STAIRS 2012
K. Kersting and M. Toussaint (Eds.)

doi:10.3233/978-1-61499-096-3-294

Neural Network-based Framework for Data Stream Mining

Bruno SILVA[a,1] Nuno MARQUES[b]
[a] *DSI/ESTSetúbal, Instituto Politécnico de Setúbal, Portugal*
[b] *CITI and Departamento de Informática, FCT, Universidade Nova de Lisboa, Portugal*

Abstract. We address the problem of mining data streams using Artificial Neural Networks (ANN). Usual data stream clustering models (eg. k-means) are too dependent on assumptions regarding cluster statistical properties (ie. number of clusters, cluster *shape*), while unsupervised ANN algorithms (Adaptive Resonant Theory — ART networks and Self-Organizing Maps — SOM) are recognized widely by their ability to discover hidden patterns, generalization capabilities and robustness to noise. However, use of ANNs with the data stream model is still poorly explored. We propose a methodology and modular framework to cluster data streams and extract other relevant knowledge. Empirical results with both synthetic and real data provide evidence of the validity of the approach.

Keywords. Neural Networks, Data Stream Mining, Unsupervised Learning, Knowledge Extraction

1. INTRODUCTION

Data streams are generated naturally within several applications as opposed to simple datasets. Network monitoring, web mining, telecommunications data management, stock-market analysis and sensor data processing are applications that have vast amounts of data arriving continuously. Data mining has become the key technique to analyze and understand data. These mining techniques help find interesting patterns, regularities and abnormalities in the data, e.g., clusters and correlations between variables. However, mining data streams pose different challenges and proposed methodologies use adaptations of traditional algorithms such as *k*-means [8,7] or specially devised ones [1].

Artificial Neural Networks (ANN) are a well-established set of, biologically inspired, mining algorithms and are recognized widely by their ability to discover hidden patterns, generalization capabilities and robustness to noise. However, using ANN for mining data streams is still a very unexplored path. In this paper we present a modular approach in mining data streams using unsupervised ANN, namely *Adaptive Resonance Theory* (ART) networks [5] and *Self-Organizing Maps* (SOM) [10], for clustering and knowledge extraction. We show that ANN are a viable and promising approach to process continuous streams of data and explain their advantages over current traditional approaches.

[1]Corresponding Author: Escola Superior de Tecnologia de Setúbal - Campus do IPS, 2910-761, Setúbal, Portugal; E-mail: bruno.silva@estsetubal.ips.pt.

Hence, our main contributions are: (i) a modular framework based on a two-phased learning process (Section 3): in the *online* part of the framework an ART network is used to produce data aggregations of the incoming stream. These are then used to create SOM models from which clusters and other knowledge can be extracted. These models are produced *offline*; (ii) a micro-clustering procedure based on ART networks and necessary modifications to the used ART algorithm (Section 3.1). The obtained micro-clusters are a compact representation of the incoming stream and form the data aggregations; (iii) a *concept drift* detection mechanism that operates on continuous aggregation results (Section 3.2). With data streams the underlying distribution of the data may not be strictly stationary, i.e., it may change over time; (iv) the ability to create SOM models trained with micro-clusters through a modification to the update rule, from which knowledge can be extracted using conventional visualization techniques (Section 3.3).

In Section 4 we provide several experimental results on both synthetic and real data that provide evidence of the validity of the approach. These results target the several modules of the framework.

2. CURRENT APPROACHES

In the data stream model the data points can only be accessed in the order in which they arrive; random access to data is not allowed; memory is assumed to be small relatively to the number of data points, thus only allowing a limited amount information to be stored. One should point out that algorithms performing on data streams are expected to produce "only" approximated models, since the data cannot be revisited to refine the generated models.

The related work pertinent to this paper concerns the task of clustering, which is mainly used to get insight into data distribution. Using ANN with data streams is still a very unexplored path and literature is very scarce. A recent work provided a qualitative exploration of weightless neural paradigms in the problem of clustering data streams [6]. They explored the potential agility of one-shot training and the reduced amount of memory needed by their architecture. However, they did not provide a quantitative approach dealing with realistic data nor a fully exploration of the cluster dynamics.

K-means is a popular algorithm in clustering data streams. In [8] a single pass *k*-means algorithm is proposed. The main idea is to use a buffer where points of the dataset are kept in a compressed way. The data stream is processed in blocks. All available space on the buffer is filled with points from the stream. Only the *k* centroids (representing the clustering results) are retained, with the corresponding *k* cluster features. In the following iterations, the buffer is initialized with the k-centroids, found in previous iteration, weighted by the *k* cluster features, and incoming data points from the stream. The single pass *k*-means is incremental, improving its solution given additional data. It uses a fixed sized buffer. Improvements of the *k*-means algorithm in stream mining can be found in [7,9]. However, *k*-means suffers from the problem that the initial *k* clusters have to be set either randomly or through other methods. This has a strong impact on the quality of the clustering process. On the other hand, ART networks do not suffer from this problem (Section 3.1).

Other works [1,2] present a relevant technique to our work based on micro-clustering. These algorithms divide the clustering process into two phases, where the first

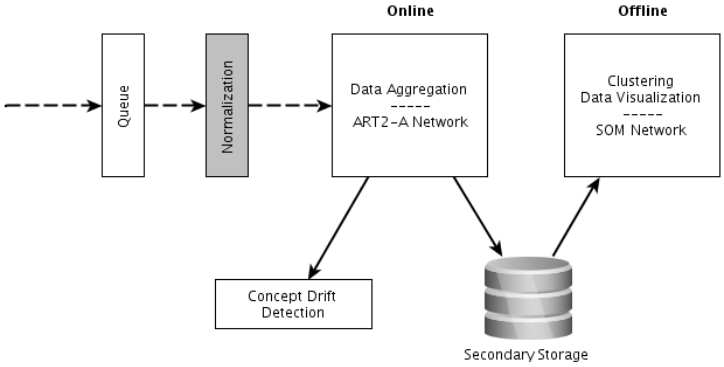


Figure 1. Proposed framework architecture and interaction between modules.

phase is online and summarizes the data stream in local models (micro-clusters) and the second offline phase generates a global cluster model from the micro-clusters. CluStream [1] is a framework for clustering data streams and the offline phase performs clustering on summarized data according to a number of user preferences, such as the time frame and number of clusters. A number of experiments on real datasets have been conducted to provide evidence of the accuracy and efficiency of the proposed algorithm. HPStream [2] is an enhancement of the later framework to target high-dimensional data streams.

3. PROPOSED FRAMEWORK

We start by describing the overall neural network-based framework, its modules and how they interact, depicted in Figure 1.

Queuing The incoming data stream is buffered in a queue. This serves as a cushion between possible bursts in the stream and the processing in subsequent modules.

Normalization Although not used in the present work, we intend to introduce later on a normalization module that scales the features to the same dynamic range. This real-time normalization is frequently ignored in most works related to mining data streams. For now, we assume that they are normalized.

Aggregation This module is responsible for producing a synopsis of the data. An ART network is responsible for generating aggregation results. This process is detailed in Section 3.1.

Concept Drift Detection This module continuously receives aggregation results, stored in a circular buffer of fixed size and determines how well the last micro-clusters "fit" into the previous ones. Section 3.2 explains this procedure.

Storage Since the stream is potentially infinite, subsequent aggregation results from the previous module can be saved in secondary storage. The stored aggregation results form a maximum *time frame* from which the offline module can extract knowledge.

Knowledge Extraction This is the offline component of the framework, where models can be generated from the aggregation results. Since it is offline, no single-pass re-

striction over the data is imposed. A SOM is trained, from which knowledge can be extracted through a variety of techniques and visualizations, namely clusters and non-linear correlations between attributes. We explain the necessary modifications to the SOM training algorithm in Section 3.3.

3.1. Online Data Aggregation

The aggregation module is responsible for the online summarization of the incoming stream and processes the stream in blocks of size S, extracted from the *Queue*. For each S observations q representative prototypes of data are created, where $q \ll S$. This relates to an incremental clustering process that is performed by an ART network. Each prototype is included in a tuple that stores other relevant information, such as the number of observations described by a particular prototype and the point in time that a particular prototype was last updated. Similar data structures were popularized in [1] and are called *micro-clusters*. We use the same name, but store different information: we create q "weighted" prototypes of data stored in tuples $Q = \{\mathcal{M}_1, ..., \mathcal{M}_j, ..., \mathcal{M}_q\}$, each containing: a prototype of data P_j; the number of inputs patterns N_j it represents and a *timestamp* T_j that contains the point in time that prototype was last accessed. Hence, a micro-cluster is denoted by $\mathcal{M}_j = \{P_j, N_j, T_j\}$. The prototype together with the number of inputs it represents (the *prototype weight*) is important to preserve the input space density if one is interested in creating offline models of the underlying distribution. The timestamp allows the creation of models from specific intervals in time.

ART is a family of neural networks that develop stable recognition categories (clusters) by self-organization in response to arbitrary sequences of input patterns. Its fast commitment mechanism and capability of learning at moderate speed guarantees a high efficiency. The common algorithm used for clustering in any kind of ART network is closely related to the *k*-means algorithm. Both use single prototypes to internally represent and dynamically adapt clusters. The *k*-means algorithm clusters a given set of input patterns into *k* groups. The parameter *k* thus specifies the coarseness of the partition. In contrast, ART uses a minimum required similarity between patterns that are grouped within one cluster. The resulting number k of clusters then depends on the distances (in terms of the applied metric) between all input patterns, presented to the network during training. This similarity parameter is called *vigilance* ρ.

More formally, a data stream is a sequence of observations $x_1, ..., x_i, ..., x_n$ that are read once in increasing order of the indexes i. If each observation contains a set of d-dimensional features, then a data stream is a sequence of $X_1^d, ..., X_i^d, ..., X_n^d$ vectors. We employ an ART2-A [5] network specially geared towards fast one-shot training, with an important modification given our goals: constrain the network on a maximum of q prototypes. ART2-A networks are extensions of the original ART network to handle continuous real-valued features. It shares the basic processing of all ART networks, which is based on *competitive learning*. ART requires the same input pattern size for all patterns, i.e., the dimension d of the input space where the clusters regions shall be placed. Starting with an empty set of prototypes $P_1^d, ..., P_j^d, ..., P_q^d$ each input pattern X_i^d is compared to the j stored prototypes in a *search* stage, in a *winner-takes-all* fashion. If the degree of similarity between current input pattern and best fitting prototype P_c is at least as high as the vigilance parameter ρ, this prototype is chosen to represent the micro-cluster containing the input. Similarity between the input pattern i and a prototype j is given by Eq. (1),

where the distance is subtracted from one to get $S(X_i, P_j) = 1$ if input and prototype are identical. The distance is normalized with the dimension d of an input vector to keep measurements of similarity independent of the number of features.

$$S(X_i, P_j) = 1 - \sqrt{\frac{1}{d}\sum_{n=1}^{d}(X_i^n - P_j^n)^2} \tag{1}$$

The degree of similarity is limited to the range $[0, 1]$. If similarity between the input pattern and the best matching prototype does not fit into the vigilance interval $[\rho, 1]$, i.e., $S(X_i, P_j) < \rho$, a new micro-cluster has to be created, where the current input is used as the prototype initialization. Otherwise, if one of the previously committed prototypes (micro-clusters) matches the input pattern well enough, it is adapted by shifting the prototype's values towards the values of the input by the update rule in Eq. (2).

$$P_c^{(new)} = \eta \cdot X_i + (1 - \eta) \cdot P_c^{(old)} \tag{2}$$

The constant learning rate $\eta \in [0, 1]$ is chosen to prevent prototype P_c from moving too fast and therefore destabilizing the learning process. However, given our goals, i.e., to perform an adaptive vector quantization, we define η dynamically in such a way that the mean quantization error of inputs represented by a prototype is minimized. Eq. (3) establishes the dynamic value of η, where N_c is the current number of assigned input patterns for best fitting prototype P_c. This way, it is expected that the prototypes converge to the mean of the assigned input patterns.

$$\eta = \frac{N_c}{N_c + 1} \tag{3}$$

This does not guarantee the convergence to local minimum, however, according to the Adaptive Vector Quantization (AVQ) convergence theorem [4], AVQ can be viewed as a way to learn prototype vector patterns of real numbers; it can guarantee that average synaptic vectors converge to centroids exponentially quickly.

Another needed modification arises from the fact that ART networks, by design, form as much prototypes as needed based on the vigilance value. At the extremes, $\rho = 1$ causes each unique input to be encoded by a separate prototype, whereas $\rho = 0$ causes all inputs to be represented by a single prototype. Therefore, for decreasing values of ρ coarser prototypes are formed. However, to achieve exactly q prototypes solely on a manually tuned value of ρ is a very hard task, mainly due to the input space density, that can change over time, and is also different from application to application.

To overcome this, we make a modification to the ART-2 algorithm to impose a restriction on creating a maximum of q prototypes and dynamically adjusting the vigilance parameter. We start with $\rho = 1$ so that a new micro-cluster is assigned to each arriving input vector. After learning an input vector, a verification is made to check if $q = n + 1$, where n is the current number of stored micro-clusters. If this condition is met, then to keep only q micro-clusters we need to merge the *nearest pair* of micro-clusters. Let

$min\{\|P_r - P_s\|^2 : r, s = 1, ..., q, r \neq s\}$ be the minimum Euclidean distance between a pair of prototypes stored in micro-clusters $\mathcal{M}_r$ and $\mathcal{M}_s$. We merge these two micro-clusters using Eq. (4).

$$\mathcal{M}_{merge} = \{P_{merge}, N_r + N_s, max\{T_r, T_s\}\} \tag{4}$$

The new prototype, calculated with Eq. (5), is a "weighted" average between P_r and P_s, based on the number of samples each one represented at merge time.

$$P_{merge} = \frac{N_r}{N_r + N_s} P_r + \frac{N_s}{N_r + N_s} P_s \tag{5}$$

With d-dimensional input vectors, Eq. (1) defines a hypersphere around any stored prototype with radius $r = (1 - \rho) \cdot \sqrt{d}$. By solving this equation in respect to ρ, we update the vigilance parameter dynamically with Eq. (6), hence $\rho^{(new)} < \rho^{(old)}$ and the radius, consequently, increases.

$$\rho^{(new)} = 1 - \frac{T_{r,s}}{\sqrt{d}} \tag{6}$$

In Section 4.1 we experimentally show that this approach seems effective in providing a summarization of the underlying distribution within the data streams.

3.2. Detecting Concept Drift

Our method assumes that if the underlying distribution is stationary than the error-rate of the learning algorithm will decrease as the number of samples increases [15]. Hence, we compute the average quantization error (AQE) at each aggregation phase of the ART network and track the changes of these errors over time. We use a circular buffer $\mathcal{B}$ of b aggregation results, such that $\mathcal{B} = \{Q_l, Q_{l-1}, ..., Q_{l-b+1}\}$, where Q_l is the last aggregation obtained. For each Q_l that arrives, we compute the average Euclidean distance between each prototype in Q_l and the closest one in $\{Q_{l-1}, ..., Q_{l-b+1}\}$. This computes the error of the last aggregation in quantifying previous aggregations in a particular point in time.

By repeating this procedure over time, we obtain a series of errors that stabilizes and/or decreases when the underlying distribution is stationary and presents increases on this curve when the underlying distribution is changing, i.e., concept drift is occurring. Larger values of b are used to detect *abrupt* changes in the underlying distribution, whereas to detect *gradual* concept drift a lower value should be adopted. We exemplify concept drift detection with this method in Section 4.2.

3.3. Offline Model Creation

The offline module of the framework is not affected by the single-pass data stream restriction of the online module, since it is produced at user-request. Hence, the set of

micro-clusters used to produce the model compose a *dataset* used to train a SOM. The model can be built using the latest aggregation results, i.e., to model the current input space, or with the aggregation results obtained for a specific timespan given time clocks t_1 and t_2, i.e., micro-clusters with $t_1 \leq T_j \leq t_2$ are drawn from secondary storage to compose the dataset. This also allows the analysis of cluster evolution, which we talk about in Section 3.3.1. Hence, micro-clusters serve as training inputs to the SOM.

The SOM converts complex, nonlinear statistical relations between high-dimensional data into simple geometric relations in a, typically, 2D map, i.e., performs a projection of high-dimensional data in a much lower dimension. It is specially powerful for the visualization of high-dimensional data. However, the topology preservation of the SOM projection is of little use when using small maps. Emergent phenomena involve by definition a large number of nodes, i.e., at least a few thousands – Emergent SOMs (ESOM)[13]. An ESOM is essentially a SOM with a large number of nodes from which knowledge extraction can be performed through a variety of visualization techniques, e.g., U-Matrix, P-Matrix and U*-Matrix [13], to detect clusters and non-linear correlations between features in a visual form. This is a very different process than using k-means, with a major advantage of not being necessary to specify the number of clusters of interest; they arise naturally in the visualizations of the map. We provide an example of this in Section 4.3. Nevertheless, the knowledge can be extracted automatically by applying other algorithms on trained maps, e.g., SOM-Ward method to identify clusters [14] and the use of component planes to identify non-linear correlations between features [11].

The offline module uses the batch [10] algorithm of the SOM training procedure. We make a slight modification to the update rule, by enabling it to take into account the "weight" of a training prototype. A SOM is composed by a set of k neurons m_k^d arranged in a rectangular lattice. For each training sample the winning neuron c is found using the nearest-neighbor rule. Around this winning neuron a neighborhood kernel is computed by a Gaussian function h_{kc}, so the magnitude of the updates decrease with lateral distance. In the batch algorithm the whole selected training set of $\mathcal{M}_n$ micro-clusters is gone through at once and only after this the map is updated. Actually, the updating is done by simply replacing the neuron vector with a weighted average over the training samples P_j, where the weighting factors are the neighborhood function values h_{kc} and the "weight" of the training sample (derived from the corresponding micro-cluster) calculated from Eq. (7). This process is repeated several times until convergence.

$$\lambda(j) = \frac{N_j}{max(N)} \tag{7}$$

In Eq. (8) we present the modified batch rule used at each update of the map at time t. This enables the SOM to model the input space density based on the information contained in the micro-clusters.

$$m_k(t+1) = \frac{\sum_{j=1}^{n} \lambda(j) h_{kc(j)} P_j}{\sum_{j=1}^{n} \lambda(j) h_{ck(j)}} \tag{8}$$

From trained maps visualizations can then be performed, as we illustrate in Section 4.3 to detect clusters.

3.3.1. Evolution Analysis

Interesting changes in an evolving data stream can be interesting to an analyst in a number of business applications [2]. For example, a financial analyst may wish to know how the data changed over the last week, month, quarter and so on. For this purpose the user needs to input two clock times t_1 and t_2, where $t_2 > t_1$ and a time horizon h over which the micro-clusters are gathered, more specifically a dataset composed by micro-clusters from data between $(t_1 - h, t_1)$ and another from $(t_2 - h, t_2)$. Then, two SOM can be trained from these two datasets and comparisons made. In Section 4.3 we provide a simple example where it is possible to see the evolution of the detected number of clusters.

4. EXPERIMENTAL RESULTS

The presented experiments aim to evaluate the frameworks ability to: (i) summarize the data correctly; (ii) to detect concept drift and; (iii) to identify clusters in an evolving data stream. For these purposes we use the following datasets that are converted into data streams by taking the data input order as the order of the streaming and assuming that flow-in at constant speed:

Artificial datasets. To test the concept drift detection mechanism we use two artificial datasets. Dataset **Gaussian** is composed of 10K points describing a Gaussian distribution with mean zero and variance one; Dataset **Clouds**, depicted in Figure 2, describes three Gaussian clouds, that vary in mean and variance over time, in 15.300 samples. The later is used to perform cluster evolution analysis.

Real datasets. We use the UCI **Adult** dataset which, after some preprocessing, contains 30.148 samples and 106 features; The **EuroStoxx** dataset is composed by gathered stock values for 50 different commodities and companies of the *Euro Stoxx Index* from a 10-year period, containing 2.928 samples with 50 features.

The framework is implemented in the Java language and all experiments were done in the same hardware platform. Parameterizations are described with each experiment.

4.1. Aggregation Evaluation

These experiments aim at quantifying the quality of the data aggregation and its scalability given an increasing number of features. Table 1 shows the average quantization errors that the set of all generated prototypes of the micro-clusters exhibit in respect to the whole processed data. These values were taken as the mean values of five runs. The parameters used were a queue size $S = 1000$, varying the number q of generated micro-clusters for each block of S samples.

Table 2 shows the mean number of prototype merges, described in Section 3.1, for the experiment presented in Table 1. One can see that the datasets composed by Gaussian clouds are the ones that need more merges, mainly due to the fact that the first input patterns that arrive in the stream have a high probability of being in the center of the clouds, therefore committing initially prototypes in those areas. When other inputs corresponding to outer regions of the clouds arrive there is a lot of adjustments (merges) that are necessary, mainly in the center of the clouds. For the real datasets there are few

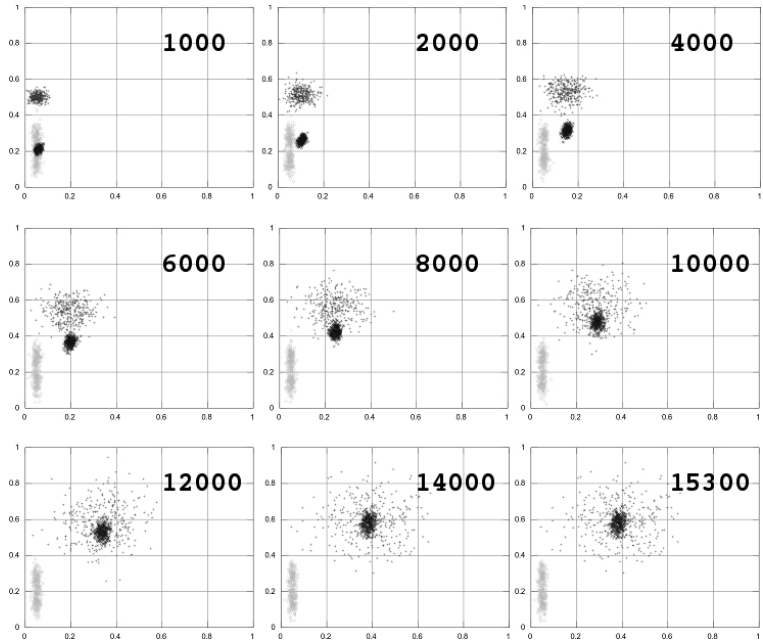


Figure 2. Evolution of clusters in the Clouds dataset. Each image was plotted with a thousand points and the numbering indicates the last input pattern used.

Table 1. Average quantization errors of all aggregation results over the entire stream for several values of q.

	Average Quantization Error			
Dataset	$q = 20$	$q = 50$	$q = 100$	$q = 200$
Gaussian	0.0168	0.0101	0.0067	0.0041
Clouds	0.0093	0.0059	0.0038	0.0022
EuroStoxx	0.0323	0.0195	0.0138	0.0092
Adult	0.1155	0.1091	0.0974	0.0663

merges needed, given that in high-dimensional spaces the density of input patterns is much lower.

Given that q is an important parameter of the framework, i.e., determines the granularity of the data aggregation performed over S input patterns at a time, we test the scalability of varying the value of q, while maintaining the value of $S = 1000$ fixed. To make this experiment with all datasets, we limited the number of input patterns used to the size of the EuroStoxx dataset (the smallest). Figure 3 shows the results and it is easy to see that linear increments in the value of q produce exponential increments in processing time. This is easily explained by the exponential number of computations needed for large values of d when computing distances. The relative processing times must be analyzed together with values in Table 2 to make sense, e.g., although the EuroStoxx dataset has approximately half the features of the Adult dataset, it produces more prototype merges, that cause a processing overhead especially visible with $q = 60$ and

Table 2. Mean number of prototype merges for all aggregations over the entire stream for several values of q.

	Mean number of merges			
Dataset	$q = 20$	$q = 50$	$q = 100$	$q = 200$
Gaussian	288.6	548.3	768.6	758.6
Clouds	801.6	866	848.6	775
EuroStoxx	52.6	70.6	94	119.3
Adult	4	6	13.6	32

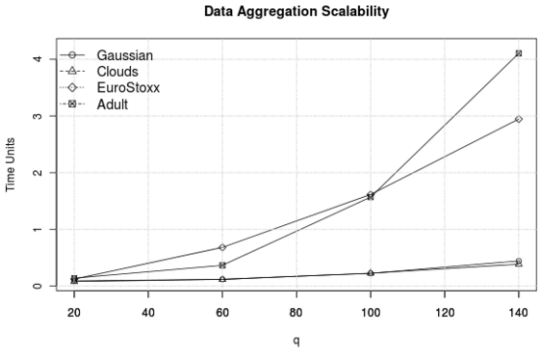


Figure 3. Scalability of the aggregation process for increasing values of q.

$q = 100$. The combination of these results show that using $S = 1000$ and $q \in [20, 50]$ should be appropriate for any stream.

4.2. Detection of Concept Drift

We experimentally performed concept drift detection on the Clouds dataset. We also provide the result on applying it to the stationary distribution of the Gaussian dataset. Figure 4 depicts the evolution of the AQE over the several aggregation results, using a circular buffer of size $b = 10$ (see Section 3.2). We experimentally obtained this value as a good compromise in detecting either abrupt and gradual drifts. For this experiment we used $S = 1000$ and $q = 50$ as we justified earlier. For the Gaussian dataset we see that the AQE is stable across time. This is an indication that no drift is occurring, which makes sense given that the distribution is stationary. However, in the Clouds dataset a progressive drift is initially occurring. By comparing the time of the aggregations curve with the dataset itself, one can verify that the concept drift is increasing while the darker cluster is moving towards the larger one (aggregation 9 corresponds to the dataset at the $9000th$ point). Once inside, it "dilutes" in the larger one and the drift lowers significantly while the dark cluster reaches the center of the larger one.

4.3. Clustering in Evolving Streams

In this experiment we show the clustering process performed by the ESOMs trained offline. Given a set of aggregation produced from the data stream, one can visualize

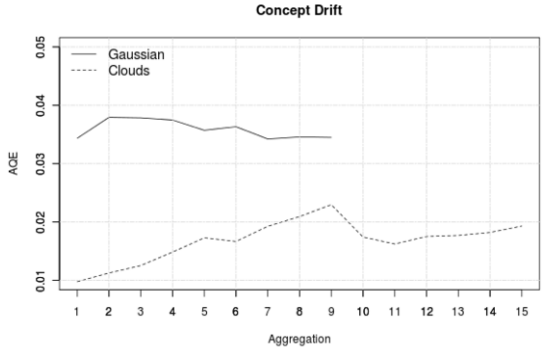
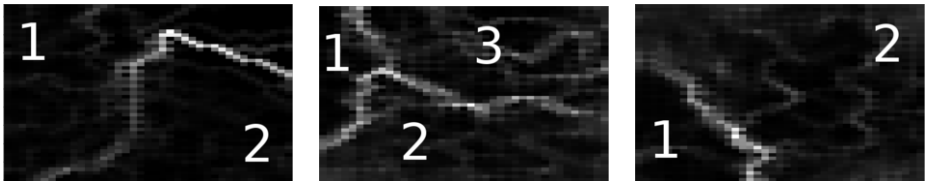


Figure 4. Concept drift in Gaussian and Clouds datasets.



(a) (b) (c)

Figure 5. U-Matrices of trained ESOM with aggregations from the Clouds dataset at different times. (a) Two approximately equal sized clusters visible. (b) Three clusters visible. Cluster 1 is smaller. (c) Two clusters visible. Cluster 2 is larger.

the clusters present in the underlying distribution in the time-frame defined by the set of aggregations. We illustrate this process using the Clouds dataset and changed the parameters to $S = 200$ and $q = 20$. This is because we're dealing with a relatively small amount of input vectors, when compared to real streams, and we need a sufficient amount of prototypes to train the ESOM.

We extracted the aggregation results produced from the Clouds dataset (please see Figure 2 since we reference the numbers in the images) in three particular points in time: *(a)* early on, when the clouds are stationary, i.e., aggregations produced until the $1000th$ input pattern; *(b)* when the darker cluster is moving, i.e., aggregations from the $2000th$ to the $8000th$ input pattern and; *(c)* when the darker cluster is absorbed, i.e., aggregations from the $12000th$ to the $15300th$ input pattern. Given the described parameterization, the ESOM for case (1) will be trained with 100 "weighted" prototypes extracted from the corresponding micro-clusters, for example. All ESOM are of dimension 20×25 and are trained for 50 epochs, i.e., 50 presentations of the training data (the micro-clusters). Figure 5 shows the U-Matrices for the three models. The U-Matrix [13] is a special type of visualization where lighter colors represent cluster separation. It is clearly visible that the ESOMs detect the expected number of clusters at those time-frames.

5. CONCLUSIONS

We propose a methodology and modular framework to cluster data streams and extract other relevant knowledge. The methodology is based on the micro-clustering technique which uses an online module to produce an aggregation summary of the incoming data stream and an offline module to generate the corresponding model. Empirical results with both synthetic and real data confirm the validity of the model and present promising research on future application of ANN models to data streams.

Future work will address the efficient storage of the aggregation results; the Normalization module; cluster evolution analysis, i.e., to automatically check if clusters in the underlying stream appeared, were merged or destroyed. Also, the SOM algorithm is time-dependent, i.e., some parameters are decreased monotonically in order for the map to converge. This decreases the *plasticity* of the network during training and limits its use in real-time applications. A real-time SOM capable of incorporating training data on-the-fly is our next path of research. Such a network could be always representing the underlying distribution and the evolution of clusters visualized in real-time.

References

[1] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, 'A framework for clustering evolving data streams', in *Proceedings of the 29th International Conference on Very Large Databases*, volume 29, pp. 81–92. Morgan Kaufmann Publishers Inc., (2003).

[2] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, 'A framework for projected clustering of high dimensional data streams', in *Proceedings of the Thirtieth International Conference on Very Large Databases*, volume 30, pp. 852–863. Morgan Kaufmann Publishers Inc., (2004).

[3] J. Gama, *Knowledge discovery from data streams*, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, (2010).

[4] K. Bart, *Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence*, Prentice-Hall of India, (1997).

[5] G.A. Carpenter, S. Grossberg, and D.B. Rosen, 'Art 2-a: An adaptive resonance algorithm for rapid category learning and recognition', *Neural networks*, **4**(4), 493–504, (1991).

[6] Douglas de O. Cardoso, Priscila M. V. Lima, Massimo De Gregorio, João Gama, and Felipe M. G. França, 'Clustering data streams with weightless neural networks', in *ESANN*, (2011).

[7] P. Domingos and G. Hulten, 'A general method for scaling up machine learning algorithms and its application to clustering', in *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 106–113. Morgan Kaufmann Publishers Inc., (2001).

[8] F. Farnstrom, J. Lewis, and C. Elkan, 'Scalability for clustering algorithms revisited', in *ACM SIGKDD Explorations Newsletter*, volume 2, pp. 51–57. ACM, (2000).

[9] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, 'Clustering data streams: Theory and practice', *IEEE Transactions on Knowledge and Data Engineering*, 515–528, (2003).

[10] Teuvo Kohonen, *Self-Organizing Maps*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, (2001).

[11] B. Silva and N. Marques, 'Feature clustering with self-organizing maps and an application to financial time series portfolio selection', in *International Conference on Neural Computation*, (2010).

[12] Alfred Ultsch, 'Self-Organizing Neural Networks for Visualization and Classification', in *Information and Classification*, (1993).

[13] Alfred Ultsch, 'Maps for the visualization of high-dimensional data spaces', in *In Proceedings Workshop on Self-Organizing Maps (WSOM 2003)*, pp. 225–230, (2003).

[14] J. Vesanto and E. Alhoniemi, 'Clustering of the self-organizing map', *Neural Networks, IEEE Transactions on*, **11**(3), 586–600, (2000).

[15] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, 'Learning with drift detection', *Advances in Artificial Intelligence–SBIA 2004*, 66–112, (2004).