Adaptive Reservoir Neural Gas: An Effective Clustering Algorithm for Addressing Concept Drift in Real-Time Data Streams

Konstantinos Demertzis[1(✉)], Lazaros Iliadis[2], and Antonios Papaleonidas[2]

[1] School of Science and Technology, Informatics Studies, Hellenic Open University, Patra, Greece
demertzis.konstantinos@ac.eap.gr

[2] School of Engineering, Department of Civil Engineering, Faculty of Mathematics Programming and General Courses, Democritus University of Thrace, Kimmeria, Xanthi, Greece
liliadis@civil.duth.gr

Abstracts. The concept drift phenomenon describes how the statistical properties of a data distribution change over time. In cybersecurity domain, where data arrives continuously and rapidly in a sequential manner, concept drift can be a significant challenge. Identifying concept drift, it enables security analysts to detect emerging attacks, respond promptly, and make informed decisions based on the changing nature of the data being analyzed. The Adaptive Reservoir Neural Gas (AR-NG) clustering algorithm is proposed in this paper to handle concept drift in real-time data streams. It is a novel approach that combines reservoir computing power with the neural gas algorithm, allowing the algorithm to automatically update its clustering structure as new data arrives. Furthermore, in order to effectively handle evolving data streams that significantly change over time in unexpected ways, the proposed method incorporates a density-based clustering mechanism (DBCM) to concept drift detection. Experiments on real-time data streams show that the proposed algorithm is effective at mitigating the impact of concept drift, making it a useful tool for real-time data analysis and decision-making in dynamic environments.

Keywords: Reservoir Computing · Neural Gas · Concept Drift · Data Streams

1 Introduction

The phenomenon of concept drift describes how the statistical properties of a data distribution change over time [1]. Concept drift can be a significant challenge in data stream mining, where data arrives continuously and rapidly in a sequential manner. The mean, variance, correlation, and probability distribution of the data are all statistical properties of the data distribution. When there is concept drift, these properties change, causing shifts in the underlying patterns, relationships, and structures of the data. Changes in user behavior, evolving trends, seasonality, or external factors influencing the data generation process can all cause concept drift. It can be gradual, with changes occurring gradually over time, or abrupt, with changes occurring suddenly and significantly [2].

L. Iliadis et al. (Eds.): ICANN 2023, LNCS 14259, pp. 152–166, 2023.
https://doi.org/10.1007/978-3-031-44223-0_13

Concept drift complicates data analysis and mining because models trained on historical data may become outdated and less accurate in predicting or clustering new data instances. It is critical to detect and adapt to changing data distributions in order to maintain model performance and validity in the presence of concept drift. On the other hand, in the cybersecurity domain, identifying concept drift is crucial because it can help detect and respond to evolving threats and attacks [3].

Specifically, by identifying concept drift, cybersecurity systems can adapt and learn from new patterns or behaviors that emerge over time. This enables the early detection of emerging threats that might go undetected by traditional rule-based or signature-based systems. Moreover, concept drift detection allows intrusion detection systems (IDS) to detect changes in the patterns of malicious behavior, helping to identify new attack vectors or evasion techniques [4]. Also, by dynamically adapting security controls based on concept drift, organizations can better protect their systems and networks. Also, when concept drift is detected, it may indicate a potential security breach or an ongoing attack. Rapid response can help mitigate the impact of the attack, contain the incident, and prevent further damage or data exfiltration [5]. Furthermore, by understanding how the data distribution or behavior has changed, security analysts can make more informed decisions regarding incident prioritization, resource allocation, and the selection of appropriate defensive measures [6]. Finally, concept drift detection can trigger the retraining or recalibration of machine learning cyber-defense models to adapt to new data distributions and ensure accurate and up-to-date detection [7].

The AR-NG clustering algorithm is used in this paper to present an innovative approach to effectively address concept drift in real-time data streams. AR-NG combines reservoir computing's strengths with the Neural Gas algorithm to create a powerful and adaptive clustering solution that can update its structure as new data arrives.

Reservoir computing is a computational framework that processes input data using a fixed, randomly initialized dynamic reservoir. The Neural Gas algorithm, on the other hand, is a clustering technique that divides data into groups based on similarities. AR-NG gains the ability to adaptively update its clustering structure in response to changes in the data distribution by incorporating reservoir computing into the Neural Gas algorithm.

AR-adaptive NG's nature enables it to capture the evolving patterns and relationships within the real-time data streams. AR-NG dynamically adjusts its clustering structure to reflect the current data distribution as new data points arrive, ensuring that the clusters remain accurate and up to date.

Also, by incorporating the density-based clustering mechanism, the proposed method enhances concept drift detection in evolving data streams. It is designed to discover clusters of arbitrary shape in a dataset, based on the density of data points in the feature space. This approach allows the method to effectively detect concept drift and adapt the clustering structure to handle unexpected changes in the data distribution.

The methodology of AR-NG algorithm is described in detail below.

2 Methodology

The AR-NG algorithm is designed to handle concept drift in real-time data streams by combining reservoir computing with the neural gas algorithm and incorporating a density-based clustering mechanism using DBCM. Specifically, reservoir computing is

a computational framework that processes sequential data using a fixed-size dynamic reservoir. The reservoir is a network of neurons that is randomly connected and serves as a computational resource [8]. The reservoir computing approach is used in the AR-NG algorithm to deal with the dynamic nature of data streams. The reservoir computing architecture consists of the input, reservoir, and output layers. The connection and input weights are chosen at random [9]. The reservoir weights are scaled so that the Echo State Property (ESP) is maintained, which is defined as the state in which the reservoir is a "echo" of the entire input history [10]. The discrete layers are only those of input $u(n)$ and output $y(n)$ as they are defined by the problem. The hidden layers are clustered in an region, and their number is indistinguishable. The neurons $x(n)$, are connected by some percentage, which determines how sparsity the reservoir computing will be [11].

The synaptic associations that link the levels together and the reservoir computing are characterized by a value that identifies the weights [12]. In the proposed system, each input neuron is connected via ${W^{in}}_{ij}$ weights (i-input neuron, j-neuron) to each neuron from the reservoir computing. Although normalized, these weights are determined randomly before training, and their values are the final ones as they do not change during training. Also, each neuron is connected to each other neuron, via weights W_{jk} (j-neuron, k-neuron, and $j \neq k$). The respective weights, although normalized, are randomly determined before training and their values do not change. We use $x^{(l)}(t) \in R^{N_R}$ to declare the status of level l at time t. By omitting the bias conditions, the first level state transition function is defined by the following equations [13]:

$$x^{(1)}(t) = \left(1 - a^{(1)}\right)x^{(1)}(t-1) + a^{(1)} \tanh\left(W_{in}u(t) + \hat{W}^{(1)}x^{(1)}(t-1)\right)$$

For each level higher than $l > 1$ equation 1, has the following form (2) [8, 14]:

$$x^{(l)}(t) = \left(1 - a^{(l)}\right)x^{(l)}(t-1) + a^{(l)} \tanh\left(W^{l}x^{l-1}(t) + \hat{W}^{(l)}x^{(l)}(t-1)\right)$$

where $W_{in} \in R^{N_R \times N_U}$ is the input weight matrix, $\widehat{W}^{(l)} \in R^{N_R \times N_R}$ is the recurrent weight matrix for layer l, $W^{(l)} \in R^{N_R \times N_R}$ is the matrix containing the connection weights between layer l–1 and l, $a^{(l)}$ is the leaky parameter of layer l and *tanh* is the Tangent Hyperbolic function. Finally, each reservoir computing neuron is connected via ${W^{out}}_{jm}$ weights (j-neuron, m-neuron input) to the neurons in the output layer. The weights, located in the readout layer, are the only ones trained to get their final values [15, 16].

The Neural Gas algorithm is a competitive learning algorithm used for clustering. It organizes data points into clusters based on their similarity [17]. In AR-NG, the Neural Gas algorithm is combined with reservoir computing to create an adaptive clustering structure that can handle concept drift. Competitive learning neural networks include a competitive layer comprising of Competitive Neurons (CNE). Every CNE_i is characterized by a weight vector $w_i = (w_{il,\ldots,}w_{id})^T$, $i = 1,\ldots, M$ and it estimates a similarity measure with the input data vector $x_i = (x_{il,\ldots,}x_{id})^T$ $x \in R$. For every input vector that is introduced to the network there is a competition between the CNE for the determination of the winning neuron. The winner is the neuron that has the higher degree of similarity between the input vector and its assigned weight vector. The output of the winning CNE is set to $o_m = 1$, whereas for the rest of the neurons the output is $o_i = 0$, $i = 1,\ldots,M$,

$i \neq m$. The default similarity function used is the inverse value of the actual Euclidean distance $\|x - w_i\|$ between the input vector x^n and the weight vector w_i [18].

Initially, the algorithm initializes the clustering structure using the Neural Gas algorithm. This involves randomly selecting initial cluster centers and updating them based on the similarity between data points and cluster centers [19]. The initialization step creates an initial set of clusters to start with. These clusters are recreated with the algorithm calculating the density around each data point. This is achieved by counting the number of points in a user-defined neighborhood (*Eps-Neighbourhood*) with the definition of thresholds [20, 21]. The purpose is to locate points in the center of the areas (core), on their borders (border), and points that involve noise (noise). The extra data points are added to the center of the regions if they are densely accessible, i.e., there is a chain of core points where each one belongs to the neighborhood (*Eps-Neighborhood*) of the next point and therefore to distinguish the extreme values for each cluster.

Thus, the user can see the streams in different periods. Specifically, the neighborhood area of a point p is defined as the set of points for which the *Euclidean* distance between the points p, q is smaller than the parameter [22, 23]:

$$N_{Eps}(p) = \{q \in D \mid dist(p, q) \leq Eps\}$$

provided that $p = (p1, p2)$ and $q = (q1, q2)$, the Euclidean distance is defined as:

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

So, a point p is considered to be reachable from a point q based on a density determined by the parameters *Eps*, *MinPts* if:

$$p \in N_{Eps}(q) \text{ and } N_{Eps}(q) \geq MinPts$$

Having calculated the cluster density, the kernel methods' property is exploited to express several samples through a symmetric and positive definite matrix, according to the similarity of two samples in each position. Thus, the array elements are in a linear space, regardless of the space they come from. This allows the introduction of one more function in the processing stage, which exploits the nonlinear structure of the features to present them as a sparse representation problem modified to work in a *Reproducible Kernel Hilbert Space* (RKHS) [24].

Specifically, as the data is in a *Euclidean* space, the problem of the sparse representation of an *L-dimensional vector* $u \in \mathbb{R}^L$ and a basis $D = [D_1, \ldots, D_m] \in \mathbb{R}^{L \times m}$ The following relationship defines it [25]:

$$\ell(h, D) := \min_a \left\| h - \sum_{i=1}^{m} D_i a_i \right\|_2^2 + \lambda \|a\|_1$$

The sparseness of this vector is required. In the above relation, the vector $a \in \mathbb{R}^m$ Represents the sparse coefficients, and the function $\ell(h, D)$ is the optimal approximation of the problem. The above relationship between the sample distances has to be modified to a nonlinear one. For this purpose, we first define a mapping function $\varphi : \mathbb{R}^L \to \mathcal{H}$. This

function maps the samples from the original space to a new Hilbert space equipped with the inner product. Therefore, using the inner product, two groups of samples, u_i, $u_j \in \mathbb{R}^L$, are mapped to a new space using the relation [26]:

$$k(u_i, u_j) = \varphi(u_i)^T \cdot \varphi(u_j) \in \mathbb{R}$$

Similarly, a set of m sequences can be expressed as a Kernel Matrix $K \in \mathbb{R}^{m \times m}$ Whose elements express the similarities between the samples. The similarity function used herein is the *Radial Basis Function* (RBF), whose variance has been estimated as the mean value of the distances of the training data [27].

But the relation of sparse coding, as presented, does not allow to work with the K register. For this reason, a modified version has been introduced, which offers the potential to work in this space based on the following equation [28]:

$$\ell(\hat{h}, \hat{D}) := \min_a \left\| \hat{h} - \hat{D}a \right\|_2^2 + \lambda \|a\|_1$$

To fully utilize the intrinsic properties of sparse representations in the above space, the proposed methodology introduces the concept of a *Spatial Pooler*, which normalizes sparse input representations by enriching the input representation with its temporal context. Specifically, the Spatial Pooler aims to prepare sparse input representations for further processing, ensuring that inputs similar to each other (have high overlap and thus high coherence) produce output vectors similar to each other. So, each input pattern is encoded by the *Spatial Pooler* into sparse representations represented as a set of A_k indices of the given pattern at iteration k. [29].

At each step, the similarity between the sparse input representations at step k and step $k + 1$ is calculated by the equation [30]:

$$s = \frac{|A_k \cap A_{k+1}|}{max(|A_k|, |A_{k+1}|)}$$

Similarity s is defined as the ratio between the number of elements (cardinality) of the same active clusters in sparse representations generated in steps k and $k + 1$ and a maximum amount of data in two comparable steps. *Spatial Pooler* is usually stable if the sparse representations of the same pattern do not change for its entire life cycle. In this case, the similarity s between all representations of the same pattern is 100%.

As the data stream arrives continuously and rapidly, the AR-NG algorithm processes each incoming data point sequentially. For each data point, the algorithm performs the following steps:

1. Neuron Activation: The reservoir neurons are activated based on the incoming data point. The activation level of each neuron in the reservoir is determined by its similarity to the current data point.
2. Competitive Learning: The activated neurons compete with each other to become the winning neuron. The winning neuron is the one with the highest activation level.
3. Clustering Initialization. Clustering initialization refers to the process of setting up the initial state of a clustering algorithm before the actual clustering procedure begins. It involves determining the initial cluster assignments or centroids for the data points being clustered. Proper initialization is important as it can significantly impact the quality and convergence speed of the clustering results.

4. Update Clustering Structure: The winning neuron and its neighboring neurons in the topological order are updated to adapt to the incoming data point. This step allows the clustering structure to dynamically evolve and adjust to concept drift.
5. Density-Based Clustering: After updating the clustering structure, a density-based clustering mechanism is employed to detect concept drift. This mechanism identifies clusters based on the density of data points. It assigns data points to clusters and identifies outliers as noise points. By analyzing the density-based clusters, the algorithm can detect significant changes in the data distribution, indicating the presence of concept drift.
6. Concept Drift Handling: If concept drift is detected, the algorithm can take appropriate actions to handle it. For example, it may merge or split clusters, update cluster centers, or adjust the clustering parameters to adapt to the new data distribution.

By combining reservoir computing, the Neural Gas algorithm, and DBCM for density-based clustering, the AR-NG algorithm provides an effective solution for handling concept drift in real-time data streams. It adapts to changing data distributions, maintains accurate clustering structures, and enables real-time analysis in dynamic environments.

The pseudocode that presented in the Appendix 1 is a high-level representation of the AR-NG algorithm. This pseudocode provides a high-level overview of the steps involved in the code, including initialization, data processing in batches, clustering, performance evaluation, concept drift detection, handling, analysis, and plotting of results.

3 Dataset and Results

Factry.io and InfluxDB were used to collect and store data about the industrial environment, such as programmable PLC controllers, SCADA systems, and construction equipment, in order to create a perfect test-bed environment for the proposed algorithm. They are created using measurements or events tracked over time, such as transactions, application performance monitoring, and server analytics [31].

The scenario was focused on collecting time series data from sensors that measure quantifiable values in an hourly manner. These sensors can be monitoring different parameters related to industrial machines' conditions and specifically to a raw water storage tank, equipped with a water level sensor and a valve. The system is configurated such that the valve opens when the water level detected by the sensor is less than or equal to 0.5 m and closes when the level is higher than 0.8 m. Additionally, included a pump that operates based on pressure levels separated by a semipermeable membrane. The pump acts as a safety device and shuts off if the water level falls below 0.25 m.

An attacker aims to manipulate the system by modifying the sensor and actuator information by creating packets to alter the sensor readings and actuator behavior. He exploits the fieldbus communication protocol to change the functionality of the devices without raising suspicions from typical detection systems that look for irregularities.

One year of data was gathered, to create a data stream that is hourly quantifiable values from the sensors, reflecting the machine's condition over time. The concept drift was related by modifying the sensor settings and actuator behavior according to the

attacker's intentions. This drift can occur gradually or in sudden changes, mimicking the evolving behavior of the compromised system.

For the experiments, configured the system to send the collected data as a data stream. In the scenario described, the data stream has the following properties [32]:

1. Batch Size: The batch size refers to the number of data points collected and processed at a given time. In the provided scenario, each hour's data is considered as a batch.
2. Time Granularity: The time granularity refers to the resolution or interval at which the data is collected and recorded. In this scenario, the time granularity is specified as hourly quantifiable values from the sensors. This means that the data points are collected and recorded at hourly intervals.
3. Data Source: The data source in this scenario is industrial sensors that monitor various parameters related to machine conditions. The sensors provide quantifiable values reflecting the state of the machines and are collected using the appropriate infrastructure.
4. Data Format: Since the data is collected from sensors, it is numerical data representing measurements or readings from those sensors.
5. Concept Drift: The concept drift refers to changes or shifts in the underlying data distribution over time. In this scenario, concept drift is introduced by the attacker who modifies the sensor and actuator information. This leads to changes in the behavior of the system, which is reflected in the data stream. The concept drift is intentional and occurs gradually (medium), abruptly (high) random with very frequent changes (chaotic) depending on the attacker's actions.

After each data point is processed, the clustering performance is evaluated using several evaluation metrics. Here are the evaluation metrics used in the code:

1. Adjusted Rand Index (ARI): The adjusted Rand index measures the similarity between the true cluster assignments and the predicted cluster assignments, taking into account all pairs of samples and their respective cluster assignments. A value of 1 indicates a perfect clustering, while a value close to 0 suggests random clustering. With overlapping entries of different clusters of modeled clustering C^m and real clustering C^r, ARI can be computed as follows [22]:

$$ARI(C^r, C^m) = \frac{\Sigma_{ij} \frac{n_{ij}}{2} - \frac{\Sigma_i a_2^{a_i} \Sigma_2^{b_j}}{2}}{\frac{1}{2}\Sigma_i a_2^{a_i} + \Sigma_j a_2^{b_j} - \frac{\Sigma_i a_i \Sigma_j b_j}{n}}$$

where, n_{ij} is the number of nodes that are present in both cluster C_i^m and C_j^r, a_i is the summation of all n_{ij} corresponding to any C_j^r of C^r and all C_i^m of C^m, and b_j is the summation of all n_{ij} corresponding to any C_i^m of C^m and all C_j^r of C^r.

2. Silhouette Coefficient: The silhouette coefficient measures how well each sample in a cluster is separated from samples in other clusters. It computes the mean distance between a sample and all other points in the same cluster (a) and the mean distance between the sample and all other points in the nearest neighboring cluster (b). The silhouette coefficient ranges from –1 to 1, where a value close to 1 indicates well-separated clusters, 0 indicates overlapping clusters, and –1 indicates incorrect cluster

assignments. Specifically, silhouette score for a datapoint i is given as [22]:

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)} & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1 & \text{if } a(i) > b(i) \end{cases}$$

where bi is the inter cluster distance defined as the average distance to closest cluster of datapoint i except for that it's a part of:

$$b_i = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

where ai is the intra cluster distance defined as the average distance to all other points in the cluster to which it's a part of:

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j)$$

3. Calinski-Harabasz Index: The Calinski-Harabasz index, also known as the variance ratio criterion, measures the ratio between the within-cluster dispersion and the between-cluster dispersion. It evaluates the compactness and separation of clusters, where a higher index value indicates better-defined clusters. The Calinski-Harabasz index is calculated as [22]:

$$CH = \frac{\frac{BGSS}{K-1}}{\frac{WGSS}{N-K}} = \frac{BGSS}{WGSS} \times \frac{N-K}{K-1}$$

where N is total number of observations, K is total number of clusters and

$$BGSS = \sum_{k=1}^{k} n_k \times \|C_k - C\|^2$$

where n_k is the number of observations in cluster k, C_k is the centroid of cluster k, C is the centroid of the dataset (barycenter) and K is the number of clusters,

$$WGSS_k = \sum_{i=1}^{n_k} \|X_{ik} - C_k\|^2$$

where n_k is the number of observations in cluster k, X_{ik} is the i-th observation of cluster k, C_k is the centroid of cluster k and then sum all individual within group sums of squares:

$$WGSS = \sum_{k=1}^{K} WGSS_k$$

where $WGSS_k$ is the within group sum of squares of cluster k and K is the number of clusters.

4. Davies-Bouldin Index: The Davies-Bouldin index measures the average similarity between each cluster and its most similar cluster, taking into account both the size and the separation between clusters. A lower index value indicates better clustering, where 0 indicates perfectly separated clusters. Calculate the Davies-Bouldin index as [22]:

$$\overline{D} = \frac{1}{N}\sum_{i=1}^{N} D_i$$

where Di chooses the worst-case scenario, and this value is equal to Ri,j for the most similar cluster to cluster i:

$$R_{ij} = \left\|A_i - A_j\right\|_p = \left(\sum_{k=1}^{n} \left|a_{k,i} - a_{k,j}\right|^p\right)^{\frac{1}{p}}$$

$R_{i,j}$ is a measure of separation between cluster C_i and cluster C_j and $a_{k,i}$ is the k th element of A_i, and there are N such elements in A for it is an n dimensional centroid.

To evaluate the method and prove its superiority, 3 different data streams with varying difficulty concept drift were used (medium, high, and chaotic) in which a comparison was made with corresponding competing algorithms, namely Density-based spatial clustering of applications with noise (DBSCAN), Online K-Means, CluStream and Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH). The results are presented in the following Table 1.

These results indicate the performance of different methods in handling concept drift in Data Streams. Higher values for ARI and Silhouette indicate better clustering quality, while higher values for Calinski-Harabasz and lower values for Davies-Bouldin indicate better cluster separation.

For Data Stream 1 (medium), AR-NG method achieved the highest scores across all metrics, indicating good performance in handling concept drift. It has the highest ARI and Silhouette Score, and it also performs well in the Calinski-Harabasz and Davies-Bouldin measures. DBSCAN, Online K-Means, CluStream, and BIRCH methods also show competitive performance, but slightly lower than AR-NG in terms of ARI and Silhouette Score.

For Data Stream 2 (high), the AR-NG method maintains its relatively high performance, although the scores decrease compared to Data Stream 1. It still outperforms other methods in most metrics. DBSCAN, Online K-Means, CluStream, and BIRCH methods show comparable performance, but they exhibit significant lower scores compared to AR-NG.

For Data Stream 3 (chaotic), all methods experience a significant drop in performance across all metrics. This is likely due to the introduction of chaotic concept drift, which poses challenges for the methods to adapt and accurately cluster the data. AR-NG method still performs better than other methods, but its scores decrease substantially compared to the previous data streams. DBSCAN, Online K-Means, CluStream, and BIRCH methods also show a decrease in performance, with DBSCAN having the lowest scores among them.

Table 1. Performance Results

Data	Method	Scores			
		ARI	Silhouette	Calinski-Harabasz	Davies-Bouldin
Data Stream 1 (Medium)	AR-NG	0.82837	0.694837	1513	0.215832
	DBSCAN	0.80212	0.669901	1487	0.270023
	Online K-Means	0.79441	0.660725	1469	0.268830
	CluStream	0.79092	0.658722	1457	0.283102
	BIRCH	0.78996	0.642399	1401	0.308972
Data Stream 2 (High)	AR-NG	0.78401	0.61774	1408	0.312113
	DBSCAN	0.74392	0.55973	1251	0.360921
	Online K-Means	0.74489	0.56130	1269	0.359872
	CluStream	0.73825	0.54671	1204	0.373321
	BIRCH	0.74003	0.55682	1238	0.359283
Data Stream 3 (Chaotic)	AR-NG	0.42007	0.36093	1102	0.380049
	DBSCAN	0.32901	0.29087	857	0.499208
	Online K-Means	0.31992	0.30011	799	0.535561
	CluStream	0.33459	0.26994	865	0.523009
	BIRCH	0.35022	0.27690	888	0.501297

Overall, the results suggest that the AR-NG method demonstrates better adaptability to concept drift compared to the other methods, at least in the provided scenarios. These performance metrics suggest that the clustering algorithm is accurately capturing the underlying structure of the data, with well-separated and meaningful clusters.

4 Conclusion

Identifying concept drift in the cybersecurity domain is essential for staying ahead of evolving threats, adapting security measures, and maintaining the effectiveness of cybersecurity systems. It enables organizations to detect emerging attacks, respond promptly, and make informed decisions based on the changing nature of the data being analyzed. The Adaptive Reservoir Neural Gas algorithm presented in this paper offers a novel and effective solution for handling concept drift in real-time data streams.

By combining the power of reservoir computing and the Neural Gas algorithm, AR-NG achieves adaptive clustering that efficiently captures the evolving data distribution. Specifically, the use of reservoir computing in the clustering algorithm provides a powerful framework for efficient and parallel processing of input data streams, making it suitable for real-time or online applications. Also, the algorithm utilizes DBCM, a density-based clustering mechanism, which can effectively discover clusters of arbitrary shapes and handle noise in the data without require specifying the number of clusters in advance.

The key important is that the algorithm incorporates adaptive parameters to handle concept drift in the data stream, which can significantly impact the clustering results. By adapting the reservoir parameters based on concept drift detection, the algorithm can maintain accurate and up-to-date clustering models. This achieves by employs competitive learning to find the winning neuron among the activated reservoir neurons in order to identify the most representative neurons for each data point, leading to improved clustering accuracy and separation.

As proven experimentally the proposed streaming approach is memory-efficient since it does not require storing the entire data stream in memory. Instead, it processes data points sequentially, updating the clustering model and adapting the parameters incrementally as new data arrives. The results of this study provide evidence of its efficacy, positioning AR-NG as a valuable tool for dynamic environments requiring real-time data analysis and decision-making.

While the above algorithm has its benefits, it also has some limitations and areas for future improvement. Particularly, as the number of data points increases, the processing and memory requirements of the algorithm may become a bottleneck. Scaling up the algorithm to handle big data efficiently would be a challenge. In addition, the algorithm relies on manually setting parameters such as the size of the reservoir, DBCM parameters (epsilon and min_samples), and the number of neighbors. Finding the optimal parameter values can be a challenging task, and these values may vary depending on the characteristics of the data stream. A more automated approach for parameter tuning would enhance the algorithm's performance and applicability. From this point of view, investigating techniques to parallelize the algorithm's computations could improve its efficiency and scalability. This could involve utilizing parallel processing frameworks or distributed computing approaches to handle the processing of data points in parallel, making the algorithm more suitable for larger-scale data streams.

Appendix 1

Pseudocode of the proposed AR-NG methodology

```
# Constants
RESERVOIR_SIZE = 100
FEATURE_SIZE = 2
# Function to initialize parameters
initialize_parameters()
# Function to initialize clustering structure using Neural Gas
initialize_neural_gas()
# Function to calculate activation level based on similarity
calculate_activation_level()
# Function to activate reservoir neurons based on data point
activate_reservoir_neurons()
# Function to perform competitive learning and find the winning neuron
competitive_learning()
# Function to update clustering structure based on winning neuron and neighbors
update_clustering_structure()
# Function to update a neuron's cluster and position
update_neuron()
# Function to generate data stream
generate_data_stream()
# Function to evaluate clustering performance
evaluate_clustering_performance()
# Main function
if __name__ == '__main__':
    # Initialize parameters
    reservoir = initialize_parameters()
    initialize_neural_gas(reservoir)
    # Data stream
    data_stream = generate_data_stream(1000)
    # Initialize variables
    clusters = []
    ari_scores = []
    silhouette_scores = []
    calinski_harabasz_scores = []
    davies_bouldin_scores = []
    # Generate true labels for evaluation
    true_labels = generate_true_labels()
    # Perform online clustering
    batch_size = 100
    num_batches = length(data_stream) / batch_size
    for batch_idx in range(num_batches):
        start_idx = batch_idx * batch_size
        end_idx = start_idx + batch_size
        # Process a batch of data points
        for i in range(start_idx, end_idx):
            data_point = data_stream[i]
            # Activate reservoir neurons
            activate_reservoir_neurons(reservoir, data_point)
            # Perform competitive learning and update clustering structure
            winning_neuron = competitive_learning(reservoir)
            update_clustering_structure(winning_neuron)
```

```
            # Store cluster assignment
            clusters[i] = winning_neuron['cluster']
        # Evaluate clustering performance for the current batch
        ari, silhouette, calinski_harabasz, davies_bouldin = evaluate_clustering_performance(
            true_labels[start_idx:end_idx], clusters[start_idx:end_idx], reservoir[:end_idx]
        )
        if ari is not None:
            ari_scores.append(ari)
            silhouette_scores.append(silhouette)
            calinski_harabasz_scores.append(calinski_harabasz)
            davies_bouldin_scores.append(davies_bouldin)
        # Detect and handle concept drift for the current batch
        if detect_concept_drift(clusters[:end_idx]):
            handle_concept_drift(clusters[:end_idx])
        # Perform analysis and decision-making for the current batch
        perform_analysis(clusters[:end_idx])
        # Plot clustering result for the current batch
        plot_clusters(data_stream[:end_idx], clusters[:end_idx])
    # Plot ARI scores over batches
    plot_ari_scores(ari_scores)
    # Plot silhouette scores over batches
    plot_silhouette_scores(silhouette_scores)
    # Plot Calinski-Harabasz scores over batches
    plot_calinski_harabasz_scores(calinski_harabasz_scores)
    # Plot Davies-Bouldin scores over batches
    plot_davies_bouldin_scores(davies_bouldin_scores)
```

References

1. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: a review. IEEE Trans. Knowl. Data Eng. **31**, 2346–2363 (2018). https://doi.org/10.1109/TKDE.2018.2876857
2. Yu, H., Liu, T., Lu, J., Zhang, G.: Automatic learning to detect concept drift. arXiv:arXiv:2105.01419 (2021). https://doi.org/10.48550/arXiv.2105.01419
3. Liu, A., Zhang, G., Lu, J.: Concept drift detection based on anomaly analysis. In: Loo, C.K., Yap, K.S., Wong, K.W., Teoh, A., Huang, K. (eds.) ICONIP 2014. LNCS, vol. 8834, pp. 263–270. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12637-1_33
4. Chauhan, R., Heydari, S.S.: Polymorphic adversarial DDoS attack on IDS using GAN. In: 2020 International Symposium on Networks, Computers and Communications (ISNCC), pp. 1–6 (2020). https://doi.org/10.1109/ISNCC49221.2020.9297264
5. Demertzis, K., Iliadis, L.: SAME: an intelligent anti-malware extension for android ART virtual machine. In: Núñez, M., Nguyen, N.T., Camacho, D., Trawiński, B. (eds.) ICCCI 2015. LNCS (LNAI), vol. 9330, pp. 235–245. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24306-1_23
6. Demertzis, K., Taketzis, D., Demertzi, V., Skianis, C.: An ensemble transfer learning spiking immune system for adaptive smart grid protection. Energies **15**(12), 4398 (2022). https://doi.org/10.3390/en15124398
7. Alhasan, S., Abdul-Salaam, G., Bayor, L., Oliver, K.: Intrusion detection system based on artificial immune system: a review. In: 2021 International Conference on Cyber Security and

Internet of Things (ICSIoT), pp. 7–14 (2021). https://doi.org/10.1109/ICSIoT55070.2021.00011
8. Hart, A.: Generalised synchronisation for continuous time reservoir computers. Rochester, NY (2021). https://doi.org/10.2139/ssrn.3987856
9. Demertzis, K., Iliadis, L., Pimenidis, E.: Geo-AI to aid disaster response by memory-augmented deep reservoir computing. Integr. Comput.-Aided Eng. **28**(4), 383–398 (2021). https://doi.org/10.3233/ICA-210657
10. Li, X., Bi, F., Yang, X., Bi, X.: An echo state network with improved topology for time series prediction. IEEE Sens. J. **22**(6), 5869–5878 (2022). https://doi.org/10.1109/JSEN.2022.3148742
11. Abu, U.A., Folly, K.A., Jayawardene, I., Venayagamoorthy, G. K.: Echo state network (ESN) based generator speed prediction of wide area signals in a multimachine power system. In: 2020 International SAUPEC/RobMech/PRASA Conference, pp. 1–5. (2020). https://doi.org/10.1109/SAUPEC/RobMech/PRASA48453.2020.9041236
12. Bala, A., Ismail, I., Ibrahim, R., Sait, S.M.: Applications of metaheuristics in reservoir computing techniques: a review. IEEE Access **6**, 58012–58029 (2018). https://doi.org/10.1109/ACCESS.2018.2873770
13. Gauthier, D.J., Bollt, E., Griffith, A., Barbosa, W.A.: Next generation reservoir computing. Nat. Commun. **12**(1), 5564 (2021). https://doi.org/10.1038/s41467-021-25801-2
14. Shao, Y., Yao, X., Wang, G., Cao, S.: A new improved echo state network with multiple output layers for time series prediction. In: 2021 6th International Conference on Robotics and Automation Engineering (ICRAE), pp. 7–11. (2021). https://doi.org/10.1109/ICRAE53653.2021.9657812
15. Demertzis, K., Iliadis, L.: Next generation automated reservoir computing for cyber Defense. In: Maglogiannis, I., Iliadis, L., MacIntyre, J., Dominguez, M. (eds.) Artificial Intelligence Applications and Innovations. AIAI 2023. IFIP Advances in Information and Communication Technology, vol. 676, pp. 16–27. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-34107-6_2
16. Demertzis, K., Iliadis, L.: An autonomous self-learning and self-adversarial training neural architecture for intelligent and resilient cyber security systems. In: Iliadis, L., Maglogiannis, I., Alonso, S., Jayne, C., Pimenidis, E. (eds.) Engineering Applications of Neural Networks. EANN 2023. Communications in Computer and Information Science, vol. 1826, pp. 461–478. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-34204-2_38
17. Li, J., Yao, X., Xu, K.: A comprehensive model integrating BP neural network and RSM for the prediction and optimization of syngas quality. Biomass Bioenergy **155**, 106278 (2021)
18. Alzubaidi, L., et al.: Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J. Big Data **8**(1), 53 (2021). https://doi.org/10.1186/s40537-021-00444-8
19. Aggarwal, C.C., Philip, S.Y., Han, J., Wang, J.: A framework for clustering evolving data streams. In: Freytag, J.-C., Lockemann, P., Abiteboul, S., Carey, M., Selinger, P., Heuer, A. (eds.) Proceedings 2003 VLDB Conference. Morgan Kaufmann, San Francisco, pp. 81–92 (2003). https://doi.org/10.1016/B978-012722442-8/50016-1
20. Aggarwal, C.C.: Neighborhood-based collaborative filtering. In: Aggarwal, C.C. (ed.) Recommender Systems: The Textbook, pp. 29–70. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-29659-3_2
21. Aumüller, M., Bernhardsson, E., Faithfull, A.: ANN-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. arXiv: https://doi.org/10.48550/arXiv.1807.05614 (2018)
22. Bifet, A., de Francisci Morales, G., Read, J., Holmes, G., Pfahringer, B.: Efficient online evaluation of big data stream classifiers. In: Proceedings of the 21th ACM SIGKDD International

Conference on Knowledge Discovery and Data Mining, in KDD '15, pp. 59–68. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2783258.2783372
23. Sabau, A.S.: Stream clustering using probabilistic data structures. arXiv: https://doi.org/10.48550/arXiv.1612.02701 (2016)
24. Stepaniants, G.: Learning partial differential equations in reproducing kernel Hilbert spaces. arXiv: https://doi.org/10.48550/arXiv.2108.11580 (2022)
25. Fujii, K., Kawahara, Y.: Dynamic mode decomposition in vector-valued reproducing kernel Hilbert spaces for extracting dynamical structure among observables. Neural Netw. **117**, 94–103 (2019). https://doi.org/10.1016/j.neunet.2019.04.020
26. Kostic, V., Novelli, P., Maurer, A., Ciliberto, C., Rosasco, L., Pontil, M.: Learning dynamical systems via Koopman operator regression in reproducing kernel hilbert spaces. arXiv: https://doi.org/10.48550/arXiv.2205.14027 (2022)
27. Hu, F., Chen, H., Wang, X.: An intuitionistic kernel-based fuzzy C-means clustering algorithm with local information for power equipment image segmentation. IEEE Access **8**(4), 4500–4514 (2020)
28. Hou, R., Tang, F., Liang, S., Ling, G.: Multi-party verifiable privacy-preserving federated k-means clustering in outsourced environment. Secur. Commun. Netw. **2021**, e3630312 (2021). https://doi.org/10.1155/2021/3630312
29. Alkathiri, M., Abdul, J., Potdar, M.B.: Kluster: Application of k-means clustering to multi-dimensional GEO-spatial data. In: 2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC), pp. 1–7 (2017). https://doi.org/10.1109/ICOMICON.2017.8279080
30. Wielgosz, M., Pietroń, M.: Using spatial pooler of hierarchical temporal memory to classify noisy videos with predefined complexity. Neurocomputing **240**, 84–97 (2017). https://doi.org/10.1016/j.neucom.2017.02.046
31. Nguyen, Q.D., Dhouib, S., Chanet, J.P., Bellot, P.: Towards a web-of-things approach for OPC UA field device discovery in the industrial IoT. In: 2022 IEEE 18th International Conference on Factory Communication Systems (WFCS), pp. 1–4 (2022). https://doi.org/10.1109/WFCS53837.2022.9779181
32. Hahsler, M., Bolaños, M., Forrest, J.: Introduction to stream: an extensible framework for data stream clustering research with R. J. Stat. Softw. **76**, 1–50 (2017). https://doi.org/10.18637/jss.v076.i14