IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 July 2024, accepted 25 August 2024, date of publication 30 August 2024, date of current version 16 September 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3452168

RESEARCH ARTICLE

Comparing Threshold Selection Methods for Network Anomaly Detection

ADRIAN KOMADINA[1], MISLAV MARTINIĆ[2], STJEPAN GROŠ[1], AND ŽELJKA MIHAJLOVIĆ[3], (Member, IEEE)

[1]Laboratory for Information Security and Privacy, Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia
[2]CS Computer Systems, 10000 Zagreb, Croatia
[3]Department of Electronics, Microelectronics, Computer and Intelligent System, Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia

Corresponding author: Adrian Komadina (adrian.komadina@fer.hr)

ABSTRACT The use of unsupervised machine learning models for anomaly detection is a common thing nowadays. While many research papers focus on improving and testing these models, there is a lack of those that deal with threshold selection, which is an important step in implementing a good anomaly detection system. In this paper, we investigate different supervised and unsupervised threshold selection methods found in the network anomaly detection literature. A total of five supervised and twenty unsupervised methods were found, all of which are described, categorized, and implemented in this paper. The unsupervised methods were further categorized according to the input data they expect, the type of output data they produce, and whether they are parametric or not, and divided into six groups according to the idea behind these methods: Statistics-based, Distribution-based, Clustering-based, Density-based, Graphical-based methods and Other. To test all the methods found, two different testing scenarios are created. The first one focuses on using data with anomalies and the second one uses only the normal data. Based on these two scenarios, tests were performed with real firewall log data containing three types of injected anomalies. The results are presented in the form of boxplots of the Matthews correlation coefficient for nine datasets. To draw a conclusion, both the method groups and the individual methods were compared in terms of evaluation metrics and execution times as well as in comparison to the methods already implemented in the *PyThresh* toolkit.

INDEX TERMS Anomaly detection, network data, threshold selection, unsupervised learning.

I. INTRODUCTION

Nowadays, cybersecurity analysts are overloaded with a huge amount of data from various sources. One of the most important tasks of cyber analysts is intrusion detection. In his work, cybersecurity analysts must detect malicious behavior by combining data from low-level logs and high-level alerts [1]. The goal of intrusion detection is to detect the unauthorized use, misuse, and compromise of computer systems by both system insiders and external intruders, preferably in real-time [2]. In other words, it is about detecting actions that attempt to compromise the confidentiality, integrity, or availability of a system/network [3].

In this work, we focus on intrusion in network data or network intrusion detection [4]. The main reason for most of these intrusions are attacks carried out by external adversaries whose goal is to gain unauthorized access to the network and steal information or disrupt the network [5]. Typically, intrusion detection systems are divided into two broad categories: Signature detection systems and Anomaly detection systems [3]. Compared to signature detection systems, anomaly detection systems are capable of detecting previously unknown attacks, which is why they are often considered one of the key components of cybersecurity intrusion detection [6] and thus are the focus of this research.

Before we define what an anomaly detection system is, we must define what an anomaly is. In the literature, an *outlier* or *anomaly* is generally considered to be a data point that is significantly different from other data points or that does not conform to the expected normal pattern [7]. Based on this, the task of anomaly detection is to determine if the test data conforms to the normal data distributions,

The associate editor coordinating the review of this manuscript and approving it for publication was Liangxiu Han.

where anomalies are nonconforming points [8]. The basic idea of anomaly detection is to compare the current traffic with predefined normal baseline traffic. Anomaly detection has two important aspects: It should detect unusual events, and these events should correlate with malicious events [1]. The main drawbacks of this type of intrusion detection system are: Building and maintaining a good profile of normal traffic, and the tendency to generate many false positives because they look for anomalous events rather than attacks [3]. To overcome these drawbacks, anomaly detection systems must overcome two major challenges. The first major challenge is to create an appropriate normal traffic profile, which is a real challenge because defining what is normal in a target network is also a difficult task [3]. The second challenge to consider is the threshold that needs to be chosen. Both of the above challenges could lead to a large number of false positives or, even worse, false negatives.

A typical unsupervised statistical and machine learning procedure for anomaly detection begins by loading some initial data containing records, some of which may be anomalous, and transforming it into a set of records optimized for use with an unsupervised learning model. The selected model is then initialized and fitted to the transformed dataset. The unsupervised model then generates anomaly scores that indicate how certain the model is that some record is anomalous (the higher the score, the more certain the model is). Based on the selected threshold (or thresholds) and the anomaly score, a prediction is then made for each record in the form of a label (zero or one). In this way, the intrusion detection system generates an alert when a given anomaly score is higher than a threshold.

Axelsson [9] has shown that the number of false alarms must be very low for an intrusion detection system to be effective. Axelsson suggested that one false alarm in 100,000 events is the minimum requirement for an effective intrusion detection system. To get a handle on the rate of positive and negative false alarms, a good threshold needs to be calculated. If we want to minimize the number of false negatives, we should set the threshold to a low value. However, this leads to many false positives and reduces the efficiency of the intrusion detection system. It also leads to extra work for the security analysts who have to examine every alert and sort out false positives [3].

There are two different approaches to threshold selection, depending on the situation we are in. The first approach is supervised threshold selection, where we know the target labels before we select the threshold, i.e., which records are anomalous and which are not. In this way, the threshold is selected to correctly predict as many labels as possible. In the second approach, the labels are not known, so we have to select the threshold without knowing which record is anomalous and which is not. Here, we need to select the threshold based only on the anomaly scores. It is important that the threshold is robust to minor changes in the network, to eliminate the problem of false positives, and be sensitive enough to detect true anomalous behavior when it occurs. Another difference between the two approaches to threshold selection is whether a method provides adaptive threshold selection or only static threshold selection. Static threshold selection methods give only one or a set of different thresholds to predict labels based on. However, it is important that the threshold is updated from time to time to adapt to non-anomalous changes in the network. Adaptive threshold selection methods, on the other hand, typically generate a different threshold for each data point. This allows them to be applied to data in the future, making them much less sensitive to changes in the distribution of network data over time. These techniques are particularly useful for data streams where data arrives one at a time and where static threshold methods may result in a high false alarm rate or low anomaly detection rate [10].

Since there is no systematic overview and grouping of threshold selection methods in the network anomaly detection literature, this is our motivation for the research that will lead to such results. Since supervised threshold selection methods are designed to maximize a specific evaluation metric, such as true-positive rate or F1 score, their use really depends on what metric we want to maximize as an outcome of our anomaly detection system. Since they are supervised, they are optimal in terms of the evaluation metric we are trying to achieve. Since we can easily obtain the optimal threshold for the desired application in this way, we focus more on the unsupervised threshold selection methods than the supervised threshold selection methods in this study.

Unsupervised threshold selection methods often use heuristic methods and are based on the particular problem and type of dataset we have. However, we believe that there are some general approaches that can be identified. From these approaches, we can derive different techniques that are optimized for the situation at hand. For this reason, we conducted this research to find as many unsupervised and supervised threshold selection methods that can be used in anomaly detection using network data. We classified the unsupervised threshold selection methods we found into six categories based on the core idea of thresholding. To test all the approaches we found, we created two test scenarios: One based only on normal (benign) data, and the other based on a mixture of normal and anomalous data. By testing all the approaches we found and their variants, we wanted to find out which methods were better at achieving a particular goal and which were better based on the data we had. In our tests, we also included already implemented methods from the *PyThresh* toolkit [11] and we compared the results of the methods found in the literature with those from this toolkit.

The contributions of this research are:

- Listing and categorizing all supervised and unsupervised approaches to threshold selection used in network anomaly detection in the literature found.
- Creating two different test scenarios and testing the found threshold selection methods and their variants on the real firewall data in combination with injected anomalies.

The structure of this paper is as follows. Related work in the target area is presented in Section II. In Sections III and IV, supervised and unsupervised threshold selection methods from the literature are enumerated, described, and discussed. Section V describes the firewall logs and the anomalies used as well as the methodology for testing the selected threshold selection methods. The results of the tests are presented in Section VI and discussed further in the VII section. Finally, conclusions are drawn in the Section VIII.

II. RELATED WORK

In reviewing the literature, our main goal was to find papers that include some sort of review of threshold selection methods for unsupervised network anomaly detection. Unfortunately, we did not find any research paper that provides a structured overview of different threshold selection methods for anomaly detection, especially when it comes to network data. We found only a few papers that included a comparison of some basic threshold selection methods that were used as a basis for comparison with their own threshold selection method implemented for a specific use case.

Bergmann et al. [12] deal with unsupervised anomalous structure detection in natural image data and have presented and compared different methods for threshold estimation, namely: the maximum method, the p-quantile method, the k-sigma method, and the max-area method, which is specific to their application because it takes into account the spatial location of the image pixels.

He et al. [13] presented a paper on a topology-aware anomaly detector for multivariate time series, in which they proposed a threshold selection method based on the distance between sets. They also compared this method with two unsupervised threshold selection methods and one supervised method that enumerates thresholds to achieve the best F1 score in the training dataset. Liu et al. [14] presented a streaming unsupervised threshold selection method based on Peaks-Over-Threshold (POT) and compared it with the k-sigma method using the value 3 for the parameter k. Yang et al. [15] compared an implemented unsupervised threshold selection method based on POT, which produces a static threshold, with the 3-sigma method and the boxplot method.

Killourhy and Maxion [16] conducted a study in which they found seven papers that use anomaly detection to analyze password timing data. Based on these papers, they classified the threshold selection methods found in them into three categories based on how they select the threshold using the Receiver Operating Characteristics (ROC) curve. The first category they found is an approach that uses detector-specific heuristics, which can be problematic when comparing the performance of different detectors [17], [18], [19], [20]. In contrast, the other two categories, i.e., equal error rate and zero-miss false-alarm rate, are detector-independent ways to select the threshold.

In contrast to other authors, Chae et al. [21] represented the network data as a bipartite graph. They developed an adaptive threshold selection method and compared it with five static threshold methods. One static threshold was determined only by using a low constant value, three others were determined by calculating averages of the degree of the nodes, and the last one was calculated using the ROC graph with the condition that a true positive rate (TPR) equals 70%. They proposed an adaptive threshold selection method to characterize the node behavior, which adapts to the dynamic node communication.

Sharma et al. [22] presented four different methods for detecting faults in sensor measurement data. For each of these methods, they gave one or more different threshold selection methods. Since these detection methods are very specific to the authors' use case, such as heuristic methods and time series analysis-based methods, these methods do not follow the approach of generating anomaly scores and selecting thresholds from them. Therefore, all presented methods for threshold selection are also very specific and cannot be used for a wide range of applications.

Rather than searching for papers that provide a structured overview of threshold selection methods, our literature search focused on identifying papers relevant to the topics of *unsupervised learning*, *anomaly detection*, and *threshold selection*. The literature search was conducted via Google Scholar with the aim of capturing the broad spectrum of threshold selection methods applicable to different anomaly detection scenarios. First, we selected papers in which threshold selection was discussed in the context of anomaly detection. To refine our selection, we selected papers presenting threshold selection methods that can be generally applied in different contexts of anomaly detection. Papers proposing methods for specific use cases or applications were excluded to ensure the general applicability of the review. From this refined selection, we identified 32 papers, of which 8 focus on supervised and 24 on unsupervised threshold selection methods. This selection ensured that the methods examined were generally applicable, allowing us to provide a comprehensive discussion in Sections III and IV.

III. SUPERVISED THRESHOLD SELECTION METHODS

Supervised threshold selection methods are based on the fact that not only can a list of anomaly scores corresponding to each data point be used to create a threshold, but also a list of labels indicating which data points are normal and which are the product of attacker actions. In this way, it is fairly easy to create one or more thresholds where the predicted labels closely match the true labels.

The main idea behind all these approaches is to maximize the desired evaluation metric. The evaluation metric to be maximized dictates the approach to be used. Since two labels must be predicted for each data point, we will refer to class one (positive) as *anomalous* and class zero (negative) as *normal (benign)* in the rest of the article.

In this section, we provide an overview of all the supervised threshold selection methods that were found while doing a review of the network anomaly detection domain. For each approach, a general motivation is given, the implementation

is discussed, and variants of the approach are presented. In addition, some general strengths and weaknesses of each approach are discussed, and how they can be applied.

A. RECEIVER OPERATING CHARACTERISTICS CURVE

One of the most popular methods for selecting the threshold in a supervised manner is the Receiver Operating Characteristics (ROC) curve. The ROC curve is a plot of true positive rate (TPR) as a function of false positive rate (FPR) [23], [24]. Each point on the plotted curve corresponds to the various thresholds that determine whether a data point is considered anomalous. The area under this curve, also known as AUC-ROC, is often used as an evaluation metric for comparing models for classification problems where different thresholds must be evaluated. The larger the AUC-ROC value, the better the model. Values of AUC-ROC range from 0, when the model always predicts incorrectly, to 1, when we have a perfect model that predicts everything correctly. If AUC-ROC has a value of 0.5, the model predicts randomly. ROC curves can be useful to compare and evaluate different unsupervised anomaly detection algorithms [25] because we do not need to define an ad hoc threshold for each algorithm. If a curve has more area under it, then it is clearly superior, regardless of the threshold chosen [26].

Finding the optimal threshold using the ROC curve can be viewed as a maximization problem, where we need to find the threshold that maximizes the difference between TPR and FPR. This is actually the same as maximizing the Youden's index [27] defined as:

$$J = \text{sensitivity} + \text{specificity} - 1 = \text{TPR} - \text{FPR}. \quad (1)$$

On this basis, the equation for determining the threshold is defined as follows:

$$T = \arg\max_{T_i}\{J(T_i)\} = \arg\max_{T_i}\{\text{TPR}(T_i) - \text{FPR}(T_i)\}, \quad \forall i \in \{0, \ldots, N_t - 1\} \quad (2)$$

where N_t is the number of thresholds considered, and $J(T_i)$, $\text{TPR}(T_i)$, and $\text{FPR}(T_i)$ are the Youden's index, true positive rate, and false positive rate, given the threshold selection T_i.

The second method of selecting the optimal threshold using the ROC curve can be performed using the Neyman-Pearson criterion [28], which works by maximizing the number of true positives while at the same time not allowing the false positives to exceed a predefined value α. To find the optimal threshold, the optimization problem must be solved to find the maximum TPR value such that the FPR value is less than or equal to α [26]. The optimal threshold value is defined by solving the equation:

$$T = \arg\max_{T_i}\{\text{TPR}(T_i)\} \mid \text{FPR}(T_i) \leq \alpha, \quad \alpha \in [0, 1], \quad \forall i \in \{0, \ldots, N_t - 1\} \quad (3)$$

where N_t is the number of thresholds considered, α is a predefined value, and $\text{TPR}(T_i)$ and $\text{FPR}(T_i)$ are true positive rate and false positive rate for the given threshold T_i. Similarly, Harshaw et al. [29] suggested that the optimal threshold should be defined as the value for which the number of known true positives (records that are truly anomalous) is maximized while the number of other records that exceed the threshold is minimized.

The third method for selecting the threshold is to minimize the FPR with the additional constraint that the TPR is 1, which is referred to in the literature as the *zero-miss false-alarm rate* [16], [30], [31]. To determine the threshold with this approach, the following optimization problem should be solved:

$$T = \arg\min_{T_i}\{\text{FPR}(T_i)\} \mid \text{TPR}(T_i) = 1, \quad \forall i \in \{0, \ldots, N_t - 1\} \quad (4)$$

where N_t is the number of thresholds considered, and $\text{TPR}(T_i)$ and $\text{FPR}(T_i)$ are true positive rate and false positive rate given the threshold T_i.

B. EQUAL ERROR RATE

Fatemifar et al. [32] have taken the approach of plotting the false acceptance rate (FAR) and the false rejection rate (FRR) for different thresholds. Here, FRR is actually defined as the same as a false negative rate (FNR) and FAR is the same as FPR. The graph obtained shows that the number of false acceptances decreases while the number of false rejections increases, and vice versa. The threshold value can be determined at the point where two lines intersect and is known as the *Equal Error Rate* (EER). At the threshold determined in this way, the percentage of false acceptances and false rejections is equal. The EER can also be determined using the ROC curve, where the threshold corresponding to ERR is a point at which FNR, which is equal to 1-TPR, and FPR are equal [16], [33].

C. PRECISION-RECALL CURVE

Similar to the ROC curve approach, there is also an approach that uses the Precision-Recall (PR) curve. The PR curve is a plot of precision as a function of recall, where recall equals TPR [24]. Here, the optimal threshold can be determined by finding a point on the PR curve for which the F-beta score is maximum [34]:

$$T = \arg\max_{T_i}\{\frac{(1+\beta^2) \times \text{precision}(T_i) \times \text{recall}(T_i)}{\beta^2 \times \text{precision}(T_i) + \text{recall}(T_i)}\}, \quad \forall i \in \{0, \ldots, N_t - 1\} \quad (5)$$

where N_t is the number of thresholds considered, β is a parameter defining the weight of precision and recall in the calculation of threshold, and $\text{precision}(T_i)$ and $\text{recall}(T_i)$ are the precision and recall values at the chosen threshold T_i.

Depending on the value of β, a different evaluation metric is maximized. For example, for $\beta = 1$, the F1 score is maximized. Although the F1 score is most commonly used [35], there are cases where recall is more important than precision. In that case, we are more likely to use the F-beta

score, where $\beta = 2$ (F2 score). In many cases, this will be the more desirable metric when detecting anomalies in network data, since it is more important that our model detects all true anomalies as anomalous, but with some degree of false positives. The degree of false positives depends on how many human resources we have to manually review all data points flagged as anomalous.

D. SUMMARY

One of the main advantages of supervised threshold selection methods is their ability to optimize the desired evaluation metric. For example, if the goal is to find the maximum value for TPR, then the ROC curve is the best approach. On the other hand, if the priority is to optimize recall or precision values, the PR curve is the best choice. To find a balance between TPR and FPR, the Equal Error Rate can be used. Although these methods can find an optimal threshold for the desired evaluation metric, this does not mean that the metric itself will always be 1 or close to 1. This is the case because the underlying model may not be able to produce anomaly scores that correctly distinguish anomalous from normal data points. In this sense, there will always be some wrong predictions, no matter how good the threshold selection method is. For this reason, we use optimization methods and decide what tradeoffs to make (and how much) to achieve the desired level of performance. The other major advantage of supervised methods is that they are very easy to implement, fast to run, and do not require any additional parameters, except when using the Neyman-Pearson criterion, which requires α as the maximum allowable FPR value.

A major disadvantage of supervised methods is, of course, their supervised nature. In many cases, our network data consists only of normal data, so there are no anomalies at the starting point. Still, we need to choose an appropriate threshold for future data that may contain anomalies. Without anomalous samples, the number of true positives and false negatives cannot be determined. Based on this, TPR and recall are undefined, FPR is 1 and precision is 0. Since we have already established that each method presented here has an evaluation metric to maximize, a valid threshold cannot be calculated. In some cases, the method used must be dynamic, that is, it must generate a different threshold for each data point or data group. This is the case when the distribution of data changes over time windows. In other cases, the threshold selection method must also be adaptive, so the threshold must be calculated for each new data point or group of data points. Due to their static nature, supervised methods cannot be applied to streaming data, which is often the case in cybersecurity [1].

When comparing approaches using the PR curve and the ROC curve, it is important to note that the ROC curve is only appropriate when data classes are balanced, while the PR curve is more appropriate for unbalanced datasets. When detecting anomalies in network data, there is much more normal data than anomalous data, so the classes are very often imbalanced. Also, the PR curve focuses more on the positive class, which is usually more important. In conclusion, methods based on the PR curve should be preferred over methods based on the ROC curve when selecting thresholds in a supervised manner.

To summarize this subsection, the Table 1 lists all supervised approaches and methods found to determine the threshold. It also indicates, for each method, whether parameters (and which ones) need to be specified before using the method to determine the threshold.

IV. UNSUPERVISED THRESHOLD SELECTION METHODS

Unlike supervised threshold selection methods, unsupervised methods do not use true labels to select thresholds. Instead, in most cases, they must use a statistical model to find the threshold that they believe is the best discriminator between two different classes in the given anomaly scores. Therefore, to achieve good threshold selection in an unsupervised manner, our anomaly scores must be divided into two groups with some statistical differences. Unlike supervised methods, some methods here are designed to work only with anomaly scores obtained from normal data. Therefore, they try to set the threshold somewhere near the limit of normal data scores. In this section, we have listed all the unsupervised threshold selection methods that we found while reviewing the literature. As in the previous section, a general motivation is given for each approach, the implementation is discussed, and variations of the approach are presented. These methods are then summarized, categorized, and grouped in a table. Here we have not included some specific heuristic approaches that can be used for a particular situation, i.e., a particular type of data or a particular anomaly detection model.

A. MAXIMUM METHOD

One of the simplest ways to calculate the threshold in an unsupervised way is to use the maximum value of the given anomaly scores. We can simply use this value and set the threshold based on this value. The final threshold value can be calculated using the following equation:

$$T = \max(X) \tag{6}$$

where X is a set of anomaly scores given by the model. Besides its simplicity, it is obvious that this method works well only when we generate anomaly scores from data that do not contain anomalies [36]. This is because a potentially high anomaly score coming from an anomalous record would generate a very high threshold, leading to many false negatives in the future.

B. PERCENTILE METHOD

The idea behind the next unsupervised method is to set the threshold at the k-th percentile of the anomaly scores. The k-th percentile is defined as a value below which a certain k percent of the data falls. The final threshold value can be

TABLE 1. List of all supervised approaches found for selecting threshold and the methods based on them that are used to determine threshold.

Approach	Method	Parameters	Reference
Receiver Operating Characteristics Curve	Maximize the difference between TPR and FPR (Youden's index).	/	[27]
	Maximize TPR while keeping FPR under predefined value alpha.	alpha = allowed FPR	[26]
	Minimize FPR while keeping TPR at the value of 1.	/	[16], [30], [31]
	Point at which FPR and FNR are equal.	/	[16], [33]
FAR and FRR Plot	Point at which FAR and FRR are equal.	/	[32]
Precision-Recall Curve	Maximize F-beta metric.	beta = defines F-beta metric	[34], [35]

calculated using the following equation:

$$T = \text{perc}(k, X) \tag{7}$$

where perc is a function to calculate the k-th percentile, where k is a positive integer, and X is the set of anomaly scores.

Unlike the previous method, this method is parametric, meaning that it requires a set of parameters to work. In this case, the only parameter is k, which determines which percentile is used. This method is subject to the same limitations as the previous one, as it is sensitive to anomalous records that produce high anomaly scores.

Some authors such as Zong et al. [37], Zenati et al. [38], Bergman and Hoshen [39], and Alvarez et al. [40] suggest that the threshold should be set based on the ratio of anomalous data in the test set α. Once α is determined, the threshold can be set as the $(1-\alpha)^{th}$ percentile. While this is a good idea, it no longer makes the method unsupervised, because while we do not need to know which records are anomalous, we at least need to know their ratio, which is sometimes hard to know or even assume.

C. INTERQUARTILE RANGE METHOD

One of the most popular methods of defining outliers, often used in making boxplots, was proposed by Tukey et al. [41]. Tukey defined an outlier as the point greater than the sum of the third quartile and 1.5 times the interquartile range (IQR) of the data, where IQR is defined as the difference between the third and first quartiles. The threshold is defined as follows:

$$T = Q_3(X) + 1.5 \times \text{IQR}, \quad \text{IQR} = Q_3(X) - Q_1(X) \tag{8}$$

where Q_1 and Q_3 are the first and third empirical quartiles of the anomaly scores X and IQR is the interquartile range. The factor of 1.5 was chosen by Tukey so that the probability of a Gaussian random variable being classified as an outlier is about 0.7% [42].

Because this method calculates the threshold using quartiles, which are essentially the same as percentiles, it is sensitive to anomalous records that produce high anomaly scores, just like the percentile method described earlier.

D. K-SIGMA METHOD

This method is the one that is easy to implement and also very popular when it comes to threshold selection [43]. Here the final threshold value is calculated as follows:

$$T = \mu + k \times \sigma \tag{9}$$

where μ is the mean and σ is the standard deviation of the anomaly scores and k is the parameter which is a positive integer. This method is often used under the assumption that the data are normally distributed. With this in mind, the parameter k is usually set to three, since 99.73% of the data is then below the calculated threshold [44]. This rule is also generalized by Chebyshev's inequality [45], which states that regardless of the probability distribution, 88.89% of the values fall within three standard deviations [46].

This method usually works better for data containing only normal records. Since anomalous data records tend to have much larger anomaly scores when combined with the normal ones, this causes the mean and standard deviation of the dataset to increase, making the final threshold value higher than it should be.

The extension of this approach can be seen in the use of windowing, i.e. selecting the subsets of a larger dataset. With windowing, we now obtain not just a threshold, but the number of thresholds corresponding to the number of windows into which we divide the anomaly scores. The prediction is now made by comparing the anomaly scores in each window with the threshold generated for that window. In the literature, we find two common types of windowing techniques. The first is classical windowing, where we divide the dataset of anomaly scores sequentially into subsets of equal size (the last one may be smaller). An example is the work of Dhiman et al. [47], in which they dealt with automated fault detection systems and used static windowing in combination with the k-sigma method for threshold selection.

The second windowing technique is the sliding window method. Here we used sliding windowing for threshold selection as suggested by Qin et al. [48]. They proposed that the threshold of the current data point is calculated using the k-sigma method based on the network traffic of the last s seconds. For us, this means that for each point we use a sliding window of size w consisting of past w values starting with one point before the current point. Based on this window, we calculate the k-sigma threshold for this point. In the end, we have thresholds whose number is equal to the number of anomaly scores, with the first threshold set to an infinite

value, so that the first data point is always normal since we do not have enough data to calculate this window. Both of the aforementioned windowing methods require an additional parameter besides the parameter k, namely the size of the window w. The complete algorithm of the described method is listed as Algorithm 1.

Similar to using windows to select thresholds, different moving averages can be used. The moving average is a calculation to create a series of averages of different subsets of the full dataset. Hamamoto et al. [49] used fuzzy logic to detect whether an anomaly exists in a time interval. They performed the threshold calculation using an exponentially weighted moving average (EWMA), a type of moving average that assigns higher weight to more recent observations. The equation to calculate the weighted average of the previous data points from the data point x_i is as follows:

$$\mu_i = \lambda \times x_i + (1 - \lambda) \times \mu_{i-1}, \quad \lambda \in (0, 1] \tag{10}$$

where λ is a so-called forgetting factor that determines how much weight we give to previous data points [50], [51]. On this basis, the variance for each data point σ_i^2 can then be calculated as follows:

$$\sigma_i^2 = \sigma^2 \times (\frac{\lambda}{2 - \lambda}) \times (1 - (1 - \lambda)^{2i}), \quad \lambda \in (0, 1] \tag{11}$$

where σ^2 is the variance of all data points and λ is the forgetting factor [50], [51]. The threshold selection method based on the equations is implemented simply by calculating the k-sigma threshold for each data point using the mean and standard deviation calculated with (10) and (11). This method has two parameters, one is the k parameter for the k-sigma rule and the other is the forgetting factor λ. The algorithm of this method is given as Algorithm 2.

E. DISTRIBUTION ADJUSTMENT METHOD

In analyzing the k-sigma method, we found that it worked well especially for normally distributed data, and we wanted to generalize this to any distribution. Our idea was to fit the anomaly score values to each possible distribution to find out which distribution our anomaly scores were most likely to come from. Once we find the probability distribution with its parameters, we can calculate the mean and standard deviation of this distribution. In this way, the mean and standard deviation are calculated from the closest probability distribution rather than directly from the anomaly scores (observations). With the mean and standard deviation calculated, we can use (9) and calculate the final threshold.

To implement the proposed method, we first need to find out which probability distribution our anomaly scores would most likely come from. Here we can use the goodness of fit test. A goodness of fit test evaluates how well a set of observed data matches a particular theoretical distribution. It is commonly used by statisticians to assess the validity of assumptions about the underlying distribution of a population based on sample data [52]. In this statistical test, the null hypothesis is that the sample was drawn from a population that follows a particular distribution and the alternative is that this is not the case.

Out of the many different goodness of fit tests [53], we opted to use three of them: Kolmogorov-Smirnov test [52], [54], [55], Anderson-Darling test [55], [56], [57], and Cramér-von Mises test [54], [55]. There is also the popular Shapiro-Wilk test [55], but it can only be used to check whether the data come from the normal distribution, which was not sufficient for our case.

For the distributions we tested, we used all available continuous distributions under the *SciPy* module [58]. Then we fitted each of these continuous distributions to the given anomaly scores. After fitting the distributions, we obtained maximum likelihood estimates of the shape parameters, location, and (if applicable) scale of the distribution. We then created a cumulative distribution function (CDF) based on the selected continuous distribution and the parameters obtained in the previous step, where the CDF of a real-valued random variable X is the probability that X takes a value less than or equal to x [59].

We use the CDF generated in this way and the given anomaly scores for one of the selected goodness of fit tests mentioned above. This test provides us with the test statistic and the p-value for rejecting or not rejecting the null hypothesis. Based on the p-values obtained, we can select the lowest p-value as an indicator of the distribution that our anomaly scores most closely follow. Once the closest distribution with the parameters is determined, we can easily obtain mean and variance values that we can use to calculate the threshold based on the k-sigma method. The pseudocode of this approach is listed as Algorithm 3.

The advantage of this approach is that we do not have to select in advance the distribution from which our anomaly scores come. Since different datasets and different unsupervised models can lead to anomaly scores from different distributions, this approach solves this problem. On the other hand, the big problem is the complexity of this approach, since there is a huge list of implemented continuous distributions in the *SciPy* package, so it is not possible to test each one. As an alternative, we created a list of all available distributions and randomly selected one from this list in each iteration to test the hypothesis. Similar to the standard k-sigma method, this method works better on datasets containing only normal data, since the anomaly scores obtained from mixed records often result in a bimodal distribution that is difficult to fit into any of the known continuous distributions.

F. PEAKS-OVER-THRESHOLD METHOD

Extreme Value Theory (EVT) [60] is a statistical theory that aims to find the law of extreme values. These extreme values are shown to follow some kind of distribution, regardless of the original distribution of the data, which means that there are no assumptions about the original data distribution, which is a major advantage of this approach. All extreme

values of common standard distributions follow Extreme Value Distributions (EVD) [61]. It is possible to evaluate the probability of potential extreme values by fitting an EVD to the tail of the input distribution. One of the most popular methods for fitting the tail of the distribution is the Peaks-Over-Threshold (POT) approach, which is based on the Pickands-Balkema-de Haan theorem [62], [63]. According to this theorem, the excess over a threshold is likely to follow a Generalized Pareto Distribution (GPD) with parameters γ and σ. With the estimated parameters $\tilde{\gamma}$ and $\tilde{\sigma}$, we can calculate the threshold T as follows:

$$T = t + \frac{\tilde{\sigma}}{\tilde{\gamma}}((\frac{q \times |X|}{|Y|})^{-\tilde{\gamma}} - 1) \tag{12}$$

where t is a high initial threshold, q is the desired probability, $|X|$ is the number of anomaly scores, and $|Y|$ is the number of peak values, i.e., the number of anomaly scores greater than t. Usually, the Maximum Likelihood Estimation (MLE) is used to estimate the parameters $\tilde{\gamma}$ and $\tilde{\sigma}$ [61].

Many papers addressing the anomaly detection problem use the POT approach for automatic threshold selection. Yang et al. [15], Su et al. [64], and Siffer et al. [61] implemented a static threshold selection method based on (12). In addition to the static threshold selection method, Siffer et al. [61] also proposed two streaming algorithms that update the threshold with incoming data. One is the streaming POT method, which works well if the distribution of incoming data does not change drastically over time, while the other is the streaming POT with drift method, which takes into account a drift component of the incoming data. Similarly, Liu et al. [14] implemented a streaming algorithm based on POT for automatic threshold selection that updates the threshold over time based on the comparison of each new data point with the current and the initial threshold.

We implemented two methods based on the POT approach. First, we implemented the static threshold selection method based on (12). For the initial threshold t, we used the p-th percentile. To estimate the parameters $\tilde{\gamma}$ and $\tilde{\sigma}$, we used the already implemented scipy.stats.rv_continuous.fit method, which by default uses MLE to estimate the shape ($\tilde{\gamma}$), location ($\tilde{\mu}$), and scale ($\tilde{\sigma}$) parameters. The algorithm of this approach is listed as Algorithm 4.

In addition to this method, we have also implemented a streaming threshold selection method based on the POT approach. This method is based on the streaming methods of Liu et al. [14] and Siffer et al. [61]. Here, the idea is to update the threshold as new data arrives. First, we use a certain percentage m of the anomaly scores and calculate the threshold for this set based on the Algorithm 4. For each new anomaly score, we consider three possible options. If the anomaly score is larger than the current threshold T, this data point is marked as an anomaly. If it is larger than the initial threshold t, we calculate the excess of this score over the t, re-estimate the parameters of GPD, and update the threshold according to (12). In the remaining cases, we do nothing because the data point with this anomaly score is considered normal. Since this is a streaming method, no static threshold is generated, but the labels are predicted based on the anomalies detected during the process. The algorithm of this method is listed as Algorithm 5.

Both of the methods described are parametric. The static method depends on the parameters p and q. In addition to these two parameters, in the streaming method, there is also the parameter m, which specifies the percentage of anomaly scores used to set the initial threshold. Since the methods based on the POT approach take into account the excess over the threshold, they should work better with data containing anomalous records in combination with normal records.

G. KERNEL DENSITY ESTIMATION METHOD

Kernel density estimation (KDE) is a method for nonparametric estimation of a probability density function (PDF) that produces a smooth empirical PDF based on the individual locations of all sample data [65]. Traditionally, histograms have been used to represent the empirical PDF. However, since these have a very subjective structure, as their shape depends on the subjective choice of the number of bins into which the range of a sample is divided, the KDE technique is more commonly used to analyze the probability distribution [66].

Wang et al. [67] proposed an approach for unsupervised anomaly detection using a self-adversarial variational autoencoder with a Gaussian prior assumption for anomalies that trains a Gaussian transformation network to synthesize anomalous but near-normal latent variables. In their work, they used the KDE technique to automatically learn the decision thresholds from normal data only. They first used KDE to determine the PDF of the training dataset consisting of only normal data points and then calculated the cumulative distribution function (CDF) $F(s)$ from the obtained PDF. Using the significance level α and the obtained CDF, they defined a final decision threshold T as a point satisfying $F(T) = 1 - \alpha$. This means that there is at least a $(1 - \alpha) \times 100\%$ probability that a sample with an anomaly score greater than or equal to the decision threshold is an outlier. The significance level α can be used to manipulate the FPR level such that a lower α results in a lower FPR.

Based on the approach described by Wang et al. [67] we implemented the KDE method to select the threshold in an unsupervised manner. Before using the KDE function, we need to specify two parameters: the kernel function to use and the bandwidth of the kernel. For the choice of the bandwidth parameter we used the criterion proposed by Silvermann [65]:

$$h = (\frac{n(d+2)}{4})^{-\frac{1}{d+4}} \tag{13}$$

where n is the number and d is the dimension of the data points. In our case, n is the number of anomaly scores, and d is equal to 1.

With the given kernel and bandwidth parameters, KDE can be initialized and fitted to the given anomaly scores. We then

created N_e uniformly distributed evaluation points in the interval between the minimum and maximum values of the given anomaly scores. The value of N_e was set to 1000, as it provides sufficient precision in terms of the number of points evaluated without affecting performance too much. We used the fitted model to calculate the probability density values for the selected evaluation points under this model. From these probability density values, we calculate the values of CDF, by calculating the cumulative sum of the elements along a given axis and dividing by the last cumulative sum element to obtain values between 0 and 1. In this way, we obtain CDF, which gives us the probability that the random variable X takes a value smaller than the value of a particular evaluation point. We defined the threshold as the evaluation point where the value of CDF is closest to the value of $(1-\alpha)$. The pseudocode of the described method for threshold selection using KDE is listed as Algorithm 6.

H. K-MEANS METHOD

Wang et al. [68] proposed an improved long short-term memory-based time-series anomaly detection method for use in the rail transportation environment. To account for the distributions of various device data, which often violate the Gaussian assumption, and to consider a very low rate of anomalies, they used a dynamic threshold selection method based on prediction errors to detect anomalies. The proposed method relies on the k-means clustering [69] for the initial threshold calculation. The selection of the final threshold is based on the idea of balancing between the number of anomalies detected and the difference between the variances of all anomaly scores and the anomaly scores that are smaller or equal to the selected threshold.

The implementation of this approach is as follows. The initial threshold is calculated using the k-means clustering, where the number of clusters is set to 3, representing the normal, transitional, and anomaly states, and the centers of these clusters are set to 0, the average value of the anomaly scores, and the maximum value of the anomaly scores, respectively. The minimum value of the third cluster was used as the initial threshold. Then, threshold candidates are generated from the initial threshold value to the maximum value of anomaly scores with a step of d. The value of d controls the granularity, i.e., how large the step we want to take when generating threshold candidates. Empirically, this value is set to 0.01 because it provides enough threshold candidates while not being too slow to execute. For each generated threshold candidate, we divide the anomaly scores into two groups: one with values less than or equal to the threshold candidate (X_l) and the other with values greater than the threshold candidate (X_h). Then we calculate the measure of goodness S of the respective threshold candidate as follows:

$$S = \frac{\sigma(X)}{\sigma(X_l) \times |X_h|} \tag{14}$$

where $\sigma(X)$ is the standard deviation of all anomaly scores, $\sigma(X_l)$ is the standard deviation of anomaly scores less than or equal to the candidate threshold, and $|X_h|$ is the number of anomaly scores greater than the candidate threshold, with the smallest value set to 5. The final threshold is determined by finding the candidate threshold with the largest measure of goodness S, which is calculated by (14). The algorithm of this approach is listed as Algorithm 7.

From the implementation of this method, we can see that it relies on data containing both normal and anomalous records, as it divides the input data into normal, transitional, and anomaly clusters. Since the method does not require any additional value other than the anomaly scores, it is non-parametric.

I. DBSCAN METHOD

We can view the anomaly scores we generated as the simplified problem of finding two classes in the data that have only one attribute which is a score (magnitude of anomaly). To find different classes in an unsupervised way, clustering methods are often used. One of the best methods that is commonly used for clustering is Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [70]. In density-based clustering, clusters are detected as regions in which the objects of the regions are dense. The clusters are separated from each other by low-density regions [71]. DBSCAN is particularly useful here because we do not have to specify the number of classes in the data in advance, but this information is provided to us by the algorithm.

We used the DBSCAN algorithm to predict labels for our anomaly scores. Since DBSCAN does not require us to specify in advance how many classes are in the data, we do not know how many different labels DBSCAN will output. Based on the number of predicted classes, we distinguish three possible cases: whether one label is found, whether two labels are found, or whether three or more different labels are found. If one label is found, it means that DBSCAN cannot distinguish between our normal and anomalous scores. In this case, we set the threshold to the sum of the mean and maximum values of the anomalous scores. If two different labels are identified, it means that DBSCAN has found two different groups, one from normal records and the other from anomalous records, so we need to set a threshold that is somewhere between the two identified groups. For this purpose, we first determine the boundaries (minimum and maximum) of the two groups. The final threshold is determined as the value equal to half the distance between the boundaries of the two groups. If DBSCAN finds more than two different labels (groups), the threshold is undefined because we cannot calculate it. Therefore, we set it to the value 0. The algorithm of this approach is listed as Algorithm 8.

The method proposed here was developed for the mixed data structure, i.e., those containing both normal and anomalous records. Otherwise, DBSCAN would likely recognize

only one class, resulting in the threshold being set to the point one standard deviation away from the maximum. Thus, it is assumed that the two classes are fairly well separated from each other and that each of them collects around their respective mean. The problem occurs when there are several small groups in one of the two classes. DBSCAN then recognizes them as separate classes and we cannot choose a suitable threshold with the proposed method.

With the described approach, not only the DBSCAN algorithm can be used, but also any other popular clustering algorithm such as k-means, agglomerative hierarchical clustering, or k-medoids clustering [72]. The main advantage of DBSCAN over these popular clustering algorithms is that the number of clusters does not need to be specified before using the algorithm. Therefore, it was chosen for the implementation of the described approach. Although the DBSCAN algorithm does not require the number of classes in the data to be specified, there are two important parameters to consider: the radius of the cluster (eps) and the minimum required points within the cluster (minPts) [71].

J. LOCAL OUTLIER FACTOR METHOD

The Local Outlier Factor (LOF) is an unsupervised anomaly detection method that calculates the local density deviation of a given data point concerning its neighbors [73]. It is local in the sense that the anomaly score depends on how isolated the object is from the surrounding neighborhood. More specifically, locality is determined by the k-nearest neighbors whose distance is used to estimate the local density. The regions of similar density are identified by comparing the local density of a data point with the local densities of its neighbors [44]. The data points that have a significantly lower density than their neighbors are considered outliers.

Laptev et al. [44] presented a generic and scalable framework for automatic anomaly detection in large time series data. In their system, an alert is generated when the prediction error is outside a specified threshold. To solve the problem of non-normally distributed data and the use of parametric methods, the method LOF was used to determine the threshold.

To implement the unsupervised threshold selection approach using LOF, we simply fitted our anomaly scores as one-dimensional data to the LOF algorithm, which is then used to predict labels indicating whether each data point is anomalous or not. Here we do not generate a threshold but make predictions directly, which can then be evaluated. The algorithm of this approach is listed as Algorithm 9.

Although Laptev et al. stated that this approach is not parametric, there is still an important parameter that must be considered when using the LOF algorithm, namely the number of neighbors m, which has a large impact on the final prediction. Since anomaly detection in this approach is based on different local densities, it can only work well if the input data contains anomalies. Therefore, it cannot make a good threshold selection when only normal data is available.

K. DISTANCE BETWEEN THE SETS METHOD

He et al. [13] proposed a topology-aware anomaly detector for multivariate time series. To achieve unsupervised anomaly detection in a cloud system, they combined graph neural networks, long short-term memory, and a variational autoencoder. They performed unsupervised threshold selection after training the model based on the assumption that there is a high-density region corresponding to normal data and a low-density region corresponding to anomalous data. On this basis, the threshold can be defined as a point that maximizes the distance between two sets of anomaly scores. The distance between the two anomaly score sets separated by a threshold is defined as:

$$d(X_l, X_h) = \frac{\min(X_h) - \max(X_l)}{\min(X_h) + \max(X_l) - 2 \times \min(X_l)} \tag{15}$$

where X_l is a set of anomaly scores with smaller values than the chosen threshold and X_h is a set of anomaly scores with larger values than the chosen threshold.

The implementation of this approach is as follows. First, we set the average of the given anomaly values as the initial threshold and the maximum value of the anomaly scores as the maximum threshold. Then, for each threshold between the initial and maximum values with a step of 0.01, we create two groups of anomaly scores separated by that threshold. Between these two groups, we calculate the distance using (15), and the final threshold is chosen as the one at which the distance between the sets is the largest. The algorithm of this approach is listed as Algorithm 10. This approach is based on the assumption that there are anomalous data points in the data and therefore does not apply to the situation where there is only normal data.

L. INFLECTION POINT METHOD

Since the given anomaly scores from which we have to select a threshold are one-dimensional values, they can be easily plotted in two dimensions. Based on the good graphical representation, we can sometimes see a good threshold value with the naked eye. A simple but good graphical representation is to sort the scores from smallest to largest and then plot them as a line graph. The points plotted in this way can be used to identify the area or region of the graph where the anomaly values begin to increase rapidly, i.e., the inflection point [74]. Ideally, the idea is to choose as a threshold the value that precedes the rapid increase of the anomaly scores.

Casas et al. [75] presented an unsupervised network anomaly detection algorithm that detects anomalous traffic in a knowledge-independent manner. They used a novel clustering technique based on sub-space density clustering to identify clusters and outliers. They ranked traffic flows by dissimilarity obtained using a correlation distance-based approach and chose a threshold by determining the value at which the slope of the sorted dissimilarity values showed a major change.

Zhong et al. [76] developed a novel gas turbine anomaly detection model that uses a weight-agnostic neural network search to find minimal and effective neural network architectures. To calculate the threshold, they first ordered the anomaly scores from smaller to larger values. Then, the sliding window was moved from left to right until the rapidly changing region was found. Within the rapidly changing region, the difference of all points in that region is calculated. The point with the largest difference value is considered as the inflection point and the anomaly value of this point is set as the threshold value.

Based on these two examples of using the inflection point for threshold selection, we have implemented three different variants of this approach. In all variants, the anomaly scores are sorted from the lowest to the highest value. The first variant uses the package *KneeLocator* [77], which provides an algorithm for detecting the knee/elbow point in a curve, i.e., finding the point of maximum curvature. The implementation can be easily invoked to provide the point of maximum curvature, from which the threshold is defined as the anomaly score for the detected knee. Here, the only parameter is the one in the *KneeLocator* function, the sensitivity parameter s, which allows you to set how aggressive the function should be in detecting knees. Smaller values for s detect knees faster, while larger values allow more flat points in the curve before a knee is detected.

In the second variant, we use a sliding window of size d and calculate for each window the difference between the first element in the window and the last. Each calculated difference is associated with the last anomaly score of the corresponding window, where the first d anomaly scores are associated with the NaN value. From the differences, we then find the peaks using the function scipy.signal.find_peaks, which finds all local maxima by simply comparing neighboring values. Of all the peaks found in this way, we select the one that has the largest value. Then we choose a list of d sequential differences from the d number of points before the largest peak position to the largest peak position. From these differences, we obtain anomaly scores and consider them as threshold candidates. Finally, we choose the k-th percentile value of these anomaly scores as the final threshold.

The last variant of this approach differs from the second only in the step of selecting the final threshold from the candidate thresholds. Instead of using the k-th percentile, we again calculate the difference between each anomaly score in the candidate thresholds as $X_{i+1} - X_i$. From the obtained differences, we select the largest one and choose the final threshold as the smaller anomaly score that generated this difference. Since the first approach is very simple, only the pseudocode for the last two variants using the inflection point approach is listed as Algorithm 11.

Since this approach relies heavily on the fact that there is a point at which sorted anomaly scores increase rapidly, it is only applicable to datasets that contain both normal and anomalous traffic. For the method to work well, it is also important that there is a significant difference between the values of the anomaly scores at some point.

M. STANDARD DEVIATIONS INFLECTION POINT METHOD

When considering different subsets of given anomaly scores, the standard deviation of each subset can be useful in determining how much a particular point in a given dataset deviates from the other points in that dataset. As you remove the largest values from the set of anomaly scores, the standard deviation of the remaining values should decrease. Starting with the rapid decrease, the decrease eventually becomes constant as we remove all the anomaly scores that have the largest impact on the standard deviation of the entire dataset, i.e., the anomaly scores generated by anomalous data points. The point at which the standard deviation stops decreasing rapidly could be used as a threshold. Based on this, we implemented an approach that finds an inflection point in the standard deviation series.

The idea behind this method is that we first sort the anomaly scores from largest to smallest, then remove the largest score one by one and calculate the standard deviation of the remaining scores. Since it is very time-consuming to remove one score at a time and recalculate the standard deviation, we use only a certain percentage p of the largest anomaly scores, since we can assume that anomalous data points produce only the largest anomaly scores. From the calculated set of standard deviations, we want to find the inflection point of the standard deviation plot. For this, we can use the same *KneeLocator* function as before. From the knee position k found in this way, we set the final threshold to the average of the anomaly scores at position k and $k + 1$. The algorithm of this approach is listed as Algorithm 12.

Similar to the inflection point approach, this method only works for datasets that contain both normal and anomalous traffic because it is based on the difference in anomaly scores between the two classes.

N. EMPIRICAL CUMULATIVE DISTRIBUTION FUNCTION METHOD

The empirical cumulative distribution function (ECDF) is an estimate of the CDF, which generated the points in the sample. The ECDF can be calculated by ordering all unique values in the sample and calculating the cumulative probability for each value as the number of values less than or equal to a given value divided by the size of the sample [78]. This yields the step function that jumps up $1/n$ for each of the data points, where n is the number of data points in the given sample. Hoang and Nguyen [79] investigated the use of PCA techniques for anomaly detection in the Internet of Things. To determine the threshold using the ECDF, they set the threshold to a value equal to the $1 - \alpha$ value of the ECDF, where α is the rate of false estimates.

In our implementation, we used the same threshold selection method as Hoang and Nguyen [79]. First, we sorted all anomaly scores in ascending order and then calculated the ECDF by simply adding $1/|X|$ for each anomaly score, where

$|X|$ is the number of the anomaly scores. After calculating the ECDF, we determined the anomaly score from the sorted set whose ECDF value was closest to $1 - \alpha$ and chose that value as the threshold. The algorithm of this approach is listed as Algorithm 13.

From the implementation, we can see that this method assumes that the data passed to it has anomaly scores that come from anomalous records. Therefore, this method depends on the rate of false estimates α, which makes it parametric. If we had only normal records, we could set α to a value of 0, but then we would simply choose the largest anomaly score as the threshold. So this method is not applicable if we have as input for threshold selection only datasets that contain only normal records.

O. SYMBOLIC AGGREGATE APPROXIMATION METHOD

Symbolic Aggregate approXimation (SAX) is a method for the symbolic representation of time series. It transforms a time series of arbitrary length n into a string of arbitrary length w, where $w < n$ typically using an alphabet A of size $a > 2$. The SAX discretization procedure uses an intermediate representation between the raw time series and the symbolic strings. It first transforms the time series data into the Piecewise Aggregate Approximation (PAA) representation and then symbolizes the PAA representation into a discrete string [80].

Yue et al. [81] focused on developing anomaly detection methods based on descriptive analytics to protect the load forecasting process from cyberattacks. They proposed a solution to detect and mitigate anomalies in long sequences caused by cyberattacks. For this purpose, they used the method SAX, to find the threshold for classifying a subsequence as an anomaly. To do this, they ran the SAX through the intact historical data to find the largest distance between different subsequences.

Based on the work of Yue et al. [81], we implemented a simple unsupervised threshold selection method using SAX. Our method uses SAX, where the number of bins generated is two, so we can obtain only two different symbols as the output of the SAX algorithm. With the symbols "a" and "b" obtained, we can easily assign them to labels 0 and 1. Since SAX assigns the first letters to lower values and the last to higher values, the symbol "a" is assigned to label 0 and the symbol "b" to label 1. Here we have skipped the part of creating the final threshold and generated predicted labels immediately. The final threshold can be obtained somewhere between the maximum value of the points with the symbol "a" and the minimum value of the points with the symbol "b". The pseudocode for predicting labels with SAX is listed as Algorithm 14.

Since this method does not explicitly specify the final threshold, but focuses only on predicting labels, it can only work well for datasets that contain both normal and abnormal data. If there is only normal data, this method will forcibly distinguish between two classes where there is only one and is therefore not useful. Since this method is based on trends, it is best suited for data that is ordered as a time series, such as events with timestamps.

P. OTSU'S METHOD

In the context of image segmentation, Otsu [82] has presented a nonparametric and unsupervised method for automatic threshold selection. Otsu's method for determining an optimal threshold for this purpose was to minimize intra-class intensity variance or maximize the inter-class variance. The intra-class variance σ_ω^2 is defined as a weighted sum of the variances of the two classes:

$$\sigma_\omega^2(T) = \omega_0(T) \times \sigma_0^2(T) + \omega_1(T) \times \sigma_1^2(T) \qquad (16)$$

where ω_0 and ω_1 are the probabilities of the two classes separated by the threshold T, and σ_0^2 and σ_1^2 are the variances of these two classes. The algorithm based on Otsu's method searches for the threshold that minimizes σ_ω^2, defined by (16).

We regard features coming from the spliced area as anomalies and iterate autoencoder-based modeling and discriminative labeling to tell them apart.

Cozzolino and Verdoliva [83] proposed a method for no-reference localization of image splices using autoencoder-based modeling and discriminative labeling to enable the detection of features that originate from the spliced area and are considered anomalies. In their work, the features are labeled by comparing the associated reconstruction error with a certain threshold value chosen using Otsu's algorithm. In the case of anomaly detection in the network data, the same algorithm can be used since we have two classes, one normal and one anomalous.

We implemented Otsu's algorithm by choosing the mean value of the anomaly scores as the initial threshold t and testing all possible thresholds with the step d until it reaches the maximum value of the anomaly scores. The value of d was empirically set to 0.01 because it provides a large number of candidate thresholds while not being too slow in execution. For each threshold candidate, we predict the classes that would be created by the choice of that threshold. For the anomaly scores belonging to each of the classes, we calculate the variances σ_0^2 and σ_1^2 and the probabilities ω_0 and ω_1, where ω_0 and ω_1 are calculated as the ratio of the number of scores in one class compared to the number of all anomaly scores. With these parameters, we calculate the intra-class variance σ_ω^2 using (16). The final threshold is defined as the one for which the calculated σ_ω^2 is smallest. The algorithm of this approach is listed as Algorithm 15.

Q. OTHER METHODS

In addition to all of the unsupervised threshold selection methods described, we also used in our tests some of the methods already implemented in the *PyThresh* toolkit [11]. *PyThresh* is a comprehensive and scalable Python thresholding toolkit for outlier detection. *PyThresh* contains more than 30 thresholding algorithms. All of these algorithms are nonparametric, so there is no need to specify a contamination level or have the user estimate the number of outliers that

might be present in the dataset. These algorithms range from simple statistical analyses such as the Z-score to more complex mathematical methods involving graph theory and topology. To use these methods correctly, the anomaly scores given must follow the rule that the higher the score, the higher the probability that this point in the dataset is anomalous. Of these methods, we used only those that gave us results for our test data in a reasonable amount of time. The methods used are enumerated in the Table 2. The first and second columns contain the name of the method and its abbreviation used in presenting the test results, and the third column contains the reference used to demonstrate the validity of the threshold methods.

TABLE 2. List of PyThresh methods used.

Method name	Abbreviation	Reference
Area Under Curve Percentage	AUCP	[84]
Chauvenet's Criterions	CHAU	[85]
Distance Shift from Normal	DSN	[86]
Elliptical Boundary	EB	[87]
Fixed Gradient Descent	FGD	[88]
Filtering Based	FILTER	[89]
Full Width at Full Minimum	FWFM	[90]
Generalized Extreme Studentized Deviate	GESD	[91]
Histogram Based	HIST	[92]
Median Absolute Deviation	MAD	[93]
Friedrichs' Mollifier	MOLL	[94], [95]
Modified Thompson Tau Test	MTT	[96]
Regression Based	REGR	[97]
Variational Autoencoder	VAE	[98]
Topological Winding Number	WIND	[99]
Yeo-Johnson Transformation	YJ	[100]

Alongside *PyThresh* methods, some methods rely on available human resources to determine the threshold for anomaly scores. Le and Zincir-Heywood [101] suggest that the selection of the threshold should be based on what they call the investigation budget. The investigation budget is defined as the available budget for investigating anomalous flags and is defined as the percentage of data with the highest anomaly scores that can be investigated by the analyst to confirm malicious behavior. When testing threshold selection methods, we did not consider these methods because they depend on available human resources, which in turn depend on the real situation we face.

R. SUMMARY

Now that we have described all the unsupervised threshold selection methods found in the literature, we present a categorization based on the few attributes that are important to us in choosing the right threshold selection method. The first of this attribute is the type of the input data that each method expects. Depending on the nature of the anomaly detection problem, we may only have normal data available and need to set the threshold to be able to detect anomalous data that may or may not occur in the future. On the other hand, we may already have a data sample that contains both normal and anomalous records, and based on the difference between them we can choose the threshold.

The second important attribute is how each method performs thresholding in the context of the method's output. Most methods perform thresholding in such a way that they provide a static threshold value at the end of their work. This value is then compared to each anomaly score to determine which data point is normal and which is anomalous. Some methods may also generate m static thresholds where m is less than the number of anomaly scores, which is usually the case when using windowing approaches. Similarly, some methods generate different thresholds for each streaming anomaly score because they recalculate the threshold for each new data point. On the other hand, some methods directly generate labels and in this way predict which data point is normal and which is anomalous. In these methods, the threshold is implicit and can only be determined by analyzing the generated labels.

Another important feature of the threshold selection method is whether it is parametric or non-parametric. Unlike non-parametric methods, parametric methods require certain parameters to be specified for the thresholding method to work well with the given dataset. These parameters are often determined empirically for each dataset or group of datasets and are rarely universal for all anomaly detection situations. The number of these parameters is also very important, as it is much easier to optimize one parameter than several at a time.

Finally, we decided to divide all unsupervised threshold selection methods into a few broad groups based on the idea and manner of thresholding. The first group called *Statistics-based* methods, includes the methods that use only simple statistics functions such as mean, standard deviation, and percentiles to determine the threshold. Second, there are *Distribution-based* methods, where the initial data is used as a sample to which a particular distribution is fitted, and the parameters of this fitted distribution are used to calculate the threshold. *Clustering-based* methods use clustering algorithms to first divide the data into classes, which are then used to determine the final threshold. *Density-based* methods assume that higher density regions correspond to normal data and lower density regions correspond to abnormal data, with density calculated based on the distance between data points. Ultimately, *Graphical-based* methods are based on observing sorted values and the point at which these values begin to deviate significantly is considered the threshold. All other methods are grouped in the *Other* category.

Based on the analyzed unsupervised threshold selection methods, we created a table containing all the methods described in this section, given as Table 3. The first and second columns of this table contain the name of the unsupervised threshold selection method and its abbreviation,

TABLE 3. List of all unsupervised methods for selecting threshold.

Method name	Abbreviation	Approach Group	Input Data Type	Output Data Type	Non-parametric	Parameters	Reference
Maximum method	MAX	Statistics-based	Normal	Static threshold	✓		[36]
Percentile method	PERC	Statistics-based	Normal	Static threshold	✗	k = percentile number	[37]–[40]
Interquartile Range method	IQR	Statistics-based	Normal	Static threshold	✓		[42]
K-sigma method	KSigma	Statistics-based	Normal	Static threshold	✗	k = multiplication factor	[43], [44]
K-sigma method with static windowing	KSigma_Window	Statistics-based	Normal	Multiple static thresholds	✗	k = multiplication factor w = window size	[47]
K-sigma method with sliding windowing	KSigma_Sliding	Statistics-based	Normal	Streaming threshold	✗	k = multiplication factor w = window size	[48]
K-sigma method with EWMA	KSigma_EWMA	Statistics-based	Normal	Streaming threshold	✗	k = multiplication factor λ = forgetting factor	[49]
Distribution adjustment method	Distribution	Distribution-based	Normal	Static threshold	✗	k = multiplication factor goodness of fit test type	-
Peaks-Over-Threshold method	POT	Distribution-based	Mixed	Static threshold	✗	p = percentile number q = desired probability	[15], [61], [64]
Streaming Peaks-Over-Threshold method	SPOT	Distribution-based	Mixed	Streaming threshold	✗	p = percentile number q = desired probability m = initial data percentage	[14], [61]
KDE method	KDE	Distribution-based	Mixed	Static threshold	✗	kernel = kernel type α = rate of false estimates	[67]
K-means method	KMeans	Clustering-based	Mixed	Static threshold	✓		[68]
DBSCAN method	DBSCAN	Clustering-based	Mixed	Static threshold	✗	eps = radius of cluster minPts = minimum points within cluster	-
LOF method	LOF	Density-based	Mixed	Predicted labels	✗	m = number of neighbors	[44]
Distance between the sets method	Dist_Sets	Density-based	Mixed	Static threshold	✓		[13]
Inflection point method	Inflection	Graphical-based	Mixed	Static threshold	✗	d = step size k = percentile number s = KneeLocator sensitivity	[75], [76]
Standard deviation inflection point method	Inflection_STD	Graphical-based	Mixed	Static threshold	✗	p = percentage of the largest scores	-
ECDF method	ECDF	Graphical-based	Mixed	Static threshold	✗	α = rate of false estimates	[79]
SAX method	SAX	Other	Mixed	Predicted labels	✓		[81]
OTSU method	OTSU	Other	Mixed	Static threshold	✓		[83]

which is used when displaying the test results. The third column shows the group to which each method belongs, based on the idea and the way it performs thresholding. The fourth column specifies the type of input data the method should work with. The fifth column shows the type of output the method produces. The sixth column indicates whether the method is non-parametric or not, and the seventh column contains the required parameters, if any. The last column contains references to papers where this method has been used or defined to select threshold.

V. METHODOLOGY

The methodology section of this study is divided into several important subsections to comprehensively explain the research approach. First, the original firewall log data is described, including the source of the data, the preprocessing steps performed, and the injection of anomalies. The methodology for testing the threshold selection methods is then presented, including a detailed explanation of the two test scenarios created for the experiments: one based on normal (benign) data and the other based on a mixture of normal and anomalous data. The anomaly scores obtained with the presented test scenarios are presented in the form of descriptive statistics and data distributions for each dataset. Finally, the evaluation metrics commonly used in this area are discussed.

A. FIREWALL LOGS AND ANOMALY GENERATION

Rather than using artificially generated datasets for anomaly and outlier detection that exist online, we used anonymized real firewall logs in combination with injected anomalies to test selected threshold selection methods. The original firewall logs are from a few Check Point firewalls deployed in the industrial control network of an electricity transmission system operator. The collected logs are organized by month and each contains about twelve million records. Each log file contains records of communication between two IP addresses, and each record contains only information about SYN segments for communication using the TCP protocol. In addition to the TCP protocol, the UDP and ICMP protocols are also included in these logs. Originally, each record contained a large number of fields, but the ones we used for anomaly detection were: connection timestamp, source and destination IP address, source destination port, protocol type, and firewall action attribute.

The transformation of the original firewall logs involves several important steps to enrich and normalize the data. A circular timestamp label is added based on a sinusoidal representation of the hour of the day. The data is then aggregated hourly and grouped by source and destination IP pairs. For each source IP within the time unit, basic characteristics are added, such as the total number of hits and destination IPs contacted. In addition, the ratio of hits per source-destination pair to the number of unique destination ports used is calculated. Ports 0-1023 are one-hot encoded, while the subsequent ranges of 1000 ports are grouped and recorded by usage. The protocol type field is also one-hot encoded. This transformation, which emerged from our earlier research, ensures that the logs are prepared for unsupervised anomaly detection algorithms.

Since the firewall logs only record connections that do not violate configured policies, we extended the original logs to include custom-generated anomalies. We generated anomalies as multiple network scans originating from machines in internal subnets and configured to target machines in subnets to which the originating machine was not directly connected. We used the tool *Nmap* [102] to scan both individual targets and the entire subnet. The resulting network traffic was recorded using *Wireshark*. We used the SYN, TCP connect, and UDP scan Nmap scanning types and scanned several combinations of ports commonly found in the target network.

The recorded network scans are then used to manually construct logs that the original firewall would have generated in a similar situation. IP addresses from the lab recordings were replaced with IP addresses of prominent targets within the original network, and timestamps were shifted to match the target logs. Since our firewall logs contain much less detailed data than PCAPS, we manually checked all properties in the generated anomalous logs to ensure that they were identical to those that would be expected in the case of real attacks in the target network. The details of the original firewall logs used, the anomalies generated, and the method used to generate and integrate anomalies into the logs can be found as part of our previous work in [103] and [104].

For this study, we used three different months of original firewall logs in combination with three types of anomalous datasets constructed. Two anomalous datasets were created using the *Nmap* scans described above, and one anomalous dataset was obtained through penetration testing on the target network. In total, we used nine different datasets here to test the threshold selection methods, with each dataset corresponding to one month of the original logs combined with one type of anomalous dataset. These datasets are publicly available in a GitHub repository [105].

B. TESTING SCENARIOS

To test all the threshold selection methods and their variants described in Sections III and IV, it is not sufficient to use only one test method, because some methods do not perform equally well in different test scenarios. Based on the assumption that some methods work better with data containing only normal traffic and others work better with mixed traffic, i.e., data containing anomalies, we created two test scenarios. The first test scenario is based on mixed traffic, and the second is based on normal traffic only. We have assumed that the methods that expect anomalies in the data perform better in the first test scenario, while the methods that do not exploit this assumption perform better in the second scenario. The implementation of each test scenario is divided

into two phases: generating anomaly scores and performing the test of threshold selection methods.

To generate anomaly scores for the first test scenario, we used firewall logs into which anomalies were inserted so that these logs presented the network in a state of potential attack. We then initialized an unsupervised anomaly detection model and fitted it to our data without labels. In all scenarios, we used the Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions (ECOD) [106], as it has been shown to be deterministic, stable and very fast in previous studies with our data [104]. The fitted model now provides us with an anomaly score for each data entry. Each of these anomaly scores is given as a float and indicates how certain the model is that this entry is anomalous. It is important to note that all generated anomaly scores are normalized before being used in the testing procedure, because the implementation of methods from the *PyThresh* toolkit expect anomaly scores in the range of 0 to 1.

In the second test scenario, we assume that the data from which we want to generate anomaly scores does not contain anomalies. In this case, we split the initial data into a training and a test dataset in a 70:30 ratio and then removed all anomalous records from the training dataset and labels. Again, we initialize the model ECOD and fit it to the modified training dataset. In the end, we generate anomaly scores for both the training dataset and the test dataset that contains anomalies, normalize them, and store them for the test procedure.

The procedure for testing threshold selection methods in the first test scenario is as follows. First, the anomaly scores generated in the previous step are loaded. Then, a threshold is generated based on these scores and the selected threshold selection method. If the method does not generate labels directly, the labels are predicted to be 0 or 1 based on the generated threshold, with the rule that label 1 is predicted if the anomaly score is strictly greater than the given threshold, otherwise label 0 is predicted. Finally, the evaluation metrics are calculated based on the predicted labels and the true labels. For each dataset and threshold selection method, the resulting evaluation metric values and execution time are stored as the result of the test procedure.

For the second test scenario, the procedure is slightly different because we have training and test datasets. Here, we only use the training dataset to select the threshold with a specific threshold selection method, and then we use the generated threshold to predict the labels in the test dataset. If the observed threshold selection method directly predicts labels or is a streaming-based method, then we use both datasets together as input. Finally, we calculate the evaluation metrics, but this time only for the test dataset and save the results as before. Comparing these two created test scenarios, the second one is a bit more realistic for real-world use. This is because many networks initially have no anomalies at all, or very few anomalies that are not related to any type of attack, so the threshold is defined based on the history of normal data.

To get a better idea of the input data that the threshold selection methods work with, we have calculated the basic descriptive statistics for each dataset used. Table 4 gives an overview of the basic descriptive statistics for nine datasets of anomaly scores obtained by combining firewall logs from three months: October, November, and December with three types of injected anomalies: two types of Nmap and pentest anomalies. For each anomaly score dataset, the following statistics are presented: the number of anomaly scores, the mean of the anomaly scores, the standard deviation of the anomaly scores, the minimum value of the anomaly scores, the 25th percentile of the anomaly scores, the 50th percentile of the anomaly scores, the 75th percentile of the anomaly scores, and the maximum value of the anomaly scores. The statistics shown are only those calculated with the datasets generated in the first test scenario, as the second test scenario involves splitting the data into a training set and a test set. In addition to the basic statistics, Fig. 1 shows the histograms of the anomaly scores datasets used. The figure shows the different distributions of the individual datasets, which play a decisive role in the selection of the threshold values. When creating the histograms, the number of bins was set to 2000.

From the descriptive statistics presented, we can see that each dataset contains about half a million anomaly scores. The mean of these scores varies depending on the type of anomaly injected, ranging from 0.05 to 0.1 for both types of Nmap and about 0.0025 for pentest anomalies. The standard deviations also have similar values to the means. All of these datasets contain a very small amount of anomalous records. On average, about 99.95% of the scores come from normal data and 0.05% of the scores come from anomalous records, i.e. on average about 300 scores have to be detected as anomalous.

The repository with the Python scripts used to generate test data based on the two scenarios described, the implementation of the threshold selection methods described, and the scripts used to test these methods are publicly available in a GitHub repository [105].

C. EVALUATION METRICS

Another important aspect of the testing procedure is the use of an evaluation metric appropriate for our case. Here, we considered three common evaluation metrics for presenting the classification results. When comparing classification results, the F1 score is usually used. This metric weights precision and recall equally and is calculated using the following equation:

$$\text{F1 score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{17}$$

From the same family of F-beta metrics comes the F2 score metric, where recall is weighted more heavily than precision. This is particularly useful because when detecting anomalies in network data, it is most important to detect all anomalous events and then reduce the number of false positives as much as possible. The equation for calculating the F2 score is as

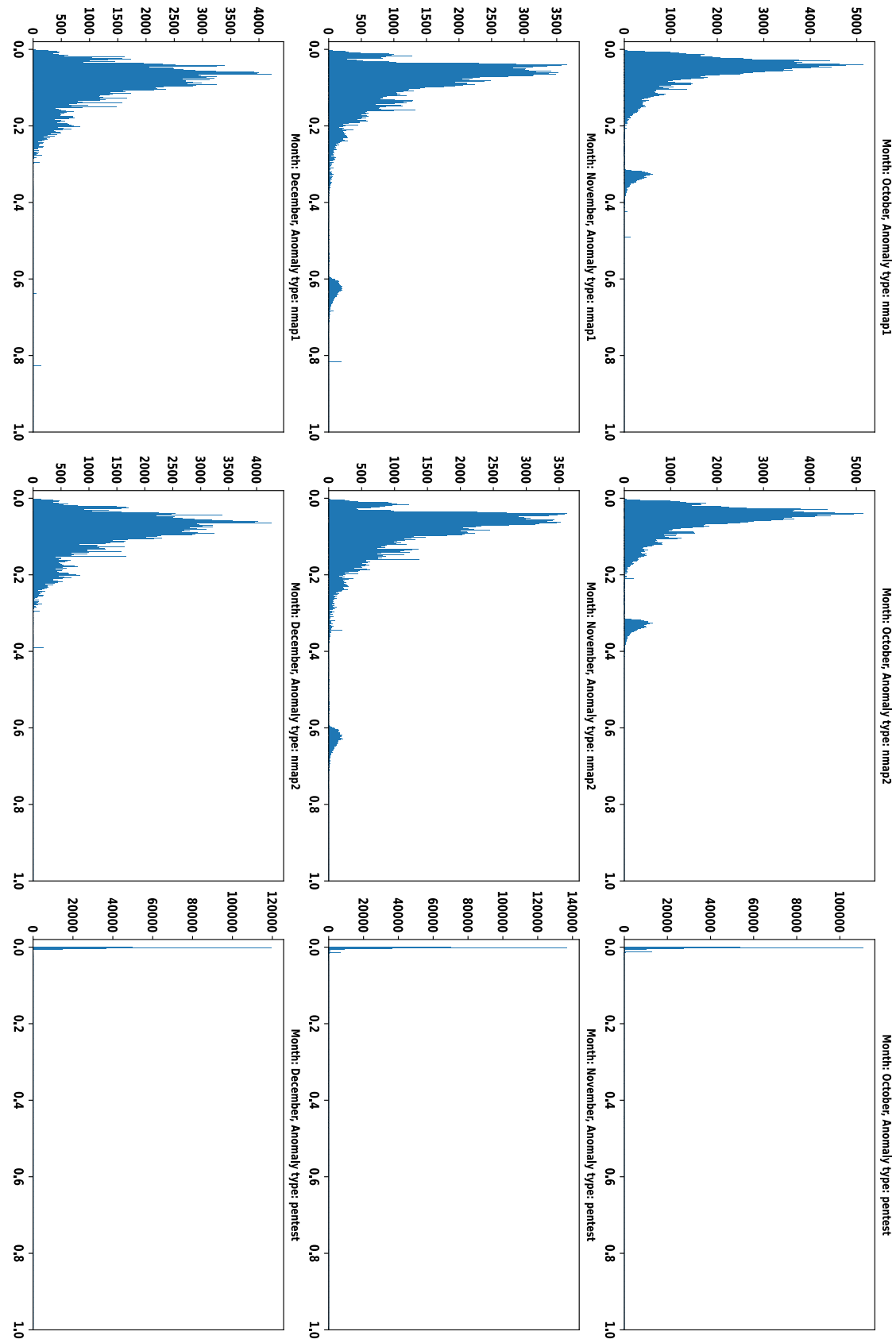

FIGURE 1. Histograms of nine anomaly scores datasets used for the testing procedure.

TABLE 4. Descriptive statistics of anomaly scores that summarize the central tendency, dispersion, and shape of a dataset.

Month	October			November			December		
Anomaly Type	Nmap1	Nmap2	Pentest	Nmap1	Nmap2	Pentest	Nmap1	Nmap2	Pentest
Count	524,104	524,117	524,229	553,854	553,867	553,979	572,505	572,518	572,630
Mean	0.0742	0.0740	0.0027	0.1076	0.1075	0.0024	0.0917	0.0916	0.0023
Standard Deviation	0.0728	0.0723	0.0056	0.1099	0.1090	0.0054	0.0529	0.0511	0.0049
Minimum	0	0	0	0	0	0	0	0	0
25th percentile	0.0355	0.0355	0.0013	0.0525	0.0526	0.0012	0.0570	0.0570	0.0014
50th percentile	0.0511	0.0511	0.0019	0.0766	0.0767	0.0017	0.0808	0.0808	0.0020
75th percentile	0.0817	0.0816	0.0030	0.1230	0.1230	0.0028	0.1130	0.1129	0.0028
Minimum	1	1	1	1	1	1	1	1	1

follows:

$$\text{F2 score} = \frac{5 \times \text{precision} \times \text{recall}}{4 \times \text{precision} + \text{recall}} \tag{18}$$

The Matthews correlation coefficient (MCC) was first introduced by Matthews [107] and is used as a measure of the quality of binary classifications. The MCC value can be calculated using the following equation:

$$\text{MCC} = \frac{(\text{TP} \times \text{TN}) - (\text{FP} \times \text{FN})}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{19}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives. The MCC can take values from -1 to 1. A value of 1 represents a perfect prediction, 0 is no better than a random prediction, and -1 represents a complete disagreement between prediction and observation [108]. For the presentation of our results, we chose to use the MCC metric because it is a more reliable statistical value and gives a high score only if the prediction performs well in all four categories of the confusion matrix: TP, FP, TN, and FN [109], [110].

In addition to the choice of MCC metric to represent the classification results, another important aspect is the execution time of each tested method. We calculated the execution time as the time difference between the time when the threshold selection method receives the anomaly scores and the time when it generates the threshold. When presenting the execution time values, we only use the values obtained with the first test scenario, as these times do not change significantly between the different scenarios.

VI. RESULTS

In this section, we present the results obtained with the two testing scenarios described in Section V. The results are divided into several parts depending on the group of threshold selection methods since the number of tested methods is so large that the histogram of all results would not be readable. First, the results of the supervised threshold selection methods are discussed. Then, the results of the Statistics-based, Distribution-based, Clustering-based, Density-based, Graphical-based, and other unsupervised threshold selection methods are presented.

A. SUPERVISED METHODS

We tested five different supervised threshold selection methods described in Section III. Four methods were tested using the ROC curve approach: the method that maximizes Youden's index, the method that maximizes the TPR while keeping the FPR below α (Neyman-Pearson Criterion), the method that minimizes the FPR while keeping the TPR at the value of 1 (Zero-Miss False-Alarm Rate), and the method that selects the threshold for which the FNR and FPR are equal (Equal Error Rate). For the value of α, we used the value of 0.018. The chosen value is based on the average percentage of false anomalies in our datasets and the criterion presented by Axelsson [9] which states that one false alarm in 100,000 events is the minimum requirement for an effective intrusion detection system.

Apart from the ROC curve, we tested the method that uses the PR curve to maximize the F1 score. The method that uses the FAR and FRR plot is the same as the method that selects the threshold for which FNR and FPR are equal and thus is not considered here.

Fig. 2 shows boxplots of the MCC metric for two test scenarios for the nine different datasets with each of the five supervised threshold selection methods selected. From the boxplots, it can be seen that the method using the PR Curve provided the best average MCC value for the first test scenario, while the Youden index-based method provided the best average MCC value for the second test scenario. Both methods yielded an average MCC value of about 0.6. On the other hand, the Zero-Miss False-Alarm Rate yielded the worst average MCC value for both test scenarios.

One of the main advantages of these methods is the fast time they take from obtaining the anomaly scores to generating the threshold. All the methods presented here generate a threshold in less than 0.5 seconds, with the EER method proving to be the fastest with an average execution

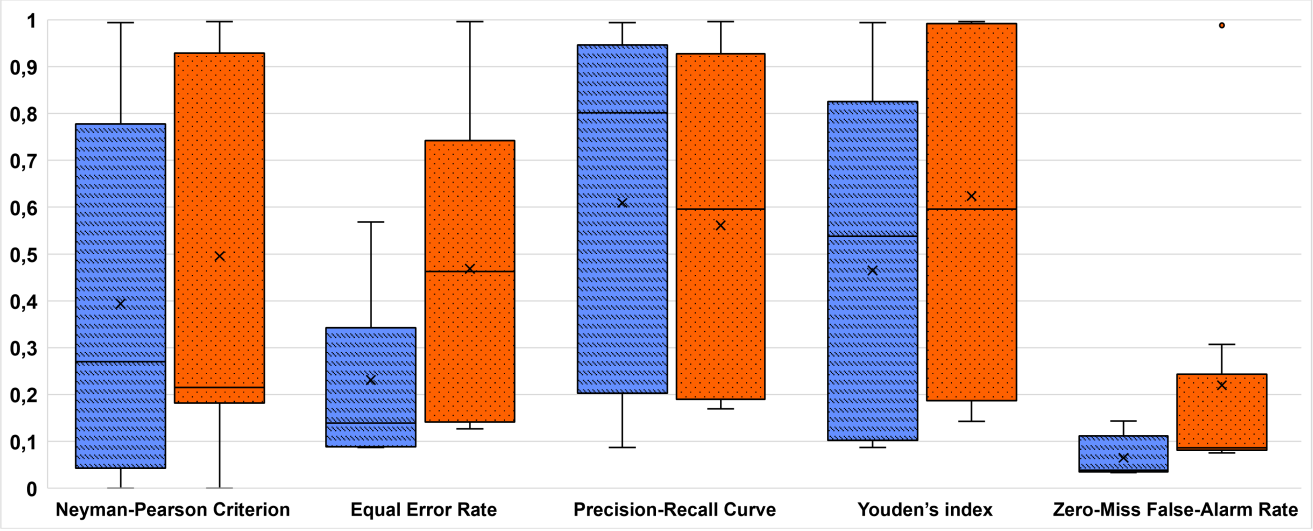

FIGURE 2. Boxplots of the MCC metric using five supervised threshold selection methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

time of 0.28 seconds. The PR curve method, on the other hand, is the slowest with an average execution time of 0.41 seconds. The other three supervised methods require between 0.3 and 0.4 seconds to generate the threshold value.

As for choosing one of the presented supervised methods, we cannot say that one of them is the best, but it depends on each application. Therefore, we must choose a method that gives the most importance to the evaluation metric that is most important to us. For example, if precision and recall are equally important to us, we should maximize the F1 score using the PR curve. On the other hand, if we want the probability of marking an actual anomaly as such to be high, we should keep the TPR at a value of 1 and minimize the FPR using the ROC curve.

These threshold selection methods can be used not only for threshold selection but also for benchmarking unsupervised threshold selection methods. To do this, you must first select the appropriate supervised method, which in our case serves as the ideal threshold method. Then, by calculating the difference between the threshold determined by each unsupervised method and the threshold determined by the supervised method, you can select an unsupervised threshold selection method that is closest to the best threshold selection method for our use case. Similar benchmarking can also be performed by calculating and averaging the differences between the MCC values obtained with the optimal supervised method and the MCC values obtained with each unsupervised method.

B. STATISTICS-BASED METHODS

Here we present the results obtained with the two test scenarios for each of the statistics-based unsupervised threshold selection methods. Since some of these methods have one or more parameters that need to be adjusted for the method to produce the best results, we tested a few different parameter combinations for each of the methods used. For the percentile method, we tested three different values for the k parameter: 97, 98, and 99. For the k-sigma method, the k-sigma method with static windowing, the k-sigma method with sliding windowing, and the k-sigma method with EWMA, we tested values for the k parameter of 3 and 5. To determine an appropriate value for the window size w for the k-sigma method with static and sliding windowing, we treated anomaly scores as signal data with the assumption that it is periodic. To determine the magnitude of the period of this signal, we used the autocorrelation function [111] and then multiplied it by an empirically determined multiplication factor, which was 20 for the k-sigma method with static windowing and 10 for the k-sigma method with sliding windowing. The forgetting factor λ for the k-sigma method with EWMA was also determined empirically, as a value of 0.2 and 0.25, depending on the parameter k.

Boxplots of the MCC metric using the nine different datasets for two test scenarios can be found in Fig. 3. Seven methods are presented: maximum method, percentile method, Interquartile Range method, k-sigma method, k-sigma method with static windowing, k-sigma method with sliding windowing, and k-sigma method with EWMA. Only the results where the best combination of parameters was used are shown, namely: the value 5 for the parameter k for the k-sigma methods and 99 for the percentile method, the value 0.25 for the forgetting factor λ and the value 20 for the window size multiplication factor for the k-sigma method with static windowing and 10 for the k-sigma method with sliding windowing.

From the boxplots in Fig. 3, we can see that the maximum method performed worst, as expected because only the

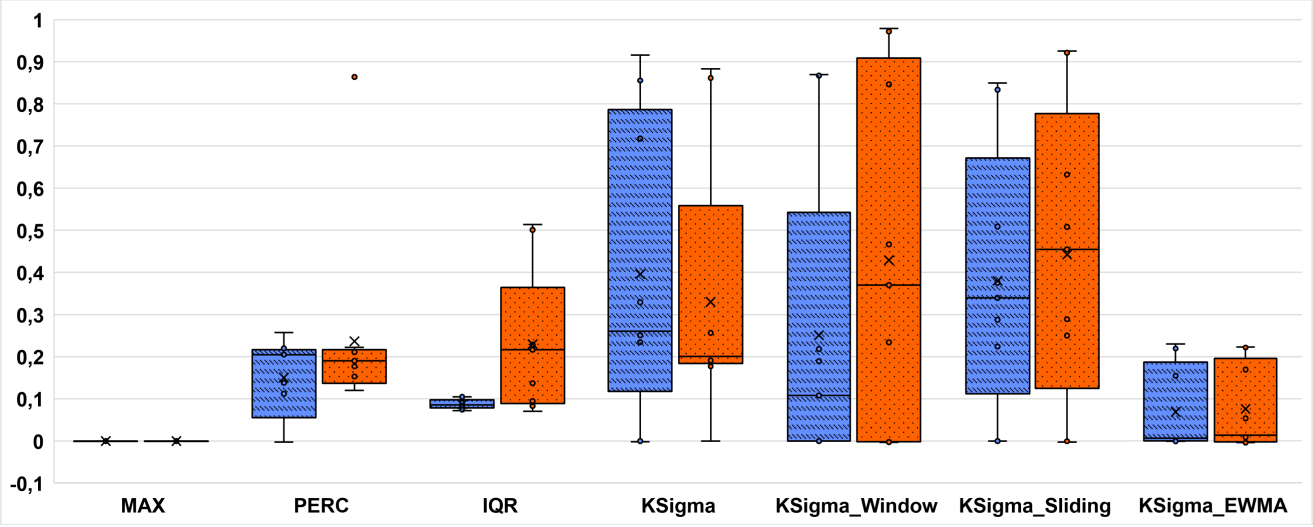

FIGURE 3. Boxplots of the MCC metric using seven statistics-based unsupervised threshold selection methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

maximum value of the anomaly scores was used as the final threshold. The percentile method showed a better average result for the second test scenario with an average MCC score of 0.24 compared to the first test scenario but with a larger standard deviation. Similarly, the IQR method yielded a better average MCC score with a value of 0.23 for the second test scenario, while the average MCC score for the first test scenario yielded a value of 0.08. When comparing the different k-sigma methods, the k-sigma method with EWMA showed the worst average MCC value with less than 0.1 for both test scenarios. For the other k-sigma methods, the simple k-sigma method gives the best result for the first test scenario with an average MCC value of almost 0.4. For the second test scenario, the k-sigma method with sliding windowing gives the best average MCC value of 0.44. From this, we can see that adding windowing to the k-sigma method generally improves the results of the second test scenario and worsens the results of the first test scenario.

Due to their low complexity, the methods presented here are among the fastest that were considered in this study. This applies in particular to the maximum, percentile, and IQR methods, all of which have an execution time of less than 0.01 seconds. The k-sigma method is also very fast, with an execution time of around 0.85 seconds. The k-sigma method with EWMA and the k-sigma method with static windowing are similar in terms of execution time, averaging between 2 and 3 seconds. In contrast, the execution of the k-sigma method with sliding windowing takes slightly longer, on average around 10 to 11 seconds.

C. DISTRIBUTION-BASED METHODS

In the group of distribution-based methods, we tested the distribution adjustment method, the Peaks-Over-Threshold method, the streaming Peaks-Over-Threshold method, and the KDE method. Again, we needed to adjust some parameters so that each of these methods could achieve the best results in our tests. For the distribution adjustment method, we again have the parameter k as in the k-sigma method and the type of goodness of fit test. Here we used the value 5 as the best for the parameter k and three different goodness of fit tests: Kolmogorov-Smirnov (KS), Anderson-Darling (AD), and Cramér-von Mises (CVM) test. The results for all these tests are presented separately. Due to the nature of this method, we could not run it completely, as it selects distributions randomly from a large number of distributions. Instead, we ran it for 30 minutes for each dataset and used the distribution with the highest p-value found during that time.

For the POT method, we have two parameters: percentile number p and desired probability q. Both are empirically set to values 98 and 7×10^{-4}, respectively. In the SPOT method, we again have the parameters p and q, in addition to the parameter for the percentage of the initial data m. The parameter p is again the same as in POT, but the value of q is set here to 5×10^{-5}. The value of parameter m depends on the test scenario. If the first test scenario is used, then m is set to 20%, but if the second test scenario is chosen, then it is set to include all anomaly scores from the training set. For the KDE method, we used six different kernels: cosine, epanechnikov, exponential, gaussian, linear, and tophat kernels. The results obtained with each of these kernels are presented separately. Apart from the choice of kernel, the value of the α parameter is determined empirically for each different kernel used. The value of α is set to 0.001 for the cosine and linear kernels, 0.002 for the epanechnikov and tophat kernels, 0.01 for the gaussian kernel, and 0.02 for the exponential kernel.

Results in the form of boxplots of the MCC metric are shown in Fig. 4 for the two test scenarios and nine different datasets. The distribution adjustment method using

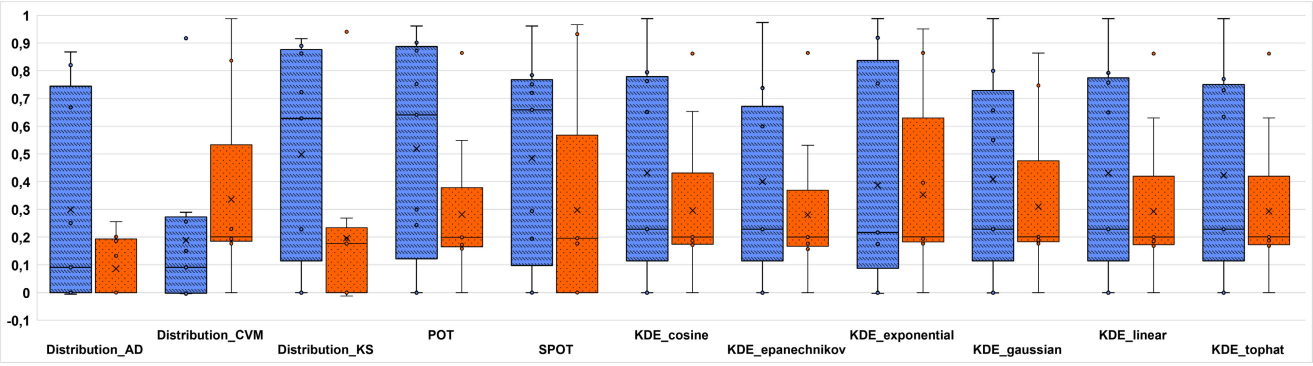

FIGURE 4. Boxplots of the MCC metric using distribution-based unsupervised threshold selection methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

the Kolmogorov-Smirnov test gives the best results for the first test scenario with an average MCC score of almost 0.5, while for the second test scenario, the Cramér-von Mises test gives the best result with an average MCC score of 0.34. Both the POT and SPOT methods provided very good results for the first test scenario with an average MCC score of about 0.5. These two methods also provided similar results for the second test scenario with an average MCC score of about 0.29, although SPOT has a slightly higher variance here. Of the different kernels used for the KDE method, the cosine and linear kernels performed best for the first test scenario with an average MCC value of 0.43, while the exponential kernel performed best for the second test scenario with an average MCC value of 0.33. The difference between the average MCC value obtained with these kernels is not drastic and is at most 0.05.

In terms of execution times, all distribution adjustment methods reach the 1800-second limit we set. Of the other distribution-based methods, the fastest is the POT method, which takes only 0.1 seconds to execute, while the slowest is the SPOT method, which takes about 520 seconds to execute on average. Of the KDE method variants, the variants with cosine, epanechnikov, linear, and tophat kernels are the fastest with an average execution time of about 3 seconds. The variants with gaussian and exponential kernels are somewhat slower and require an average of 17 and 20 seconds respectively for execution.

D. CLUSTERING-BASED METHODS

For clustering-based methods, only two different methods were tested: the k-means method and the DBSCAN method. When searching for the optimal parameters for the DBSCAN method, we find the tendency that a higher eps value leads to a higher threshold value, and minPts hardly influences the threshold value. We found the optimal values for the parameters eps and minPts to be 0.05 and 15 respectively.

In Fig. 5, the MCC values obtained with two different test scenarios and the two clustering-based methods mentioned above for nine datasets are presented in the form of boxplots.

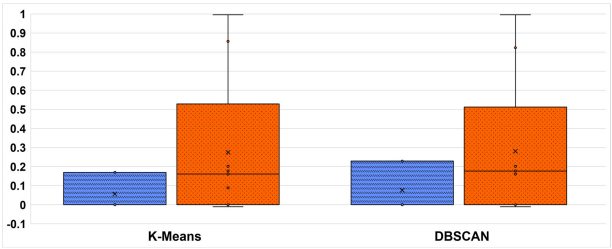

FIGURE 5. Boxplots of the MCC metric using two clustering-based unsupervised threshold selection methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

Based on these results, we can see that both methods provide almost identical results for both test scenarios, with an average MCC score of about 0.06 for the first test scenario and 0.27 for the second test scenario. Although the MCC score results are almost identical, the execution times are quite different. The k-means method takes about 25 seconds to execute, while the DBSCAN method takes only 0.7 seconds on average.

E. DENSITY-BASED METHODS

Of the methods based on the high and low-density regions, we tested two different methods: the LOF method and the method based on the distance between sets. Here, only the LOF method is parametric, where the parameter m, which represents the number of neighbors, must be optimized. The value of m was determined empirically and set to the value of 1 to obtain the best performance.

The results in the form of boxplots for these two methods are shown in Fig. 6 for each of the two test scenarios for nine datasets. As can be seen from these boxplots, the LOF method performed significantly better for the first test scenario with an average MCC value of about 0.35, while for the second test scenario, both approaches yielded a similar average MCC value of about 0.28. Looking at the execution times, the LOF method is significantly slower with an average execution

time of around 19 seconds, while the method based on the distance between sets only requires 4 seconds for execution on average.

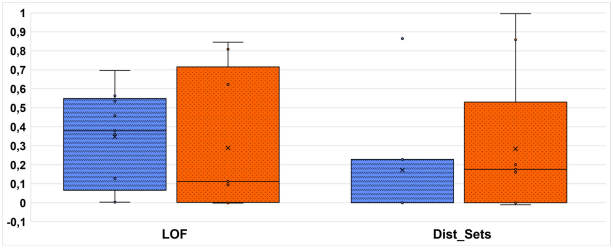

FIGURE 6. Boxplots of the MCC metric using two density-based unsupervised threshold selection methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

F. GRAPHICAL-BASED METHODS

Of the methods based on graphical representation, we tested three different methods: inflection point method, standard deviation inflection point method, and ECDF method. For the inflection point method, we tested all three implemented variants: using *KneeLocator* (KNEE), using a sliding window with differences (DIFF), and using a sliding window with differences in combination with the k-th percentile (PERC). For the KNEE variant of the inflection point method, there is only one parameter s that defines the sensitivity of *KneeLocator*, and this was empirically set to the value 20. For the DIFF variant, there is also only one parameter, namely the step size d, which was empirically set to the value 20. The PERC variant has two parameters, again the step size d and the percentile number k. For best performance, we choose 99.5 as the value of k and 10% of the anomaly scores size for the value of d. For the standard deviation inflection point method, the percentage of the largest anomaly values p must be determined. Since the percentage of true anomalous records in our datasets is very small, we can safely use 1% as the value for p. Finally, the ECDF method depends on the parameter α, which represents the rate of false estimates. Empirically, the value of α was set to 0.0005.

Fig. 7 shows the boxplots obtained for the two test scenarios using all the described graphical-based methods for nine different datasets. If we consider only the inflection point variants, we can see from the boxplots that for the first test scenario, the PERC variant performed best with an average MCC of almost 0.5, while the KNEE variant performed worst. In contrast, for the second test scenario, the DIFF variant performed the best with an average MCC of 0.37, while the PERC variant performed the worst this time. The standard deviation inflection point method performed similarly to the DIFF variant in the first test scenario, but with a lower variance, while in the second test scenario, it performed worse than all three inflection point variants. The ECDF method performed best of all the graphical-based methods in the first test scenario with an average MCC of almost 0.52, while in the second test scenario, it performed similarly to the PERC variant of the inflection point method.

When comparing the average execution time of these methods, all variants of the inflection point method and the ECDF method performed very quickly, executing in just 0.1 to 0.2 seconds. The standard deviation inflection point method, on the other hand, took an average of almost 40 seconds to execute.

G. OTHER METHODS

In the category of other methods, we have the SAX and the OTSU methods. Both are non-parametric, so this time no parameters need to be optimized. The boxplots showing the MCC metric obtained with these methods for the nine datasets are shown in Fig. 8 for both test scenarios. From these boxplots, it is easy to see that both methods perform very poorly for the first test scenario and are not suitable for this task. For the second test scenario, the SAX method also performs very poorly, while the OTSU method delivers an average MCC value of 0.21. While the MCC values provided are not satisfactory, the execution times of both methods are very good. The SAX method takes on average about 0.7 seconds to execute, while the OTSU method takes about 2 seconds.

In addition to these two implemented methods, we also tested 16 unsupervised threshold selection methods from the *PyThresh* toolkit. Fig. 9 shows the boxplots of the MCC metrics obtained with these methods using the two test scenarios for nine different datasets. Of the 16 methods tested using the first test scenario, the GESD method was the best with an average MCC metric of 0.33. Of the remaining methods, only EB, MOLL VAE, and YJ achieved an average MCC value greater than 0.25 in the first test scenario. Among the results obtained using the second test scenario, MAD performed best with an average MCC value of 0.41. Close to this result are also the methods MOLL, VAE, and YJ with an average MCC value around 0.4.

The average execution time of the tested PyThresh methods varies greatly from method to method. The slowest was the DSN method with an average execution time of over 9000 seconds. The AUCP, FGD, FWFM, VAE, WIND, and YJ methods are also very slow with an average execution time of between 4300 and 5800 seconds. The GESD method is also quite slow with an average execution time of 275 seconds, while the MTT method is slightly faster with an average execution time of 65 seconds. The fastest methods are CHAU, HIST, MAD, and REGR with an execution time of only about 0.01 seconds. The MOLL, FILTER, and EB methods are slightly slower, but also fast, with an average execution time of 1, 2, and 4 seconds respectively.

VII. DISCUSSION

Now that we have presented the results in the form of boxplots of the MCC values obtained for each of the two test scenarios using all the threshold selection methods

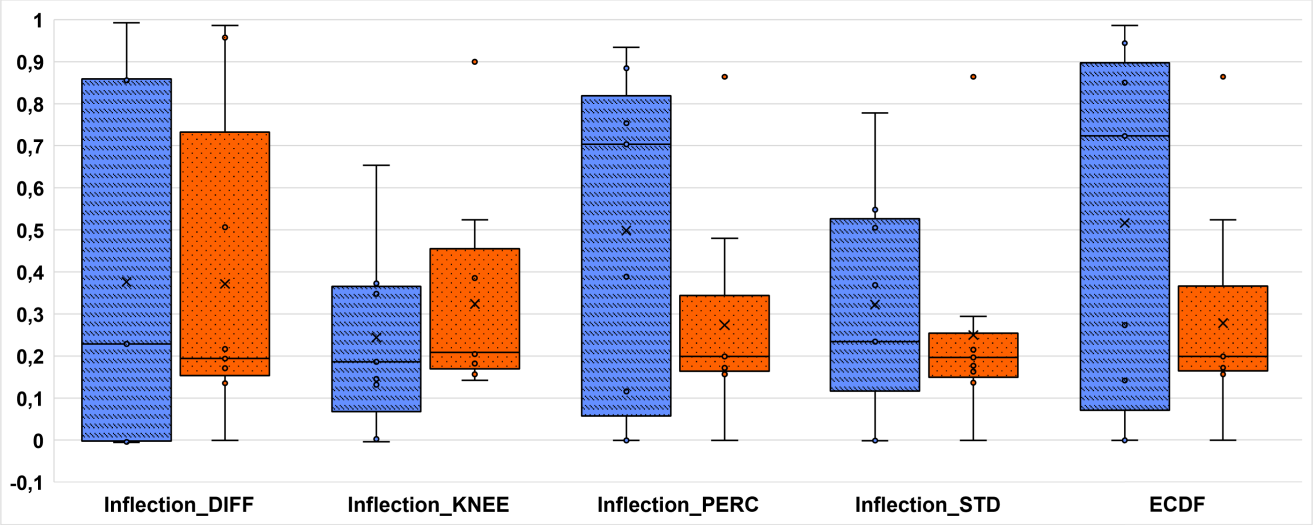

FIGURE 7. Boxplots of the MCC metric using graphical-based unsupervised threshold selection methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

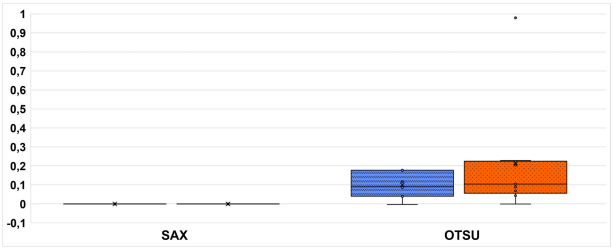

FIGURE 8. Boxplots of the MCC metric using the SAX and OTSU methods for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

implemented, in this section we focus on summarizing the results obtained in the context of comparing the average results and execution time between the groups of methods presented in IV-R, determining the best method among all those tested here, comparing the execution times of the methods, and comparing the results of the implemented methods with the methods of the *PyThresh* toolkit.

First, we performed an overall comparison of the results between each group of threshold selection methods listed in IV-R. Looking at the average MCC value of each group in the first test scenario, the distribution-based and graphical-based methods produced the best results with an average MCC of 0.4, while the clustering-based methods produced the worst results with an average MCC of only 0.07 across all nine datasets used. In the second test scenario, the average MCC for both the distribution-based and graphical-based methods deteriorated to about 0.3. In contrast, the statistics-based methods had better average performance with a jump from 0.2 to 0.25 average MCC value. The clustering-based methods also had a jump to 0.27 average MCC value. The density-based methods remain fairly stable as the test scenarios change, with an average MCC of 0.26. The methods from the *PyThresh* toolkit also improved their performance in the second test scenario, reaching an average MCC of 0.23 (previously 0.16).

So far, we have compared the results of each threshold selection method with the methods of its group. Now it is time to compare all methods to determine the best method based on our test scenarios. In the analysis of the first test scenario, there are two methods whose average MCC score exceeds 0.5 across nine datasets. These are POT with 0.53 and ECDF with an average MCC score of 0.52. Close to the average MCC score of 0.5 are also the distribution adjustment method using the Kolmogorov-Smirnov test, SPOT, and the inflection point PERC method with an average MCC score of about 0.49. To get a better sense of these results, we also converted the results to the average percentage relative to the best possible MCC scores for each dataset, which can be calculated using the following equation:

$$p = \frac{\sum_{i=1}^{n} \frac{\text{MCC}_i}{\text{MCC\_OPT}_i}}{n} \tag{20}$$

where p is the average percentage achieved, MCC_i is the MCC value achieved by a given method for dataset i, MCC_OPT_i is the MCC value achieved for dataset i using the optimal threshold selection method that maximized the MCC value, and n is the number of datasets tested. On this basis, the POT method achieved on average almost 70% of the best possible MCC value, while the ECDF method, the distribution adjustment method using the Kolmogorov-Smirnov test, the inflection point PERC method, and SPOT achieved about 60%. When considering the second test scenario, the results are worse. Only four methods provided an average MCC value above 0.4, namely the MAD and the

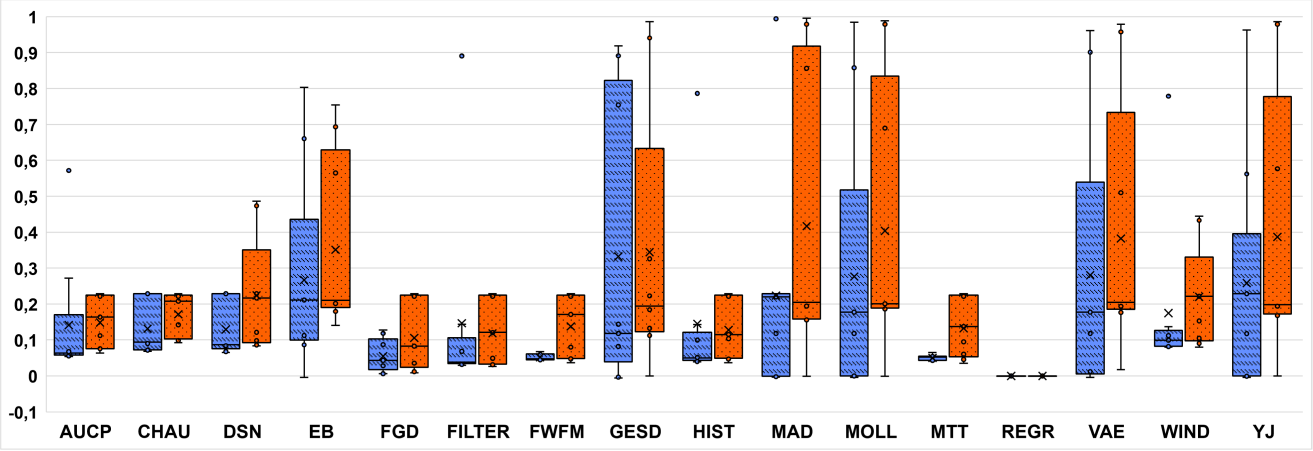

FIGURE 9. Boxplots of the MCC metric using 16 unsupervised threshold selection methods from the PyThresh toolkit for the two testing scenarios across nine datasets. The x-axis represents the different threshold selection methods. The y-axis measures the MCC values. The results of the first testing method are shown as blue bars with diagonal stripes, and those of the second testing method as orange bars with dots.

MOLL methods from the *PyThresh* toolkit, as well as the k-sigma method with static and sliding windowing. Of these methods, the k-sigma method with sliding windowing is the best with an average MCC of 0.44. The average percentage, defined by (20), shows that the MAD the MOLL methods perform best with an average of 57% of the best possible MCC value. Of the other methods, only the YJ and VAE methods from the *PyThresh* toolkit and the k-sigma method with static and sliding windowing achieved an average percentage close to 55%.

We also performed an analysis of the initial assumption about the input data that each method expects based on its implementation. This can be seen in the third column of Table 3 where we divided the input data into data containing only normal examples and data containing both normal examples and anomalies (mixed data). We assumed that methods using mixed data should perform better in the first test scenario, while methods using only normal data should perform better in the second test scenario. Based on the results we have, we can now verify that this assumption is indeed true in most cases, with our assumption being true in 80% of the cases. The methods that expect normal data achieved a better average MCC score in the second test scenario than in the first scenario. Six methods do not meet our assumption, namely OTSU, the distance between the sets method, the k-means method, the inflection point KNEE method, and the distribution adjustment method with the Anderson-Darling and Kolmogorov-Smirnov tests. The results we obtained with the distribution adjustment method with the Anderson-Darling and Kolmogorov-Smirnov tests could be purely random since this method randomly selects the distributions when generating the threshold. The same is true for the inflection point KNEE method since the other variants of this method follow our assumption. From the implementations of the OTSU, distance between the sets, and k-means methods, it is clear that they assume the existence of different groups of data (normal and abnormal). It remains to be analyzed why these methods provide such results for these test scenarios.

The next question is how the results of the methods implemented in Section IV compare to the results of the methods already implemented in the *PyThresh* toolkit. Of the *PyThresh* methods, we have already found that the GESD method performs best in terms of performance and stability, with an average MCC metric of 0.33 in the first test scenario and 0.34 in the second test scenario. In analyzing our results, we found that many of the implemented methods outperform the GESD method in the first test scenario, but there are only three implemented methods that perform better in both test scenarios, namely: KDE with exponential kernel, k-sigma method with sliding windowing where k is set to 5, and inflection point DIFF method. Looking at the results individually, we can see that the inflection point method outperforms the other two methods overall and achieves better or similar results compared to the GESD method on seven datasets in the first test scenario and on all datasets in the second test scenario.

Finally, we compare the execution times between the tested methods and between the method groups. From the results, we conclude that these execution times vary greatly among the observed methods, as some such as DSN, FGD, FWFM, VAE, WIND, and YJ require more than 5000 seconds to generate a threshold from anomaly scores, which is too much and impractical for real-world use. All other methods, except AUCP, GESD, MTT, SPOT, and the distribution adjustment method, take less than 60 seconds to execute, which is suitable for real-world use. When we compared different groups of methods by their execution time, we found that the methods from the *PyThresh* toolkit have the longest execution time, with an average of more than 2500 seconds and a median of 70 seconds. Density-based methods also have a slightly higher average execution

Algorithm 1 Unsupervised Threshold Selection Method Using Sliding Windowing and k-Sigma Method

procedure SLIDING_K-SIGMA_METHOD(X, k, w)
  $t \leftarrow$ []
  $i \leftarrow 0$
  while $i < |X|$ **do**
    $\mu \leftarrow$ Inf
    $\sigma \leftarrow$ Inf
    if $i \neq 0$ **then**
      $j \leftarrow i - w$
      if $j < 0$ **then**
        $j \leftarrow 0$
      $W \leftarrow \{X_j, \ldots, X_{i-1}\}$
      $\mu \leftarrow$ mean(W)
      $\sigma \leftarrow$ stdev(W)
    $t_i \leftarrow \mu + k \times \sigma$
    $i \leftarrow i + 1$
  labels $\leftarrow$ predict(X, t)
  return labels

Algorithm 2 Unsupervised Threshold Selection Method Using EWMA and k-Sigma Method

procedure EWMA_K-SIGMA_METHOD(X, k, λ)
  $t \leftarrow$ []
  $\mu \leftarrow$ []
  $s \leftarrow$ variance(X)
  $i \leftarrow 0$
  while $i < |X|$ **do**
    if $i = 0$ **then**
      $\mu_i \leftarrow \lambda \times X_i$
    else
      $\mu_i \leftarrow \lambda \times X_i + (1 - \lambda) \times \mu_{i-1}$
    $\sigma \leftarrow \sqrt{s \times (\frac{\lambda}{2-\lambda}) \times (1 - (1 - \lambda)^{2i})}$
    $t_i \leftarrow \mu_i + k \times \sigma$
    $i \leftarrow i + 1$
  labels $\leftarrow$ predict(X, t)
  return labels

time of 260 seconds with a median of 3.5 seconds, mainly due to the SPOT method. Clustering-based and density-based methods follow with an average and median execution time of about 12 seconds. The graphical-based methods take about 8 seconds on average with a median of 0.17, with the outlier being the standard deviation inflection point method, which takes 40 seconds to execute. Due to their simplicity, the fastest methods are the statistics-based methods with an average execution time of 2.4 seconds and a median of 0.85 seconds. The outlier here is the k-sigma method with sliding windowing, which takes about 10 seconds to generate the threshold.

An important limitation of this study is the potential lack of generalizability to other network environments. Since our test data comes from a limited set of real firewall logs, the effectiveness of the presented methods in other networks with different characteristics remains uncertain. Furthermore, we did not investigate the correlation between the type, shape, or distribution of the data and the threshold selection method, nor did we investigate the correlation between the type of unsupervised anomaly detection model and the threshold selection method or group. Understanding these correlations could be useful for generalizing our results.

Moreover, our research mainly focused on static approaches, which may not be as effective as streaming methods in real-world scenarios where dynamic network conditions prevail. Additionally, we aimed to present general approaches rather than describing in detail each specific threshold selection method found in the literature. Therefore, selecting a method is only the first step in the threshold selection process. The next crucial step is to adapt these general approaches to specific datasets and real dynamic network environments to ensure their effectiveness and applicability.

VIII. CONCLUSION

In this paper, we have provided an overview of all threshold selection methods used in unsupervised anomaly detection that we found while reviewing the literature. All methods found were classified as supervised or unsupervised, and for each, a motivation is given, the implementation is discussed, and variants of the approach are presented. In the end, we were able to find five different supervised approaches and twenty different unsupervised approaches. The unsupervised methods were additionally categorized by the input data they expect, the type of output data they produce, and whether or not they are parametric. Furthermore, these methods were grouped according to the idea behind the method and the type of thresholding procedure. We formed six groups: statistics-based, distribution-based, clustering-based, density-based, graphical-based methods, and others.

To test all threshold selection methods, we created two test scenarios, each tailored for a specific use case. The first test scenario focuses on the situation where we have data that contains anomalies, and the second scenario was created to simulate a situation where we only have normal data available for threshold selection. For the data used to create the test scenarios, we used three months of real firewall logs in which three types of anomalies were inserted. In total, we used nine different datasets, with each dataset corresponding to one month of the original firewall logs combined with one type of anomaly dataset. To generate anomaly scores, we used the unsupervised anomaly detection model ECOD. In addition to the methods found in the literature, we also included sixteen unsupervised methods from the *PyThresh* toolkit in our testing process. Instead of using the popular F1 score, we chose to present the results using the MCC metric, which is considered more reliable and yields a high score only if the prediction performs well in all four categories of the confusion matrix. In addition to the MCC metric, the results also show the time taken by each method to generate a threshold. All datasets used in this work as well as the

Algorithm 3 Unsupervised Threshold Selection Method Using Distribution Adjustment Method

procedure DISTRIBUTION_ADJUSTMENT_METHOD($X, k,$ goodness_of_fit_test)
    all_distribution_names ← get_all_SciPy_distributions()
    shuffle(all_distribution_names)
    $\mu, \sigma \leftarrow 0$
    $p_{min} \leftarrow$ Inf
    for distribution_name in all_distribution_names **do**
        parameters ← distribution(distribution_name).fit(X)
        cdf ← distribution(distribution_name, parameters).cdf
        $m, s \leftarrow$ distribution(distribution_name, parameters).stats()
        $p \leftarrow$ goodness_of_fit_test(X, cdf)
        if $p < p_{min}$ **then**
            $p_{min} \leftarrow p$
            $\mu, \sigma \leftarrow m, s$
    T ← $\mu + k \times \sigma$
    return T

Algorithm 4 Unsupervised Threshold Selection Method Using Peaks-Over-Threshold

procedure POT_METHOD(X, p, q)
    $t \leftarrow$ perc(X, p)
    $Y \leftarrow \{X_i - t | X_i > t\}$
    $\tilde{\gamma}, \tilde{\mu}, \tilde{\sigma} \leftarrow$ stats.genpareto.fit(Y)
    $T \leftarrow t + \frac{\tilde{\sigma}}{\tilde{\gamma}}((\frac{q \times |X|}{|Y|})^{-\tilde{\gamma}} - 1)$
    return T

Algorithm 5 Unsupervised Streaming Threshold Selection Method Using Peaks-Over-Threshold

procedure SPOT_METHOD(X, p, q, m)
    $n \leftarrow$ round($m \times |X|$)
    $M \leftarrow \{X_0, \ldots, X_{n-1}\}$
    $t \leftarrow$ perc(M, p)
    $Y \leftarrow \{M_i - t | M_i > t\}$
    $\tilde{\gamma}, \tilde{\mu}, \tilde{\sigma} \leftarrow$ stats.genpareto.fit(Y)
    $T \leftarrow t + \frac{\tilde{\sigma}}{\tilde{\gamma}}((\frac{q \times n}{|Y|})^{-\tilde{\gamma}} - 1)$
    $i \leftarrow n$
    $A \leftarrow$ []
    labels ← zeroes($|X|$)
    while $i < |X|$ **do**
        if $X_i > T$ **then**
            $A_i \leftarrow X_i$
            labels[i] ← 1
        else if $X_i > t$ **then**
            $Y_i \leftarrow X_i - t$
            $i \leftarrow i + 1$
            $\tilde{\gamma}, \tilde{\mu}, \tilde{\sigma} \leftarrow$ stats.genpareto.fit(Y)
            $T \leftarrow t + \frac{\tilde{\sigma}}{\tilde{\gamma}}((\frac{q \times i}{|Y|})^{-\tilde{\gamma}} - 1)$
        else
            $i \leftarrow i + 1$
    return labels

Algorithm 6 Unsupervised threshold selection method using Kernel Density Estimation

procedure KDE_METHOD(X, kernel, α)
    $h \leftarrow (\frac{3 \times |X|}{4})^{-\frac{1}{5}}$
    $d \leftarrow \frac{\max(X) - \min(X)}{1000 - 1}$
    $E \leftarrow$ []
    $i \leftarrow 0$
    $N_e \leftarrow 1000$
    while $i < N_e$ **do**
        $E_i \leftarrow \min(X) + d \times i$
        $i \leftarrow i + 1$
    kde ← KernelDensity(h, kernel)
    kde.fit(X)
    $p_d \leftarrow$ kde.evaluate(E)
    $C \leftarrow$ cummulative_sum(p_d)
    $p \leftarrow \frac{C}{C_{|C|-2}}$
    $i \leftarrow 0$
    $d_{min} \leftarrow$ Inf
    $T \leftarrow 0$
    while $i < |E|$ **do**
        $d \leftarrow \text{abs}(p_i - (1 - \alpha))$
        if $d < d_{min}$ **then**
            $T \leftarrow E_i$
            $d_{min} \leftarrow d$
        $i \leftarrow i + 1$
    return T

scripts for creating the test datasets, the implementation of the threshold selection methods, and the scripts for testing these methods are available in the public GitHub repository [105].

We concluded that the main advantage of supervised threshold selection methods is that they can optimize the desired evaluation metric, are simple and fast to execute, and do not require additional parameters. On the other hand, a major disadvantage of these methods is, of course, that they are supervised. Moreover, they do not handle datasets containing only normal data, and they produce only static thresholds. Of the observed supervised methods, we recommend those based on the PR curve over those based on the ROC curve because they work better with

Algorithm 7 Unsupervised Threshold Selection Method Using k-Means Algorithm

procedure K-MEANS_METHOD(X)
  centers ← [0, mean(X), max(X)]
  kmeans ← KMeans(k=3, init=centers)
  kmeans.fit(centers) ▷ To keep fixed centers as initialized
  labels ← kmeans.predict(X)
  $t, T \leftarrow$ min(X[labels=2]) ▷ Minimum value of anomaly scores labeled as 2
  $T_{max} \leftarrow$ max(X)
  $d \leftarrow 0.01$
  $S_{max} \leftarrow 0$
  while $t \leq T_{max}$ **do**
    $X_l \leftarrow \{X_i | X_i \leq t\}$
    $X_h \leftarrow \{X_i | X_i > t\}$
    $S \leftarrow \frac{\sigma(X)}{\sigma(X_l)} \times \frac{1}{|X_h|}$
    if $S \geq S_{max}$ **then**
      $T \leftarrow t$
      $S_{max} \leftarrow S$
    $t \leftarrow t + d$
  return T

Algorithm 8 Unsupervised Threshold Selection Method Using DBSCAN Algorithm

procedure DBSCAN_METHOD(X, eps, minPts)
  dbscan ← DBSCAN(eps,minPts)
  labels ← dbscan.fit_predict(X)
  if |labels| = 1 **then**
    $T \leftarrow$ max(X) + stdev(X)
  else if |labels| = 2 **then**
    $\min_0 \leftarrow$ min(X[labels=0])
    $\max_0 \leftarrow$ max(X[labels=0])
    $\min_1 \leftarrow$ min(X[labels=1])
    $\max_1 \leftarrow$ max(X[labels=1])
    if $\min_0 < \min_1$ **then**
      $d \leftarrow \text{abs}((\min_1 - \max_0)/2)$
      $T \leftarrow \max_0 + d$
    else
      $d \leftarrow \text{abs}((\min_0 - \max_1)/2)$
      $T \leftarrow \max_1 + d$
  else
    $T \leftarrow 0$
  return T

Algorithm 9 Unsupervised Threshold Selection Method Using Local Outlier Factor

procedure LOF_METHOD(X, m)
  lof ← LocalOutlierFactor(m)
  labels ← lof.fit_predict(X)
  return labels

unbalanced datasets, which are more common in network anomaly detection.

In this study, we presented the results in the form of boxplots of the MCC metric across nine datasets for each unsupervised threshold selection method for the two test scenarios compared with the other methods from their group. We also compared the average MCC metric between all methods to determine which was best for each test scenario. The results showed that the POT and ECDF methods were the best for the first test scenario, with the POT method averaging nearly 70% of the best possible MCC value. On the other

Algorithm 10 Unsupervised Threshold Selection Method Using the Distance Between the Sets

procedure DBS_METHOD(X)
  $t \leftarrow$ mean(X)
  $T_{max} \leftarrow$ max(X)
  $d \leftarrow 0.01$
  $T, s_{max} \leftarrow 0$
  while $t \leq T_{max}$ **do**
    $X_l \leftarrow \{X_i | X_i \leq t\}$
    $X_h \leftarrow \{X_i | X_i > t\}$
    $s \leftarrow \frac{\min(X_h) - \max(X_l)}{\min(X_h) + \max(X_l) - 2 \times \min(X_l)}$
    if $s \geq s_{max}$ **then**
      $T \leftarrow t$
      $s_{max} \leftarrow s$
    $t \leftarrow t + d$
  return T

Algorithm 11 Unsupervised threshold selection method using inflection point approach

procedure INFLECTION_POINT_METHOD(variant, X, d, k)
  $XS \leftarrow$ sort(X, ascending)
  $S \leftarrow$ []
  $i \leftarrow 0$
  while $i < d$ **do**
    $S_i \leftarrow NaN$
    $i \leftarrow i + 1$
  $i \leftarrow 0$
  while $i < |X| - d$ **do**
    $S_{i+d} \leftarrow XS_{i+d} - XS_i$
    $i \leftarrow i + 1$
  $P \leftarrow$ find_peaks(S)
  $p \leftarrow$ find_largest_index(P, S) ▷ Finds index of the largest peak
  threshold_candidate_indexes ← []
  $i \leftarrow p - d$
  while $i < p$ **do**
    threshold_candidate_indexes.put(i)
    $i \leftarrow i + 1$
  $TC \leftarrow X$[threshold_candidate_indexes]
  if variant = "perc" **then**
    $T \leftarrow$ perc(TC, k)
  else if variant = "diff" **then**
    $S \leftarrow$ []
    $i \leftarrow 0$
    while $i < |TC|$ **do**
      $s \leftarrow TC_{i+1} - TC_i$
      S.put(s)
      $i \leftarrow i + 1$
    $i \leftarrow$ threshold_candidate_indexes [argmax(S)]
    $T \leftarrow X_i$
  return T

Algorithm 12 Unsupervised Threshold Selection Method Using Standard Deviations Inflection Point

procedure STANDARD_DEVIATIONS_METHOD(X, p)
  $i \leftarrow 0$
  $m \leftarrow p \times |X|$
  $S \leftarrow$ []
  $XS \leftarrow$ sort(X, descending)
  while $i < m$ **do**
    $s \leftarrow$ stdev(XS)
    $S_i \leftarrow s$
    $XS \leftarrow \{XS_0, \ldots, XS_{|XS|-2}\}$
    $i \leftarrow i + 1$
  $k \leftarrow$ KneeLocator($\{0, \ldots, |S|-1\}, S$).knee
  $T \leftarrow \frac{X_k + X_{k+1}}{2}$
  return T

Algorithm 13 Unsupervised Threshold Selection Method Using ECDF

procedure ECDF_METHOD(X, α)
  $XS \leftarrow$ sort(X, ascending)
  $Y \leftarrow$ []
  $i \leftarrow 1$
  while $i \leq |X|$ **do**
    $Y_{i-1} \leftarrow \frac{i}{|X|}$
    $i \leftarrow i + 1$
  $d_{min} \leftarrow$ Inf
  $i \leftarrow 0$
  while $i < |XS|$ **do**
    $d \leftarrow$ abs($Y_i - (1 - \alpha)$)
    if $d < d_{min}$ **then**
      $T \leftarrow XS_i$
      $d_{min} \leftarrow d$
    $i \leftarrow i + 1$
  return T

Algorithm 14 Unsupervised Threshold Selection Method Using Symbolic Aggregate approXimation

procedure SAX_METHOD(X)
  sax $\leftarrow$ SymbolicAggregateApproximation (n_bins=2)
  S $\leftarrow$ sax.fit_transform(X)
  labels $\leftarrow$ []
  $i \leftarrow 0$
  while $i < |S|$ **do**
    if $S_i =$ "a" **then**
      labels.put(0)
    else if $S_i =$ "b" **then**
      labels.put(1)
    $i \leftarrow i + 1$
  return labels

Algorithm 15 Unsupervised Threshold Selection Method Using Otsu's Method

procedure OTSU_METHOD(X)
  $T_{max} \leftarrow$ max(X)
  $t, T \leftarrow$ mean(X)
  $\sigma^2_{min} \leftarrow$ Inf
  $d \leftarrow$ 0.001
  while $t \leq T_{max}$ **do**
    labels $\leftarrow$ predict(X, t)
    $n_0 \leftarrow 0$
    $i \leftarrow 0$
    while $i < |X|$ **do**
      if labels$_i = 0$ **then**
        $n_0 \leftarrow n_0 + 1$
      $i \leftarrow i + 1$
    $\omega_0 \leftarrow \frac{n_0}{|X|}$
    $\omega_1 \leftarrow 1 - \omega_0$
    $\sigma^2_0 \leftarrow$ variance(X[labels=0])
    $\sigma^2_1 \leftarrow$ variance(X[labels=1])
    $\sigma^2 \leftarrow \omega_0 \times \sigma^2_0 + \omega_1 \times \sigma^2_1$
    if $\sigma^2 < \sigma^2_{min}$ **then**
      $T \leftarrow t$
      $\sigma^2_{min} \leftarrow \sigma^2_\omega$
    $t \leftarrow t + d$
  return T

hand, MAD, MOLL, and the k-sigma method with sliding windowing were the best for the second test scenario, with the MAD and MOLL methods achieving on average 57% of the best MCC value. We also compared the implemented threshold selection methods with those already implemented in the *PyThresh* toolkit. When compared to the overall best *PyThresh* method, GESD, we concluded that the inflection point DIFF method outperforms it on average in both test scenarios. The results also showed that in 80% of the cases, the input data that the method expects affects the difference between the results of the two test scenarios, so that the methods expecting data with anomalies perform better on average in the first test scenario, while the methods expecting only normal data perform better in the second test scenario.

We also compared the different method groups according to the average MCC value and execution time. We concluded that the best results in both test scenarios were obtained by the distribution-based and graphical-based methods, while the clustering-based methods performed the worst. Looking at the test scenarios, we found that the distribution-based and graphical-based methods performed better overall in the first test scenario, while the clustering-based, statistics-based, and *PyThresh* methods performed better in the second test scenario. When comparing the execution times between the method groups, we found that the methods from the *PyThresh* toolkit had the longest execution time on average, while the statistics-based methods had the shortest. Looking at the individual methods, some methods such as DSN, AUCP, MTT, FGD, FWFM, VAE, WIND, YJ, SPOT, and GESD take a lot of time to execute and are not suitable for use in practice, as well as the distribution adjustment method that consumes time depending on the time limit set. Each of the other methods takes less than a minute to select the threshold and therefore can be used in an anomaly detection system.

For future work, it is important to perform all the tests described in this study with different publicly available

network anomaly detection datasets, e.g., the KDD-99 Cup dataset or the NSL-KDD dataset [112]. Instead of selecting the threshold only from the anomaly scores generated by ECOD, some other popular unsupervised models should also be used, as the other models can potentially generate anomaly scores that come from different distributions. Apart from these additional tests, it should be analyzed why the results of certain methods such as OTSU, the distance between the sets method, and the k-means method do not correlate with the data they expect as input. It may also be beneficial to correlate which of the threshold selection methods works best with each of the unsupervised anomaly detection model types.

APPENDIX. PSEUDOCODES

In this section, we provided pseudocodes for each unsupervised threshold selection method presented in Table 3.

REFERENCES

[1] E. M. Ferragut, J. Laska, and R. A. Bridges, "A new, principled approach to anomaly detection," in *Proc. 11th Int. Conf. Mach. Learn. Appl.*, vol. 2, Dec. 2012, pp. 210–215.

[2] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *IEEE Netw.*, vol. 8, no. 3, pp. 26–41, May 1994.

[3] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Comput. Netw.*, vol. 51, no. 12, pp. 3448–3470, Aug. 2007.

[4] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *Comput. Secur.*, vol. 28, nos. 1–2, pp. 18–28, Feb. 2009.

[5] P. Gogoi, D. K. Bhattacharyya, B. Borah, and J. K. Kalita, "A survey of outlier detection methods in network anomaly identification," *Comput. J.*, vol. 54, no. 4, pp. 570–588, Apr. 2011.

[6] K. Alrawashdeh and C. Purdy, "Toward an online anomaly intrusion detection system based on deep learning," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 195–200.

[7] H. Wang, M. J. Bah, and M. Hammad, "Progress in outlier detection techniques: A survey," *IEEE Access*, vol. 7, pp. 107964–108000, 2019.

[8] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, and S.-K. Ng, "MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks," in *Artificial Neural Networks and Machine Learning—ICANN 2019: Text and Time Series*, I. V. Tetko, V. Kůrková, P. Karpov, and F. Theis, Eds., Cham, Switzerland: Springer, 2019, pp. 703–716.

[9] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 3, pp. 186–205, Aug. 2000.

[10] J. Clark, Z. Liu, and N. Japkowicz, "Adaptive threshold for outlier detection on data streams," in *Proc. IEEE 5th Int. Conf. Data Sci. Adv. Anal. (DSAA)*, Oct. 2018, pp. 41–49.

[11] D. Kulik, S. Schmidland, and L. Perini, "Kulikdm/Pythresh: V0.3.6," Zenodo, Feb. 2024, doi: 10.5281/zenodo.10613189.

[12] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, "The MVTec anomaly detection dataset: A comprehensive real-world dataset for unsupervised anomaly detection," *Int. J. Comput. Vis.*, vol. 129, no. 4, pp. 1038–1059, Apr. 2021.

[13] Z. He, P. Chen, X. Li, Y. Wang, G. Yu, C. Chen, X. Li, and Z. Zheng, "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 4, pp. 1705–1719, Apr. 2023.

[14] S. Liu, B. Zhou, Q. Ding, B. Hooi, Z. Zhang, H. Shen, and X. Cheng, "Time series anomaly detection with adversarial reconstruction networks," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4293–4306, Apr. 2023.

[15] C. Yang, Z. Du, X. Meng, X. Zhang, X. Hao, and D. A. Bader, "Anomaly detection in catalog streams," *IEEE Trans. Big Data*, vol. 9, no. 1, pp. 294–311, Feb. 2023.

[16] K. S. Killourhy and R. A. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2009, pp. 125–134.

[17] R. Joyce and G. Gupta, "Identity authentication based on keystroke latencies," *Commun. ACM*, vol. 33, no. 2, pp. 168–176, Feb. 1990.

[18] L. C. F. Araujo, L. H. R. Sucupira, M. G. Lizarraga, L. L. Ling, and J. B. T. Yabu-Uti, "User authentication through typing biometrics features," *IEEE Trans. Signal Process.*, vol. 53, no. 2, pp. 851–855, Feb. 2005.

[19] S. Bleha, C. Slivinsky, and B. Hussien, "Computer-access security systems using keystroke dynamics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 12, pp. 1217–1222, Dec. 1990.

[20] S. Haider, A. Abbas, and A. K. Zaidi, "A multi-technique approach for user identification through keystroke dynamics," in *Proc. SMC Conf. IEEE Int. Conf. Syst., Man Cybern., Cybern. Evolving Syst., Hum., Org., Complex Interact.*, vol. 2, Oct. 2000, pp. 1336–1341.

[21] Y. Chae, N. Katenka, and L. Dipippo, "Adaptive threshold selection for trust-based detection systems," in *Proc. IEEE 16th Int. Conf. Data Mining Workshops (ICDMW)*, Dec. 2016, pp. 281–287.

[22] A. B. Sharma, L. Golubchik, and R. Govindan, "Sensor faults: Detection methods and prevalence in real-world datasets," *ACM Trans. Sensor Netw.*, vol. 6, no. 3, pp. 1–39, Jun. 2010.

[23] V. Bewick, L. Cheek, and J. Ball, "Statistics review 13: Receiver operating characteristic curves," *Critical Care*, vol. 8, no. 6, pp. 1–5, 2004.

[24] J. Cook and V. Ramadas, "When to consult precision-recall curves," *Stata J., Promoting Commun. Statist. Stata*, vol. 20, no. 1, pp. 131–148, Mar. 2020.

[25] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLoS ONE*, vol. 11, no. 4, Apr. 2016, Art. no. e0152173.

[26] A. Soule, K. Salamatian, and N. Taft, "Combining filtering and statistical methods for anomaly detection," in *Proc. Internet Meas. Conf.*, Berkeley, CA, USA, 2005, p. 1.

[27] M. D. Ruopp, N. J. Perkins, B. W. Whitcomb, and E. F. Schisterman, "Youden index and optimal cut-point estimated from observations affected by a lower limit of detection," *Biometrical J.*, vol. 50, no. 3, pp. 419–430, Jun. 2008.

[28] Q. Yan and R. S. Blum, "Distributed signal detection under the Neyman–Pearson criterion," *IEEE Trans. Inf. Theory*, vol. 47, no. 4, pp. 1368–1377, May 2001.

[29] C. R. Harshaw, R. A. Bridges, M. D. Iannacone, J. W. Reed, and J. R. Goodall, "GraphPrints: Towards a graph analytic method for network anomaly detection," in *Proc. 11th Annu. Cyber Inf. Secur. Res. Conf.*, New York, NY, USA, Apr. 2016, pp. 15:1–15:4, Paper 15.

[30] E. Yu and S. Cho, "GA-SVM wrapper approach for feature subset selection in keystroke dynamics identity verification," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 3, 2003, pp. 2253–2257.

[31] S. Cho, C. Han, D. H. Han, and H.-I. Kim, "Web-based keystroke dynamics identity verification using neural network," *J. Organizational Comput. Electron. Commerce*, vol. 10, no. 4, pp. 295–307, Dec. 2000.

[32] S. Fatemifar, M. Awais, A. Akbari, and J. Kittler, "Developing a generic framework for anomaly detection," *Pattern Recognit.*, vol. 124, Apr. 2022, Art. no. 108500.

[33] P. Kang, S.-S. Hwang, and S. Cho, "Continual retraining of keystroke dynamics based authenticator," in *Advances in Biometrics*, S. W. Lee and S. Z. Li, Eds., Berlin, Germany: Springer, 2007, pp. 1203–1211.

[34] C.-C. Tsai, T.-H. Wu, and S.-H. Lai, "Multi-scale patch-based representation learning for image anomaly detection and segmentation," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. (WACV)*, Jan. 2022, pp. 3065–3073.

[35] D. H. Blevins, P. Moriano, R. A. Bridges, M. E. Verma, M. D. Iannacone, and S. C. Hollifield, "Time-based CAN intrusion detection benchmark," in *Proc. 3rd Int. Workshop Automot. Auto. Vehicle Secur.*, 2021, pp. 38–44.

[36] J. Bernacki and G. Kołaczek, "Anomaly detection in network traffic using selected methods of time series analysis," *Int. J. Comput. Netw. Inf. Secur.*, vol. 7, no. 9, pp. 10–18, Aug. 2015.

[37] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding Gaussian mixture model for unsupervised anomaly detection," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–19.

[38] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially learned anomaly detection," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 727–736.
[39] L. Bergman and Y. Hoshen, "Classification-based anomaly detection for general data," 2020, *arXiv:2005.02359*.
[40] M. Alvarez, J.-C. Verdier, D. K. Nkashama, M. Frappier, P.-M. Tardif, and F. Kabanza, "A revealing large-scale evaluation of unsupervised anomaly detection algorithms," 2022, *arXiv:2204.09825*.
[41] J. W. Tukey, *Exploratory Data Analysis*, vol. 2. Reading, MA, USA: Addison-Wesley, 1977.
[42] J.-M. Bardet and S.-F. Dimby, "A new non-parametric detector of univariate outliers for distributions with unbounded support," *Extremes*, vol. 20, no. 4, pp. 751–775, Dec. 2017.
[43] G. Bovenzi, G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Network anomaly detection methods in IoT environments via deep learning: A fair comparison of performance and robustness," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103167.
[44] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2015, pp. 1939–1947.
[45] P. Olukanmi, F. Nelwamondo, T. Marwala, and B. Twala, "Automatic detection of outliers and the number of clusters in k-means clustering via chebyshev-type inequalities," *Neural Comput. Appl.*, vol. 34, no. 8, pp. 5939–5958, Apr. 2022.
[46] L. Yang, R. Ma, H. M. Zhang, W. Guan, and S. Jiang, "Driving behavior recognition using EEG data from a simulated car-following experiment," *Accident Anal. Prevention*, vol. 116, pp. 30–40, Jul. 2018.
[47] H. S. Dhiman, D. Deb, S. M. Muyeen, and I. Kamwa, "Wind turbine gearbox anomaly detection based on adaptive threshold and twin support vector machines," *IEEE Trans. Energy Convers.*, vol. 36, no. 4, pp. 3462–3469, Dec. 2021.
[48] T. Qin, Z. Liu, P. Wang, S. Li, X. Guan, and L. Gao, "Symmetry degree measurement and its applications to anomaly detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, no. 1, pp. 1040–1055, Aug. 2019.
[49] A. H. Hamamoto, L. F. Carvalho, L. D. H. Sampaio, T. Abrão, and M. L. Proença, "Network anomaly detection system using genetic algorithm and fuzzy logic," *Expert Syst. Appl.*, vol. 92, pp. 390–402, Feb. 2018.
[50] R. Matias, A. M. M. Carvalho, L. B. Araujo, and P. R. M. Maciel, "Comparison analysis of statistical control charts for quality monitoring of network traffic forecasts," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2011, pp. 404–409.
[51] D. C. Montgomery, *Introduction to Statistical Quality Control*. Hoboken, NJ, USA: Wiley, 2020.
[52] F. J. Massey, "The Kolmogorov–Smirnov test for goodness of fit," *J. Amer. Stat. Assoc.*, vol. 46, no. 253, p. 68, Mar. 1951.
[53] C. Huber-Carol, N. Balakrishnan, M. Nikulin, and M. Mesbah, *Goodness-of-Fit Tests and Model Validity*. Cham, Switzerland: Springer, 2012.
[54] D. A. Darling, "The Kolmogorov–Smirnov, Cramer-von Mises tests," *Ann. Math. Statist.*, vol. 28, no. 4, pp. 823–838, Dec. 1957.
[55] N. M. Razali and Y. B. Wah, "Power comparisons of Shapiro–Wilk, Kolmogorov–Smirnov, Lilliefors and Anderson–Darling tests," *J. Stat. Model. Anal.*, vol. 2, no. 1, pp. 21–33, 2011.
[56] T. W. Anderson and D. A. Darling, "A test of goodness of fit," *J. Amer. Stat. Assoc.*, vol. 49, no. 268, pp. 765–769, 1954.
[57] T. W. Anderson, "Anderson–Darling tests of goodness-of-fit," in *International Encyclopedia of Statistical Science*. Cham, Switzerland: Springer, 2011, pp. 52–54.
[58] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, Jan. 2020.
[59] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2020.
[60] L. Haan and A. Ferreira, *Extreme Value Theory: An Introduction*, vol. 3. Cham, Switzerland: Springer, 2006.
[61] A. Siffer, P.-A. Fouque, A. Termier, and C. Largouet, "Anomaly detection in streams with extreme value theory," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Aug. 2017, pp. 1067–1075.
[62] A. A. Balkema and L. de Haan, "Residual life time at great age," *Ann. Probab.*, vol. 2, no. 5, pp. 792–804, Oct. 1974.
[63] J. P. Iii, "Statistical inference using extreme order statistics," *Ann. Statist.*, vol. 3, no. 1, pp. 119–131, Jan. 1975.
[64] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, Jul. 2019, pp. 2828–2837.
[65] B. W. Silverman, *EmphDensity Estimation for Statistics and Data Analysis*, 1st ed., Evanston, IL, USA: Routledge, 1998.
[66] S. Węglarczyk, "Kernel density estimation and its application," in *Proc. ITM Web Conf.*, vol. 23, 2018, p. 37.
[67] X. Wang, Y. Du, S. Lin, P. Cui, Y. Shen, and Y. Yang, "AdVAE: A self-adversarial variational autoencoder with Gaussian anomaly prior knowledge for anomaly detection," *Knowl.-Based Syst.*, vol. 190, Feb. 2020, Art. no. 105187.
[68] Y. Wang, X. Du, Z. Lu, Q. Duan, and J. Wu, "Improved LSTM-based time-series anomaly detection in rail transit operation environments," *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 9027–9036, Dec. 2022.
[69] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-means clustering algorithm," *Appl. Statist.*, vol. 28, no. 1, p. 100, 1979.
[70] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, "DBSCAN: Past, present and future," in *Proc. 5th Int. Conf. Appl. Digit. Inf. Web Technol. (ICADIWT)*, Feb. 2014, pp. 232–238.
[71] A. Karami and R. Johansson, "Choosing DBSCAN parameters automatically using differential evolution," *Int. J. Comput. Appl.*, vol. 91, no. 7, pp. 1–11, Apr. 2014.
[72] K. M. A. Patel and P. Thakral, "The best clustering algorithms in data mining," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2016, pp. 2042–2046.
[73] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA, 2000, pp. 93–104.
[74] W. E. Wecker, "Predicting the turning points of a time series," *J. Bus.*, vol. 52, no. 1, p. 35, Jan. 1979.
[75] P. Casas, J. Mazel, and P. Owezarski, "UNADA: Unsupervised network anomaly detection using sub-space outliers ranking," in *NETWORKING 2011*, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, Eds., Berlin, Germany: Springer, 2011, pp. 40–51.
[76] S. Zhong, D. Liu, L. Lin, M. Zhao, X. Fu, and F. Guo, "A novel anomaly detection method for gas turbines using weight agnostic neural network search," in *Proc. Asia–Pacific Int. Symp. Adv. Rel. Maintenance Modeling (APARM)*, Aug. 2020, pp. 1–6.
[77] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan, "Finding a 'kneedle' in a haystack: Detecting knee points in system behavior," in *Proc. 31st Int. Conf. Distrib. Comput. Syst. Workshops*, 2011, pp. 166–171.
[78] W. J. Conover, *Practical Nonparametric Statistics* (Wiley series in probability and statistics), 3rd ed., Hoboken, NJ, USA: Wiley, 1999.
[79] D. H. Hoang and H. D. Nguyen, "A PCA-based method for IoT network traffic anomaly detection," in *Proc. 20th Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2018, pp. 381–386.
[80] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: A novel symbolic representation of time series," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 107–144, Aug. 2007.
[81] M. Yue, T. Hong, and J. Wang, "Descriptive analytics-based anomaly detection for cybersecure load forecasting," *IEEE Trans. Smart Grid*, vol. 10, no. 6, pp. 5964–5974, Nov. 2019.
[82] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 1, pp. 62–66, Jan. 1979.
[83] D. Cozzolino and L. Verdoliva, "Single-image splicing localization through autoencoder-based anomaly detection," in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Dec. 2016, pp. 1–6.
[84] K. Ren, H. Yang, Y. Zhao, W. Chen, M. Xue, H. Miao, S. Huang, and J. Liu, "A robust AUC maximization framework with simultaneous outlier detection and feature selection for positive-unlabeled classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 10, pp. 3072–3083, Oct. 2019.
[85] L. N. Bol'shev and M. Ubaidullaeva, "Chauvenet's test in the classical theory of errors," *Theory Probab. Appl.*, vol. 19, no. 4, pp. 683–692, 1975.
[86] D. Amagata, M. Onizuka, and T. Hara, "Fast and exact outlier detection in metric spaces: A proximity graph-based approach," in *Proc. Int. Conf. Manage. Data*, New York, NY, USA, Jun. 2021, pp. 36–48.
[87] M. Friendly, G. Monette, and J. Fox, "Elliptical insights: Understanding statistical methods through elliptical geometry," *Stat. Sci.*, vol. 28, no. 1, pp. 1–39, Feb. 2013.

[88] Z. Qi, D. Jiang, and X. Chen, "Iterative gradient descent for outlier detection," *Int. J. Wavelets, Multiresolution Inf. Process.*, vol. 19, no. 4, Jul. 2021, Art. no. 2150004.
[89] N. Hashemi, E. V. German, J. Pena Ramirez, and J. Ruths, "Filtering approaches for dealing with noise in anomaly detection," in *Proc. IEEE 58th Conf. Decis. Control (CDC)*, Dec. 2019, pp. 5356–5361.
[90] M. Joneidi, "Sparse auto-regressive: Robust estimation of AR parameters," 2013, *arXiv:1306.3317*.
[91] M. J. Alrawashdeh, "An adjusted Grubbs' and generalized extreme studentized deviation," *Demonstratio Mathematica*, vol. 54, no. 1, pp. 548–557, Dec. 2021.
[92] K. K. Thanammal, R. R. Vijayalakshmi, S. Arumugaperumal, and J. S. Jayasudha, "Effective histogram thresholding techniques for natural images using segmentation," *J. Image Graph.*, vol. 2, no. 2, pp. 113–116, 2015.
[93] A. N. and S. S. Pawar, "Periodicity detection of outlier sequences using constraint based pattern tree with MAD," 2015, *arXiv:1507.01685*.
[94] H. Karcher, "Riemannian center of mass and mollifier smoothing," *Commun. Pure Appl. Math.*, vol. 30, no. 5, pp. 509–541, Sep. 1977.
[95] Michiel A. Keyzer and B. G. J. S. Sonneveld, "Using the mollifier method to characterize datasets and models: The case of the universal soil loss equation," *ITC J.*, vol. 3, no. 4, pp. 263–272, 1997.
[96] D. Rengasamy, B. C. Rothwell, and G. P. Figueredo, "Towards a more reliable interpretation of machine learning outputs for safety-critical systems using feature importance fusion," *Appl. Sci.*, vol. 11, no. 24, p. 11854, Dec. 2021.
[97] C. C. Aggarwal, "Linear models for outlier detection," in *Outlier Analysis*. Cham, Switzerland: Springer, 2017, pp. 65–110.
[98] Zhisheng Xiao, Qing Yan, and Yali Amit, "Likelihood regret: An out-of-distribution detection score for variational auto-encoder," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., Red Hook, NY, USA: Curran Associates, 2020, pp. 20685–20696.
[99] A. Jacobson, L. Kavan, and O. Sorkine-Hornung, "Robust inside-outside segmentation using generalized winding numbers," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–12, Jul. 2013.
[100] J. Raymaekers and P. J. Rousseeuw, "Transforming variables to central normality," *Mach. Learn.*, vol. 113, no. 8, pp. 4953–4975, Aug. 2024.
[101] D. C. Le and N. Zincir-Heywood, "Exploring anomalous behaviour detection and classification for insider threat identification," *Int. J. Netw. Manage.*, vol. 31, no. 4, Jul. 2021, Art. no. e2109, doi: 10.1002/nem.2109.
[102] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Sunnyvale, CA, USA: Insecure, 2009.
[103] I. Kovačević, A. Komadina, B. Štengl, and S. Groš, "Light-weight synthesis of security logs for evaluation of anomaly detection and security related experiments," in *Proc. 16th Eur. Workshop Syst. Secur.*, New York, NY, USA, May 2023, pp. 30–36.
[104] A. Komadina, I. Kovačević, B. Štengl, and S. Groš, "Comparative analysis of anomaly detection approaches in firewall logs: Integrating light-weight synthesis of security logs and artificially generated attack detection," *Sensors*, vol. 24, no. 8, p. 2636, Apr. 2024.
[105] A. Komadina, May 2024, "AnomalyThresholdSelector," doi: 10.5281/zenodo.7893199.
[106] Z. Li, Y. Zhao, X. Hu, N. Botta, C. Ionescu, and G. Chen, "ECOD: Unsupervised outlier detection using empirical cumulative distribution functions," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 12, pp. 12181–12193, May 2023.
[107] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica Biophysica Acta (BBA)-Protein Struct.*, vol. 405, no. 2, pp. 442–451, Oct. 1975.
[108] D. Chicco, N. Tötsch, and G. Jurman, "The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation," *BioData Mining*, vol. 14, no. 1, p. 13, Feb. 2021.
[109] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, p. 6, Dec. 2020.
[110] T. Klinsuwan, W. Ratiphaphongthon, R. Wangkeeree, R. Wangkeeree, and C. Sirisamphanwong, "Evaluation of machine learning algorithms for supervised anomaly detection and comparison between static and dynamic thresholds in photovoltaic systems," *Energies*, vol. 16, no. 4, p. 1947, Feb. 2023.
[111] M. Vlachos, P. Yu, and V. Castelli, "On periodicity detection and structural periodic similarity," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2005, pp. 449–460.
[112] R. R. Devi and M. Abualkibash, "Intrusion detection system classification using different machine learning algorithms on KDD-99 and NSL-KDD datasets—A review paper," *Int. J. Comput. Sci. Inf. Technol.*, vol. 11, no. 3, pp. 65–80, Jun. 2019.

ADRIAN KOMADINA was born in Zagreb, Croatia. He received the B.S. and M.S. degrees from the Faculty of Electrical Engineering and Computing, University of Zagreb, in 2019 and 2021, respectively. He is currently pursuing the Ph.D. degree. He started his professional career as a Research Associate with the Faculty of Electrical Engineering and Computing, where he worked in the fields of cybersecurity, data science, machine learning, and information visualization, which also coincides with his academic and research interests. He is a Security Operations Center (SOC) Engineer with Infigo IS.

MISLAV MARTINIĆ received the bachelor's and master's degrees from the Department of Mathematics, Faculty of Science, University of Zagreb, in 2019 and 2021, respectively. Transitioning into the professional realm, he began his journey as a Full-Stack Developer with the Financial Agency, contributing from 2021 to 2022. In 2022, he pivoted toward the role of a Data Scientist with CS Computer Systems. His research interests include software engineering, machine learning, cybersecurity, large language models, and anomaly detection.

STJEPAN GROŠ is currently an Assistant Professor with the Faculty of Electrical Engineering and Computing, University of Zagreb. He also deals with issues of research and development management. He participates in the implementation of several EU-funded projects in the field of cyber security, with a focus on the study of attacker behavior and automation using machine learning algorithms. He has published a number of articles in the field of information security, computer networks, and operating systems. His scientific and professional interests include the field of information and cyber security and the application of advanced methods in solving problems in these areas.

ŽELJKA MIHAJLOVIĆ (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the Faculty of Electrical Engineering, University of Zagreb, in 1988, 1993, and 1998, respectively. She is currently a Full Professor with the Department of Electronics, Microelectronics, Computer and Intelligent Systems, Faculty of Electrical Engineering and Computing, University of Zagreb. She participates in teaching several undergraduate and graduate courses, including Interactive Computer Graphics, Computer Graphics, Computer Animation, and Visualization. She has been collaborating on several projects funded by the Ministry of Science, Education and Sports of the Republic of Croatia. She leads project financed by Croatian Agency for SMEs, innovation and investments, and international project with automotive industry. Her primary research interests include computer graphics, visualization, volume rendering and interpolation, and engineering.