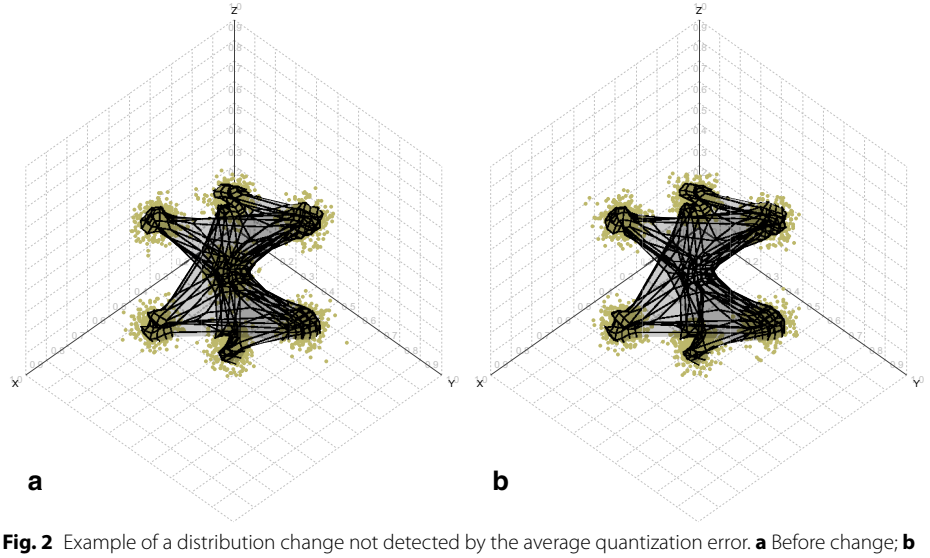

Fig. 2 Example of a distribution change not detected by the average quantization error. **a** Before change; **b** after the disappearance of the inner cluster, where it is visible a region of unused neurons

As a result, a decrease in $\overline{\lambda}(t)$ indicates that there are neurons that are not being used to quantize the data stream. While it is not unusual to obtain these "dead-units" with stationary data after the map has converged, the decreasing trend should alert for changes in the underlying distribution.

The drift function

The previous metrics $\overline{qe}(t)$ and $\overline{\lambda}(t)$ are both weighed in a drift function that is used by the UbiSOM to estimate learning parameters. In short:

$\overline{qe}(t)$ The average quantization error gives an indication of how well the map is currently quantifying the underlying distribution, previously defined in Eq. (6). In most situation where the underlying data stream is stationary, $\overline{qe}(t)$ is expected to decrease and stabilize, i.e., the map is converging. If the shape of the distribution changes, $\overline{qe}(t)$ is expected to increase.

$\overline{\lambda}(t)$ The average neuron utility is an additional measure which gives an indication of the proportion of neurons that are actively being updated, previously defined in Eq. (8). The decrease of $\overline{\lambda}(t)$ indicates neurons are being underused, which can reflect changes in the underlying distribution not detected by $\overline{qe}(t)$.

The drift function is defined as

$$d(t) = \beta\,\overline{qe}(t) + (1-\beta)\,(1-\overline{\lambda}(t)) \tag{9}$$

where $\beta \in [0,1]$ is a weighting factor that establishes the balance of importance between the two metrics. Since both $\overline{qe}(t)$ and $\overline{\lambda}(t)$ are only obtained after T observations, so is $d(t)$.

A quick analysis of $d(t)$ should be made: with high learning parameters, specially the neighborhood σ value, $\overline{\lambda}(t)$ is expected to be ≈ 1, which practically eliminates the second term of the equation. Consequently, the drift function in only governed by $\overline{qe}(t)$. When the neuron utility decreases the second term contributes to the increase of $d(t)$ in proportion to the chosen β value. Ultimately, if $\beta = 1$ then the drift function is only defined by the $\overline{qe}(t)$ metric. Empirically, β should be parameterized with relatively high values, establishing $\overline{qe}(t)$ as the main measure of "fit" and using $\overline{\lambda}(t)$ as a failsafe mechanism.

The neighborhood function

The UbiSOM algorithm uses a normalized neighborhood radius σ learning parameter and a truncated neighborhood function. The latter is what effectively allows $\overline{\lambda}(t)$ to be computed.

The classical SOM neighborhood function relies on a σ value that is problem-dependent, i.e., the used values depend on the lattice size. This complicates the parameterization of σ for different values of $\mathcal{K}$, i.e., $width \times height$.

The performed normalization is based on the maximum distance between any two neurons in the lattice. In rectangular maps the farthest neurons are the ones at opposing diagonals, e.g., positions (0, 0) and $(width-1, height-1)$ in Fig. 3. Hence distances within the lattice are normalized by the Euclidean norm of the vector $\mathbf{diag} = (width-1, height-1)$, defined as

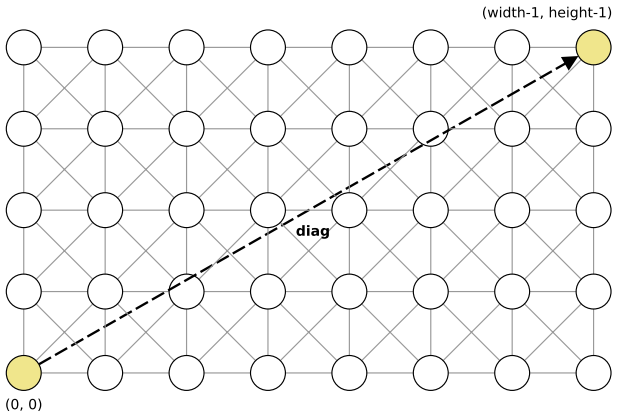

Fig. 3 Maximum lattice distance used for normalization of σ

$$\|\mathbf{diag}\| = \sqrt{(width-1)^2 + (height-1)^2}. \tag{10}$$

This effectively limits the maximum neighborhood width the UbiSOM can use and establishes $\sigma \in [0, 1]$.

The neighborhood function of the UbiSOM variant is given by

$$h'_{ck}(t) = e^{-\left(\frac{\|r_c - r_k\|)}{\sigma \|\mathbf{diag}\|}\right)^2} \tag{11}$$

where r_c is the position in the grid of the BMU for observation $\mathbf{x}_t$. To get a grasp on how different σ values determine the influence region around the BMU, Fig. 4 depicts Eq. (11) for different σ values. Neurons whose values of $h'_{ck}(t)$ are below a threshold of 0.01 are not updated. This is critical for the computation of $\lambda(t)$, since $h'_{ck}(t)$ is a continuous function and as $h'_{ck}(t) \to 0$ all other neurons would still be updated with very small values. The truncated neighborhood function is also a performance improvement, avoiding negligible updates to prototypes.

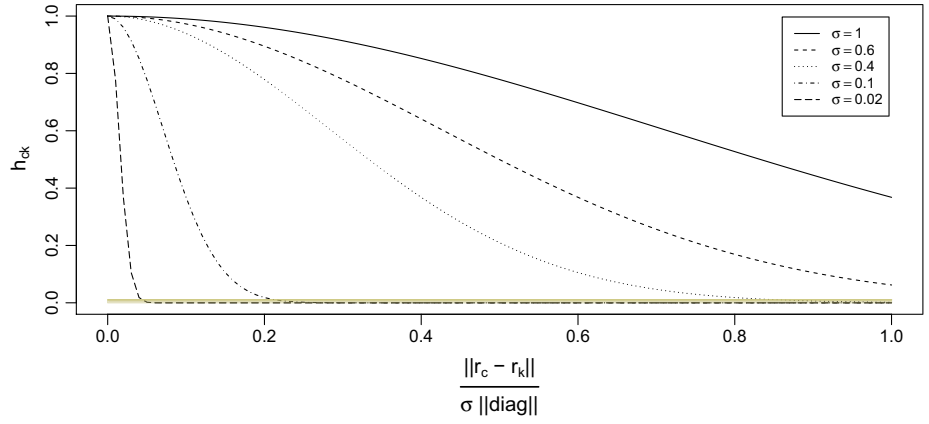

Fig. 4 The UbiSOM neighborhood function. The threshold value is also depicted, below which no updates to prototypes are performed

States and transitions

The UbiSOM algorithm implements a finite state-machine, i.e., it can switch between two states. This design was, on one hand, imposed by the initial delay in obtaining values for the assessment metrics and, as a consequence, for the drift function $d(t)$; on the other hand, seen as a desirable mechanism to conform to Kohonen's proposal of an ordering and a convergence phase for the SOM [8] and to deal with drastic changes that can occur in the underlying distribution.

The two possible states of the UbiSOM algorithm, namely ordering state and learning state are depicted in Fig. 5 and described next. Both use a similar update equation as the classical algorithm, but with the neighborhood function defined in Eq. (11), as defined in Eq. (12). Please note that the prototypes are only updated above the neighborhood function threshold.

$$\mathbf{w}_k(t+1) = \begin{cases} \mathbf{w}_k(t) + \eta(t) h'_{ck}(t)[\mathbf{x}_t - \mathbf{w}_k(t)] & h'_{ck}(t) > 0.01 \\ \mathbf{w}_k(t) & \textit{otherwise} \end{cases} \tag{12}$$

However, each state estimates learning parameters with different functions for $\eta(t)$ and $\sigma(t)$.

Ordering state

The ordering state is the initial state of the UbiSOM algorithm and to where it possibly reverts if it can not recover from an abrupt change in the data stream. It endures for T observations where learning parameters are estimated with a monotonically decreasing function, i.e., time-dependent, similar to the classical SOM. Thus, the parameter T simultaneously defines the window length of the assessment metrics, as well as dictates the duration of the ordering state. The parameters should be relatively high, so the map can order itself from a totally unordered initialization regarding the underlying distribution. This phase also allows for the first value of the drift function $d(t)$ to be available. After T observations the algorithm switches to the learning state.

Let t_i and $t_f = t_i + T - 1$ be the first and last iterations of the ordering phase, respectively. This state requires choosing appropriate parameter values for η_i, η_f, σ_i and σ_f, which are, respectively, the initial and final values for the learning rate and the normalized neighborhood radius. The choice of values will greatly impact the initial ordering of the prototypes and will affect the estimation of parameters of the learning state. Any

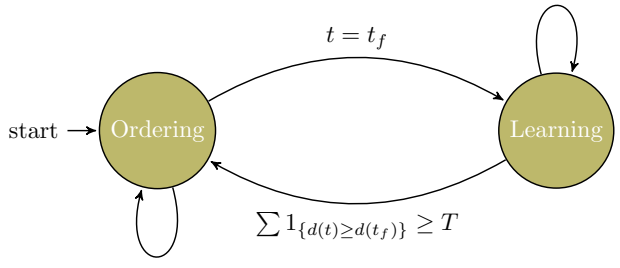

Fig. 5 UbiSOM states and transitions

monotonically decreasing function can be used, although in this research the following were used:

$$\sigma(t) = \sigma_i \left(\frac{\sigma_f}{\sigma_i}\right)^{t/t_f}, \quad \eta(t) = \eta_i \left(\frac{\eta_f}{\eta_i}\right)^{t/t_f} \qquad \forall t \in \{t_i, t_{i+1}, \ldots, t_f\} \tag{13}$$

At the end of the t_f iteration, the first value of the drift function is obtained, i.e., $d(t_f)$, and the UbiSOM algorithm transitions to the learning state.

Learning state

The learning state begins at $t_f + 1$ and is the main state of the UbiSOM algorithm, during which learning parameters are estimated in a time-independent manner. Here learning parameters are estimated solely based on the drift function $d(t)$, decreasing or increasing relative to the first computed value $d(t_f)$ and final values (η_f, σ_f) of the ordering state.

Given that in this state the map is expected to start converging, the values of $d(t)$ should also decrease. Hence, the value $d(t_f)$ is used as a reference value establishing a threshold above which the map is considered to be irrecoverably diverging from changes in the underlying distribution, e.g., in some abrupt changes the drift function can increase rapidly to very high values. Consequently, it also limits the maximum values that learning parameters can attain during this state

Learning parameters $\eta(t)$ and $\sigma(t)$ are estimated for an observation presented at time t by Eq. (14), where $d(t)$ is defined as in Eq. (9). One can easily derive that learning parameters are estimated proportionally to $d(t)$. Also, final values of the ordering state for η_f and σ_f establish an upper bounded for the learning parameters in this state.

$$\eta(t) = \begin{cases} \frac{\eta_f}{d(t_f)}\, d(t) & d(t) < d(t_t) \\ \eta_f & \textit{otherwise} \end{cases} \quad \sigma(t) = \begin{cases} \frac{\sigma_f}{d(t_f)}\, d(t) & d(t) < d(t_f) \\ \sigma_f & \textit{otherwise.} \end{cases} \tag{14}$$

The outcome of these equations is that if the distribution is stationary the learning parameters accompany the decrease of the drift function values, allowing the map to converge to a stable state. On the contrary, if changes occur, the drift function values rise, consequently increasing the learning parameters, and increase the plasticity of the map to a point where $d(t)$ should decrease again. The increased plasticity should allow the map to adjust to the distribution change.

However, there may be cases of abrupt changes from where the map cannot recover, i.e., the map does not resume convergence with decreasing $d(t)$ values. Therefore, if we detect that learning parameters are in their peak values during at least T iterations, i.e., $\sum 1_{\{d(t) \geq d(t_f)\}} \geq T$, then this situation is confirmed and the UbiSOM transitions back to the ordering state.

Time and space complexity

The UbiSOM algorithm (and model) does not increase the time complexity of the classical SOM algorithm, since all the potentially penalizing additional operations, namely the computations of the assessment metrics, can be obtained in $O(1)$. Regarding space complexity, it increases the space needed for: (1) storing an additional timestamps for each

neuron k; (2) storing two queues for the assessment metrics $\overline{qe}(t)$ and $\overline{\lambda}(t)$, each of length T. Therefore, after the initial creation of data structures (map and queues) in $O(K)$ time and $O(Kd + 2K + 2T)$ space, every observation $\mathbf{x}_t$ is processed in constant $O(2Kd)$ time and constant space. No observations are kept in memory.

Hence, the UbiSOM algorithm is scalable in respect to the number of observations N, since the cost per observations is kept constant. However, the increase of the number of neurons K, i.e., the size of the lattice, and the dimensionality d of the data stream will increase this cost linearly.

Results and discussion

A series of experiments was conducted using artificial data streams to assess the UbiSOM parameterization and performance over stationary and non-stationary data, while comparing it to current proposals, namely the classical Online SOM, PLSOM and DSOM. With artificial data we can establish the ground truth of the expected outcome and illustrate some key points. Afterwards we apply the UbiSOM to a real-world electric power consumption problem where we further illustrate the potential of the UbiSOM when dealing with sensor data in a streaming environment.

Table 1 summarizes the artificial data streams used in the presented results. These are two- and three-dimensional for the purpose of easy visualization of obtained maps. The *Gauss* data stream, as the name suggests, describes a Gaussian cluster of points and is used to check if the algorithms can map the input space density properly; *Chain* describes two inter-locked rings—this represents a cluster structure that partitional clustering algorithms, e.g., k-means, fail to cluster properly; *Hepta* describes a distribution of 7 evenly spaced Gaussian clusters, where at time $t = 50\,000$ one cluster disappears, previously depicted in Fig. 2, and; *Clouds* contains an evolving cluster structure with separation and merge of clusters (between $t = 50\,000$ and $t = 150\,000$) and aims at evaluating how the different tested algorithms react to continuous changes in the distribution. All data streams were normalized such that $\mathbf{x}_t \in [0, 1]^d$.

The parameterization of any SOM algorithm is mainly performed empirically, since only rules of thumb exist towards finding good parameters [8]. In the next section present an initial parameter sensitivity analysis of the new parameters introduced in the UbiSOM, e.g., T and β, while empirically setting the remaining parameters, shared at some extent with the classical SOM algorithm. Concerning the lattice size it should be rectangular in order to minimize projection distortions, hence we use a 20×40 lattice for all algorithms, which also allows for a good quantization of the input space. In the ordering state of the UbiSOM algorithm we have empirically set $\eta_i = 0.1$, $\eta_f = 0.08$, $\sigma_i = 0.6$ and $\sigma_f = 0.2$, based on the recommendation that learning parameters should

Table 1 Summary of artificial data streams used in presented experiments

Name	*d*	*N*	Stationary?	Number clusters
Gauss	2	100,000	Yes	1
Chain	2	100,000	Yes	2
Hepta	3	150 000	No	7/6
Clouds	3	200,000	No	2/3/2

be initially relatively high to allow the unfolding of the map over the underlying distribution. These values have shown optimal results in the presented experiments and many others not included in this paper. A parameter sensitivity analysis including these parameters is reserved for future work.

Regarding the other algorithms, after several tries the best parameters for the chosen map size and for each compared algorithm were selected. The classical *Online* SOM uses $\eta_i = 0.1$ and $\sigma_i = 2\sqrt{K}$, decreasing monotonically to $\eta_f = 0.01$ and $\sigma_f = 1$ respectively; PLSOM uses a single parameter γ called neighborhood range and the values yielding the best results for the used lattice size were $\gamma = (65, 37, 35, 130)$ for the *Gauss*, *Chain*, *Hepta* and *Clouds* data streams, respectively. DSOM was parameterized as in [15] with *elasticity* $= 3$ and $\varepsilon = 0.1$, but since it fails to unfold from an initial random state, it was left out of further experiments. The authors admit that their algorithm has this drawback.

Maps across all experiments use the same random initialization of prototypes at the center of the input space, so no results are affected by different initial states.

Parameter sensitivity analysis

We present a parameter sensitivity analysis for parameters T and β introduced in the UbiSOM. The first establishes the length of the sliding window used to compute the assessment metrics, and consequently weather it uses a short, medium or long-term trend to estimate learning parameters. While a shorter window is more sensitive to the variance of the $E'(t)$ and to noise, a longer window increases the reaction time of the algorithm to true change in the underlying distribution. It also implicitly dictates the duration of the ordering state, where Kohonen recommends, as another rule-of-thumb, that it should not cover less that 1 000 examples [8]. The later weights the importance of both assessment metrics in the drift function $d(t)$ and, as discussed earlier, we should use higher values so as to favor the $\overline{qe}(t)$ values while estimating learning parameters. Hence, we chose $T = \{500, 1000, 1500, 2000, 2500, 3000\}$ and $\beta = \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ as the sets of values from where to perform the parameter sensitivity analysis.

To shed some light on how these parameters could affect learning, we opted to measure the mean quantization error [(Mean $E'(t)$], so as to obtain a single value that could characterize the quantization procedure across the entire stream. Similarly, we used the mean neuron activity [(Mean $\lambda(t)$] to measure in a single value the proportion of utilized neurons during learning from stationary and non-stationary data streams.

Thus, we were interested in finding ideal intervals for the tested parameters that could simultaneously minimize the mean quantization error, while maximizing the mean neuron utility. We also computed $\overline{qe}(t)$ for the different values of T to obtain a grasp on the delay in convergence imposed by this parameter. From the minimum $\overline{qe}(t)$ obtained throughout the stream, we computed the iteration where the $\overline{qe}(t)$ value falls within 5 % of the minimum (Convergence t), as a temporal indicator of convergence.

The results for all combinations of the chosen parameter values for *Chain* and *Clouds* data streams are presented in Tables 2 and 3, respectively. It comes at no surprise that for increasing T, the convergence happens latter in the stream. More importantly, results empirically suggest that $T = \{1500, 2000\}$ and $\beta = \{0.6, 0.7, 0.8\}$ exhibit the best compromise between the minimization of the error and the maximization of neuron utility.

Table 2 Parameter sensitivity analysis with the *chain* data stream

T	β	Mean $E'(t)$	Mean $\lambda(t)$	Convergence t
500	0.5	1.5542e−02	9.9451e−01	5119
	0.6	1.5377e−02	9.9251e−01	6669
	0.7	1.5260e−02	9.8955e−01	6667
	0.8	1.5201e−02	9.8422e−01	6661
	0.9	1.5356e−02	9.7235e−01	6699
	1.0	1.5915e−02	8.7072e−01	7326
1000	0.5	1.5759e−02	9.9663e−01	7029
	0.6	1.5787e−02	9.9537e−01	7509
	0.7	1.5780e−02	9.9355e−01	7480
	0.8	1.5824e−02	9.9083e−01	7537
	0.9	1.5888e−02	9.8480e−01	7573
	1.0	1.6085e−02	9.3651e−01	7709
1500	0.5	1.6260e−02	9.9753e−01	9483
	0.6	1.6261e−02	9.9671e−01	9561
	0.7	1.6260e−02	9.9555e−01	9543
	0.8	1.6288e−02	9.9361e−01	9569
	0.9	1.6319e−02	9.9002e−01	9608
	1.0	1.6436e−02	9.6240e−01	9582
2000	0.5	1.6864e−02	9.9818e−01	8264
	0.6	1.6892e−02	9.9746e−01	7976
	0.7	1.6902e−02	9.9661e−01	8114
	0.8	1.6887e−02	9.9603e−01	8352
	0.9	1.6904e−02	9.9403e−01	8307
	1.0	1.6974e−02	9.7860e−01	8264
2500	0.5	1.7317e−02	9.9872e−01	10,022
	0.6	1.7339e−02	9.9836e−01	10,006
	0.7	1.7361e−02	9.9808e−01	9993
	0.8	1.7356e−02	9.9756e−01	10,035
	0.9	1.7383e−02	9.9613e−01	10,001
	1.0	1.7391e−02	9.8976e−01	10,027
3000	0.5	1.7819e−02	9.9910e−01	10,541
	0.6	1.7833e−02	9.9892e−01	10,518
	0.7	1.7839e−02	9.9856e−01	10,535
	0.8	1.7855e−02	9.9811e−01	10,510
	0.9	1.7856e−02	9.9701e−01	10,543
	1.0	1.7863e−02	9.9405e−01	10,532

After experimentally trying these values, we opted for $T = 2000$ and $\beta = 0.7$ since it consistently gave good results across a variety of data streams, some not included in this paper. Hence, all the remaining experiments use these parameter values.

Density mapping

We illustrate the modeling and quantization process of all tested algorithms in Fig. 6, for the stationary *Gauss* data stream. It can be seen that only the Online SOM and the Ubi-SOM are able to model the input space density correctly, assigning more neurons to the denser area of points. The inability of PLSOM to map the density limits its applicability

Table 3 Parameter sensitivity analysis with the *clouds* data stream

T	β	Mean $E'(t)$	Mean $\lambda(t)$	Convergence t
500	0.5	5.6533e−03	9.9748e−01	6093
	0.6	5.3362e−03	9.9665e−01	6939
	0.7	5.0489e−03	9.9483e−01	6913
	0.8	5.2061e−03	9.9247e−01	6896
	0.9	4.9515e−03	9.8354e−01	6934
	1.0	4.6026e−03	8.9388e−01	6905
1000	0.5	5.2879e−03	9.9811e−01	7163
	0.6	5.0090e−03	9.9766e−01	7161
	0.7	5.1416e−03	9.9697e−01	7164
	0.8	5.0338e−03	9.9457e−01	7178
	0.9	4.8834e−03	9.8795e−01	7186
	1.0	4.5776e−03	9.3157e−01	7201
1500	0.5	5.2028e−03	9.9854e−01	9457
	0.6	5.1877e−03	9.9828e−01	9466
	0.7	5.0920e−03	9.9719e−01	9471
	0.8	5.0337e−03	9.9470e−01	9487
	0.9	4.8891e−03	9.8845e−01	9496
	1.0	4.6342e−03	9.3501e−01	9495
2000	0.5	5.2332e−03	9.9885e−01	9794
	0.6	5.2783e−03	9.9822e−01	9794
	0.7	5.3479e−03	9.9714e−01	9794
	0.8	5.1034e−03	9.9575e−01	9794
	0.9	4.9776e−03	9.8838e−01	9794
	1.0	4.6787e−03	9.0427e−01	9794
2500	0.5	5.2758e−03	9.9870e−01	10,176
	0.6	5.2640e−03	9.9811e−01	10,176
	0.7	5.2271e−03	9.9810e−01	10,176
	0.8	5.1438e−03	9.9707e−01	10,176
	0.9	5.1787e−03	9.9203e−01	10,176
	1.0	4.9083e−03	9.5121e−01	10,176
3000	0.5	5.6079e−03	9.9816e−01	10,725
	0.6	5.4242e−03	9.9846e−01	11,830
	0.7	5.1422e−03	9.9844e−01	12,017
	0.8	5.2570e−03	9.9645e−01	10,725
	0.9	5.2056e−03	9.9229e−01	10,725
	1.0	4.9657e−03	9.5392e−01	10,725

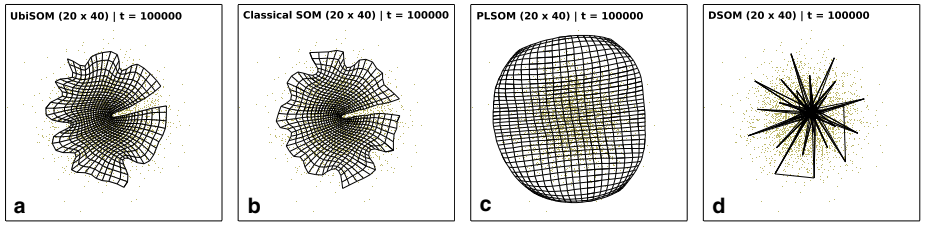

Fig. 6 Final maps obtained for the stationary Gaussian data stream. **a** UbiSOM; **b** online SOM; **c** PLSOM; **d** DSOM

to exploratory analysis with some visualization techniques illustrated in this work. It can also be seen that DSOM fails to unfold the map to cover this distribution.

Convergence with stationary and non-stationary data

The following results compare the UbiSOM algorithm against the Online SOM and the PLSOM algorithms across all artificial data streams. Table 4 summarizes the obtained values for the previously used measures, namely the mean $E'(t)$ and mean $\lambda(t)$ values. While PLSOM exhibits a lower mean $E'(t)$ on all data streams, except *Gauss*, it does so at the expense of not mapping the input space density, as previously demonstrated. The density mapping of the vector projection and the quantization process can be seen as conflicting goals. On the other hand, the UbiSOM algorithm performs very similarly in this measure, but consistently exhibits higher mean $\lambda(t)$ values, most importantly in the non-stationary data streams, which may indicate that the other algorithms may have not performed so well in the presence of changes in the underlying distribution.

The previous measures can establish a baseline comparison between algorithms, but are not conclusive regarding the quality of the obtained maps. Consequently, we computed the average quantization error $\overline{qe}$ with $T = 2000$ for all algorithms and data streams. Please note that although this is the value that the UbiSOM also uses, we consider this fair for all algorithms, since it is simply evaluating the trend of the quantization error for each algorithm. The results are depicted in Fig. 7 and it can be seen that the UbiSOM algorithm generally converges faster to stationary phases of the distributions, while the PLSOM converges less steadily and slower in half of the streams, and the convergence of the Online SOM is dictated by the monotonic decrease of the learning parameters. In the *Hepta* data stream only the UbiSOM algorithm is able to detect the disappearance of the cluster, which we can derive by the increase of the $\overline{qe}$ values. Please note that this was a consequence of the contribution of the average neuron activity into the drift function. Similarly, in the *Clouds* data stream the UbiSOM algorithm quickly reacts to the start of the gradual change in the position of the clusters through the $\overline{qe}$ metric, while the average neuron utility is responsible for the observed "spikes" when it detects currently unused regions of the map. Given that the UbiSOM learning

Table 4 Comparison of the UbiSOM, Online SOM and PLSOM algorithms across all data streams

Dataset	Algorithm	Mean $E'(t)$	Mean $\lambda(t)$
Gauss	UbiSOM	7.7362e−3	1
	Online	1.4056e−2	9.9998e−1
	PLSOM	8.2531e−3	9.8253e−1
Chain	UbiSOM	1.6902e−2	9.9661e−1
	Online	3.3196e−2	9.9954e−1
	PLSOM	1.2752e−2	9.7863e−1
Hepta	UbiSOM	1.3709e−2	9.9703e−1
	Online	2.4628e−2	9.6564e−1
	PLSOM	1.1616e−2	9.8411e−1
Clouds	UbiSOM	5.3479e−3	9.9714e−1
	Online	7.3585e−3	9.0615e−1
	PLSOM	4.0952e−3	9.6890e−1

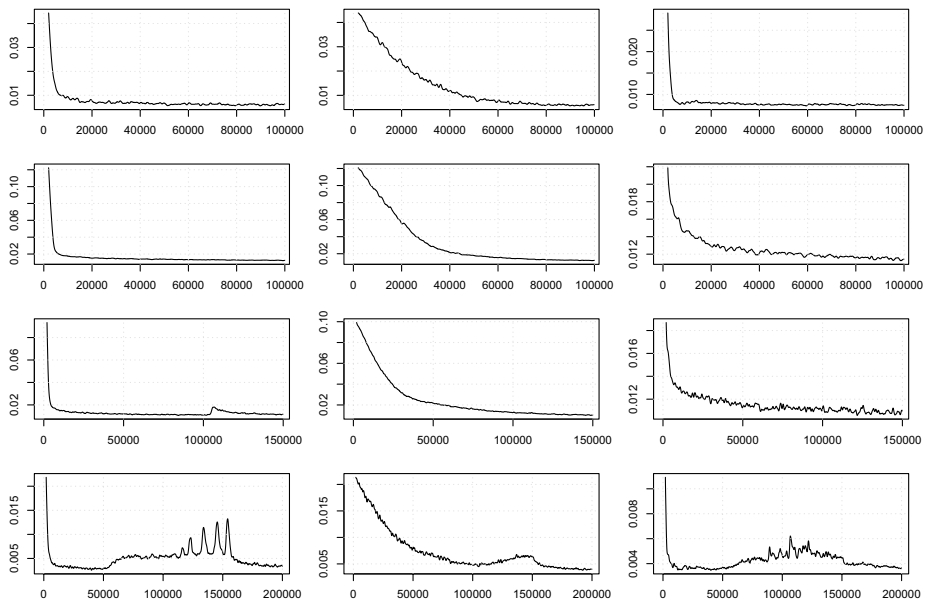

Fig. 7 Average quantization error $\overline{qe}(t)$ of algorithms across all data streams. Results for the UbiSOM, online SOM and PLSOM in *left*, *center* and *right* columns, respectively. The *rows* regard the Gauss, Chain, Hepta and Clouds data streams, respectively

parameters are mainly estimated through $\overline{qe}$, we can get a very close idea of the evolution of the learning parameters across these different datasets.

In order to support the above inferences, Fig. 8 illustrates the UbiSOM over the *Clouds* data stream, confirming it models the changing underlying distribution correctly over time, maintaining density mapping and topological ordering of prototypes with few signs of unused portions of the map. On the other hand, the final Online SOM and PLSOM maps for this data stream are presented in Fig. 9. Neither is able to correctly model the distribution in its final state. Whereas the Online SOM is progressively less capable to model changes due to decreasing learning parameters, the PLSOM suffers from the fact it also uses an estimation of the input space diameter, which also is changing, to compute learning parameters.

As an additional example of the clustering of the UbiSOM through exploratory analysis, in Fig. 10 we illustrate the final map obtained for the *Chain* data stream and corresponding *U-matrix* [3], a color-scaled visualization that can be used here to correctly identify the two clusters. Warmer colors translate to higher distances between neurons, consequence of the vector projection, establishing borders between clusters.

Exploratory analysis in real-time

A real world demonstration is achieved by applying the UbiSOM to the real-world Household electric power consumption data stream from the UCI repository [16], comprising 2 049 280 observations of seven different measurements (i.e., $d = 7$), collected to the minute, over a span of 4 years. Only features regarding sensor values were used, namely *global active power* (kW), *voltage* (V), *global intensity* (A), *global reactive power* (kW) and *sub-meterings* for the *kitchen*, *laundry room* and *heating* (W/h). The Household data stream contains several drifts in the underlying distribution, given the nature

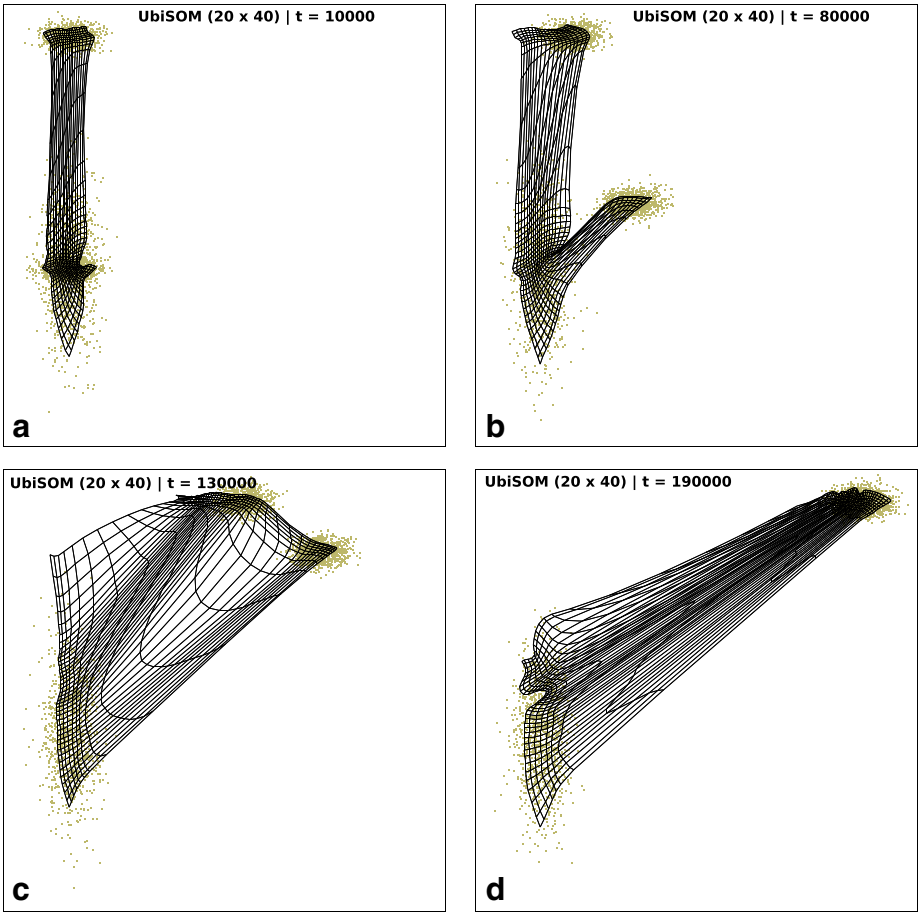

Fig. 8 Evolution of the UbiSOM over the Clouds data stream. Maps at **a** $t = 10{,}000$; **b** $t = 80{,}000$; **c** $t = 130{,}000$, and; **d** $t = 190{,}000$

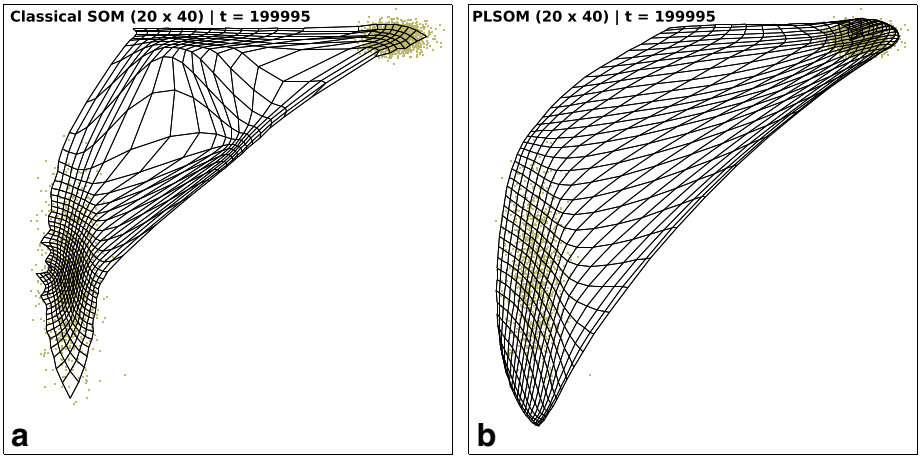

Fig. 9 Final maps for the Online SOM and PLSOM over the Clouds data stream **a** Online SOM; **b** PLSOM

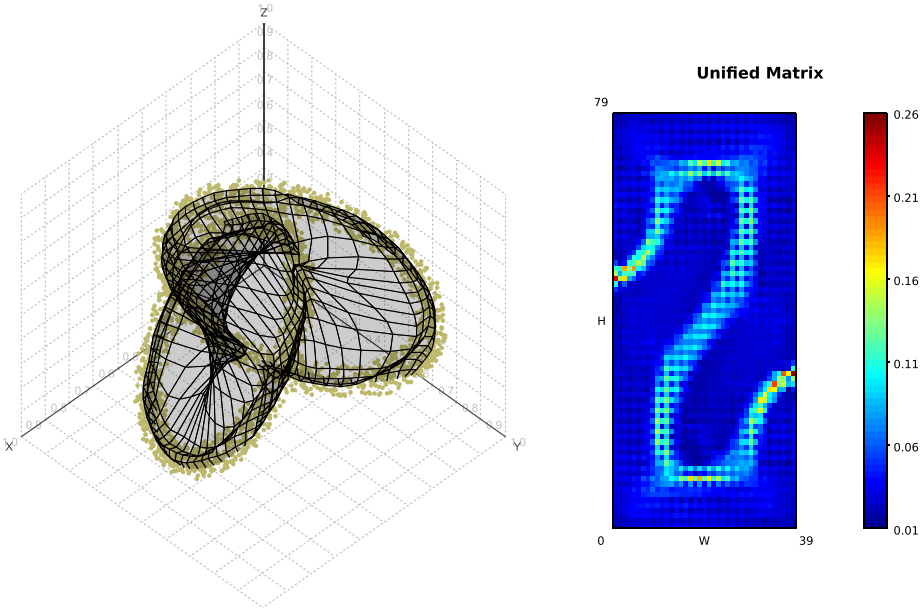

Fig. 10 Obtained UbiSOM for the Chain data stream and corresponding U-matrix. Two clear clusters can be derived from this visualization

of electric power consumption, and we believe these are the most challenging environments where UbiSOM can operate.

Here, we briefly present another visualization technique called component planes [3], that further motivates the application of UbiSOM to a non-stationary data stream. Component planes can be regarded as a "sliced" version of the SOM, showing the distribution of different features values in the map, through a color scale. This visualization can be obtained at any point in time, providing a snapshot of the model for the present and recent past. Ultimately, one can take several snapshots and inspect the evolution of the underlying stream.

Figure 11 depicts the last 5000 presented examples (∼3.5 days) of the Household data stream, i.e., from $t = 495\,000$ to $t = 500\,000$, while Fig. 12 shows the component planes obtained at $t = 500\,000$ using the UbiSOM. For illustration purposes we discarded the *global reactive power* feature. These visualizations indicate correlated features, namely *Global active power* and *Global intensity* are strongly correlated (identical component planes), while exhibiting some degree of inverse correlation to *Voltage*. Since the UbiSOM is able to map the input space density, the component planes of the heating sensors indicate their relative overall usage right before that period of time, e.g., *Heating* has a high consumption approximately 2/3 of the time. Since this point in time concerns the month of December 2008, this seems self-explanatory.

Component planes also show that *Global active power* has its highest values when *Kitchen* (Sub_metering_1) and *Heating* (Sub_metering_3) are active at the same time; the overlap of higher values for *Laundry room* (Sub_metering_2) *Kitchen* (Sub_metering_1) is low, indicating that they are not used very often at the same time. All these empirical inductions from the exploratory analysis of the component planes seem correct looking

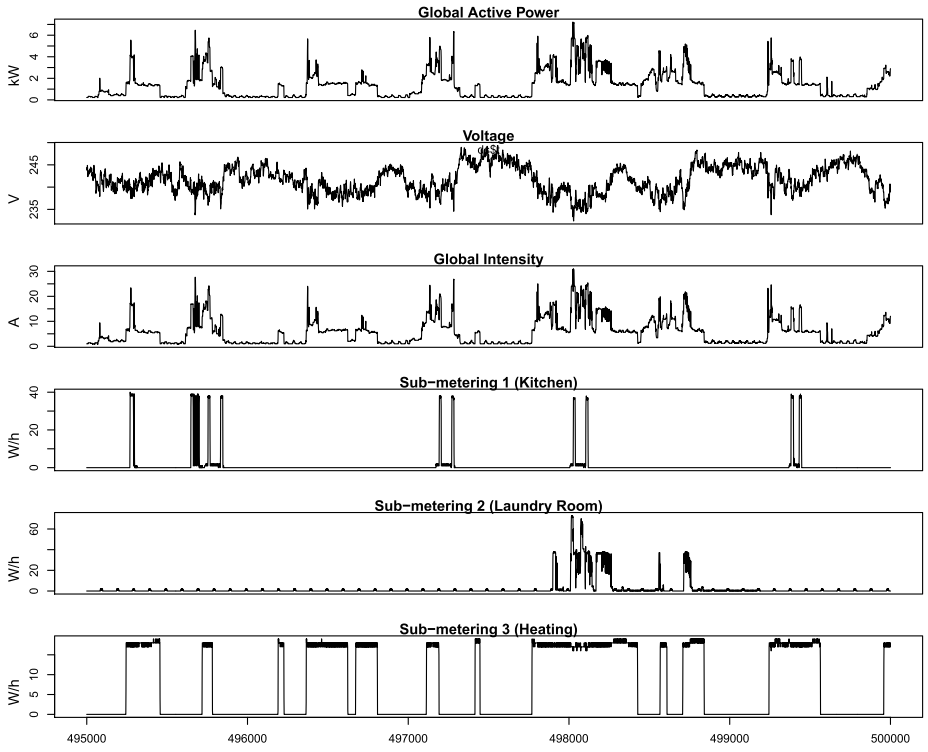

Fig. 11 Household data stream from $t = 495\,000$ to $t = 500\,000$

at the plotted data in Fig. 11, and highlight the visualization strengths of UbiSOM with streaming data.

Conclusions

This paper presented the improved version of the ubiquitous self-organizing map (UbiSOM), a variant tailored for real-time exploratory analysis over data streams. Based on literature review and the conducted experiments, it is the first SOM algorithm capable of learning stationary and non-stationary distributions, while maintaining the original SOM properties. It introduces a novel *average neuron utility* assessment metric in addition to the previously used *average quantization error*, both used in a drift function that measures the performance of the map over non-stationary data and allows for learning parameters to be estimated accordingly. Experiments show this is a reliable method to achieve the proposed goal and the assessment metrics proved fairly robust. The UbiSOM outperforms current SOM algorithms in stationary and non-stationary data streams.

The real-time exploratory analysis capabilities of the UbiSOM are, in our opinion, extremely relevant to a large set of domains. Besides cluster analysis, the component-plane based exploratory analysis of the Household data stream exemplifies the relevancy of the proposed algorithm. This points to a particular useful usage of UbiSOM in many practical applications, e.g., with high social value, including health monitoring, powering a greener economy in smart cities or the financial domain. Coincidently, ongoing work is targeting the financial domain to model the relationships between a wide variety of asset prices for portfolio selection and to signal changes in the model over time as an

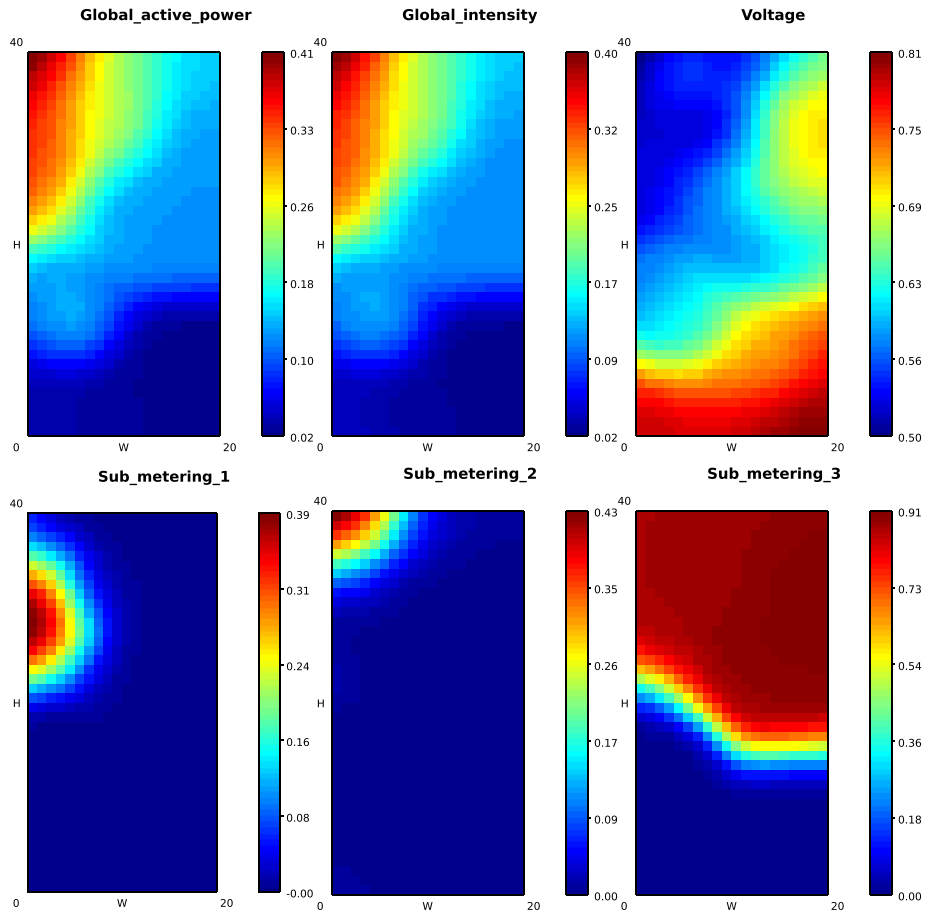

Fig. 12 UbiSOM obtained component planes at $t = 500\,000$ with the Household data stream

alert mechanism. In parallel, we continue conducting research with distributed air quality sensor data in Portugal.

Authors' contributions

BS is the principal researcher for the work proposed in this article. His contributions include the underlying idea, background investigation, initial drafting of the article, and results implementation. NCM supervised the research and played a pivotal role in writing the article. Both authors read and approved the final manuscript.

Author details

[1] DSI/ESTSetúbal, Instituto Politécnico de Setúbal, Campus do IPS, Estefanilha, 2914-761 Setúbal, Portugal. [2] NOVA Laboratory for Computer Science and Informatics, DI/FCT, Universidade Nova de Lisboa, Monte da Caparica, Portugal.

Acknowledgements

The research of BS was partially funded by Fundação para a Ciência e Tecnologia with the Ph.D. scholarship SFRH/BD/49723/2009. The authors would also like to thank Project VeedMind, funded by QREN, SI IDT 38662.

Competing interests

The authors declare that they have no competing interests.

Received: 5 June 2015 Accepted: 30 October 2015
Published online: 14 December 2015

References

1. Kohonen T. Self-organized formation of topologically correct feature maps. Biol Cybern. 1982;43(1):59–69.

2. Pöllä M, Honkela T, Kohonen T. Bibliography of self-organizing map (som) papers: 2002–2005 addendum. Neural Computing Surveys. 2009.
3. Ultsch A, Herrmann L. The architecture of emergent self-organizing maps to reduce projection errors. In: Verleysen M, editor. Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2005); 2005. pp. 1–6.
4. Ultsch A. Self organizing neural networks perform different from statistical k-means clustering. In: Proceedings of GfKl '95. 1995.
5. Silva B, Marques NC. Ubiquitous self-organizing map: learning concept-drifting data streams. New contributions in information systems and technologies. Advances in Intelligent Systems and Computing: Springer; 2015. p. 713–22.
6. Aggarwal CC. Data streams: models and algorithms, vol. 31. Springer; 2007.
7. Gama J, Rodrigues PP, Spinosa EJ, de Carvalho ACPLF. Knowledge discovery from data streams. Chapman and Hall/CRC Boca Raton. 2010.
8. Kohonen T. Self-organizing maps, vol 30. New York: Springer; 2001.
9. Fritzke B. A self-organizing network that can follow non-stationary distributions. In: Artificial Neural Networks—ICANN 97. Springer. 1997. p. 613–618
10. Deng D, Kasabov N. Esom. An algorithm to evolve self-organizing maps from on-line data streams. Neural Networks, IEEE-INNS-ENNS International Joint Conference on IEEE Computer Society, vol. 6; 2000. p. 6003.
11. Deng D, Kasabov N. On-line pattern analysis by evolving self-organizing maps. Neurocomputing. 2003;51:87–103.
12. Furao S, Hasegawa O. An incremental network for on-line unsupervised classification and topology learning. Neural Netw. 2006;19(1):90–106.
13. Furao S, Ogura T, Hasegawa O. An enhanced self-organizing incremental neural network for online unsupervised learning. Neural Netw. 2007;20(8):893–903.
14. Berglund E. Improved plsom algorithm. Appl Intell. 2010;32(1):122–30.
15. Rougier N, Boniface Y. Dynamic self-organising map. Neurocomputing. 2011;74(11):1840–7.
16. Bache K, Lichman M. UCI machine learning repository. 2013. http://archive.ics.uci.edu/html.