Information Sciences 677 (2024) 120964

Contents lists available at ScienceDirect

Information Sciences

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

A soft prototype-based autonomous fuzzy inference system for network intrusion detection

Xiaowei Gu [a,*], Gareth Howells [b], Haiyue Yuan [c]

[a] School of Computer Science and Electronic Engineering, University of Surrey, Guildford GU2 7XH, UK
[b] School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, UK
[c] School of Computing, University of Kent, Canterbury CT2 7NZ, UK

ARTICLE INFO

Keywords:
Data stream
Fuzzy rule
Fuzzy inference
Intrusion detection
Soft prototype

ABSTRACT

Nowadays, cyber-attacks have become a common and persistent issue affecting various human activities in modern societies. Due to the continuously evolving landscape of cyber-attacks and the growing concerns around "black box" models, there has been a strong demand for novel explainable and interpretable intrusion detection systems with online learning abilities. In this paper, a novel soft prototype-based autonomous fuzzy inference system (SPAFIS) is proposed for network intrusion detection. SPAFIS learns from network traffic data streams online on a chunk-by-chunk basis and autonomously identifies a set of meaningful, human-interpretable soft prototypes to build an IF-THEN fuzzy rule base for classification. Thanks to the utilization of soft prototypes, SPAFIS can precisely capture the underlying data structure and local patterns, and perform internal reasoning and decision-making in a human-interpretable manner based on the ensemble properties and mutual distances of data. To maintain a healthy and compact knowledge base, a pruning scheme is further introduced to SPAFIS, allowing itself to periodically examine the learned solution and remove redundant soft prototypes from its knowledge base. Numerical examples on public network intrusion detection datasets demonstrated the efficacy of the proposed SPAFIS in both offline and online application scenarios, outperforming the state-of-the-art alternatives.

1. Introduction

Thanks to the rapid development in electronic manufacturing and information technology, the Internet has become an essential part of everyday life for billions of individuals in modern societies. The Internet has greatly transformed the way people communicate, network and access information. However, the on-going digitalization in the world has also led to a significant rise in cyber-attacks. According to the Cyber Security Breaches Survey published by the UK government in April 2023 [1], 59 % of medium businesses, 69 % of large businesses and 56 % of high-income charities have encountered cybersecurity breaches and/or cyber-attacks in the last 12 months. Nowadays, the escalating cyber-attacks have posed a major and persistent threat to individuals, businesses and organizations on the Internet. The need for effective techniques to protect information security is highly pronounced.

Intrusion detection systems (IDSs) are one of the most effective security techniques to prevent cyber-attacks [2]. The function of an IDS is to monitor the network and identify malicious activities. Traditional IDSs are primarily based on signatures. Such IDSs utilize

* Corresponding author.
E-mail addresses: xiaowei.gu@surrey.ac.uk (X. Gu), gareth.howells@essex.ac.uk (G. Howells), h.yuan957@googlemail.com (H. Yuan).

https://doi.org/10.1016/j.ins.2024.120964
Received 29 October 2023; Received in revised form 12 January 2024; Accepted 5 June 2024
Available online 10 June 2024

pattern matching methods to compare current activities against signatures of previous intrusions stored in the database [3]. Signature-based IDSs are highly effective in detecting known attacks, but they are unable to detect novel attacks because of the lack of matching signature in the database. As the technological evolution of cybercrime has made cyber-attacks more sophisticated and difficult to detect, traditional signature-based IDSs have become insufficient in real-world scenarios [4].

Machine learning techniques are capable of learning normal and malicious patterns from empirically observed network activities to constructing accurate predictive models with less human involvement [4]. Conventional machine learning methods, such as decision tree (DT) [5], random forest (RF) [6], support vector machine (SVM) [7], k-nearest neighbour (KNN) [8], etc., have been extensively used for identifying cyber-attacks. IDSs based on conventional machine learning have achieved many successes, but they generally struggle with large-scale, complex intrusion detection problems [9]. Due to the evolving landscape of cyber-attacks, characterized by the increasing sophistication and complexity, there has been a rapidly growing demand for IDSs that leverage more advanced machine learning techniques.

On the other hand, deep neural networks (DNNs, or artificial network networks, ANNs) have demonstrated eye-catching performances on a variety of practical problems involving visual and audio information. Due to their appealing multi-level representation learning capabilities, there have been a significant increase in the number of works in the literature that utilize DNNs for network intrusion detection in the last decade [4,9]. DNN-based IDSs have demonstrated promising results in detecting sophisticated and complex cyber-attacks that the conventional machine learning based IDSs may struggle with [10,11]. However, it is also widely recognized that DNNs are the typical type of highly complex "black box" models [12,13]. DNNs usually have huge amounts of hyper-parameters that cannot be associated to the practical problems, and decisions made by such models cannot be explained to/by humans easily. Although there have been a number of post-hoc approaches proposed in the literature attempting to provide some insights to the internal reasoning of DNNs, e.g., layer-wise relevance propagation [14], saliency [15], explanations provided by these approaches are aligned to the model behaviours rather than human understanding and, hence, they often are misleading and meaningless [12]. The high complexity and low interpretability also make it extremely challenging for human experts to pinpoint the causes of prediction errors made by DNNs for a straightforward fixing. Concerns on the lack of interpretability and explainability of DNNs have largely limited their wider adaption in high-stake real-world applications, such as network intrusion detection, despite of their superior performances over traditional machine learning approaches.

Another issue associated with most of the conventional machine learning and DNN models is the lack of capability to self-adapt to new data patterns. Although these models typically can perform decision-making in real time, a full retraining is usually required when dealing with new data of unseen patterns. The ability to self-learn and self-update from streaming data is of great importance for an IDS, especially in the context of constantly evolving cyber-attack behaviours [2]. Without such feature, the performance of the IDS will inevitably decrease over time due to the shifts and/or drifts of data patterns [16].

In this paper, a soft prototype-based autonomous fuzzy inference system (SPAFIS) is proposed for network intrusion detection. The proposed SPAFIS is a novel zero-order evolving fuzzy system (EFS) to autonomously learn from data streams on a chunk-by-chunk basis. It is well known that zero-order EFSs are designed to simultaneously self-organize and self-update the system structure and *meta*-parameters online from data streams in a single pass manner utilizing prototype-based IF-THEN fuzzy rules for classification [13,17]. Prototypes play a key role in the decision-making and internal reasoning of zero-order EFSs [18,19]. They are highly informative data samples representing the local peaks of multimodal data distributions and form the knowledge base of the systems. Thanks to the transparent prototype-based system structure and explainable internal reasoning performed based on the mutual distances of data, zero-order EFSs provide an effective solution towards explainable AI (XAI) for application scenarios concerning data streams with the attractive ability to agilely self-adapt to the dynamically changing data patterns in nonstationary environments [20].

A zero-order EFS identifies prototypes from data streams by online data partitioning in a crisp manner, where each empirically observed data sample can only belong to a single prototype. Crisp prototypes identified by zero-order EFSs are mutually exclusive and they might struggle to accurately represent the local patterns of data with complex and uncertain distributions [21]. Different from existing zero-order EFSs, SPAFIS utilizes soft prototypes extracted from data, where each data sample forms a fraction of every soft prototype in the knowledge base of the system with a certain membership coefficient. This is similar to the concept of fuzzy clustering [21], but these soft prototypes are learned from data streams online in a single pass manner without iterative optimization as required by the vast majority of fuzzy clustering approaches. Compared with crisp prototypes, soft prototypes can better summarize the underlying structure and local patterns of data, thereby, enabling SPAFIS to achieve greater intrusion detection performance from network data streams. Furthermore, to enhance the computational efficiency of SPAFIS and reduce system obesity during online learning, a soft prototype pruning mechanism is introduced to SPAFIS, which periodically examines the knowledge base and removes soft prototypes with higher similarity to maintain a more compact knowledge base. With this pruning scheme, older soft prototypes that are spatially closer to newer ones will be pruned from the knowledge base, and the information carried by them will be dissolved and fused into the remaining soft prototypes to better preserve the learned knowledge.

To summarize, unique features of the proposed SPAFIS-based IDS that distinguish it from the state-of-the-art (SOTA) IDSs based on conventional machine learning and deep learning techniques include:

1) a transparent rule-based structure composed of human-interpretable soft prototypes with internal reasoning and decision-making performed based on mutual distances of data, and;
2) the capability to continuously self-expand its knowledge base from data streams and self-adapt to changing data patterns in nonstationary environments.

Key contributions of this study to the field of EFSs are outlined as follows:

3) the utilization of soft prototypes to better summarize the underlying structure and local patterns of data with complex distributions;
4) an online, non-iterative approach to learn soft prototypes from data streams on a chunk-by-chunk basis;
5) the capability to periodically prune soft prototypes with high spatial similarity for a healthier knowledge base and greater computational efficiency.

Numerical experiments on popular public benchmark datasets for network intrusion detection demonstrated the superior performance of the proposed SPAFIS-based IDS in both offline and online setting, outperforming the SOTA IDSs based on conventional machine learning models and cutting-edge DNNs.

The remainder of this paper is organized as follows. A review of related works is given by Section 2. Section 3 describes the technical details of the proposed SPAFIS. Numerical examples are presented in Section 4 to demonstrate the performance of SPAFIS on network intrusion detection. This paper is concluded by Section 5, and directions for future work is also discussed in this section.

2. Related works

In this section, a review of related works on network intrusion detection is presented, providing the background and context of this research. Due to the limited length of this paper, it is practically impossible to cover all the works in this area. Hence, the literature review is primarily focused on representative works in the following three aspects, namely, conventional machine learning-based methods, DNN-based methods and EFS-based methods. Interested readers are referred to the recently published review papers [3,4] for more details about the latest developments in using machine learning and deep learning methods for intrusion detection.

In the past decades, a number of IDSs utilizing conventional machine learning techniques have been proposed in the literature and have achieved lots of success. For example, the KNN classifier is employed in [22] for learning program behaviours defined as collections of system calls over each program execution and classifying each new program behaviour into either normal or intrusive class. In [23], a IDS utilizing DT learned by J48 algorithm is proposed to classify different network packets in the Kyoto 2006 + dataset into three major categories, namely, benign, known attack and unknown attacks. A SVM-based IDS is proposed in [7], where a hierarchical clustering algorithm and a feature selection algorithm exploiting the so-called "leave-one-out" strategy are employed to reduce both the number of samples and the number of features of the network data. A RF classifier combined with synthetic minority oversampling technique (SMOTE) and feature selection is proposed in [24] for intrusion detection with improved performance on minority class, where SMOTE is employed to restore class balance in the training set of NSL-KDD dataset [25] and feature selection based on information gain is used to obtain a reduced feature subset to facilitate the classifier training. In [26], an ensemble system composed of multiple SVM and KNN classifier with particle swarm optimization generated base classifier weights is proposed for detecting network intrusions in the KDD99 dataset. A duel ensemble classifier combining multiple gradient boosting decision tree ensembles via bootstrap aggregation (bagging) is introduced in [24], offering greater intrusion detection performance.

Thanks to the superior performances of DNNs over conventional machine learning techniques on many highly challenging problems, there has been an increasing number of works proposed in the recent years exploiting DNNs for network intrusion detection. Popular DNN models for constructing IDSs include, but are not limited to, feedforward neural networks (FNNs) [11], recurrent neural networks (RNNs) [9,27], convolutional neural networks (CNNs) [10]. DNN-based IDSs have demonstrated superior performances in binary and multiclass intrusion classification on various benchmark problems, e.g., KDD99, NLS-KDD [25], CICIDS2017 [28], outperforming a variety of conventional machine learning techniques. However, as aforementioned, one of the key drawbacks of DNNs is the lack of explainability, which is a critical issue for high-stake applications. To partially alleviate this issue whilst maintaining the same-level performance, IDSs based on hybridizations of DNNs and conventional machine learning models are constructed, where DNNs are used for extracting high-level abstract features from network data and the conventional machine learning models are used for classifying the data instances into different categories based on the extracted features [29,30]. Nevertheless, it is also widely known that conventional machine learning models suffer from a number of weaknesses when applied to large-scale, complex problems, such as lower transparency, lower explainability and lower computational efficiency.

As cyber-attacks are consistently evolving, it is necessary for IDSs to be capable of continuously self-evolving the system structure and *meta*-parameters to self-adapt to new data patterns. It is also important for IDSs to be able to perform decision-making in a human understandable manner and provide high-level explainability. As a result, researchers are exploring alternative approaches to construct explainable IDSs that address the evolving threat landscape effectively. EFSs, as a powerful tool widely used for real-time non-stationary problem approximation and a promising approach towards XAI [17], have been increasingly used for network intrusion detection in nonstationary environments. In [31,32], an EFS-based approach named evolving agent behaviour classification based on distributions of relevant events (EvABCD) is proposed to create and recognize automatically the behaviour profiles of different computer users by representing the observed behaviours of computer users as adaptive distributions of their relevant atomic behaviours. EvABCD can effectively monitor the time varying behaviours of different computer users, thereby detecting abnormalities and identifying masquerades. An evolving possibilistic Cauchy clustering algorithm is introduced in [33] to learn a self-evolving predictive model online from network data streams for cyber-attack detection. In [34], an evolving Gaussian fuzzy classifier is implemented to autonomously analyse the time-varying data from the Tier-1 Bologna computer centre and detect anomalies in the log records in real time for predictive maintenance.

3. Proposed SPAFIS

In this section, technical details of the proposed SPAFIS are presented. As aforementioned, one key feature that differentiates

SPAFIS from alternative zero-order EFSs is the utilization of soft prototypes in its internal reasoning and decision making. Compared with crisp prototypes utilized by conventional prototype-based models [19,20,35], soft prototypes have a greater capability to summarize the underlying patterns of empirically observed data and preserve the data structure, enabling the resulting SPAFIS to achieve better prediction performance.

First of all, let $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \cdots, \boldsymbol{x}_k, \cdots\}$ ($\boldsymbol{x}_k = \left[x_{k,1}, x_{k,2}, .., x_{k,M}\right]^T \in \mathbf{X}$) be a particular data stream/static dataset in the M-dimensional data space $\mathscr{R}^M$, where the subscript k denotes the time instance at which $\boldsymbol{x}_k$ is observed. It is assumed that samples of the data stream $\mathbf{X}$ continuously arrive in chunks, namely, $\mathbf{X}_n = \left\{\boldsymbol{x}_{n,1}, \boldsymbol{x}_{n,2}, \cdots, \boldsymbol{x}_{n,L_n}\right\}$ ($n = 1, 2, 3, \cdots; L_n$ is the cardinality of $\mathbf{X}_n$). $\mathbf{X}$ is composed of data samples of C different classes with $\mathbf{Y} = \left\{y_1, y_2, y_3, \cdots, y_k, \cdots\right\}$ being the corresponding class labels of $\mathbf{X}$, where y_k denotes the class label of $\boldsymbol{x}_k$ and there is $y_k \in \{1, 2, \cdots, C\}$. According to the class labels $\mathbf{Y}_n$, data samples of $\mathbf{X}_n$ can be further divided into C non-overlapping subsets, namely, $\mathbf{X}_n^c = \left\{\boldsymbol{x}_{n,1}^c, \boldsymbol{x}_{n,2}^c, \cdots, \boldsymbol{x}_{n,L_n^c}^c\right\}$ ($c = 1, 2, \cdots, C$); L_n^c is the cardinality of $\mathbf{X}_n^c$), and there are $\mathbf{X}_n^i \cap \mathbf{X}_n^j = \varnothing \forall i \neq j$, and $\mathbf{X}_n^1 \cup \mathbf{X}_n^2 \cup \cdots \cup \mathbf{X}_n^C = \mathbf{X}_n$. By default, this study employs city block distance as the default distance measure, namely, $\|\boldsymbol{x} - \boldsymbol{y}\|_1 = \sum_{m=1}^{M} |x_m - y_m|$, same as [20]. However, one may consider using other commonly used distance metrics, such as Euclidean distance, Mahalanobis distance, etc.

3.1. Architecture

The general architecture of SPAFIS is visualized in Fig. 1, which is composed of the following components:

1) A self-adaptive threshold learner;

The self-adaptive threshold learner derives the data-driven distance threshold from the received data chunks based on the mutual distances of data and the level of granularity set by the users [19].

2) C knowledge summarizers;

The knowledge summarizers learn from received data chunks to build a knowledge base composed of highly distinctive and informative soft prototypes preserving the underlying structure and local patterns of the data streams. The knowledge summarizers also periodically remove these soft prototypes with high spatial similarity from the knowledge base to maintain its compactness and healthiness.

3) C soft prototype-based IF-THEN rules, and;

The soft prototype-based IF-THEN rules are the core of SPAFIS, created from the learned knowledge base. They serve as the inference engine and are formulated in the following form [19]:

$$R^c: \begin{array}{l} \text{IF } \left(\boldsymbol{x} \sim \boldsymbol{s}_1^c\right) \text{ OR } \left(\boldsymbol{x} \sim \boldsymbol{s}_2^c\right) \text{ OR} \cdots \text{OR } \left(\boldsymbol{x} \sim \boldsymbol{s}_{S^c}^c\right) \\ \text{THEN } (y = c) \end{array} \tag{1}$$

where $c = 1, 2, \cdots, C$; "$\sim$" denotes similarity; $\boldsymbol{s}_j^c$ is the j th soft prototype of the c th IF-THEN rule, R^c; S^c is the total number of soft prototypes identified from the empirically observed data samples of the c th class; $\mathbf{S}^c$ is the collection of soft prototypes associated with R^c. As one can see from Eq. (1), each IF-THEN rule is, in fact, a combination of S^c simpler IF-THEN rules with the same singleton consequent part integrated by logical "OR" connectives as follows ($j = 1, 2, \cdots, S^c$):

$$R_j^c: \begin{array}{l} \text{IF } \left(\boldsymbol{x} \sim \boldsymbol{s}_j^c\right) \\ \text{THEN } (y = c) \end{array} \tag{2}$$

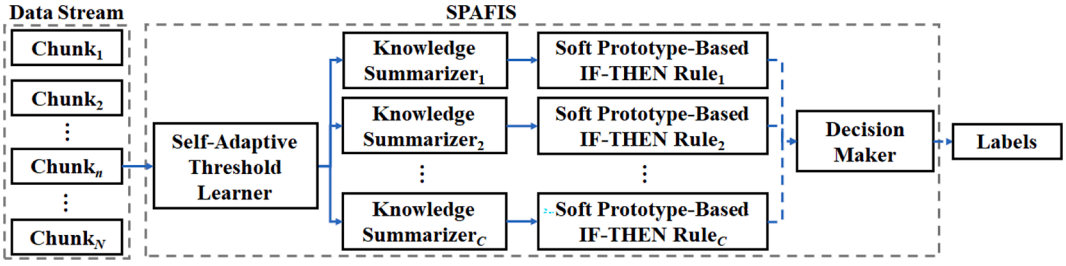

Fig. 1. General architecture of SPAFIS.

4) A decision maker

The decision maker predicts class labels of unlabelled samples based on the confidence scores produced by the IF-THEN rules during the testing stage [36]. The confidence scores are calculated based on the spatial similarity between unlabelled samples and soft prototypes, and the class labels are determined by the decision maker based on the highest confidence scores accordingly.

The algorithmic procedures of the system identification and decision-making schemes are detailed in the following two subsections. It is worth noting that to maintain its computation- and memory- efficiency, SPAFIS discards all the processed historical data chunks and only keeps the extracted prototypes in its knowledge base. In addition, SPAFIS will periodically examine the spatial similarity between identified prototypes and disassemble these ones that share high similarity with their neighbours to improve the compactness of the learned knowledge base and further reduce the computational complexity.

3.2. System identification

The system identification process of SPAFIS is composed of the following five recurring steps, namely, 1) self-adaptive threshold learning; 2) soft prototype identification; 3) knowledge base updating; 4) soft prototype pruning, and 5) rule base updating. The detailed system identification process is presented as follows.

Step 1. Self-adaptive threshold learning. Once the n th data chunk, $\mathbf{X}_n$ arrives, the self-adaptive threshold learner firstly calculates the pairwise distances between any two data samples of $\mathbf{X}_n$ and obtain the following $L_n \times L_n$ dimensional pairwise distance matrix $\mathbf{d}_n$:

$$\mathbf{d}_n = \left[\|\boldsymbol{x}_{n,k} - \boldsymbol{x}_{n,j}\|_1^2\right]_{j=1:L_n}^{k=1:L_n} = \left[\left(\sum_{m=1}^{M} |x_{n,k,m} - x_{n,j,m}|\right)^2\right]_{j=1:L_n}^{k=1:L_n} \tag{3}$$

Then, the average distance $\overline{d}_{n,G}$ between any two data samples that can be viewed as neighbours under the G th level of granularity can be calculated by Eq. (4) in an iterative manner ($g = 1, 2, \cdots, G$):

$$\overline{d}_{n,g} = \frac{1}{\sum_{k=1}^{L_n-1}\sum_{j=k+1}^{L_n} w_{g,k,j}} \sum_{k=1}^{L_n-1}\sum_{j=k+1}^{L_n} w_{g,k,j}\|\boldsymbol{x}_{n,k} - \boldsymbol{x}_{n,j}\|_1^2 \tag{4}$$

where $w_{g,k,j} = \begin{cases} 1, & if\ \|\boldsymbol{x}_{n,k} - \boldsymbol{x}_{n,j}\|_1^2 \le \overline{d}_{n,g-1} \\ 0, & else \end{cases}$; $\overline{d}_{n,0} = \frac{2}{L_n(L_n-1)}\sum_{k=1}^{L_n-1}\sum_{j=k+1}^{L_n}\|\boldsymbol{x}_{n,k} - \boldsymbol{x}_{n,j}\|_1^2$.

If $\mathbf{X}_n$ is the very first data chunk (namely, $n = 1$), the self-adaptive distance threshold γ_G is set as: $\gamma_G \leftarrow \overline{d}_{n,G}$. Otherwise, γ_G is updated via Eq. (5):

$$\gamma_G \leftarrow \frac{\gamma_G \bullet \sum_{k=1}^{n-1} L_k + L_n \bullet \overline{d}_{n,G}}{\sum_{k=1}^{n} L_k} \tag{5}$$

Note that γ_G serves as an estimation of the maximum distance between any two data samples that are viewed as neighbours under the G th level of granularity set by users (G is a nonnegative integer). $G = 6$ is considered in this study. The greater the level of granularity is, the smaller γ_G is, and SPAFIS will focus more on the local patterns of data and identify more soft prototypes. Compared with prefixed hard thresholds, which are commonly used by existing works [35], the self-adaptive distance threshold γ_G is always guaranteed to be meaningful thanks to its data-driven nature [37]. Very important, γ_G can be determined based on users' preferences and requires no prior knowledge about the given problem.

Once γ_G is initialized/updated, the system identification process enters Step 2.

Step 2. Soft prototype identification. In this step, $\mathbf{X}_n$ is divided into C nonoverlapping subsets (denoted as $\mathbf{X}_n^1$, $\mathbf{X}_n^2$, ..., $\mathbf{X}_n^C$) according to the corresponding class labels, $\mathbf{Y}_n$. The C subsets are then passed to the corresponding knowledge summarizers. Each knowledge summarizer follows the exact same procedure to learn soft prototypes from the received data of the corresponding class. After the c th knowledge summarizer has collected the corresponding subset of data $\mathbf{X}_n^c$, the $L_n^c \times L_n^c$ dimensional pairwise distance matrix $\mathbf{d}_n^c$ is firstly sliced from $\mathbf{d}_n$ based on samples' indices:

$$\mathbf{d}_n^c = \left[\|\boldsymbol{x}_{n,k}^c - \boldsymbol{x}_{n,j}^c\|_1^2\right]_{j=1:L_n^c}^{k=1:L_n^c} \tag{6}$$

Next, the symmetric adjacency matrix $\mathbf{A}_n^c$ is derived from $\mathbf{d}_n^c$:

$$\mathbf{A}_n^c = \left[A_{n,k,j}^c\right]_{j=1:L_n^c}^{k=1:L_n^c} \tag{7}$$

where $A_{n,k,j}^c = \begin{cases} 1, & if\ \|\boldsymbol{x}_{n,k}^c - \boldsymbol{x}_{n,j}^c\|_1^2 \le \gamma_G \\ 0, & else \end{cases}$ and there is $A_{n,k,j}^c = A_{n,j,k}^c\ \forall j,k$.

The data density of every data sample of $\mathbf{X}_n^c$ is calculated using Eq. (8) based on $\mathbf{d}_n^c$ and $\mathbf{A}_n^c$ ($k = 1, 2, \cdots, L_n^c$):

$$D\left(\boldsymbol{x}_{n,k}^{c}\right)=\sum_{j=1}^{L_n^c}A_{n,k,j}^{c}\alpha\left(\boldsymbol{x}_{n,k}^{c},\boldsymbol{x}_{n,j}^{c}\right) \tag{8}$$

where $\alpha(\boldsymbol{x},\boldsymbol{z})=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{z}\|_1^2}{\gamma_G}\right)$.Then, the local peaks of the data density are identified by Condition (1) [20].

$$\begin{aligned}\textit{Cond.}\,(1):\quad &\textit{if}\left(\max_{j=1,2,\cdots,L_n^c}\left(A_{n,k,j}^{c}D\left(\boldsymbol{x}_{n,j}^{c}\right)\right)=D\left(\boldsymbol{x}_{n,k}^{c}\right)\right)\\ &\textit{then}\left(\boldsymbol{x}_{n,k}^{c}\ \textit{is a local peak}\right)\end{aligned} \tag{9}$$

Condition (1) finds out the data samples with the locally maximum data density in their neighbours. Such samples are aligned tightly with the peaks of multimodal density distribution of data and can be used for constructing soft prototypes. Assuming that a total of P_n^c local peaks are identified by Condition (1) (denoted as $\widehat{\boldsymbol{x}}_{n,1}^{c}$, $\widehat{\boldsymbol{x}}_{n,2}^{c}$, …, $\widehat{\boldsymbol{x}}_{n,P_n^c}^{c}$), the c th knowledge summarizer will extract P_n^c soft prototypes from $\mathbf{X}_n^c$ to represent these identified local peaks using Eq. (10):

$$\boldsymbol{p}_{n,k}^{c}=\sum\nolimits_{j=1}^{L_n^c}\alpha^{*}\left(\widehat{\boldsymbol{x}}_{n,k}^{c},\boldsymbol{x}_{n,j}^{c}\right)\boldsymbol{x}_{n,j}^{c} \tag{10}$$

and the support of $\boldsymbol{p}_{n,k}^{c}$, namely, the number of data samples associated with $\boldsymbol{p}_{n,k}^{c}$, denoted as $\rho_{n,k}^{c}$ is obtained as:

$$\rho_{n,k}^{c}=\sum_{j=1}^{L_n^c}\frac{\alpha^{*}\left(\widehat{\boldsymbol{x}}_{n,k}^{c},\boldsymbol{x}_{n,j}^{c}\right)}{\sum_{l=1}^{P_n^c}\alpha^{*}\left(\widehat{\boldsymbol{x}}_{n,l}^{c},\boldsymbol{x}_{n,j}^{c}\right)} \tag{11}$$

where $\alpha^{*}\left(\widehat{\boldsymbol{x}}_{n,k}^{c},\boldsymbol{x}_{n,j}^{c}\right)=\frac{\alpha\left(\widehat{\boldsymbol{x}}_{n,k}^{c},\boldsymbol{x}_{n,j}^{c}\right)}{\sum_{l=1}^{L_n^c}\alpha\left(\widehat{\boldsymbol{x}}_{n,k}^{c},\boldsymbol{x}_{n,l}^{c}\right)}$; $k=1,2,\cdots,P_n^c$. The collection of soft prototypes extracted from $\mathbf{X}_n^c$ is denoted as $\mathbf{P}_n^c$. It can be seen from Eq. (10) that each soft prototype is a weighted combination of all data samples of $\mathbf{X}_n^C$ with the respective weights calculated based on their distances to the local peak it represents. Similar to soft clustering, e.g., fuzzy c-means [38,39], in SPAFIS, each data sample is associated with every soft prototype with a certain membership coefficient. Such additional flexibility gives the soft prototype stronger capability to preserve the data structure and the underlying patterns compared with crisp prototypes used by conventional approaches [19,20,35].

Step 3. Knowledge base updating. Once the knowledge summarizers have extracted soft prototypes from the current data chunk, $\mathbf{X}_n$, the knowledge base of SPAFIS will be initialized/updated.

If $\mathbf{X}_n$ is the first data chunk (namely, $n=1$), the knowledge base in the form of soft prototypes $(\mathbf{S}^1,\mathbf{S}^2,\cdots,\mathbf{S}^C)$ will be initialized as follows ($c=1,2,\cdots,C$):

$$\mathbf{S}^{c}\leftarrow\mathbf{P}_n^c \tag{12}$$

Here ν_j^c is the support of $\boldsymbol{s}_j^c$ ($j=1,2,\cdots,S^c$).

Otherwise, namely, $n>1$, the knowledge base will be updated with the newly identified prototypes, $\mathbf{P}_n^1,\mathbf{P}_n^2,\cdots,\mathbf{P}_n^C$ by integrating $\mathbf{P}_n^c$ into $\mathbf{S}^c$ according to their mutual distances ($c=1,2,\cdots,C$). To do so, Condition (2) is checked for each soft prototype $\boldsymbol{p}_{n,k}^{c}$ within $\mathbf{P}_n^c$ ($k=1,2,\cdots,P_n^c$):

$$\begin{aligned}\textit{Cond.}\,(2):\quad &\textit{if}\left(\min_{\boldsymbol{s}\in\mathbf{S}^c}\left(\left\|\boldsymbol{p}_{n,k}^{c}-\boldsymbol{s}\right\|_1^2\right)>\gamma_G\right)\\ &\textit{then}\left(\mathbf{S}^{c}\leftarrow\mathbf{S}^{c}\cup\left\{\boldsymbol{p}_{n,k}^{c}\right\};\mathbf{P}_n^{c}\leftarrow\mathbf{P}_n^{c}\backslash\left\{\boldsymbol{p}_{n,k}^{c}\right\}\right)\end{aligned} \tag{13}$$

If $\boldsymbol{p}_{n,k}^{c}$ satisfies Condition (2), $\boldsymbol{p}_{n,k}^{c}$ is spatially distant to any of the existing soft prototypes learned from historical data chunks, and it is highly likely that $\boldsymbol{p}_{n,k}^{c}$ represents a novel data pattern that has not been seen in the historical data. Hence, $\boldsymbol{p}_{n,k}^{c}$ is added to $\mathbf{S}^c$ to incorporate the new knowledge in the knowledge base ($S^c\leftarrow S^c+1$) and, meanwhile, remove from $\mathbf{P}_n^c$ ($P_n^c\leftarrow P_n^c-1$).

After all the soft prototypes of $\mathbf{P}_n^c$ that satisfy Condition (2) have been selected to join $\mathbf{S}^c$, the remaining members of $\mathbf{P}_n^c$ will be used for updating the members of $\mathbf{S}^c$ using Eq. (14) ($j=1,2,\cdots,S^c;k=1,2,\cdots,P_n^c$):

$$\boldsymbol{s}_j^{c}\leftarrow\frac{\nu_j^{c}\boldsymbol{s}_j^{c}+\sum_{k=1}^{P_n^c}\alpha_{k,j}^{*}\rho_{n,k}^{c}\boldsymbol{p}_{n,k}^{c}}{\nu_j^{c}+\sum_{k=1}^{P_n^c}\alpha_{k,j}^{*}\rho_{n,k}^{c}};\quad \nu_j^{c}\leftarrow\nu_j^{c}+\sum\nolimits_{k=1}^{P_n^c}\alpha_{k,j}^{*}\rho_{n,k}^{c} \tag{14}$$

where $\alpha_{k,j}^{*}=\frac{\alpha\left(\boldsymbol{p}_{n,k}^{c},\boldsymbol{s}_j^{c}\right)}{\sum_{i=1}^{S^c}\alpha\left(\boldsymbol{p}_{n,k}^{c},\boldsymbol{s}_i^{c}\right)}$. It can be seen from Eq. (14) that if $\boldsymbol{p}_{n,k}^{c}$ fails to satisfy Condition (2), it will be disassembled for updating the knowledge base . Every soft prototype $\boldsymbol{s}_j^{c}$ within $\mathbf{S}^c$ will receive a portion of $\boldsymbol{p}_{n,k}^{c}$ for parameter updating. The closer $\boldsymbol{s}_j^{c}$ is to $\boldsymbol{p}_{n,k}^{c}$, the

bigger portion $\boldsymbol{s}_j^c$ receives from $\boldsymbol{p}_{n,k}^c$.

After $\mathbf{P}_n^1, \mathbf{P}_n^2, \cdots, \mathbf{P}_n^C$ have been integrated into $\mathbf{S}^1, \mathbf{S}^2, \cdots, \mathbf{S}^C$, the knowledge summarizers proceed to the next step for maintaining the knowledge base.

Step 4. Soft prototype pruning. Due to the nonstationary nature of data streams, the areas of influence of some soft prototypes may gradually overlap with others because of parameter updating from new data. This is typically caused by the drifts and/or shifts in the data streams and is a common issue in online model identification [16]. Keeping these soft prototypes with high spatial similarity in the knowledge base increases the computational and memory costs but does not improve the inference. To maintain a more compact and healthier knowledge base, the knowledge summarizers will perform soft prototype pruning periodically.

To do so, Condition (3) is used to examine the soft prototypes of $\mathbf{S}^c$ ($c = 1, 2, \cdots, C$) one by one from the oldest (identified from $\mathbf{X}_1^c$) to the newest (identified from $\mathbf{X}_n^c$):

$$\begin{aligned} \textit{Cond.}\,(3):\ & \textit{if}\left(\min_{k=j+1,2,\cdots,S^c}\left(\left\|\boldsymbol{s}_j^c-\boldsymbol{s}_k^c\right\|_1^2\right)<\omega_o\bullet\gamma_G\right) \\ & \textit{then}\left(\mathbf{R}^c\leftarrow\mathbf{R}^c\cup\left\{\boldsymbol{s}_j^c\right\};\mathbf{S}^c\leftarrow\mathbf{S}^c\Big/\left\{\boldsymbol{s}_j^c\right\}\right) \end{aligned} \tag{15}$$

where $j = 1, 2, \cdots, S^c - 1$; ω_o ($0 \le \omega_o < 1$) is a small non-negative value controlling the tolerance of SPAFIS towards the similarity between soft prototypes; $\mathbf{R}^c = \left\{\boldsymbol{r}_1^c, \boldsymbol{r}_2^c, \cdots, \boldsymbol{r}_{R^c}^c\right\}$ denotes the collection of soft prototypes to be removed from $\mathbf{S}^c$; R^c is the cardinality of $\mathbf{R}^c$; β_j^c is the support of $\boldsymbol{r}_j^c$. If a soft prototype, i.e., $\boldsymbol{s}_j^c$ satisfies Condition (3), it suggests that $\boldsymbol{s}_j^c$ shares very high similarity with some of soft prototypes identified from later data chunks. Hence, $\boldsymbol{s}_j^c$ will be removed from $\mathbf{S}^c$ to keep the knowledge base compact.

A smaller ω_o gives the knowledge summarizers greater tolerance towards these soft prototypes that are spatially close to each other and enables the system to preserve more soft prototypes in its knowledge base. On the other hand, a greater ω_o means that the knowledge summarizers will maintain a smaller-scale knowledge base consisted of more distinctive soft prototypes. However, the knowledge summarizers may remove too many soft prototypes from its knowledge base if ω_o is over large, causing the loss of knowledge learned from data and deteriorating the prediction performance of SPAFIS. In this study, $\omega_o = 0.01$ is used by default such that SPAFIS will only prune these highly overlapping soft prototypes with minimal impact on the predictive performance. One may also notice that the knowledge summarizers tend to keep these soft prototypes that are more recently identified in the knowledge base and prefer to remove these older soft prototypes when possible overlaps are spotted. The main reason for this is that these more recently identified soft prototypes tend to better represent the latest patterns of data streams, enabling SPAFIS to self-adaptive from data streams more effectively.

After the set of soft prototypes satisfying Condition (3), namely, $\mathbf{R}^c$ have been identified, the c th knowledge summarizer will disassemble them to update the remaining soft prototypes within $\mathbf{S}^c$ rather than directly discarding them ($c = 1, 2, \cdots, C$). In this way, the knowledge preserved by $\mathbf{R}^c$ will be used for enhancing the soft prototypes remaining in $\mathbf{S}^c$. For each soft prototype $\boldsymbol{s}_j^c \in \mathbf{S}^c$, it will be updated by $\mathbf{R}^c$ using Eq. (16):

$$\boldsymbol{s}_j^c\leftarrow\frac{\nu_j^c\boldsymbol{s}_j^c+\sum_{k=1}^{R^c}\alpha_{k,j}^{**}\beta_{n,k}^c\boldsymbol{r}_{n,k}^c}{\nu_j^c+\sum_{k=1}^{R^c}\alpha_{k,j}^{**}\beta_{n,k}^c};\quad \nu_j^c\leftarrow\nu_j^c+\sum_{k=1}^{R^c}\alpha_{k,j}^{**}\beta_{n,k}^c \tag{16}$$

where $j = 1, 2, \cdots, S^c$; $\alpha_{k,j}^{**} = \frac{a\left(\boldsymbol{r}_k^c, \boldsymbol{s}_j^c\right)}{\sum_{i=1}^{S^c} a\left(\boldsymbol{r}_k^c, \boldsymbol{s}_i^c\right)}$.

Note that, for greater computational efficiency, SPAFIS may only perform Step 4 after every T data chunks have been processed. In this study, $T = 10$ is considered.

Step 5. Rule base updating. If $\mathbf{X}_n$ is the first data chunk SPAFIS receives, the IF-THEN rule base (R^1, R^2, …, R^C) will be initialized with $\mathbf{S}^1, \mathbf{S}^2, \cdots, \mathbf{S}^C$. Otherwise, namely, the IF-THEN rule base will be updated to reflect the latest changes in $\mathbf{S}^1, \mathbf{S}^2, \cdots, \mathbf{S}^C$. Then, the current learning cycle is completed and SPAFIS starts a new learning cycle going back to Step 1 to continue process the next available data chunk ($n \leftarrow n + 1$) or it starts to make predictions on the unlabelled testing data.

The algorithmic procedure of the system identification process of SPAFIS is summarized in the following pseudo code for visual clarity. Note that except for Step 1, which is performed by the self-adaptive threshold learner, the other four steps are performed by the C knowledge summarizers in parallel. Therefore, one can greatly improve the computational efficiency of SPAFIS by implementing each of the knowledge summarizers on a separate computing node via distributed computation.

Algorithm 1. System identification of SPAFIS

while ($\mathbf{X}_n$ is available) **do**
Step 1. Self-adaptive threshold learning
  derive $\mathbf{d}_n$ from $\mathbf{X}_n$ using (3);
  calculate $\bar{d}_{n,G}$ from $\mathbf{d}_n$ using (4);
  if ($n = 1$) **then**
    $\gamma_G \leftarrow \bar{d}_{n,G}$;
  else
    update γ_G with $\bar{d}_{n,G}$ using (5);

(*continued on next page*)

(*continued*)

Algorithm 1. System identification of SPAFIS

end if
for $c = 1$ to C **do**
Step 2. Soft prototype identification
obtain $\mathbf{X}_n^c$ from $\mathbf{X}_n$;
derive $\mathbf{d}_n^c$ from $\mathbf{d}_n$;
derive $\mathbf{A}_n^c$ from $\mathbf{d}_n^c$ using (7);
calculate $D\left(\boldsymbol{x}_{n,k}^c\right)$ for each $\boldsymbol{x}_{n,k}^c \in \mathbf{X}_n^c$ using (8);
identify $\widehat{\mathbf{x}}_{n,1}^c$, $\widehat{\boldsymbol{x}}_{n,2}^c$, …, $\widehat{\boldsymbol{x}}_{n,P_n^c}^c$ using Condition (1);
extract $\mathbf{P}_n^c$ from $\mathbf{X}_n^c$ using (10);
Step 3. Knowledge base updating
if $(n = 1)$ **then**
$\mathbf{S}^c \leftarrow \mathbf{P}_n^c$;
else
integrate $\mathbf{P}_n^c$ into $\mathbf{S}^c$ using Condition (2);
update $\mathbf{S}^c$ with the remaining $\mathbf{P}_n^c$ using (14);
end if
Step 4. Soft prototype pruning
if $(n\%T = 0)$ **then**
remove $\mathbf{R}^c$ from $\mathbf{S}^c$ using Condition (3);
update $\mathbf{S}^c$ with $\mathbf{R}^c$ using (16);
end if
Step 5. Rule base updating
initialize/update R^c with $\mathbf{S}^c$;
end for
$n \leftarrow n + 1$;
end while

3.3. Decision making

The decision making process of SPAFIS is detailed in this subsection. For each unlabelled testing sample, denoted as $\boldsymbol{x}_k$, each soft prototype-based IF-THEN rule $\Re^c$ $(c = 1, 2, \cdots, C)$ will produce a confidence score based on the spatial similarity between $\boldsymbol{x}_k$ and soft prototypes associated with it. The confidence score produced by $\Re^c$ is calculated by Eq. (17):

$$\lambda_c(\boldsymbol{x}_k) = \frac{1}{S^c}\sum\nolimits_{i=1}^{S^c} \alpha\left(\boldsymbol{x}_k, \boldsymbol{s}_i^c\right) = \frac{1}{S^c}\sum\nolimits_{i=1}^{S^c} \exp\left(-\frac{\|\boldsymbol{x}_k - \boldsymbol{s}_i^c\|_1^2}{\gamma_G}\right) \tag{17}$$

The class label of $\boldsymbol{x}_k$ is determined by the IF-THEN rule that produces the greatest confidence score by Eq. (18) following the "winner takes all" principle:

$$\widehat{y}_k = \underset{c=1,2,\cdots,C}{\operatorname{argmax}}\left(\lambda_c(\boldsymbol{x}_k)\right) \tag{18}$$

3.4. Computational complexity analysis

As the system identification process of SPAFIS is conducted on a chunk-wise manner, it is assumed that the computational complexity analysis is performed at the time instance when the n th data chunk $\mathbf{X}_n$ is received. The Step 1 of the current learning cycle is to initialize/update the self-adaptive threshold from $\mathbf{X}_n$. The computational complexity of calculating $\mathbf{d}_n$ is $O\left(M^2 L_n^2\right)$ and that of deriving $\overline{d}_{n,G}$ from $\mathbf{d}_n$ is $O\left(GL_n^2\right)$. Compared with calculating $\mathbf{d}_n$ and $\overline{d}_{n,G}$, the complexity of initializing/updating γ_G is negligible. Hence, the overall complexity of Step 1 is $O\left(\left(G + M^2\right)L_n^2\right)$. In Step 2, soft prototypes $\mathbf{P}_n^1$, $\mathbf{P}_n^2$, $\cdots$, $\mathbf{P}_n^C$ are identified from $\mathbf{X}_n$. The computational complexity of converting $\mathbf{d}_n^c$ to $\mathbf{A}_n^c$ and calculating the data density is $O\left(\sum_{c=1}^{C}\left(L_n^c\right)^2\right)$. The complexity of identifying local peaks using Condition (1) is $O\left(\sum_{c=1}^{C} L_n^c\right)$ and that of extracting soft prototypes is $O\left(M\sum_{c=1}^{C} L_n^c P_n^c\right)$. Hence, the overall complexity of Step 2 is $O\left(M\sum_{c=1}^{C} L_n^c P_n^c + \sum_{c=1}^{C}\left(L_n^c\right)^2\right)$. The computational complexity of selecting members of $\mathbf{P}_n^c$ to join $\mathbf{S}^c$ using Condition (2) $(c = 1, 2, \cdots, C)$ and updating $\mathbf{S}^c$ using the remaining members of $\mathbf{P}_n^c$ in Step 3 is $O\left(M\sum_{c=1}^{C} S^c P_n^c\right)$. Different from other steps, Step 4 is only activated once every T learning cycles. At the learning cycle that Step 4 is activated, the complexity of removing $\mathbf{R}^c$ from $\mathbf{S}^c$ using Condition (3) is $O\left(M\sum_{c=1}^{C}\left(S^c\right)^2\right)$, and the complexity of updating $\mathbf{S}^c$ with $\mathbf{R}^c$ is $O\left(M\sum_{c=1}^{C} R^c S^c\right)$. Hence, the complexity of Step 4 is $O\left(M\sum_{c=1}^{C}\left(R^c + S^c\right)S^c\right)$. Compared with the previous four steps, the computational complexity of Step 5 is negligible. Therefore, the computational complexity to process a data chunk for SPAFIS is $O\left(\left(G + M^2\right)L_n^2 + M\sum_{c=1}^{C}\left(S^c + L_n^c\right)P_n^c + \sum_{c=1}^{C}\left(L_n^c\right)^2\right)$ if Step 4 is not activated and $O\left(\left(G + M^2\right)L_n^2 + M\sum_{c=1}^{C}\left(\left(S^c + L_n^c\right)P_n^c + \left(R^c + S^c\right)S^c\right) + \sum_{c=1}^{C}\left(L_n^c\right)^2\right)$ otherwise.

During the decision making stage, for each unlabelled testing sample, $\boldsymbol{x}_k$, the computational complexity of calculating the confidence scores by the C IF-THEN rules is $O\left(M^2\sum_{c=1}^{C}S^c\right)$, and that of determining the class label is $O(C)$. Hence, given L testing samples, the overall computational complexity to determine their class labels is $O\left(LM^2\sum_{c=1}^{C}S^c\right)$.

4. Experimental investigation

4.1. Configuration

A. Data Description: To demonstrate the performance of SPAFIS, numerical examples based on publicly available benchmark datasets for network intrusion detection are presented. In particular, the following four widely recognized benchmark datasets are employed for experiments [4]:

1) KDDCUP99 dataset[1] [25]. KDDCUP99 is derived from the Defense Advanced Research Projects Agency (DARPA) 1998 datasets [40]. It is a standardized and widely recognized benchmark for evaluating the performance of network intrusion detection systems. This dataset has approximately five million records, covering attacks of four major categories, namely, denial of service (DoS), user to root (U2R), remote to local (R2L), and probe. The main issue of KDDCUP99 is the redundancy of the records.
2) NSL-KDD dataset.[2] [25] NSL-KDD is a distilled version of KDDCUP99 dataset by redundance removal and size reduction. NSL-KDD contains three subsets, which include 1) a full training set (KDDTrain^{+}), 2) a full testing set (KDDTest^{+}), and; 3) a more challenging subset of the testing set (KDDTest^{-21}).
3) UNSW-NB15 dataset.[3] [41] UNSW-NB15 is a modern benchmark dataset constructed by researchers at the Australian Centre for Cyber Security at the University of New South Wales. The dataset was created using the IXIA PerfectStorm platform to create a combination of real modern regular activities and synthetic recent attack behaviours, hence, providing a better representation of contemporary traffic patterns. UNSW-NB15 has one training set and one testing set.
4) HIKARI-2021 dataset.[4] [42] HIKARI-2021 is one of the latest benchmark datasets for evaluating the performances of IDSs built by researchers at Keio University, Japan. This dataset contains a mix of encrypted synthetic attacks and benign real traffic, reflecting the up-to-date landscape of cyber-attacks.

Key information of the datasets is summarized in Table 1.

Following the common practice [9,10], KDDCUP99, NSL-KDD and UNSW-NB15 have been pre-processed by converting the categorical attributes to numerical ones via one-hot encoding. For HIKARI-2021 dataset, the four dataset specific columns including source IP address (originh), source port (originp), destination IP address (responh), and destination port (responp) are removed in advance following [43]. The value ranges of all attributes in the four datasets are further standardized to eliminate the influence of the measurement unit on the model training [41]. Due to the very large size of KDDCUP99 dataset, it will be only used for experimental demonstration in online scenarios. To facilitate experimental simulation, 50 % of HIKARI-2021 dataset is randomly selected for each experiment and the training–testing split ratio is set as 4:1 [42].

In this study, intrusion detection from large-scale network activities is considered as a binary classification task [9], where the main aim is to classify each observed network activity into either the normal class or malicious class. To evaluate the performance of the proposed SPAFIS on multi-class classification tasks, data samples of KDD99 and NSL-KDD datasets are dividing the data into five major categories, which include normal, DoS, U2R, R2L and probe, according to the specific attack types following the common practice [10,44] (see Table 2).

B. Parameter Setting for SPAFIS: As mentioned in Section 3, SPAFIS has three externally controlled parameters to be predefined by users, namely, G, ω_o and T. In particular, G controls the level of granularity in soft prototype identification. A greater G enables SPAFIS to identify more soft prototypes, thereby, focusing on the local patterns of data, and a smaller G helps SPAFIS to focus on the global patterns of data and identify less soft prototypes. Both ω_o and T are related to soft prototype pruning. SPAFIS will keep more distinctive soft prototypes in the knowledge base with a greater ω_o, and will be more tolerant towards soft prototypes with higher similarities if ω_o is set to be a small value. T controls the frequency of SPAFIS to perform pruning. A greater T means SPAFIS will perform soft prototype pruning less often. On the other hand, a smaller T helps SPAFIS to remove redundant soft prototypes timely and maintain a more compact knowledge base. However, performing soft prototype pruning too frequently can inevitably decrease the computational efficiency of SPAFIS. The recommended values for the three parameters are given as $G = 6$, $\omega_o = 0.001$ and $T = 10$. Unless specifically declared otherwise, the experimental results reported in this paper are obtained using the recommended setting by default.

Although the size of each data chunk, L_n ($n = 1, 2, 3, \cdots$) is related to the nature of the data streams to some extent, the sizes of different data chunks are assumed to be uniformly the same, namely, $L_n = L_o \forall n$ for simplification. In this study, $L_o = 2000$ is used. If the amount of remaining data samples is less than L_o, all the remaining samples will be included in the final data chunk for algorithm training/testing.

[1] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_kddcup99.html.
[2] Available at: https://www.unb.ca/cic/datasets/nsl.html.
[3] Available at: https://research.unsw.edu.au/projects/unsw-nb15-dataset.
[4] Available at: https://www.kaggle.com/datasets/kk0105/allflowmeter-hikari2021.

Table 1
Key details of benchmark datasets for network intrusion detection.

Dataset		#(Samples)			#(Attributes)
		Total	Normal	Attack	
KDDCUP99		4,894,831	3,925,650	972,781	38 numerical ones + 3 categorical ones + 1 class label
NSL-KDD	$KDDTrain^{+}$	125,973	67,343	58,640	
	$KDDTest^{+}$	22,544	9711	12,833	
	$KDDTest^{-21}$	11,850	2152	9698	
UNSW-NB15	Training	175,341	56,000	119,341	39 numerical ones + 3 categorical ones + 1 class label
	Testing	82,332	37,000	45,332	
HIKARI-2021		555,278	517,582	37,696	83 numerical ones + 1 class label

Table 2
Five major categories of KDD99 and NSL-KDD [10,44].

Category	Attack Types
Normal	normal
DoS	apache2, back, land, mailbomb, neptune, pod, smurf, teardrop, worm, processtable, udpstorm
U2R	bufferoverflow, ps, perl, loadmodule, rootkit, sqlattack, xterm
R2L	spy, warezclient, ftpwrite, guesspasswd, imap, httptunnel, multihop, named, phf, warezmaster, sendmail, snmpgetattack, snmpguess, wxlock, xsnoop
Probe	ipsweep, nma, nmap, portsweep, satan, saint

C. Parameter Settings for SOTA Comparative Algorithms: The following nine SOTA offline classification algorithms are employed for performance comparison:

1) Decision tree (DT) [5];
2) Random forest (RF) [6];
3) K-nearest neighbour classifier (KNN) [8];
4) Support vector machine (SVM) [45];
5) Extreme gradient boosting (XGBoost) [46];
6) Recurrent neural network (RNN) [27];
7) Long short-term memory network (LSTM) [27];
8) Convolutional neural network (CNN) [10], and;
9) Bidirectional long short-term memory with attention mechanism (BAT) [9].

In running the numerical experiments, the maximum depth is set to be $L-1$ for DT to allow the tree structure to fully grow (L is the total amount of labelled data samples presented to DT). RF is composed of 40 DTs with the maximum depth set as $L-1$. The number of nearest neighbours, k is set to be 3 for KNN. SVM uses the linear kernel, and the box constraint is 1. The number of DTs in XGBoost is set as 40, $\eta = 0.3$, and the maximum depth of DTs is set as 40. For RNN, the same architecture and parameter setting as given by [27] is adopted. As [27] does not give the exact setting of batch size, the batch size for RNN is set as 50 in this study. The LSTM follows the same parameter setting and architecture as the RNN except that the recurrent neurons are replaced by LSTM neurons. Similarly, the same architecture and parameter setting of CNN given by [10] are used, and the batch size is set as 50 due to the same reason. The setting of BAT follows [9] with the kernel size for 1D convolutional layers set as 4 due to the lack of precise parameter setting given by the original literature.

In addition to the nine offline algorithms, the following six EFSs that are designed to learn from data streams in nonstationary environments are also involved in performance comparison:

10) Self-adaptive fuzzy learning system (SAFL) [17];
11) Statistically evolving fuzzy inference system (SEFIS) [47];
12) Zero-order evolving fuzzy rule-based classifier (eClass0) [18];
13) Self-organizing fuzzy belief inference system (SOFBIS) [20];
14) eClass0 ensemble classifier (eEnsemble) [48], and;
15) Self-organizing fuzzy inference ensemble system (SOFEnsemble) [36].

Among the six EFSs for experimental comparison, SOFBIS, eEnsemble and SOFEnsemble are zero-order EFSs for data stream classification. SAFL, SEFIS and ESAFIS are first-order EFSs widely used for regression tasks. Generally, zero-order EFSs have higher computational efficiency than first-order ones because they are prototype-based and do not have trainable parameters in their consequent parts. In this study, SAFL and SEFIS use the commonly used "one-versus-rest" strategy for classification. SAFL, eClass0, SOFBIS and SOFEnsemble follow the same settings given by [17,18,20,36], respectively. The externally controlled parameters of SEFIS are set as: $K = 0.5$; $\delta_1 = 0.5$; $\delta_2 = 0.5$, and $p_0 = 2$. eEnsemble is composed of 10 base components and other parameters follow the

setting of [48].

The externally controlled parameter settings for the proposed SPAFIS and 15 comparative algorithms involved in numerical experiments are summarized in Table 3 for clarity.

D. Performance Measures: For performance evaluation, four standard criteria for classification are employed, which include 1) accuracy (*acc*); 2) balanced accuracy (*bacc*) [49]; 3) F1 score (*F*1), and; 4) Matthew's correlation coefficient (*mcc*) [50]. Note that the weighted F1 score is used for multi-class problems because the F1 score was originally designed for binary problems. By default, all the reported results in this study are obtained as the average of 10 Monte Carlo experiments by randomly shuffling the order of training data samples to allow a certain degree of randomness.

The proposed SPAFIS was implemented using Python 3.9. The performance evaluation was conducted on a laptop with i7-12700H processor, 64 GB RAM and RTX 3050 Ti GPU.

4.2. Sensitivity analysis

To understand the influence of the four parameters (namely, G, ω_o, T and L_o) on the performance of SPAFIS, a sensitivity analysis is carried out in this subsection. To facilitate computation, the first 20 % data samples of KDDTrain$^+$ and KDDTest$^+$ of the NSL-KDD datasets are selected as the respective training and testing sets for conducting the experiments in this subsection.

Firstly, the influence of the level of granularity, G on the performance of SPAFIS is investigated. In this experiment, the value of G is varied from 2 to 9, and the other three parameters follow the default setting, namely, $\omega_o = 0.001$, $T = 10$ and $L_o = 2000$. The classification performance of SPAFIS with different values of G achieved on the first 20 % data of KDDTest$^+$ is reported in Table 4 in terms of the four criteria. The total number of prototypes identified from training data by SPAFIS, denoted as S ($S = \sum_{c=1}^{C} S^c$) is reported in the same table.

It can be seen from Table 4 that with a higher level of granularity, SPAFIS is able to identify more soft prototypes from data, and achieves greater classification performance on the testing data. On the other hand, SPAFIS tends to be overfitted if the value of G is too large. With an overlarge G, SPAFIS may identify too many soft prototypes from data, and focus too much on unnecessarily details of the local patterns of data, leading to poorer performance. As suggested by Table 4, a suitable value range of G is between 4 and 7.

Next, the influence of ω_o on the performance of SPAFIS is investigated. In this experiment, the value of ω_o is reduced from 0.1 to 0.0005 with a nonlinear step size. G, T and L_o use the default setting. The performance of SPAFIS with different ω_o values are reported in Table 5 in terms of S, *acc*, *bacc*, *F*1 and *mcc*. Since both ω_o and T are used for soft prototype pruning, the influence of T on the system performance is also investigated in this example. Similar to the previous experiments, the other three parameters G, ω_o and L_o follow the default setting, and the value of T is increased from 2 to 12 with a step size of 2. The obtained experimental results by SPAFIS are also reported in Table 5 for visual clarity.

As aforementioned, ω_o controls the tolerance of SPAFIS on the similarity between soft prototypes, and T determines how frequently the soft prototype pruning is performed. One can see from Table 5 that a small ω_o enables more soft prototypes being preserved in the knowledge base after pruning, whilst a small T enables SPAFIS to perform pruning more frequently, resulting in a smaller knowledge base. Nevertheless, one can see that both ω_o and T have marginal influence on the classification performance of SPAFIS.

Finally, the influence of chunk size, L_o on the performance of SPAFIS is analysed. In this example, the value of L_o is changed from 500 to 4000 with the interval of 500 and the other three parameters are set as $G = 6$, $\omega_o = 0.0001$ and $T = 10$. The results obtained by SPAFIS with different L_o values are presented in Table 6 based on the same five measures used in Tables 4 and 5. Table 6 shows that the value of L_o can influence the size of the knowledge base of SPAFIS. Generally, a smaller chunk size will make SPAFIS identify more soft prototypes because it makes small drifts and/or shifts in the underlying patterns of data carried by successive chunks more noticeable.

Although the best parameter setting is always different from problem to problem depending on the nature of data, it will be demonstrated by the numerical examples presented in the following section that SPAFIS is able to achieve high-level classification performance surpassing, or at least on par with the SOTA algorithms involved in experimental comparison with the recommended

Table 3
Externally controlled parameter settings of the algorithms for numerical experiments.

Algorithm	Parameter Setting	Algorithm	Parameter Setting
SPAFIS	$G = 6$; $\omega_o = 0.001$; $T = 10$; $L_o = 2000$;	CNN	*batch size* = 50;other parameters same as [10];
DT	*max depth* $= L - 1$;	BAT	1*D conv kernel size* = 1; other parameters same as [9];
RF	*number of estimator* = 40; *max depth* $= L - 1$;	SAFL	same as [17];
KNN	$k = 3$	SEFIS	$K = 0.5$; $\delta_1 = 0.5$; $\delta_2 = 0.5$; $p_0 = 2$;
SVM	*box constraint* = 1; *linear*;	eClass0	same as [18];
XGBoost	*number of estimator* = 40; $\eta = 0.3$; *max depth* = 40;	SOFBIS	same as [20];
RNN	*batch size* = 50; others same as [27];	eEnsemble	*number of estimator* = 10; other parameters same as [9]
LSTM	*batch size* = 50;other parameters same as [27];	SOFEnsemble	same as y[36];

Table 4
Influence of G on the performance of SPAFIS.

Meas.	*G*							
	2	3	4	5	6	7	8	9
S	283.6	517.8	996.2	1653.6	2589.4	3829.7	5474.5	7698.1
acc	0.7583	0.7954	0.8030	0.8124	0.8083	0.7982	0.7833	0.7694
bacc	0.7858	0.8170	0.8231	0.8317	0.8281	0.8192	0.8062	0.7941
*F*1	0.7364	0.7874	0.7978	0.8090	0.8038	0.7913	0.7723	0.7541
mcc	0.5945	0.6427	0.6521	0.6672	0.6613	0.6460	0.6242	0.6039

Table 5
Influence of ω_o and T on the performance of SPAFIS.

Meas.	ω_o					
	0.1	0.05	0.01	0.005	0.001	0.0005
S	2449.2	2491.5	2546.2	2568.8	2581.8	2596.9
acc	0.8084	0.8060	0.8063	0.8077	0.8086	0.8076
bacc	0.8282	0.8256	0.8263	0.8276	0.8283	0.8275
*F*1	0.8040	0.8017	0.8014	0.8030	0.8042	0.8029
mcc	0.6614	0.6560	0.6581	0.6604	0.6617	0.6603

Meas.	*T*					
	2	4	6	8	10	12
S	2574.8	2582.3	2592.1	2592.3	2585.5	2588.8
acc	0.8078	0.8092	0.8042	0.8088	0.8096	0.8050
bacc	0.8276	0.8289	0.8235	0.8286	0.8292	0.8243
*F*1	0.8032	0.8050	0.8002	0.8045	0.8054	0.8012
mcc	0.6605	0.6625	0.6515	0.6620	0.6632	0.6528

Table 6
Influence of L_o on the performance of SPAFIS.

Meas.	L_o							
	500	1000	1500	2000	2500	3000	3500	4000
S	2769.2	2700.1	2627.6	2575.9	2538.2	2567.7	2549.8	2534.4
acc	0.8099	0.8050	0.8067	0.8071	0.8063	0.8069	0.8078	0.8075
bacc	0.8295	0.8242	0.8262	0.8266	0.8259	0.8268	0.8277	0.8274
*F*1	0.8058	0.8011	0.8025	0.8031	0.8021	0.8021	0.8032	0.8030
mcc	0.6638	0.6528	0.6570	0.6578	0.6566	0.6591	0.6605	0.6599

Table 7
Binary classification performance comparison on NSL-KDD.

Algorithm	$KDDTest^{+}$				$KDDTest^{-21}$			
	acc	*bacc*	*F*1	*mcc*	*acc*	*bacc*	*F*1	*mcc*
SPAFIS	**0.8064**	**0.8252**	**0.8020**	**0.6551**	**0.6334**	0.7103	**0.7246**	0.3244
DT	0.7988	0.8188	0.7922	0.6448	0.6210	**0.7127**	0.7104	**0.3281**
RF	0.7736	0.7979	0.7579	0.6124	0.5693	0.6896	0.6554	0.2952
KNN	0.7695	0.7888	0.7625	0.5835	0.5618	0.6059	0.6672	0.1633
SVM	0.7507	0.7718	0.7389	0.5530	0.5271	0.5802	0.6323	0.1239
XGBoost	0.7918	0.8136	0.7821	0.6380	0.6041	0.7075	0.6926	0.3205
RNN	0.7956	0.8130	0.7925	0.6294	0.6123	0.6582	0.7113	0.2454
LSTM	0.8020	0.8191	0.7997	0.6405	0.6250	0.6744	0.7222	0.2700
CNN	0.7909	0.8077	0.7889	0.6172	0.6027	0.6335	0.7068	0.2064
BAT	0.7758	0.7942	0.7705	0.5928	0.5740	0.6122	0.6794	0.1732
SAFL	0.7770	0.7950	0.7814	0.5940	0.5772	0.6131	0.3652	0.1745
SEFIS	0.7426	0.7614	0.7483	0.5334	0.5604	0.6477	0.3945	0.2316
eClass0	0.6843	0.7203	0.7304	0.4914	0.4289	0.6191	0.3728	0.2061
SOFBIS	0.7549	0.7802	0.7720	0.5797	0.5357	0.6522	0.3952	0.2383
eEnsemble	0.6337	0.6744	0.6961	0.4112	0.3683	0.5628	0.3325	0.1193
SOFEnsemble	0.7675	0.7870	0.7746	0.5802	0.5580	0.6026	0.3560	0.1582

parameter setting given in Section 4.1. However, to achieve the best performance, one may need to utilize prior knowledge of the problem to adjust the externally controlled parameters accordingly.

4.3. Performance demonstration in offline scenarios

In this subsection, numerical examples on the aforementioned large-scale benchmark datasets for network intrusion detection are presented to demonstrate the performance of SPAFIS under the standard experimental protocols [10]. A number of SOTA algorithms are employed for performance comparison. All the experiments are conducted in offline manner, namely, the algorithms are firstly trained with all the training samples and then evaluated on the testing samples based on the four performance criteria.

Firstly, the binary classification performance of SPAFIS is evaluated on NSL-KDD dataset. In running the experiments, all the data samples of KDDTrain$^+$ are used for training SPAFIS to distinguish the anomalous activities from normal ones. Then, the performance of the trained model is evaluated on the data samples of KDDTest$^+$ and KDDTest^{-21}. The results obtained by SPAFIS on the two testing sets in terms of the four classification performance criteria are tabulated in Table 7. The classification results obtained by the 15 SOTA classifiers on KDDTest$^+$ and KDDTest^{-21} under the same experimental protocol are reported in Table 7 for comparison. The best result per dataset per criterion is in bold for visual clarity. The average time consumption (t_{exe}, in seconds) for each algorithm to learn from the labelled training data are presented in Fig. 2 in the form of a bar chart.

One can see from Table 7 that SPAFIS trained on KDDTrain$^+$ outperforms all other 15 comparative algorithms by offering the best classification performance on KDDTest$^+$ in terms of all four criteria, and its classification performance on KDDTest^{-21} is also better than alternative algorithms in terms of *acc* and *F*1, despite that DT achieves slightly better results than SPAFIS measured using *bacc* and *mcc*. Fig. 2 shows that the computational efficiency of SPAFIS is significantly higher than SVM and neural network-based approaches such as RNN, LSTM, CNN and BAT. The average time cost for SPAFIS to learn from KDDTrain$^+$ is only slightly higher than the zero-order EFS-based competitors due to the use of soft prototypes.

Next, the binary classification performance of SPAFIS is evaluated on UNSW-NB15 and HIKARI-2021 datasets. In running the experiments, SPAFIS is firstly trained using the training sets of the two benchmark problems, and then the performance is evaluated on the respective testing sets with the aim of identifying anomalous activities within the testing sets. The classification results obtained by SPAFIS and the 15 SOTA comparative algorithms are reported in Table 8 (the best results are in bold), and the average training time costs of the 16 algorithms on the two datasets are presented in Figs. 3 and 4.

It is shown by Table 8, Figs. 3 and 4 that SPAFIS is the best-performing model on the testing set of the UNSW-NB15 problem by offering the greatest anomaly detection results in all four criteria, and it also achieves the best performance on HIKARI-2021 with the best *bacc* and *mcc*. It is also worth noting that the computational efficacy of SPAFIS is also on the same level as alternative zero-order EFS-based approaches.

The fuzzy rule base learned by SPAFIS from the HIKARI-2021 problem during one particular experiment is presented in Table 9 for better illustration. It can be seen from Table 9 that two IF-THEN fuzzy rules are identified by SPAFIS from data (one rule per class). One rule is composed of 21,555 soft prototypes learned from data samples of normal class (benign) and the other rule is composed of 1756 soft prototypes learned from data samples of malicious class (attack). As aforementioned, these soft prototypes represent the unique local patterns of data and can be directly associated to the practical problems and, hence, are always meaningful and interpretable to/by humans. Predictions made by SPAFIS are based on the spatial similarity between soft prototypes and unlabelled data samples. Therefore, the reasoning process of SPAFIS can be traced and explained.

On the other hand, one may find that the number of soft prototypes learned from data might be still large despite being much less than the number of original training data, which could be challenging to fully comprehend. This is due to the high complexity and high dimensionality of the HIKARI-2021 problem. To enhance the interpretability of the proposed model, users can lower the level of granularity to encourage the learning system to focus more on the global patterns of data, thereby reducing the size of the knowledge base (also see Table 4). However, a trade-off between the interpretability and prediction performance will need to be made. Another potential solution is to arrange these soft prototypes in multi-level hierarchies based on their respective descriptive abilities, but this is out of the scope of this study.

Finally, to evaluate the multi-class classification performance of SPAFIS, NSL-KDD dataset is converted into a multi-class classification problem based on Table 2 [10,44]. The multi-class classification performances of SPAFIS and the 15 SOTA comparative algorithms on NSL-KDD are reported in Table 10, following the same experimental protocol used by the numerical example presented in

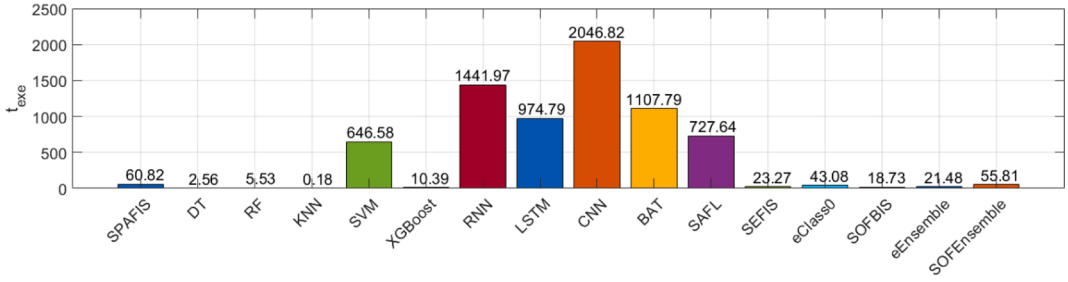

Fig. 2. Average training time consumption of 16 machine learning algorithms on NSL-KDD (binary).

Table 8
Binary classification performance comparison on UNSW-NB15 and HIKARI-2021.

Algorithm	UNSW-NB15				HIKARI-2021			
	acc	*bacc*	*F*1	*mcc*	*acc*	*bacc*	*F*1	*mcc*
SPAFIS	**0.8887**	**0.8897**	**0.8969**	**0.7769**	0.8623	**0.9172**	0.4923	**0.5223**
DT	0.8640	0.8536	0.8856	0.7322	0.8964	0.5761	0.2123	0.1571
RF	0.8727	0.8603	0.8948	0.7563	0.9073	0.5800	0.2278	0.1812
KNN	0.8487	0.8362	0.8748	0.7050	0.9126	0.6073	0.2833	0.2392
SVM	0.8108	0.7897	0.8532	0.6554	0.9326	0.5000	0.0000	0.0000
XGBoost	0.8683	0.8560	0.8910	0.7464	0.9054	0.5764	0.2198	0.1719
RNN	0.8213	0.8035	0.8580	0.6667	0.9125	0.6291	0.2577	0.2637
LSTM	0.8295	0.8153	0.8611	0.6774	0.9197	0.5557	0.1454	0.1406
CNN	0.8656	0.8537	0.8884	0.7395	**0.9333**	0.6079	0.3123	0.3114
BAT	0.8400	0.8240	0.8722	0.6996	0.9321	0.5079	0.0293	0.0395
SAFL	0.7792	0.7611	0.7034	0.5702	0.9275	0.5250	**0.9622**	0.1051
SEFIS	0.6064	0.6114	0.5722	0.2420	0.8431	0.5952	0.9081	0.1510
eClass0	0.6937	0.6867	0.6290	0.3945	0.7453	0.7554	0.8403	0.2918
SOFBIS	0.8433	0.8285	0.7964	0.7002	0.9236	0.5678	0.9598	0.1980
eEnsemble	0.6994	0.6835	0.6034	0.4051	0.7445	0.7702	0.8423	0.3024
SOFEnsemble	0.8505	0.8384	0.8120	0.7079	0.9243	0.5638	0.9603	0.1975

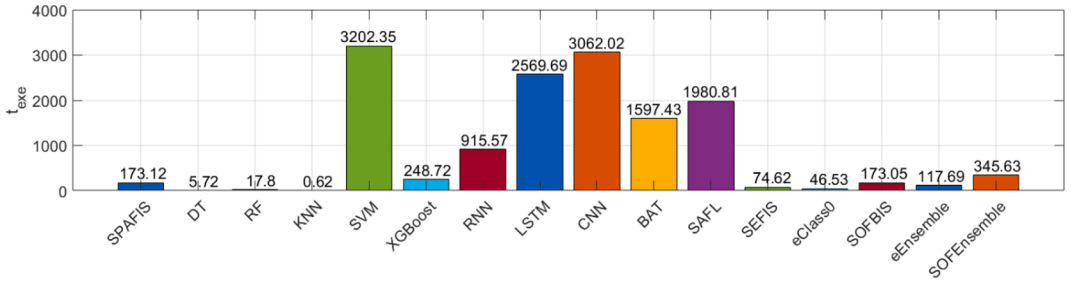

Fig. 3. Average training time consumption of 16 machine learning algorithms on UNSW-NB15 (binary).

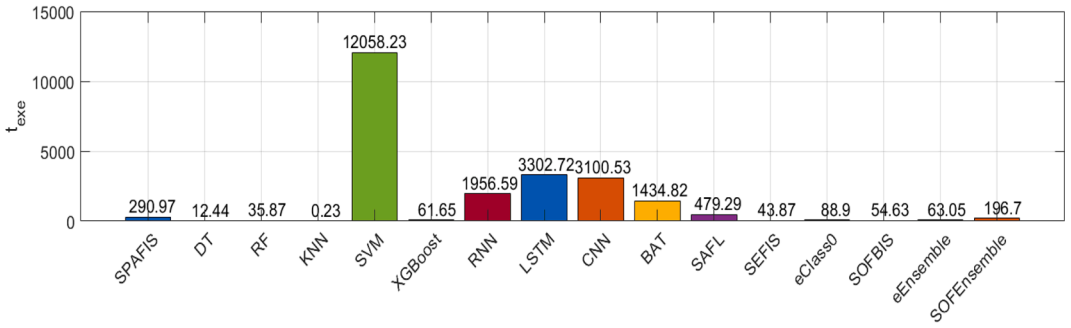

Fig. 4. Average training time consumption of 16 machine learning algorithms on HIKARI-2021 (binary).

Table 7, where the best results are also highlighted. Similarly, the average training time costs of the 16 algorithms are presented in Fig. 5. It can be seen from Table 10 that the *acc* rate and *F*1 score of the classification result obtained by SPAFIS on KDDTest^{+} and KDDTest^{-21} surpass all other comparative algorithm, despite that SVM and XGBoost offer slightly better results than SPAFIS based on *bacc* and *mcc*, respectively.

The numerical examples presented in this section collectively demonstrate the supervisor performance of SPAFIS on network intrusion detection. In particular, Tables 7 and 8 suggest that SPAFIS is able to offer superior anomaly detection performance on NSL-KDD, UNSW-NB15 and HIKARI-2021 datasets with considerably high computational efficiency (see Figs. 2–5), outperforming a variety of SOTA algorithms. Table 9 also shows that the knowledge base learned by SPAFIS can be visualized in the form of human-understandable IF-THEN fuzzy rules, offering great model transparency and interpretability. In addition, numerical example presented in Table 10 also suggests that SPAFIS can further accurately group the detected cyber-attacks into different major categories according to their statistic characteristics.

Table 9
IF-THEN fuzzy rules learned from HIKARI-2021 data by SPAFIS.

Class	IF-THEN Fuzzy Rule
Benign	$\text{IF}\left(\begin{bmatrix} \textit{flow\_duration} \\ \textit{fwd\_pkts\_tot} \\ \textit{bwd\_pkts\_tot} \\ \textit{fwd\_data\_pkts\_tot} \\ \textit{bwd\_data\_pkts\_tot} \\ \vdots \\ \textit{fwd\_last\_window\_size} \end{bmatrix} \begin{bmatrix} 3.1381 \\ 44.8081 \\ 39.0001 \\ 8.9719 \\ 32.8355 \\ \vdots \\ 0.0000 \end{bmatrix}\right)$
	$\text{OR}\left(\begin{bmatrix} \textit{flow\_duration} \\ \textit{fwd\_pkts\_tot} \\ \textit{bwd\_pkts\_tot} \\ \textit{fwd\_data\_pkts\_tot} \\ \textit{bwd\_data\_pkts\_tot} \\ \vdots \\ \textit{fwd\_last\_window\_size} \end{bmatrix} \begin{bmatrix} 0.6328 \\ 30.9921 \\ 28.9921 \\ 3.0000 \\ 24.9921 \\ \vdots \\ 0.0000 \end{bmatrix}\right)$
	$\vdots$
	$\text{OR}\left(\begin{bmatrix} \textit{flow\_duration} \\ \textit{fwd\_pkts\_tot} \\ \textit{bwd\_pkts\_tot} \\ \textit{fwd\_data\_pkts\_tot} \\ \textit{bwd\_data\_pkts\_tot} \\ \vdots \\ \textit{fwd\_last\_window\_size} \end{bmatrix} \begin{bmatrix} 275.8640 \\ 32.0000 \\ 39.0000 \\ 16.0000 \\ 24.0000 \\ \vdots \\ 112.0000 \end{bmatrix}\right)$
	$\text{THEN}(y=0)$
Attack	$\text{IF}\left(\begin{bmatrix} \textit{flow\_duration} \\ \textit{fwd\_pkts\_tot} \\ \textit{bwd\_pkts\_tot} \\ \textit{fwd\_data\_pkts\_tot} \\ \textit{bwd\_data\_pkts\_tot} \\ \vdots \\ \textit{fwd\_last\_window\_size} \end{bmatrix} \begin{bmatrix} 3.2849 \\ 151.0010 \\ 145.0010 \\ 6.0000 \\ 142.0010 \\ \vdots \\ 0.0000 \end{bmatrix}\right)$
	$\text{OR}\left(\begin{bmatrix} \textit{flow\_duration} \\ \textit{fwd\_pkts\_tot} \\ \textit{bwd\_pkts\_tot} \\ \textit{fwd\_data\_pkts\_tot} \\ \textit{bwd\_data\_pkts\_tot} \\ \vdots \\ \textit{fwd\_last\_window\_size} \end{bmatrix} \begin{bmatrix} 1.5286 \\ 3.0000 \\ 0.0000 \\ 3.0000 \\ 0.0000 \\ \vdots \\ 0.0000 \end{bmatrix}\right)$
	$\vdots$
	$\text{OR}\left(\begin{bmatrix} \textit{flow\_duration} \\ \textit{fwd\_pkts\_tot} \\ \textit{bwd\_pkts\_tot} \\ \textit{fwd\_data\_pkts\_tot} \\ \textit{bwd\_data\_pkts\_tot} \\ \vdots \\ \textit{fwd\_last\_window\_size} \end{bmatrix} \begin{bmatrix} 4.2378 \\ 157.9990 \\ 3152.9990 \\ 6.0000 \\ 149.9990 \\ \vdots \\ 0.0000 \end{bmatrix}\right)$
	$\text{THEN}(y=1)$

4.4. Performance demonstration in online scenarios

In this subsection, numerical experiments are carried out in online scenarios to demonstrate the performance of SPAFIS on network intrusion detection from data streams.

Firstly, the online classification performance of SPAFIS is evaluated on NSL-KDD, UNSW-NB15 and IKARI-2021 datasets. In this example, SPAFIS is firstly trained with the training set and then evaluated on the testing set following the standard prequential test-then-train experimental protocol. The test-then-train performance of SPAFIS on the testing sets of the three benchmark problems are

Table 10
Multi-class classification performance comparison on NSL-KDD.

Algorithm	KDDTest^{+}				KDDTest^{-21}			
	acc	*bacc*	*F*1	*mcc*	*acc*	*bacc*	*F*1	*mcc*
SPAFIS	**0.7764**	0.5610	**0.7542**	0.6713	**0.5780**	0.5032	**0.5767**	0.4812
DT	0.7609	0.5453	0.7236	0.6526	0.5505	0.4984	0.5407	0.4757
RF	0.7479	0.4927	0.7057	0.6367	0.5234	0.4430	0.5123	0.4565
KNN	0.7490	0.5403	0.7068	0.6302	0.5232	0.4627	0.5057	0.4197
SVM	0.7656	**0.5752**	0.7310	0.6553	0.5544	**0.5033**	0.5519	0.4610
XGBoost	0.7748	0.5415	0.7414	**0.6743**	0.5722	0.4941	0.5665	**0.5006**
RNN	0.7600	0.5287	0.7236	0.6459	0.5446	0.4574	0.5355	0.4444
LSTM	0.7731	0.5686	0.7420	0.6661	0.5710	0.5011	0.5642	0.4746
CNN	0.7722	0.5525	0.7362	0.6653	0.5669	0.4947	0.5552	0.4759
BAT	0.7415	0.4963	0.6982	0.6193	0.5091	0.4181	0.4899	0.4059
SAFL	0.7573	0.5342	0.7133	0.6412	0.5404	0.4617	0.5156	0.4343
SEFIS	0.6215	0.3721	0.5618	0.4266	0.4015	0.3270	0.3495	0.2393
eClass0	0.5577	0.5283	0.5571	0.3984	0.2897	0.3938	0.2920	0.1459
SOFBIS	0.7436	0.4845	0.7030	0.6271	0.5142	0.4306	0.5074	0.4346
eEnsemble	0.6039	0.5338	0.6031	0.4449	0.3403	0.4119	0.3553	0.2073
SOFEnsemble	0.7410	0.5109	0.6973	0.6163	0.5077	0.4339	0.4896	0.3965

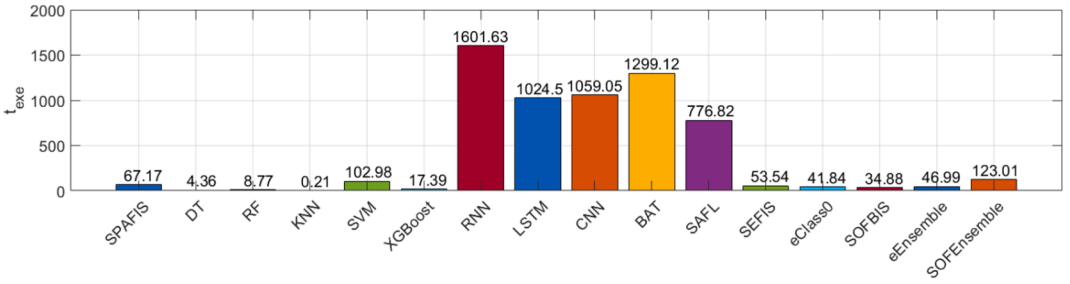

Fig. 5. Average training time consumption of 16 machine learning algorithms on NSL-KDD (multi-class).

tabulated in Table 11 in terms of the four performance criteria used before. Similarly, NSL-KDD is further converted to a multi-class classification problem based on Table 2. Then, the experiments are repeated under the same protocol and the obtained results are also reported in Table 2. For performance comparison, the six EFSs used in the previous examples are involved in this numerical example, and their test-then-train performances obtained under the same experimental protocol are reported in Table 11. The best result per problem per criteria is highlighted in bold.

One can see from Table 11 that SPAFIS outperforms its EFS competitors on the three binary classification problems in online environments by offering the best test-then-train classification results in terms of *mcc* with at least one of the other three criteria higher than the alternative EFS-based approaches. Although its performance on the multi-class NSL-KDD is slightly worse than SOFBIS in terms of *acc, F*1 and *mcc*, SPAFIS still offers the highest *bacc* on this problem. The performance comparison presented in Table 11 shows the great potential of SPAFIS to accurately detect anomalous activities from real-time network data streams with imbalanced class distributions whilst self-adapting its system and *meta*-parameters to changing data patterns.

Next, KDD99 dataset is further used to test the online learning capability of SPAFIS from data streams. To facilitate simulation, 10 % of the data is randomly selected for training SPAFIS from scratch in each experiment and the level of granularity, G is set to be 3. The test-then-train performance of SPAFIS on KDD99 over the learning process is reported in Table 12. Similar to NSL-KDD in the previous example, KDD99 is also converted to a multi-class classification problem using Table 2, and the experiments are repeated with the obtained results reported in Table 12 as well. The experimental results obtained by the six EFSs on KDD99 under the same experimental protocol are tabulated in Table 12 for comparison.

One can see from Table 12 that SPAFIS outperforms all other comparative EFS-based approaches on the task of identifying anomalous activities from normal ones in KDD99. At the same time, it can be observed that the performance of SPAFIS on distinguishing the identified attacks in KDD99 into different major categories is slightly lower than SOFBIS, but still among the best performing models.

The numerical examples presented in this section collectively shows that SPAFIS has strong capability of learning autonomously from data streams and self-updating its structure and parameters to adapt to new data patterns for greater classification performance.

4.5. Additional analysis

Finally, in this section, ablation analysis is conducted to justify the utilization of soft prototypes by SPAFIS. As aforementioned,

Table 11
Test-then-train classification performance comparison on NSL-KDD, UNSW-NB15 and HIKARI-2021.

Dataset	Algorithm	*acc*	*bacc*	*F*1	*mcc*
NSL-KDD	SPAFIS	**0.9511**	0.9538	**0.9560**	**0.9023**
(Binary)	SAFL	0.9019	0.9091	0.8940	0.8103
	SEFIS	0.8038	0.8049	0.7812	0.6055
	eClass0	0.7205	0.7525	0.7520	0.5438
	SOFBIS	0.9502	**0.9544**	0.9446	0.9020
	eEnsemble	0.7310	0.7630	0.7609	0.5654
	SOFEnsemble	0.9239	0.9281	0.9156	0.8494
UNSW-NB15 (Binary)	SPAFIS	**0.8981**	**0.8969**	**0.9076**	**0.7941**
	SAFL	0.7847	0.7673	0.7132	0.5797
	SEFIS	0.5661	0.5502	0.4254	0.1084
	eClass0	0.6885	0.6797	0.6311	0.3658
	SOFBIS	0.8880	0.8784	0.8628	0.7810
	eEnsemble	0.6768	0.6608	0.5826	0.3415
	SOFEnsemble	0.8504	0.8571	0.8473	0.7119
HIKARI-2021	SPAFIS	0.8617	**0.9177**	0.8909	**0.5208**
(Binary)	SAFL	**0.9286**	0.5336	**0.9628**	0.1357
	SEFIS	0.8462	0.6118	0.9146	0.1653
	eClass0	0.7047	0.7752	0.8127	0.2932
	SOFBIS	0.9230	0.5674	0.9595	0.1963
	eEnsemble	0.7308	0.7656	0.8328	0.2911
	SOFEnsemble	0.9261	0.6079	0.9610	0.2798
NSL-KDD	SPAFIS	0.9414	**0.8507**	0.9454	0.9144
(Multi-Class)	SAFL	0.8971	0.7042	0.8915	0.8492
	SEFIS	0.6231	0.4328	0.6227	0.4437
	eClass0	0.5364	0.5068	0.5175	0.3862
	SOFBIS	**0.9491**	0.8018	**0.9471**	**0.9254**
	eEnsemble	0.6939	0.6067	0.7049	0.5606
	SOFEnsemble	0.9127	0.7102	0.9113	0.8706

Table 12
Test-then-train classification performance comparison on KDD99.

Dataset	Algorithm	*acc*	*bacc*	*F*1	*mcc*
KDD99	SPAFIS	**0.9992**	0.9990	**0.9995**	**0.9974**
(Binary)	SAFL	0.9943	0.9951	0.9859	0.9825
	SEFIS	0.5807	0.5764	0.3470	0.1245
	eClass0	0.9450	0.9106	0.8605	0.8303
	SOFBIS	0.9991	**0.9993**	0.9977	0.9972
	eEnsemble	0.9651	0.9215	0.9045	0.8875
	SOFEnsemble	0.9858	0.9897	0.9654	0.9572
KDD99	SPAFIS	0.9989	0.7812	0.9990	0.9968
(Multi-Class)	SAFL	0.9904	0.5995	0.9910	0.9714
	SEFIS	0.5399	0.2299	0.5889	0.1075
	eClass0	0.8934	0.7561	0.9246	0.7363
	SOFBIS	**0.9991**	0.7529	**0.9991**	**0.9973**
	eEnsemble	0.6768	0.6608	0.5826	0.3415
	SOFEnsemble	0.8504	**0.8571**	0.8473	0.7119

Table 13
Ablation analysis results.

Dataset			Algorithm	*acc*	*bacc*	*F*1	*mcc*
Binary	NSL-KDD	KDDTest^{+}	SPAFIS	**0.8064**	**0.8252**	**0.8020**	**0.6551**
			CPAFIS	0.7535	0.7645	0.7601	0.5265
		KDDTest^{-21}	SPAFIS	**0.6334**	**0.7103**	**0.7246**	**0.3244**
			CPAFIS	0.6173	0.6769	0.7139	0.2730
	UNSW-NB15		SPAFIS	**0.8887**	0.8897	**0.8969**	0.7769
			CPAFIS	0.8857	**0.8935**	0.8872	**0.7852**
Multi-Class	NSL-KDD	KDDTest^{+}	SPAFIS	**0.7764**	**0.5610**	**0.7542**	**0.6713**
			CPAFIS	0.6984	0.5250	0.6784	0.5486
		KDDTest^{-21}	SPAFIS	**0.5780**	**0.5032**	**0.5767**	**0.4812**
			CPAFIS	0.5726	0.4966	0.5716	0.4706

compared with crisp prototypes used by conventional prototypes, soft prototypes can better preserve the underlying structure and local patterns of data, thereby helping SPAFIS achieve better classification performance. In this example, an alternative version of SPAFIS that uses crisp prototypes is implemented for performance demonstration, denoted as crisp prototype-based autonomous fuzzy inference system (CPAFIS). The classification performances of SPAFIS and CPAFIS are compared on NSL-KDD and UNSW-NB15 under the same experimental protocols used before and the obtained results are reported in Table 13 with the best result per criterion per dataset in bold.

Ablation analysis results presented in Table 13 show that the utilization of soft prototypes effectively help SPAFIS achieve greater classification performance. In particular, SPAFIS outperforms CPAFIS on NSL-KDD dataset in both binary and multi-class cases in terms of all four performance criteria. Both SPAFIS and CPAFIS achieve comparable performances on UNSW-NB15 dataset, but SPAFIS achieves slightly higher *acc* and $F1$ on this problem. This comparison demonstrates the advantages of soft prototypes over crisp prototypes.

5. Conclusion

In this paper, a novel zero-order EFS named SPAFIS has been proposed for network intrusion detection. SPAFIS is able to learn a set of human-interpretable IF-THEN fuzzy rules from network activities in real-time, and its system structure and *meta*-parameters are consistently self-evolving to adapt to new data patterns. Thanks to the utilization of soft-prototypes, SPAFIS can better approximate the multimodal distributions of data compared with conventional zero-order EFSs utilizing crisp prototypes. Numerical examples on four public benchmark datasets for network intrusion detection demonstrated the superior intrusion detection performance of the proposed SPAFIS-based IDS in both offline and online settings.

There are several considerations for future work. First, the optimality of the soft prototypes learned by SPAFIS needs to be investigated. As mentioned in Section 3, each soft prototype represents a local peak of the multimodal distribution of data. However, considering the nonstationary nature of data streams, soft prototypes learned by SPAFIS may not be (locally) optimal because of the chunk-wise online learning mechanism. Therefore, it would be very helpful to perform soft prototype optimization periodically to maintain the local optimality of the learned solution, for example, by using genetic algorithms. Second, as one can see from the sensitivity analysis in Section 4.2, the level of granularity is the most important parameter externally controlled by users as its setting can greatly influence the performance of SPAFIS. Although this study has given a recommended setting, the best setting is always different from problem to problem. Therefore, it will be extremely helpful to develop a novel scheme to allow SPAFIS self-determine the best value setting for the level of granularity based on the nature of data. Third SPAFIS has been equipped with a soft prototype pruning scheme to remove redundant soft prototypes from the knowledge base, but its computational efficiency will decrease inevitably with more distinctive soft prototypes learned from data. One possible solution to address this is to utilize ensemble learning framework such that a number of ensemble components can share the computation burden. How to design an effective ensemble scheme for online learning that helps SPAFIS to achieve greater prediction accuracy with higher computational efficiency remains a question to be answered. Last, but not the least, currently SPAFIS needs to be trained with labelled training data to build a predictive model. However, in real applications, the amount of labelled data may be very limited whilst unlabelled data is abundant. This is especially the case for network intrusion detection. Therefore, it would be extremely useful to extend the SPAFIS with a semi-supervised learning scheme such that SPAFIS can utilize the large amount of unlabelled data to self-improve its knowledge base with minimal human supervision after being primed with a small amount of labelled training data.

CRediT authorship contribution statement

Xiaowei Gu: Conceptualization, Formal analysis, Funding acquisition, Methodology, Writing – original draft, Writing – review & editing. **Gareth Howells:** Conceptualization, Funding acquisition, Methodology, Writing – original draft, Writing – review & editing. **Haiyue Yuan:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Xiaowei Gu, Gareth Howells, Haiyue Yuan reports financial support was provided by Defence Science and Technology Laboratory. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

Research funded by Frazer-Nash Consultancy Ltd. on behalf of the Defence Science and Technology Laboratory (Dstl) which is an executive agency of the UK Ministry of Defence providing world class expertise and delivering cutting-edge science and technology for the benefit of the nation and allies. The research supports the Autonomous Resilient Cyber Defence (ARCD) project within the Dstl

Cyber Defence Enhancement programme.

References

[1] "Cyber security breaches survey," *UK Government*, 2023. https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2023.
[2] F. Noorbehbahani, A. Fanian, R. Mousavi, H. Hasannejad, An incremental intrusion detection system using a new semi-supervised stream classification method, Int. J. Commun. Syst. 30 (4) (2017) 1–26.
[3] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, Survey of intrusion detection systems: techniques, datasets and challenges, Cybersecurity 2 (1) (2019) 1–22.
[4] K. Shaukat, S. Luo, V. Varadharajan, I. Hameed, M. Xu, A Survey on machine learning techniques for cyber security in the last decade, IEEE Access 8 (2020) 222310–222354.
[5] S. Safavian, D. Landgrebe, A survey of decsion tree classifier methodology, IEEE Trans. Syst. Man Cybern. 21 (3) (1990) 660–674.
[6] L. Breiman, Random forests, Mach. Learn. Proc. 45 (1) (2001) 5–32.
[7] S. Horng, et al., A novel intrusion detection system based on hierarchical clustering and support vector machines, Expert Syst. Appl. 38 (1) (2011) 306–313.
[8] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, ACM SIGMOD Rec. (2000) 427–438.
[9] T. Su, H. Sun, J. Zhu, S. Wang, Y. Li, BAT: deep learning methods on network intrusion detection Using NSL-KDD dataset, IEEE Access 8 (2020) 29575–29585.
[10] Y. Ding, Y. Zhai, Intrusion detection system for NSL-KDD dataset using convolutional neural networks, in: International Conference on Computer Science and Artificial Intelligence, 2018, pp. 81–85.
[11] M. Data, M. Aritsugi, T-DFNN: an incremental learning algorithm for intrusion detection systems, IEEE Access 9 (2021) 154156–154171.
[12] C. Rudin, Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, Nat. Mach. Intell. 1 (5) (2019) 206–215.
[13] X. Gu, J. Han, Q. Shen, P. Angelov, Autonomous learning for fuzzy systems: a review, Artif. Intell. Rev. (2022) 1–47.
[14] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, K. Müller, Layer-wise relevance propagation: an overview, in: Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, 2019, pp. 193–209.
[15] T. Szandala, Enhancing deep neural network saliency visualizations with gradual extrapolation, IEEE Access 9 (2021) 95155–95161.
[16] E. Lughofer, P. Angelov, Handling drifts and shifts in on-line data streams with evolving fuzzy systems, Appl. Soft Comput. 11 (2) (2011) 2057–2068.
[17] X. Gu, Q. Shen, A self-adaptive fuzzy learning system for streaming data prediction, Inf. Sci. (NY) 579 (2021) 623–647.
[18] P. Angelov, X. Zhou, Evolving fuzzy-rule based classifiers from data streams, IEEE Trans. Fuzzy Syst. 16 (6) (2008) 1462–1474.
[19] X. Gu, P. Angelov, Self-organising fuzzy logic classifier, Inf. Sci. (NY) 447 (2018) 36–51.
[20] X. Gu, P. Angelov, Q. Shen, Self-organizing fuzzy belief inference system for classification, IEEE Trans. Fuzzy Syst. 30 (12) (2022) 5473–5483.
[21] E. Mansoori, FRBC: a fuzzy rule-based clustering algorithm, IEEE Trans. Fuzzy Syst. 19 (5) (2011) 960–971.
[22] Y. Liao, V. Vemuri, Use of k-nearest neighbor classifier for intrusion detection, Comput. Secur. 21 (5) (2002) 439–448.
[23] S. Sahu, B. Mehtre, Network intrusion detection system using J48 decision tree, in: International Conference on Advances in Computing, Communications and Informatics, 2015, pp. 2023–2026.
[24] A. Tesfahun, D. Bhaskari, Intrusion detection using random forests classifier with SMOTE and feature reduction, in: International Conference on Cloud and Ubiquitous Computing and Emerging Technologies, 2013, pp. 127–132.
[25] M. Tavallaee, E. Bagheri, W. Lu, A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 1–6.
[26] A. Aburomman, M. Bin Ibne Reaz, A novel SVM-kNN-PSO ensemble method for intrusion detection system, Appl. Soft Comput. J. 38 (2016) 360–372.
[27] C. Yin, Y. Zhu, J. Fei, X. He, A deep learning approach for intrusion detection using recurrent neural betworks, IEEE Access 5 (2017) 21954–21961.
[28] I. Sharafaldin, A. Lashkari, A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, in: International Conference on Information Systems Security and Privacy, 2018, pp. 108–116.
[29] M. Yousefi-azar, V. Varadharajan, L. Hamey, U. Tupakula, Autoencoder-based feature learning for cyber security applications, in: International Joint Conference on Neural Networks, 2017, pp. 3854–3861.
[30] S. Sivatha Sindhu, S. Geetha, A. Kannan, Decision tree based light weight intrusion detection using a wrapper approach, Expert Syst. Appl. 39 (2012) 129–141.
[31] J. Iglesias, P. Angelov, A. Ledezma, A. Sanchis, Creating evolving user behavior profiles automatically, IEEE Trans. Knowl. Data Eng. 24 (5) (2012) 854–867.
[32] J. Iglesias, A. Ledezma, A. Sanchis, Evolving systems for computer user behavior classification, in: IEEE Conference on Evolving and Adaptive Intelligent Systems, 2013, pp. 78–83.
[33] I. Škrjanc, S. Ozawa, T. Ban, D. Dovžan, Large-scale cyber attacks monitoring using evolving Cauchy possibilistic clustering, Appl. Soft Comput. 62 (2018) 592–601.
[34] L. Decker, D. Leite, L. Giommi, D. Bonacorsi, Real-time anomaly detection in data centers for log-based predictive maintenance using an evolving fuzzy-rule-based approach, in: IEEE International Conference on Fuzzy Systems, 2020, pp. 1–8.
[35] J. Shao, F. Huang, Q. Yang, G. Luo, Robust prototype-based learning on data streams, IEEE Trans. Knowl. Data Eng. 30 (5) (2018) 978–991.
[36] X. Gu, P. Angelov, Z. Zhao, Self-organizing fuzzy inference ensemble system for big streaming data classification, Knowledge-Based Syst. 218 (2021) 106870.
[37] D. Ge, X. Zeng, A self-evolving fuzzy system which learns dynamic threshold parameter by itself, IEEE Trans. Fuzzy Syst. 27 (8) (2018) 1625–1637.
[38] J. Bezdek, R. Ehrlich, W. Full, FCM: the fuzzy c-means clustering algorithm, Comput. Geosci. 10 (2–3) (1984) 191–203.
[39] E. Ruspini, J. Bezdek, J. Keller, Fuzzy clustering: a historical perspective, IEEE Comput. Intell. Mag. 14 (1) (2019) 45–55.
[40] W. Lee, S. Stolfo, K. Mok, Adaptive intrusion detection: a data mining approach, Artif. Intell. Rev. 14 (6) (2000) 533–567.
[41] N. Moustafa, J. Slay, The evaluation of network anomaly detection systems: statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set, Inf. Secur. J. 25 (1–3) (2016) 18–31.
[42] A. Ferriyan, A. Thamrin, K. Takeda, J. Murai, Generating network intrusion detection dataset based on real and encrypted synthetic attack traffic, Appl. Sci. 11 (17) (2021) 7868.
[43] M. Verkerken, L. Dhooge, T. Wauters, B. Volckaert, F. De Turck, Towards model generalization for intrusion detection: unsupervised machine learning techniques, J. Netw. Syst. Manag. 30 (1) (2022) 1–25.
[44] R. Elhefnawy, H. Abounaser, A. Badr, A hybrid nested genetic-fuzzy algorithm framework for intrusion detection and attacks, IEEE Access 8 (2020) 98218–98233.
[45] N. Cristianini, J. Shawe-Taylor, An introduction to support vector machines and other kernel-based learning methods, Cambridge University Press, Cambridge, 2000.
[46] T. Chen, C. Guestrin, Xgboost: a scalable tree boosting system, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
[47] Z. Yang, H. Rong, P. Angelov, Z. Yang, Statistically evolving fuzzy inference system for non-Gaussian noises, IEEE Trans. Fuzzy Syst. 30 (4) (2022) 2649–2664.
[48] J. Iglesias, A. Ledezma, A. Sanchis, Ensemble method based on individual evolving classifiers, in: IEEE Conference on Evolving and Adaptive Intelligent Systems, 2013, pp. 56–61.
[49] K. Brodersen, C. Ong, K. Stephan, J. Buhmann, The balanced accuracy and its posterior distribution, in: International Conference on Pattern Recognition, 2010, pp. 3121–3124.
[50] D. Chicco, G. Jurman, The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation, BMC Genomics 21 (1) (2020) 1–13.