Network IDS Alert Classification with Active Learning Techniques

Risto Vaarandi
Centre for Digital Forensics and Cyber Security
Tallinn University of Technology
Tallinn, Estonia
risto.vaarandi@taltech.ee

Alejandro Guerra-Manzanares
Center for Interacting Urban Networks
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
alejandro.guerra@nyu.edu

***Abstract*—A Network Intrusion Detection System (NIDS) is a widely used security monitoring technology for detecting attacks against network services, beaconing activity of infected end user nodes, malware propagation, and other types of malicious network traffic. Unfortunately, NIDS technologies are known to generate a large number of alerts, with a significant proportion of them having low importance. During the last two decades, many machine learning and data mining based approaches have been proposed for highlighting high-importance alerts that require human attention. However, NIDS alert classification systems based on active learning have received marginal attention in the specialized research literature. This neglects the potential benefits of active learning which involves a human expert in the machine learning model life cycle. The current paper fills this research gap and studies the use of active learning techniques for NIDS alert classification.**

***Index Terms*—NIDS alert classification, active learning**

I. Introduction

A Network Intrusion Detection System (NIDS) is a widely used network security monitoring technology that has been adopted by many organizations, with a number of open-source and commercial NIDS platforms available on the market [1]–[3]. Unfortunately, NIDS is known to generate large volumes of alerts, with most of them having low importance. For example, in [4], the authors described a Suricata NIDS that generated about 200,000 alerts per day, with over 95% of them reflecting well-known attacks or being false positives.

In a recent study about Security Operations Centers (SOCs) [31], NIDS was identified as one of the main SOC technologies, and the volume of low-priority NIDS alerts as one of the main challenges for SOC analysts. One particular challenge is the large number of true alerts (i.e, not false positives, but reflecting real attacks) which have low importance for the given environment [31]. For these reasons, *alert prioritization* has been identified as one of the most important processes in a modern SOC [32]. It is worth to make a distinction between the application of machine learning (ML) for NIDS alert classification and network intrusion detection. Machine learning based NIDS involves classifying network flows (or packets) that serve as *input* for the NIDS, whereas NIDS alert classification focuses on the processing of the NIDS *output*. Thus, as highlighted in [33], they are both NIDS-related but distinct ML classification tasks with different inputs, methodology and overall purpose. In this paper, we focus on machine learning for alert prioritization purposes, assuming that the NIDS which creates these alerts already exists. More precisely, we propose *Outlier-N*, a novel methodology based on active learning, to enhance machine learning algorithms for NIDS alert classification.

In order to highlight alerts of high importance which represent successful intrusions and unusual attacks originating from sophisticated adversaries, several machine learning and data mining based approaches have been proposed [5]–[16]. Unfortunately, supervised machine learning approaches (e.g., [6], [10]–[13]) require large labeled data sets that are laborious and time-consuming to create for human experts. Data mining methods (e.g., [15], [16]) have a similar drawback – the knowledge detected by these methods has to be interpreted and turned into alert processing rules by a human expert. As discussed in [33], unsupervised methods tend to feature higher false positive rates and can thus be used for ancillary tasks.

The need for large labeled data sets is the main shortcoming of supervised machine learning methods, and for addressing this issue, active learning has been proposed. Active learning aims for training a high-performance supervised classification model on a small number of labeled data samples in an iterative fashion and with the support of a human annotator [17], [18]. Since active learning significantly reduces the workload of human experts, it has been used in the domain of network security for various tasks like network intrusion detection [34], data set generation purposes [19], [20], IoT botnet detection [21], and classification of NIDS signatures based on their relevance [22]. However, the use of active learning for NIDS alert classification has not received enough attention in prior work.

This paper addresses this research gap and studies active learning techniques for classifying NIDS alerts. The remainder of this paper is organized as follows – Section II covers related work, Section III describes the NIDS alert data set and the experimental setup, Section IV presents the evaluation of different active learning techniques on the data set, Section V

discusses the evaluation results, and Section VI concludes the paper.

II. Related Work and Background Knowledge

A. Related Work on NIDS Alert Classification

Kidmose et al. [6] proposed a supervised method that treated NIDS alerts as textual strings without applying any feature engineering to them. First, labeled training alerts were used to learn a mapping function for converting textual alerts into a vector space. For the mapping function, long short-term memory (LSTM) recurrent neural network and latent semantic analysis (LSA) were employed. The converted alerts were then clustered with DBSCAN algorithm, so that the most frequently occurring alert label was assigned to the entire cluster. During the NIDS alert classification phase, an incoming alert was assigned the label of the DBSCAN core point within the ϵ-distance, and if there was no such point, the alert was regarded as a false positive.

Ban et al. [11] evaluated several supervised and unsupervised machine learning algorithms on a large NIDS alert data set composed of 672,000 alerts. The data set was highly imbalanced, with less than 1% of alerts having high importance. The evaluated algorithms included support vector machine (SVM), weighted support vector machine (WSVM), decision tree, AdaBoost, linear discriminant analysis, naive Bayes, k-Nearest Neighbors, and two isolation forest (IF) based algorithms. According to the experiments, the best algorithms were SVM, WSVM, and AdaBoost. IF based algorithms featured the lowest precision.

Wang et al. [10] proposed a graph based method for detecting important features in NIDS alert data that helped to identify false alerts. The experiments were conducted on an imbalanced data set with less than 6% of alerts having high importance. According to the authors, gradient boosting tree models enabled the identification of true and false positive alerts effectively.

Feng et al. [13] described a user-centric organizational machine learning framework that processed NIDS alerts and other security alerts to find users at risk. Security events were converted into data points that represented users, and data points were labeled with the help of text mining and label propagation techniques. The authors evaluated multilayer neural network (MNN), SVM, random forest (RF), and logistic regression (LR). MNN and RF provided the best results. Shin et al. [12] described another organizational machine learning framework that employed binary SVM and one-class SVM methods for NIDS alert processing.

Vidović et al. [35] suggested a supervised NIDS alert prioritization method which was based on the importance of organizational devices. Each device was described with a feature vector, where features reflected the behavior of the device in the network (e.g., the number of bytes sent from the device). The human expert assigned importance levels to devices in the training data set, and after training, the ranking algorithm was used to predict the device importance for prioritizing relevant NIDS alerts.

Al-Mamory and Zhang [15] proposed a data mining method that assigned NIDS alerts with the same root cause to one cluster, so that each detected cluster was represented by a generalized alert. Generalized alerts were then interpreted by human experts to write filters for reducing the number of false positives with the same root cause. Ma et al. [16] suggested another data mining method that discovered frequent signature ID patterns from NIDS alert data. Detected patterns were used by human experts to develop rules for detecting frequent attack scenarios.

Tjhai et al. [14] suggested an unsupervised method that employed a self-organizing map (SOM) neural network for converting NIDS alerts into a two-dimensional map. The map was then clustered using the k-means algorithm, so that the alerts from the same activity were assigned to the same cluster. After that, the detected clusters were turned into data points, and SOM and k-means clustering were applied again to identify true and false positive alerts. Shittu et al. [7] proposed another unsupervised graph based method for NIDS alert prioritization involving the assignment of similar alerts into the same graph. For detecting anomalous alert groups, graphs were compared with each other, and graphs dissimilar to others received a high priority score.

In [8], an unsupervised framework was described that employed frequent itemset mining and clustering algorithms to discover frequent NIDS alert patterns. These patterns represented alerts of low importance and were used for distinguishing important alerts from irrelevant ones. Spathoulas and Katsikas [9] proposed an unsupervised clustering method for NIDS alerts that visualized detected clusters for easing the detection of security incidents.

In [5], SCAS stream clustering algorithm was proposed for real-time processing of an incoming NIDS alert stream in an unsupervised fashion. SCAS has been used in a real Security Operations Center (SOC) for several years, enabling a significant reduction of the number of NIDS alerts investigated by human analysts [4]. SCAS first aggregates alerts appearing for the same external host within a short time frame (e.g., 5 minutes), creating a data point from these alerts. Data points that represent frequently occurring NIDS alert patterns are detected as clusters, and if an incoming data point does not belong to any of the previously identified clusters, it is highlighted as an outlier that deserves closer attention.

Van Ede et al. [36] developed a semi-automated method for classifyings NIDS alerts and other security events which first identified event sequences originating from the same device. For each event, preceding events in the same sequence were analyzed with a deep learning model for identifying correlations between events, building the attention vector, and calculating the total attention for each contextual event. The attention information was then used for clustering event sequences with DBSCAN algorithm. Detected clusters were labeled by a human analyst. Labeled clusters allowed for semi-automated classification of further event sequences. Event sequences without a matching cluster were manually processed by a human analyst, updating the database of labeled clusters

during the process.

Although the approach proposed by [36] involved an iterative updating of the ML model that resembles active learning, the approach did not involve the use of a supervised classifier to select data points for labeling, a central element of active learning. To better illustrate the inner-working and conceptualize active learning, the following section provides a formal definition of it and describes the most common active learning methods.

B. Background Knowledge on Active Learning

Active learning is an iterative machine learning based modeling process that involves the interaction between a machine learning model and a human expert annotator as described in Fig. 1. In the active learning approach, a supervised classifier is initially trained on a small labeled data set (i.e., the so-called *seed*). The classifier is then used to select data point(s) from an unlabeled data *pool* and query the label(s) from the human expert. The human expert labels the selected point(s) and updates the training data set. The classifier is then retrained on the updated training data set, and the iterative process of data point selection and annotation is repeated until convergence or a specific performance threshold is achieved. Besides, in order to keep the workload of the human expert minimal, the active learning process can also be performed until the training data set reaches a specific size.

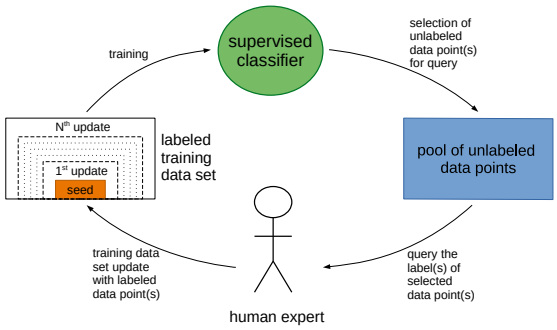

Fig. 1. Active learning.

One of the most commonly used strategies to select data points for labeling is *uncertainty sampling* which selects one point at a time based on an uncertainty measure [17]. With this strategy, the classifier predicts the labels for all points in the unlabeled pool, and selects the data point with the least certain prediction (i.e., the data sample for which the classifier has the least confident prediction).

There are several ways to measure the degree of uncertainty of the prediction for the data point x with the predicted label $\hat{y}$. One option is the *classification uncertainty* score which is calculated as

$$U(x) = 1 - P(\hat{y}|x), \tag{1}$$

and based on it, the data point with the highest score is selected.

When using the *classification margin* score, the two most likely labels $\hat{y}_1$ and $\hat{y}_2$ for the data point x are considered (note that $\hat{y} = \hat{y}_1$). This score is calculated as

$$M(x) = P(\hat{y}_1|x) - P(\hat{y}_2|x), \tag{2}$$

and the data point with the smallest score is selected (i.e., the data point with the smallest difference between the two most likely predictions).

Suppose that $\hat{y}_1, ..., \hat{y}_k$ are all possible labels (i.e., there are k classes) and $p_i = P(\hat{y}_i|x)$. Using the *classification entropy* score, the data point with the highest score is selected, and the entropy based score is calculated as follows:

$$E(x) = -\sum_{i=1}^{k} p_i * log(p_i) \tag{3}$$

Another commonly used strategy for selecting a data point for labeling is *query by committee* [17]. This strategy involves two or more independent classifiers (i.e., the so-called *committee*) that are trained on the same data set but modeling different hypotheses from the hypothesis space, with each classifier providing class predictions for all the unlabeled data points in the pool. Using this strategy, in order to select the data point for labeling, a *disagreement* score is calculated that reflects the degree of disagreement among the classifiers on the data point class.

In this regard, to calculate the *vote entropy* score, Eq. 3 is used with $p_i = m/n$, where m is the number of classifiers predicting the label $\hat{y}_i$ and n is the total number of classifiers. An alternative scoring measure is the *consensus entropy* score, for which Eq. 3 is used with p_i set to the average probability of predicting the label $\hat{y}_i$ by the classifiers in the committee. In other words, if $\theta^1, ..., \theta^n$ are classifiers in the committee, and $P_{\theta^j}(\hat{y}_i|x)$ is the probability that the data point x has the label $\hat{y}_i$ according to classifier θ^j, then $p_i = \frac{\sum_{j=1}^{n} P_{\theta^j}(\hat{y}_i|x)}{n}$ in Eq. 3. When using vote entropy and consensus entropy scores, the data point with the highest score is selected for labeling. The *maximum disagreement* score relies on the Kullback-Leibler divergence and is calculated as follows [17]:

$$\begin{aligned} KL &= \frac{1}{n}\sum_{j=1}^{n} D(P_{\theta^j}||P_C), \quad where: \\ D(P_{\theta^j}||P_C) &= \sum_{i=1}^{k} P_{\theta^j}(\hat{x}_i|x) * log\frac{P_{\theta^j}(\hat{x}_i|x)}{P_C(\hat{x}_i|x)}, \\ P_C(\hat{x}_i|x) &= \frac{1}{n}\sum_{j=1}^{n} P_{\theta^j}(\hat{x}_i|x) \end{aligned} \tag{4}$$

The data point with the highest maximum disagreement score is selected for labeling.

All the aforementioned methods have the limitation of selecting one data point at a time for annotation. However,

in some cases it might be desirable to select a larger batch of data points (e.g., 10 or 20), as that may allow the division of the labeling workload between several human experts. Such scenarios are common in SOC environments, where a group of security analysts are investigating NIDS alerts in parallel [21].

In order to address such scenarios, Cardoso et al. [18] proposed the *ranked batch-mode active learning* strategy. Suppose that L denotes the data points that have already been labeled and U the data points in the unlabeled pool of data. With the ranked batch-mode active learning strategy, a group of unlabeled data points (*batch*) are selected based on the following ranking function which assigns a relevance score to each data point x:

$$score(x) = \alpha * (1 - \Phi(x, L)) + (1 - \alpha) * U(x) \quad (5)$$

In Eq. 5, $U(x)$ denotes the uncertainty score for x, while $\Phi(x, L)$ denotes the highest similarity of x with some already labeled data point from L. Note that $0 \leq \Phi(x, L) \leq 1$, and $\alpha = \frac{|U|}{|L|+|U|}$.

Due to the nature of Eq. 5, α is close to 1 at the beginning of the active learning training cycle, and thus the similarity $\Phi(x, L)$ has a much larger impact on the data point selection process than the uncertainty score $U(x)$. Furthermore, according to Eq. 5, the data points that have a low similarity with points from L are preferred. In other words, in the early phases this strategy tries to select as many different data points as possible and explore unknown parts of the data point space. In the later stages of the active learning training cycle, the classification uncertainty measure $U(x)$ will be given more weight when points are selected and the similarity $\Phi(x, L)$ will become less important.

Cardoso et al. proposed several similarity measures, including traditional similarity functions that range from 0 to 1, but also distance metrics like Manhattan and Euclidean distance [18]. In the latter case, the distance metrics must be converted to similarity measures (e.g., distance of 0 yields a similarity of 1).

C. Related Work on Active Learning

As mentioned in Section I, active learning has not received the deserved attention in the context of NIDS alert classification. For the sake of completeness, this section provides an overview of some prominent active learning studies in other fields of network security.

Beaugnon et al. [19] proposed the use of active learning for the creation of representative labeled data sets with binary labels (i.e., *malicious* and *benign*) from large unlabeled data collections. For this purpose, the method used the LR classifier, and in order to ensure that all types of *malicious* data points were included in the labeled data set, the LR model was trained with sample weights. The proposed method was implemented in the ILAB system and used for labeling a large NetFlow data set from a production environment.

Torres et al. [20] suggested another active learning based methodology for labeling network traffic data sets that relies on the RF classification model. The authors used Stratosphere IPS encoding for representing network flows that included the information about the packet size and the duration and periodicity of packet exchange. The method was implemented in the RiskID system and tested on 22 publicly available network traffic data sets.

Guerra-Manzanares and Bahsi [21] studied the application of active learning to detect botnet traffic in IoT networks. For the experiments, a balanced data set of 150,000 data points with binary labels was used. The authors investigated all major active learning strategies described in Section II-B, using RF classifiers. Accoding to their results, active learning based algorithms achieved high performance with small labeled data sets and were more resilient to labeling mistakes by human experts than fully supervised models trained on larger data sets.

While the aforementioned papers focused on network traffic data, the work by Kawaguchi et al. [22] addressed NIDS signature classification (NIDS signatures are not to be confused with NIDS alerts, since signatures are human-created rules that generate alerts). Signatures might have different importance in SOC environments, and the purpose of the proposed method was to establish the importance level (i.e., *low*, *medium* or *high*) for new signatures. The method focused on some textual fields of NIDS signatures and converted them to numeric vectors. During the experiments, several neural network types were evaluated in active learning setup. According to the authors, a system with Monte Carlo dropout (i.e., a deep learning based uncertainty estimation method) produced the best results.

The systematization of knowledge by Appruzzese et al. [34] reviewed semi-supervised machine learning work in the domains of network intrusion detection, malware detection, and phishing web site detection. In addition, the work defined several requirements for evaluating semi-supervised machine learning methods which are summarized next. In the context of active learning, evaluations should present the performance of the classifier that was trained on the seed only, as it constitutes the lower bound for performance (henceforth denoted as μ_1). Also, the performance of the classifier that was trained on the entire labeled training data set constitutes the upper bound for performance (henceforth denoted as μ_2). Note that during evaluation the seed and pool are drawn from the entire labeled training data set. Therefore, following [34], if μ is the performance of the active learning model, then the following condition should be met:

$$\mu_1 < \mu \leq \mu_2 \quad (6)$$

In other words, the active learning cycle should improve the initial performance of the classifier (μ_1) while generally the final performance of the active learning model does not exceed the performance of the fully supervised model (μ_2).

In addition, if we define μ_3 as the performance of the active learning model where data points are selected for labeling using random sampling, then the following condition should be met:

$$\mu_3 < \mu \tag{7}$$

Consequently, if Eq. 7 does not hold, the usage of the classifier to select data points for labeling can be deemed as irrelevant.

Other evaluation requirements described by [34] include the assessment of the statistical significance of the evaluation results, provide information about the composition of evaluation data sets, ensure that experiments are reproducible (e.g., by sharing the data sets and experiment code), and conduct evaluations using multiple settings. All these requirements are satisfied in our work and presented throughout the following sections.

III. Methodology

A. Data Set

The NIDS alert data set that was used in our experimental setup was obtained from a production Suricata NIDS in the SOC of Tallinn University of Technology [4]. At the data collection time, the NIDS had more than 50,000 signatures and was running on the external organizational network perimeter. The data set was collected between January 20 and March 20, 2022 (60 days). It contains 1,395,324 data points with human-assigned binary labels *important* and *irrelevant* which denote the priority level of the NIDS alert. The data set is highly imbalanced, as only 20,952 data points (1.5% of data) are labeled as *important*. The data set is publicly available from [23].

To generate data points from textual NIDS alert data, a customized version of the SCAS stream clustering algorithm (henceforth CSCAS) was utilized [24]. CSCAS is essentially the SCAS algorithm [5] augmented with data point generation functionality. Data point features are outlined in Table I. In this regard, while data point features were generated by CSCAS, the data point labels were set by human experts.

CSCAS processes NIDS alerts in a real-time fashion, aggregating alerts triggered by the same external attacking host and the same signature into the same *alert group*. Each alert group contains alerts from a short time frame (i.e., max 5 minutes). Longer attack activity produces several alert groups.

After producing an alert group, CSCAS assigns it into a cluster, creating a new cluster if necessary. Each new CSCAS cluster represents a NIDS signature that has matched frequently over a longer period of time (e.g., at least once every hour during the last 10 days). According to previous research [5], [8], [14], [25], such signatures are highly likely to produce well-known alerts of low importance and false positives.

If the alert group *G* was triggered by the signature *S* for which there is a cluster, CSCAS regards *G* as *inlier* and assigns it to the cluster that represents *S*, otherwise *G* is regarded as *outlier* and assigned to the special *cluster of outliers*. Also, in order to adjust to environment changes, CSCAS drops a cluster if the signature it represents is no longer frequently matching.

TABLE I
NIDS Alert Data Point Features

Feature name and type	Description
SignatureID	numeric ID of the signature
SignatureMatchesPerDay	average number of matches per day by the signature *SignatureID*
AlertCount	number of alerts represented by the current data point
Proto	numeric transport protocol ID (e.g., 6 denotes TCP and 17 denotes UDP)
ExtIP	IP address of the attacker in integer format (IP address A.B.C.D is represented as: $A * 256^3 + B * 256^2 + C * 256 + D$)
ExtPort	port number at the attacking host (set to -1 if attacks involved multiple ports)
IntIP	IP address of the victim in integer format (set to -1 if the attacker *ExtIP* targeted multiple victim hosts)
IntPort	port number at victim host(s) (set to -1 if attacks targeted multiple ports)
⟨*Attr*⟩*Similarity* (34 features in total)	similarity for the NIDS alert attribute ⟨*Attr*⟩. Set to -1 if the signature *signatureID* does not set ⟨*Attr*⟩, otherwise ranges from 0 to 1. The feature reflects the degree of similarity for ⟨*Attr*⟩ between the current data point and recent data points from the same cluster.
Similarity	overall similarity of the current data point with recent data points from the same cluster (calculated as an average of ⟨*Attr*⟩*Similarity* values that are not set to -1)
OutlierIndicator	set to 1 if data point is an outlier, and set to 0 if data point is an inlier

At the time of data set generation which lasted for 60 days, the number of clusters fluctuated between 40 and 43. CSCAS detected 1,322,652 inliers (94.8% of data) that were generated by 44 signatures and 72,672 outliers (5.2% of data) from 547 signatures. In the case of inliers, 77.9% were generated by just 5 signatures which detect well-known low-priority attacks such as scans for old vulnerabilities. In other words, most NIDS alerts have low importance and most of them are generated by a small number of prolific signatures (the same observation has been made by previous works in the research domain [5], [8], [14], [25]).

Since CSCAS clusters correspond to such signatures, inliers usually represent NIDS alerts of low importance. For example, in the NIDS alert data set, only 195 inliers (i.e., 0.01% from all 1,322,652 inliers) are labeled as *important*. Also, the set of outliers contains the vast majority of *important* data points (20,757 out of 20,952, i.e., 99.1%). On the other hand, many outliers are found *irrelevant* (51,915 out of 72,672, i.e., 71.4%).

When the alert group is assigned to a cluster (or the cluster of outliers), CSCAS generates the data point representative for the alert group, using the information about the cluster members seen in the past (Table I describes the data point features).

While most features are numerical (e.g., ⟨*Attr*⟩*Similarity*), the data point has also some categorical features (e.g., *Proto*, *ExtIP*, and *ExtPort*).

From the generated features which are described in Table I, it is worth highlighting that ⟨*Attr*⟩*Similarity* features report a measure of a frequency that NIDS alert attribute values have been observed in the past. For example, if the *ExtIPSimilarity* feature is set to 0.5, then 50% of the most recently seen alert groups from the given cluster originated from the same attacking host as the current alert group. For the sake of brevity, we defer to the original paper for a thorough description of the algorithm inner-working [5].

Besides, some ⟨*Attr*⟩*Similarity* features might not be relevant for the given alert group and are thus set to -1 in the generated data point. For instance, if the alert group has been triggered by the signature that matches SMTP traffic, then HTTP related similarity features like *HttpUserAgentSimilarity* and *HttpUrlSimilarity* are meaningless (i.e., not applicable) and are thus set to -1.

B. Experimental Setup

In order to benchmark different active learning methods, we divided the NIDS alert data set into training and test data sets as described in Table II. The training data set was composed of all alerts from the first 12 days (January 20 – January 31, 2022), corresponding to 20% of the whole data set. The remaining data was included in the test data set, thus involving alerts from the following 48 days (February 1 – March 20, 2022) and corresponding to 80% of the collected data set.

TABLE II
NIDS ALERT DATA SET

	Training data	**Test data**
Time	Jan 20 – Jan 31 (12 days)	Feb 1 – Mar 20 (48 days)
# of data points	278,385 (20.0% of data)	1,116,939 (80.0% of data)
# of important data points	3,227 (1.2% of training data)	17,725 (1.6% of test data)
# of irrelevant data points	275,158 (98.8% of training data)	1,099,214 (98.4% of test data)
# of signatures	239	563

The chronological arrangement mimics the NIDS alert handling in production environments, where human analysts use past alert data to train classification models without any knowledge about the nature of future alerts. Also, this arrangement allows the evaluation of the suitability of a machine learning model to handle new threats emerging over time [26]. For example, as Table II illustrates, alerts in the training data set were generated by 239 signatures, while the test data set contained alerts from 563 signatures (i.e., more than half of these 563 signatures did not trigger any alerts during the 12 days that the training data covered). Therefore, as the training data originates from an earlier time frame, it becomes possible to evaluate on test data how accurately NIDS alerts from previously unseen signatures are classified. This realistic scenario is usually not addressed in the experimental design of related work, which use the standard machine learning procedure to split the dataset (i.e., random training/test split), and, consequently, neglect the impact of time and evolution of threats, reporting over-inflated and unrealistic results provoked by temporal snooping [27].

For all active learning experiments, we employed the *modAL* active learning library [28]. The supervised classification model used in the active learning training cycle (see Fig. 1) was implemented using the *scikit-learn* default class for RF with 100 estimators [29]. RF was chosen as classification model since it can easily handle data sets that contain categorical features. Also, based on recent studies, it has demonstrated the best performance on security data sets in active learning implementations [20], [21]. For the sake of repeatability, we make publicly available the code used for our experiments [1].

As can be seen from Table II, the NIDS alert data are highly imbalanced. To address this issue, for all experiments where training data was not balanced, we trained RF with corresponding class weights inversely proportional to class frequencies (this training mode was implemented using the *class_weight=balanced* setting of the class object [29]). To assess the performance of RF classifiers, we evaluated their generalization capabilities on the test data set (described in Table II), using F1-score as a comparative performance metric. The F1-score is the harmonic mean of the precision P and recall R, and is calculated as follows:

$$F1 = 2 * \frac{P * R}{P + R} \tag{8}$$

The nature of the F1-score allowed us to report the performance of an RF classifier with a single metric, leading to a more compact presentation of the experiment results (in some cases, we also reported the precision and recall for the sake of a more detailed comparison). Because all experiments described in this paper involved stochastic processes like random sampling, every particular experiment was repeated 10 times, and all F1-scores reported in this study are average values of 10 iterations performed per individual experiment (i.e., particular settings). All experiments were conducted on a Rocky Linux 8 server with the Intel Xeon E5-2630Lv2 CPU and 64GB of memory.

IV. RESULTS

A. Baseline Performance

In order to establish a performance baseline before evaluating active learning methods, we trained RF in a traditional supervised learning fashion, using the whole labeled training data set (described by the middle column in Table II). Note that this baseline represents the upper bound for the performance of active learning models (i.e., μ_2 in Section II-C, Eq. 6).

Since the training data was imbalanced, we trained the RF model with class weights as discussed before. In addition, we also trained a regular RF model (i.e., without employing the

[1] https://github.com/ristov/nids-al-scripts

class weights) on balanced data sets obtained with random undersampling and oversampling balancing techniques. With random undersampling, the majority class *irrelevant* was reduced to 3,227 data points to achieve a balanced data set, whereas with random oversampling, instances of the minority class *important* were replicated for having the balanced data set of 275,158 data points per class. Table III presents the performance of the above three methods in terms of F1-score.

TABLE III
SUPERVISED LEARNING: DATASET AND PERFORMANCE

	RF with class weights	**RF with oversampling**	**RF with undersampling**
Training set	278,385	550,316	6,454
F1-score	0.915	0.927	0.855

According to Table III, RF with random undersampling featured the lowest F1-score of 0.855. This result illustrates the fact that learning from smaller labeled data sets can be challenging, and that undersampling might exclude important knowledge from the original training data. On the other hand, the F1-scores of RF with class weights and oversampling are similar, exceeding 0.91.

B. Uncertainty Sampling: Fixed Seed and Varied Pool Sizes

The first active learning method we evaluated was uncertainty sampling (henceforth US). We used a seed (initial labeled training data set) of 100 labeled data points and experimented with varied pool sizes of [1,000, 2,000, ..., 10,000] unlabeled data points. Both the seed and the pool were randomly sampled from the training data set (see the middle column in Table II).

During the active learning based training experiment, up to 1,000 data points were selected from the pool and presented to the human expert as a labeling query. For selecting a data point to query, the classification uncertainty score was used (as described in Section II-B, Eq. 1). The labeling work of the human expert was simulated by using the ground-truth labels present in the data set. Fig. 2 depicts the performance of 10 active learning training cycles with different pool sizes.

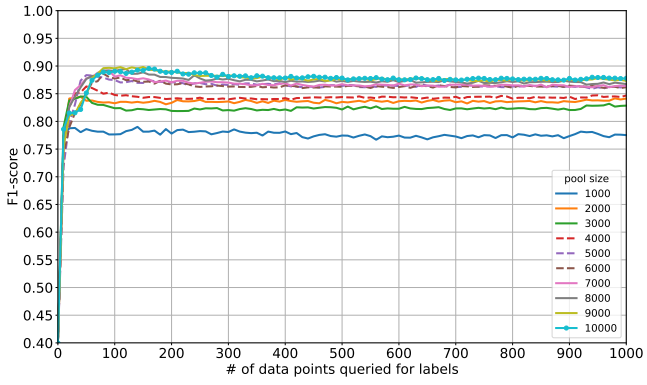

Fig. 2. F1-score of US with random sampling based seed and pool

In addition, we conducted the same experiments using active learning with random sampling strategy, i.e. the data points for labeling were selected at random from the pool (note that the performance of this model represents μ_3 as defined in Section II-C, Eq. 7). The results of these experiments are displayed in Fig. 3.

As the comparison of Fig. 2 and Fig. 3 indicates, US based data point selection yielded better performance than random sampling based data point selection – the performance of US models improved faster and featured significantly higher F1-scores. In other words, traditional active learning, where a classifier is leveraged to select data points for labeling, offered clear performance advantage over random data point selection.

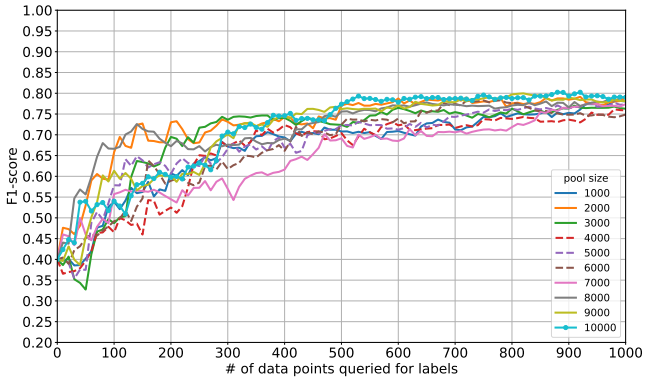

Fig. 3. F1-score of random sampling based data point selection

As Fig. 2 shows, initial F1-scores of US models (i.e., trained only on the seed data set) were around 0.4. Note that these F1-scores correspond to the lower performance bound, μ_1 as defined in Section II-C, Eq. 6. However, as data points were selected from the pool, queried for labels, and added to the training data set, the performance quickly improved. All models reached their peak performance with less than 100 queried data points and no improvement was observed with additional queries.

Based on Fig. 2, the largest pool, composed of 10,000 data points, yielded the best performance. Fig. 2 also indicates that increasing the pool size was usually providing a better final performance. Even though US allowed us to train a supervised classifier with a moderate labeling effort (e.g., the models involved the labeling of at most 1,100 data points), the final performance of the models in Fig. 2 was inferior to the baseline performance (see Table III). In addition, when we evaluated US with classification margin (Section II-B, Eq. 2) and classification entropy scores (Section II-B, Eq. 3), the observed performance was similar as in Fig. 2.

Therefore, despite enabling fast learning with a significantly smaller training data set, this standard active learning strategy did not outperform the baseline. This observation raises the following question – can the performance of active learning models *for NIDS alert classification be improved beyond the upper bound that semi-supervised methods generally have* (μ_2

in Section II-C, Eq. 6)? The following section proposes a method for addressing that question.

C. Enhancing Active Learning with Outlier-Centric Data Sets

When investigating the performance of the active learning models described in the previous section, we also studied the precision and recall of the models (see Fig. 4 and Fig. 5).

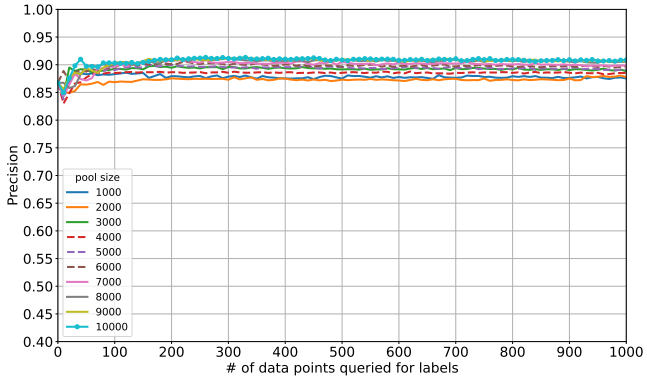

Fig. 4. Precision of US with random sampling based seed and pool

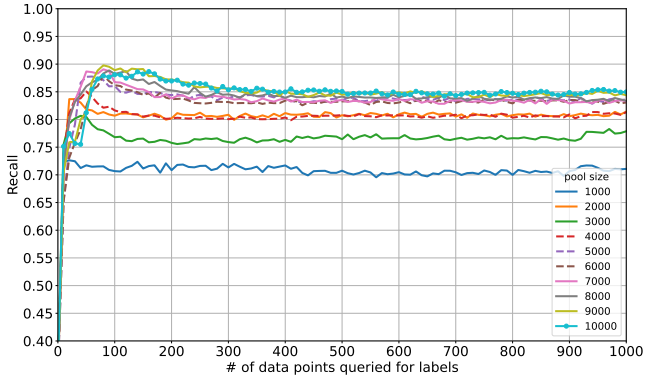

Fig. 5. Recall of US with random sampling based seed and pool

As shown in Fig. 4, all models featured a final precision within the range 0.87–0.91. In contrast, recall values were significantly lower, with the smallest pool yielding a final recall value of about 0.7 (see Fig. 5).

Since low recall indicates that a significant proportion of *important* data points are misclassified, we hypothesized that seed and pool data sets might not have included enough *important* data points for the model to learn how to discriminate them effectively. This hypothesis is also supported by Fig. 2 – the initial performance of the models is low, and classification performance plateaus around 100 queries in all cases.

As a strategy to improve the performance of active learning models so that they overcome fully supervised learning models, this paper proposes a method to create *outlier-centric* data sets for seeds and pools. The proposed method relies on the following observation – as discussed in Section III-A, a data point that is an inlier is highly likely to belong to the *irrelevant* class. While outliers do not belong predominantly to a particular class (*irrelevant* or *important*), the set of outliers contains the vast majority of *important* data points. Therefore, increasing the proportion of outliers in the training data set would also increase the proportion of *important* data points, thus providing a more representative and larger sample of data points to the classification model to learn how to discriminate this class effectively.

In contrast, using random sampling to generate the seed and pool yields data sets where the proportion of *important* data points is the same as in the original data set. Since *important* data points constitute only 1.2% of the original training data (see the middle column in Table II), random sampling based seeds and pools are unlikely to include enough *important* data points to induce high-performance active learning models.

Given the above considerations, we propose the following method, called *Outlier-N*, to generate the seed and pool data sets. Let $N \in \mathbb{R}$ and $N \in (0, 100]$, where N reflects the desired proportion of outliers in the seed and pool data sets. In order to generate the data set, *Outlier-N* takes the following two steps:

- use random sampling to select $N\%$ data points from the set of outliers,
- use random sampling to select the remaining $(100-N)\%$ data points from the set of inliers.

Note that using higher N values allows for making seeds and pools *outlier-centric*. Also, the *Outlier-N* method does not require the application of an outlier detection algorithm to the training data set, because the *OutlierIndicator* feature reflects whether a data point is an outlier or inlier (see Table I). Therefore, the *Outlier-N* method has a low computational cost.

The performance of the suggested methodology, *Outlier-N*, to build seeds and pools is evaluated in the following section.

D. Active Learning with Outlier-N Method

This section explores the performance of *Outlier-N* for active learning strategies described in section II-B: US (uncertainty sampling), query by committee, and ranked batch-mode active learning.

1) Uncertainty Sampling with Outlier-N method:

In Section IV-B, we evaluated active learning for seeds and pools created with traditional random sampling based method, using US to select data points for labeling queries. To illustrate the benefits of the *Outlier-N* method for creating seeds and pools, we have evaluated it for US based active learning as described in Section IV-B. For the sake of conciseness and interpretability of the results, only the results for pool sizes of 1,000, 2,000, 5,000, and 10,000 data points are reported. The results for other pool sizes were close to these and have been thus omitted. The selection of unlabeled points from the pool was based on the classification uncertainty score (as described in Section II-B, Eq. 1). We tested the *Outlier-N* method for

data sets generation using the following values of N: 30, 50, 70, and 90. The seed and pool for each iteration were built with the same value of N.

Fig. 6 depicts the performance of US active learning with the *Outlier-N* method. Note that the results for classification margin (Section II-B, Eq. 2) and classification entropy (Section II-B, Eq. 3) scoring strategies closely matched the results reported in Fig. 6 and have been omitted for the sake of brevity.

As the comparison of Fig. 2 and Fig. 6 reveals, *Outlier-N* provided better performance than random sampling for all evaluated pool sizes. For example, the smallest pool size provided F1-scores in the range from 0.85 to 0.9 for *Outlier-N*, while with random sampling F1-score values remained below 0.8. For the largest pool size, the *Outlier-70* and *Outlier-90* methods yielded F1-scores of 0.95 and higher, while for random sampling the F1-score remained below 0.9.

Furthermore, the *Outlier-N* method provided a significantly higher recall than random sampling. For example, as the comparison of Fig. 5 and Fig. 7 shows, the final recall improved from 0.7 to 0.87 with *Outlier-90* for the smallest pool size and from 0.85 to 0.92 for the largest pool size. The *Outlier-N* method also yielded higher precision than random sampling for all pool sizes.

More importantly, unlike random sampling, the usage of *Outlier-N* to build seeds and pools allowed the active learning strategy to outperform traditional supervised classification models (see Table III). For the pool size of 10,000, *Outlier-N* provided better performance for all evaluated values of N, while for the pool size of 5,000, *Outlier-50*, *Outlier-70*, and *Outlier-90* produced a better performance than supervised classifiers.

In relation to seed generation, *Outlier-N* based seeds provided better initial performance for the US active learning strategy than random sampling based seeds, especially for larger values of N. The initial F1-score for random sampling based seed was around 0.4, while for *Outlier-50*, *Outlier-70*, and *Outlier-90* it was above 0.75, with *Outlier-90* producing the best results. However, although *Outlier-N* based seeds offered significantly better initial performance, the active learning strategy reached its peak performance more slowly in the case of larger pools. For example, according to Fig. 2, the highest performance was achieved with less than 100 queried data points, while with *Outlier-N* 300-500 queried data points were needed for pool sizes of 5,000 and 10,000. On the other hand, although with smaller pool sizes of 1,000 and 2,000 the highest performance was reached faster, the performance started to degrade after the peak.

Similar to random sampling based pools (Fig. 2), larger *Outlier-N* based pools usually yielded better performance than smaller pools. The only deviations from this trend occurred at early stages of the active learning training cycle, where F1-scores for smaller pool sizes were occasionally slighty higher.

Finally, as can be observed in Fig. 6, increasing the value of N (i.e., the proportion of outliers in the seed and pool) had a direct impact on the performance, increasing it. Larger values of N also decreased the performance gap between different pool sizes (e.g., performance difference for pool sizes of 5,000 and 10,000 was smallest for $N = 90$).

To provide a full coverage of the impact of the hyper-parameter N, we also evaluated the *Outlier-100* method which builds seeds and pools solely from outliers. Fig. 8 depicts the performance of this method. As can be noticed in Fig. 8, US models were not stable and featured significantly lower performance when compared to data from Fig. 6. The primary reason for this was the low precision of the models, with a large number of *irrelevant* data points being mistakenly classified as *important*. As Fig. 8 indicates, the presence of inliers is important in seeds and pools, since it allows for proper detection of *irrelevant* data points (as discussed in Section III-A, the vast majority of inliers belong to this class).

Since the labeling work performed by human experts is expensive, we also investigated a scenario where the initial annotation effort was minimal, and a small seed of only 10 data points was used. As in the case of seeds of 100, *Outlier-90* offered the best final performance (see Fig. 9). The performance results for *Outlier-30*, *Outlier-50*, and *Outlier-70* methods showed similar trends, and have been omitted for the sake of brevity.

As can be observed in Fig. 9, the initial performance for the 10 point seed was very low, with the F1-score value being about 0.2 for all models. However, with fewer than 50 queried data points, the performance rapidly improved. Interestingly, the model with the largest pool size of 10,000 performed the worst when fewer than 200 data points had been queried, but became the best model after 400 queries. When comparing this model to the model with the 100 point seed and 10,000 point pool (Fig. 6), it is worth to note that the model with a smaller seed reached the F1-score of 0.95 quicker (after 400 queried data points) than the model with a larger seed which achieved this performance after 500 queried data points. In addition, although the peak performance of the models was comparable, the model with a smaller seed occasionally featured a higher F1-score (e.g., after 800 queried data points).

To summarize the main findings for the US based active learning strategy with the *Outlier-N* method, the results demonstrated that larger values of N, which increase the proportion of outliers in the seed and pool, yielded the best peak performance. Larger pools yielded better peak performance than smaller pools, and also outperformed traditional supervised classifiers (see Table III). Despite that more queried data points were needed to reach the peak with larger pools, the performance of these models did not degrade significantly after the peak. Finally, the use of a small seed (e.g., 10 data points) offered a fast learning curve and a similar peak performance to a larger seed (e.g., 100 data points), but provided lower performance in the early stages of the active learning training cycle.

To assess the computational cost associated with the training of US based models, we measured the CPU time that was spent to train these models with the *Outlier-90* method. For four pool sizes evaluated in this section, the CPU time consumed by the entire active learning cycle ranged from 249 to 418

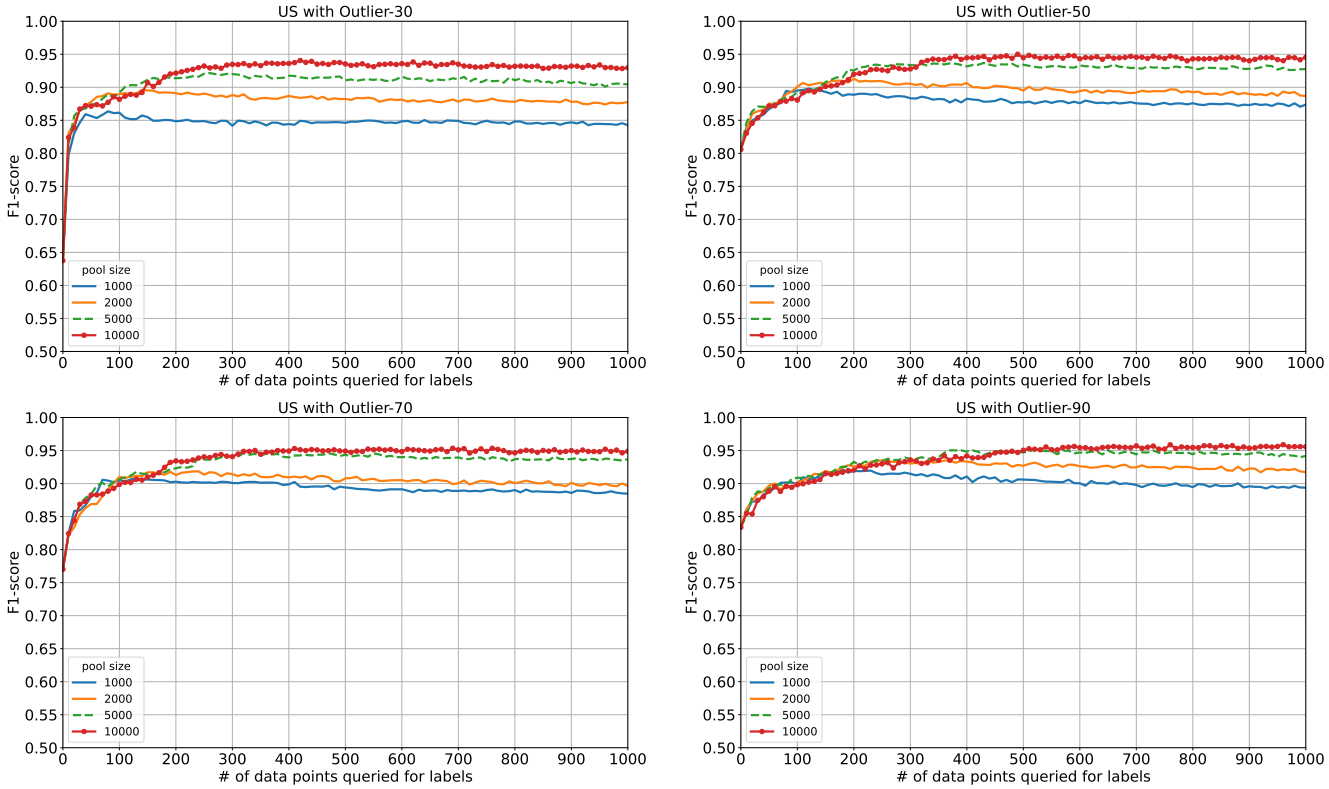

Fig. 6. F1-score of US with *Outlier-N*.

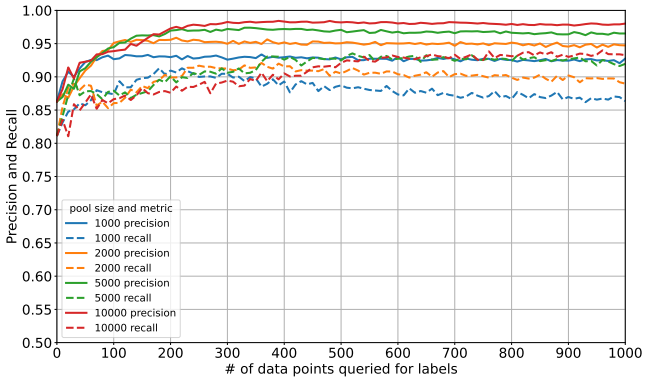

Fig. 7. Precision and Recall of US with *Outlier-90*.

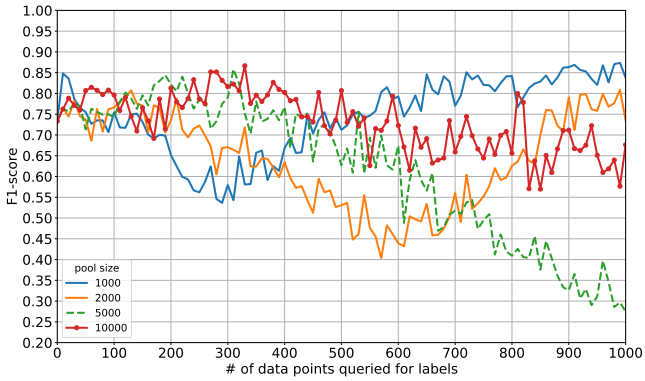

Fig. 8. F1-score of US with *Outlier-100*.

seconds (about 4-7 minutes). Smaller pools required less CPU time, since the RF classifier had less data points to process for uncertainty sampling.

We also measured the CPU time that was spent by the models to classify the test data set (see right column in Table II). In all cases, less than 10 seconds of CPU time were needed. These findings indicate that US based models with RF classifiers involve low computational costs and can be used on commodity hardware.

Since the active learning models discussed in the remainder of this section showed similar CPU time for test data classification (less than 10 seconds), only the computational cost of training is discussed in the following subsections.

2) Query by Committee with Outlier-N Method:

We also evaluated the impact of the *Outlier-N* method on the query by committee (henceforth QbC) based active learning. In this active learning setup, a committee, which consists of

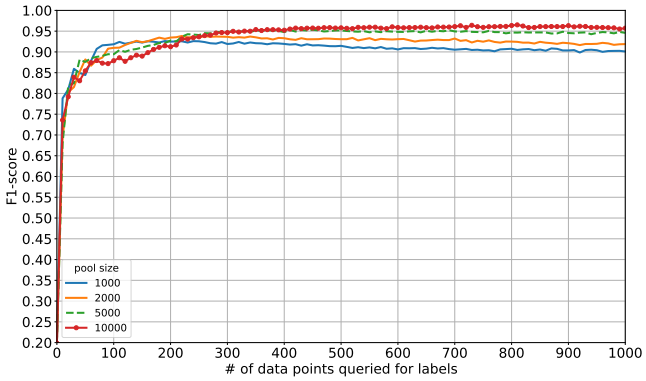

Fig. 9. F1-score of US with small seed and *Outlier-90*

two or more independent classifiers, is employed to select data points from the unlabeled pool for labeling queries. The query point selection is based on a score that reflects the degree of disagreement of the committee members on the data point class (i.e., a measure of label uncertainty).

For these experiments, we evaluated committee sizes of 2, 3, 5, and 10, with each committee member being an independent RF classifier. For query data point selection, QbC models with three different disagreement scores (discussed in Section II-B) were evaluated – consensus entropy score (henceforth QbC-CE), vote entropy score (henceforth QbC-VE), and maximum disagreement score (henceforth QbC-MD). As in the previous experiments the largest pool yielded the best performance results, we used a fixed pool of 10,000 data points. The seed size was set to 100.

Table IV displays the CPU time needed to complete the whole training for different QbC models. As can be seen, larger committees needed more computational resources, and the highest recorded CPU time consumption was about 95 minutes (observed for QbC-CE with 10 committee members). In contrast, the smallest committees, composed of 2 members, consumed only about 12-18 minutes of CPU time.

TABLE IV
TRAINING CPU TIME FOR QBC METHODS (SECONDS)

Committee size	QbC-CE	QbC-VE	QbC-MD
2	1,084	812	718
3	1,160	1,387	1,341
5	2,330	2,281	2,000
10	5,696	5,022	3,649

Fig. 10 displays the performance of evaluated models. Since the *Outlier-N* method yielded better performance for larger values of *N*, we have only reported data for *Outlier-30* and *Outlier-90* in Fig. 10 to compare the methods with the lowest and highest performance. In this regard, as Fig. 10 shows, *Outlier-90* method outperformed *Outlier-30* in all models, producing a better initial performance and reaching a higher peak performance.

Of the three evaluated QbC scoring metrics, QbC-CE (the first row in Fig. 10) featured the best peak performance, both for *Outlier-30* and *Outlier-90*. For *Outlier-90*, the F1-score value was the higher and more stable in the later stages of the active learning training cycle, with all four models converging to an F1-score value of 0.96.

In the case of QbC-CE, the size of the committee did not notably influence the performance (apart from a few minor differences in the earlier stages of the active learning training cycle). Since training a model with a larger committee requires more computational resources, QbC-CE allows for building low-cost active learning models that feature high performance.

For QbC-VE (the second row in Fig. 10), the size of the committee had a slight impact on performance, especially in the case of the *Outlier-30* method where F1-scores could differ by 2%. For *Outlier-90*, performance differences were more noticeable in the early stages, with fewer than 100 queried data points, with the largest committee clearly outperforming smaller configurations. However, the performance of QbC-VE models started to slowly degrade after a few hundred queries. For example, the best performance for QbC-VE (i.e., F1-score of 0.95) was produced by *Outlier-90* after about 300 queries. After that, the F1-score value slowly decreased by about 2-3%.

Regarding the QbC-MD models (the third row in Fig. 10), the committee size had a clear implact on the model performance, with larger committees providing better performance metrics. However, the difference was less noticeable for *Outlier-90* with committee sizes of 5 and 10. Although the best performance for QbC-MD models (i.e., F1-score of 0.94) was lower than observed for QbC-CE and QbC-VE, the models featured a monotonously increasing curve for F1-score (that stands in contrast with results for QbC-VE models).

To summarize the main findings for QbC based active learning with the *Outlier-N* methodology, the best performance was produced by *Outlier-90* with peak F1-scores of 0.96, 0.95, and 0.94 for QbC-CE, QbC-VE, and QbC-MD respectively. This implies that QbC based active learning with the *Outlier-90* method outperformed traditional supervised classifiers (Table III) for all disagreement scores. QbC-CE models demonstrated high F1-scores for small committees, with all evaluated committee sizes producing almost identical performance. This brings an opportunity for building cost-efficient and high-performance active learning models with small committees. In addition to providing the highest F1-scores, *Outlier-90* based QbC-CE models also featured a high degree of stability in the later stages of the active learning training cycle.

3) Ranked Batch-Mode Active Learning with Outlier-N Method:

In our last round of experiments, we evaluated the impact of the *Outlier-N* method on the ranked batch-mode active learning strategy (henceforth RBMAL). As for the QbC models, we used a fixed seed of 100 data points and a fixed pool of 10,000 data points. We conducted experiments using three different batch sizes – 10, 20, and 50 data points. In other

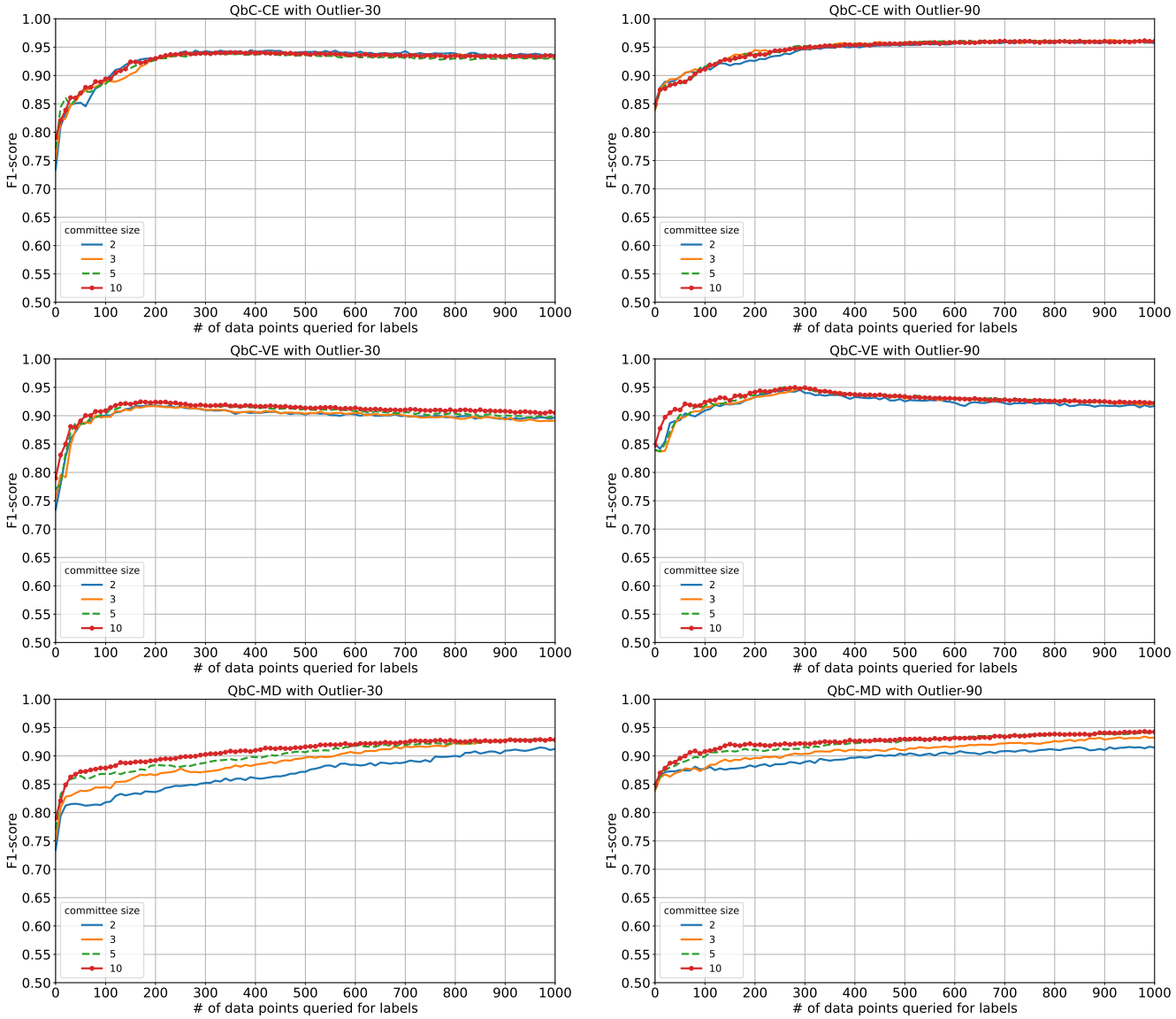

Fig. 10. F1-score of QbC with *Outlier-30* and *Outlier-90*.

words, for each labeling query the given number of unlabeled data points were selected from the pool, that is, a *batch* of data points instead of a single data point as in the previous strategies.

The *modAL* active learning library used during the experiments allowed us to specify a distance function to measure the similarity between data points. As in our case the data points have a significant number of categorical features, traditional distance metrics like Euclidean or Manhattan distance are not meaningful for measuring the similarity between data points. We experimented with three custom distance functions instead.

As discussed in Section II-B, the purpose of the distance function is to allow the effective sampling of unexplored regions of the data point space in the early stages of the active learning training cycle. In this regard, let's suppose that x and y are data points. Also, suppose that $sid(x)$ and $sim(x)$ denote the values of the *SignatureID* and *Similarity* features for data point x respectively. Provided that a distance between data points x and y needs to be calculated, all three custom distance functions we used for experiments return values in the range [0,1] and are defined as follows:

$$f_{const}(x, y) = 0 \tag{9}$$

$$f_{sigid}(x, y) = \begin{cases} 0, & if \quad sid(x) = sid(y) \\ 1, & if \quad sid(x) \neq sid(y) \end{cases} \tag{10}$$

$$f_{sim}(x, y) = |sim(x) - sim(y)| \tag{11}$$

The function f_{const} regards all points entirely similar (distance 0 is converted to similarity 1), and thus only uncertainty score is considered when selecting data points from the pool with the ranking function (as described in Section II-B, Eq. 5).

The function f_{sigid} regards points for the same signature ID entirely similar, and the ranking function thus prefers to include data points for as many signatures as possible in the training data set. In other words, since each NIDS signature represents a different attack scenario, f_{sigid} tries to cover as many scenarios as possible, so that selecting such diverse training data would lead to a high-performance classification model.

The *Similarity* feature of a data point reflects its similarity with other data points from the same cluster. As discussed in Section III-A, all outliers are assigned to a special cluster of outliers. However, as outliers are usually dissimilar to each other, the *Similarity* feature tends to have a lower value for outliers. In contrast, inliers are usually similar to other cluster members, and therefore the *Similarity* feature tends to have a higher value for inliers. Given the nature of the *Similarity* feature, f_{sim} will try to cover data points with a varying degree of similarity to other cluster members. In that way, f_{sim} will make sure that the training data set includes unusual inliers and outliers (i.e., inliers which are dissimilar and outliers which are similar to other cluster members).

Table V displays the CPU time for training different RBMAL models during the active learning cycle. As discussed in [18], building the batch involves calculating pair-wise distances between labeled data points in the training data set and unlabeled data points in the pool. Since this step is computationally expensive, RBMAL models needed more CPU time for training than QbC and US models. Besides, as Table V shows, the complexity of the distance function had a direct impact on time consumption. The simplest function, f_{const}, needed the least amount of CPU time (about 86-96 minutes), while f_{sim} required about 161-169 minutes to complete the whole training.

TABLE V
TRAINING CPU TIME FOR RBMAL MODELS (SECONDS)

Batch size	f_{const}	f_{sigid}	f_{sim}
10	5,189	8,144	9,641
20	5,761	8,402	10,130
50	5,501	7,528	9,880

Fig. 11 displays the performance of the RBMAL strategy using the aforementioned distance functions for the *Outlier-30* and *Outlier-90* methods. Note that as batch sizes of 10 and 20 samples featured similar performance, we have omitted the results for batch size 20 from Fig. 11 for the sake of brevity. *Outlier-90* produced a better performance than *Outlier-30*, featuring a significantly higher initial F1-score of over 0.8, while for *Outlier-30* the initial F1-score remained below 0.65. Besides, *Outlier-90* reached a higher peak F1-score (about 0.95), outperforming *Outlier-30* by about 2%. According to Fig. 11 and Table III, RBMAL models with the *Outlier-90* method reached a better peak performance than traditional supervised classifiers, while RBMAL with *Outlier-30* offered a similar performance to the best supervised classifier.

As Fig. 11 indicates, the batch size did not influence the peak performance of RBMAL which was typically achieved during the later stages of active learning. On the other hand, the largest batch size of 50 tended to yield a lower performance than smaller batch sizes during the early stages of the active learning training cycle. For example, with *Outlier-30*, the F1-score for the batch size of 50 improved more slowly when the first 100 data points were queried for labels. For *Outlier-90* and batch size of 50, the F1-score for the f_{const} distance function dropped suddenly after 150 queried data points to a lower value than for other batch sizes. Also, the F1-score for the f_{sim} distance function and *Outlier-90* was the lowest for the batch size of 50 between 200 and 300 queried data points. These observations suggest that larger batch sizes can lead to longer learning curves for RBMAL models, thus making the training less efficient.

When comparing the performance for the evaluated distance functions, we observed some differences for the earlier stages of the active learning training cycle, while in later stages the performance for the three distance functions was almost identical (Fig. 11). The f_{const} distance function performed well, despite that it only considered uncertainty score for selecting the query data points. Except for the largest batch size of 50, f_{const} offered the best overall performance. As for other distance functions, f_{sigid} tended to offer a slightly better performance than f_{sim}. The f_{sigid} function was particularly well suited for the batch size of 50, outperforming other distance functions in the earlier stages of the active learning training cycle.

Since identifying the right distance function for a data set with categorical features is not always an easy task, these findings illustrate that for NIDS alert data a trivial distance function like f_{const} which only considers classification uncertainty, can yield RBMAL models with high F1-scores. Furthermore, for larger batch sizes, similar peak F1-scores can be achieved by a simple distance function like f_{sigid} which attempts to include data points from as many different signatures as possible in the training data set. Finally, using simple distance functions can help reduce the computational cost of training RBMAL models.

V. DISCUSSION

This section summarizes the experimental results presented in Section IV and highlights our main findings, comparing the performance of different active learning techniques discussed in Section II-B.

Based on the experimental results reported in Section IV-A, specific traditional supervised learning algorithms (i.e., RF with class weights and RF with oversampling) can achieve high F1-scores on a NIDS alert data set ranging from 0.91 to 0.93 (Table III). The evaluation of the US active learning models with seeds and pools that are built using random

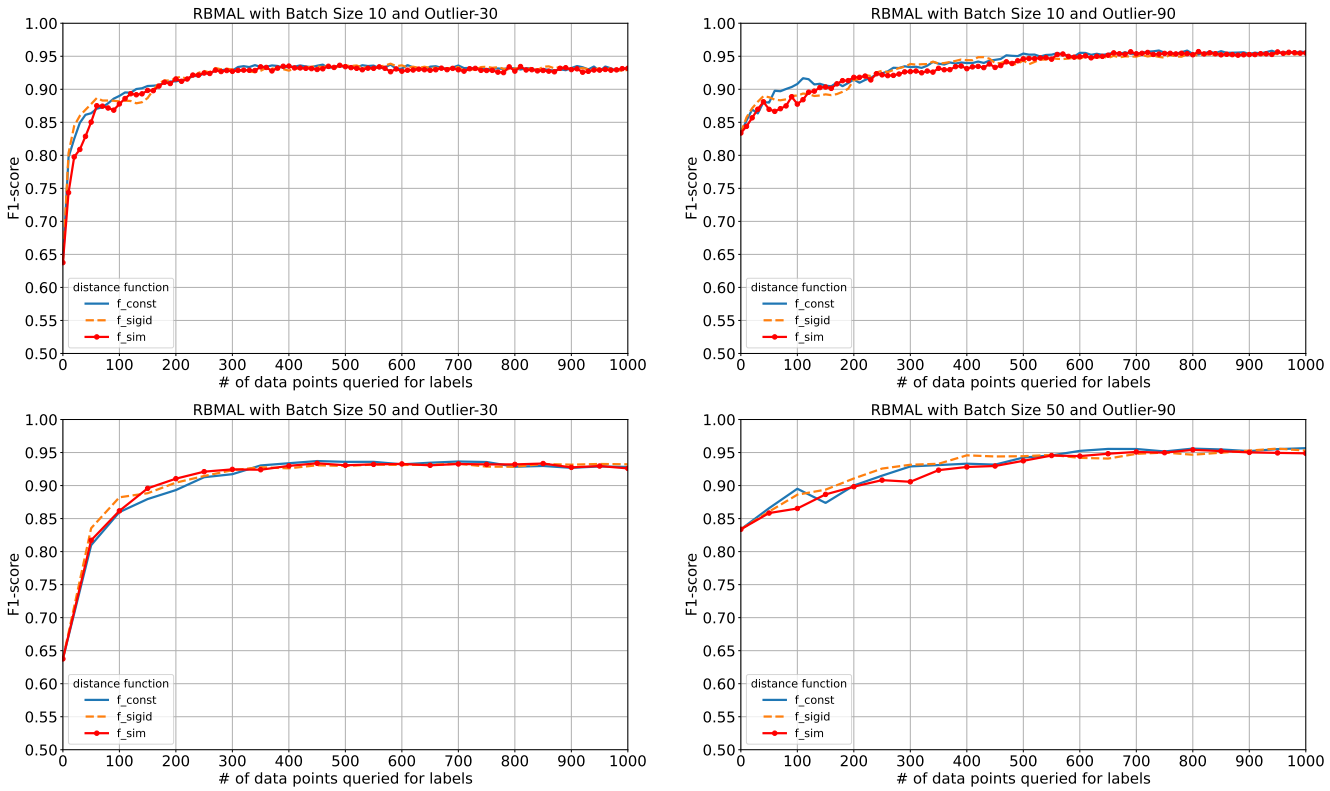

Fig. 11. F1-score of RBMAL with *Outlier-30* and *Outlier-90*

sampling (Section IV-B), demonstrated the learning limitations of such traditional approach. The F1-scores of US models did not exceed 0.90 and thus remained inferior to traditional supervised learning.

These results motivated us to propose the *Outlier-N* method for building more suitable-for-learning seeds and pools. This methodology ensures that *N%* of data points are outliers in the seed and pool data sets. Compared to the traditional approach to generate seed and pool data sets using random sampling, such outlier-centric seeds and pools contain more data points generated by less frequently matching signatures, thus inducing better learning. Unlike inliers, outliers do not belong predominantly to one particular class, and as we hypothesized in Section IV-C, outlier-centric seeds and pools contain more useful information for active learning based training. This hypothesis is empirically supported by the results obtained and reported in Section IV-D.

When experimenting with different pool sizes for the US active learning strategy, we found that increasing the pool size was usually beneficial for performance (Fig. 6). For the sake of comparison, most of the experiments were conducted with a seed of 100 data points. However, we also experimented with a small seed of 10 points. In this latter regard, while US models with small seeds learned rapidly and eventually reached the same performance as models with larger seeds, their initial performance was very low, with F1-scores around 0.2 (Fig. 9).

When comparing customary random sampling based seeds and pools with the ones built with the *Outlier-N* method, we found that larger values of *N* and larger pool sizes yielded better performance than random sampling, and also allowed outperforming traditional supervised learning (see Table III, Fig. 2, and Fig. 6). The highest peak performance was produced by the US model with the pool of size 10,000 built using the *Outlier-90* method. The F1-score of such model converged to 0.95. Similar to US models, the best performance for QbC and RBMAL active learning strategies was achieved with the *Outlier-90* method. In the case of the best QbC models, F1-scores converged to 0.96 with the pool size of 10,000, whereas for the best RBMAL models the F1-scores converged to 0.95.

Table VI provides a pair-wise comparison of the top performing classifiers for all the strategies evaluated in this work. More precisely, the following models are compared:

- Three supervised models described in Section IV-A: (i) RF with oversampling (henceforth, S-Over), (ii) RF with undersampling (S-Under), and (iii) RF with class weights (S-WRF),
- The best US active learning model using traditional random sampling based seed and pool as described in Section IV-B (henceforth, US-trad),
- The best models for US, QbC, and RBMAL with the *Outlier-N* method (described in Section IV-D).

For proper comparison, we performed 20 runs for each model and recorded the peak F1-score achieved. For US, QbC, and RBMAL strategies, the best classifiers were the following: US with *Outlier-90* (henceforth, US-90), QbC-CE with 3 committee members with *Outlier-90* (QbC-CE-90), and RBMAL with batch size of 10, f_{const} distance function, and *Outlier-90* (RBMAL-90).

Following [34], [37], [38], we used Wilcoxon signed-ranks test to evaluate the statistical significance of the difference in performance scores between pairs of classifiers trained using different strategies (i.e., clf_A, clf_B), i.e., hypothesis testing [37]. The null hypothesis (H_0) is that the performance of the classifiers is equivalent, i.e., there is no significant difference between the classifiers ($F1\text{-}score(clf_A) = F1\text{-}score(clf_B)$, while the alternative hypothesis (H_1) is that they are not equivalent ($F1\text{-}score(clf_A) \neq F1\text{-}score(clf_B)$, i.e., there is significant difference between their performance scores. The test outcome enables to accept or reject the null hypothesis and, therefore, evaluate the difference between the training approaches. The first two columns of Table VI report the average and standard deviation of each classification model for 20 runs ($n = 20$). The remaining columns report the matrix of $p - values$ as the outcome of the Wilcoxon test, which is symmetric over the main diagonal (gray cells). For the sake of understanding, reported $p - values$ are equated to confidence levels in the footnote and highlighted with different colors in the matrix, i.e., different green tones are associated to different levels of statistical significance as specified in the table footnote.

As can be observed, active learning strategies in combination with the *Outlier-90* method provided the highest performance scores, yielding significantly better results than all supervised and traditional active learning approaches. Among the three strategies using *Outlier-90*, uncertainty (US-90) and query by committee (QbC-CE-90) can be deemed as equivalent, with no statistical difference between their performance scores. The ranked-batch active learning strategy (RBMAL-90), despite outperforming supervised and active learning baselines, ranks in third position, being inferior to US-90 and QbC-CE-90.

Therefore, when using outlier-centric seeds and pools, all active learning techniques described in Section II-B achieved higher performance than traditional supervised learning methods and, as Table VI illustrates, these findings are statistically significant. Since active learning requires significantly less human labeling effort, using it with outlier-centric seeds and pools is a relevant and efficient alternative to traditional supervised learning when NIDS alerts are classified in SOC environments, in which a limited amount of human resources are available for labeling tasks.

Regarding QbC based active learning, we found that except for QbC-MD models, the committee size did not influence model performance significantly. Besides, unlike other models, the performance of QbC-VE started to deteriorate after a few hundred queries. Among the evaluated QbC active learning strategies, QbC-CE offered the best performance and the greatest degree of stability (Fig. 10).

As for RBMAL, we observed that the batch size did not influence the model performance significantly during the later stages of the active learning training cycle. Its effect was more noticeable during the earlier stages, with the largest batch size of 50 showing longer learning curves. Concerning the distance function, we found that a trivial distance function f_{const} (Section IV-D-3, Eq. 9) produced the best overall performance for all batch sizes, except for the largest one, despite that this distance function disables similarity based data point selection and only considers the classification uncertainty score. For the largest evaluated batch size, the f_{sigid} distance function (Section IV-D-3, Eq. 10) outperformed other distance functions.

When comparing US, QbC, and RBMAL models for the *Outlier-90* based 100-data-points seed and 10,000-data-points pool (see Fig. 6, Fig. 10, and Fig. 11), QbC-CE converged to the highest F1-score of 0.96, outperforming US and RBMAL strategies. However, QbC-VE and QbC-MD had a lower peak performance than US and RBMAL. In addition to yielding the highest F1-scores and a great degree of stability, QbC-CE models also featured the fastest learning rate – the F1-score of 0.95 was achieved after 300 queries, while US and RBMAL models required 500-600 queried data points.

These findings suggest that if NIDS alert data points are labeled one-by-one during the active learning process, QbC-CE models might be the best choice. Also, since the performance of QbC-CE was not found to be sensitive to the committee size in our experimental setup, committees of small size (e.g., 2-3) can be employed to reduce the computational resources required for model training.

On the other hand, if the labeling task is divided among several human experts, RBMAL models can be effectively utilized with smaller batch sizes (e.g., 10-20). Finally, as we have demonstrated, the use of RBMAL does not require the development of an elaborate distance function to assess the similarity between data points. Simple distance functions such as the ones used in this paper (e.g., a trivial distance function given by Eq. 9 in Section IV-D-3) can yield high-performance models with lower training costs.

VI. Limitations and Future Work

The NIDS alert classification approach proposed in this paper is able to highlight NIDS alert data points of high importance. However, in the case of DDoS attacks malicious activities might trigger a large volume of NIDS alerts that would normally be labelled as *irrelevant* by human analysts. In this regard, one of the limitations of the method proposed in this paper is its inability to detect an unexpected sharp increase in the number of *irrelevant* alerts.

To address this issue, different approaches should be applied. For example, Viinikka et al. proposed the use of time series analysis methods for that purpose [25]. As an example of another solution for that issue, SEC event correlation engine [30] with temporal attack detection rules can be used in SOC environments [4].

TABLE VI
STATISTICAL COMPARISON OF CLASSIFIERS FOR DIFFERENT STRATEGIES ($p - value$)

Average	**Std. dev.**	**Model**	*US-90*	*QbC-CE-90*	*RBMAL-90*	*S-Over*	*S-Under*	*S-WRF*	*US-trad*
0.9701	0.0026	US-90							
0.9699	0.0026	QbC-CE-90	0.9563						
0.9663	0.0028	RBMAL-90	0.0012	0.0031					
0.9267	0.0035	S-Over	1.91E-6	1.91E-6	1.91E-6				
0.8524	0.0154	S-Under	1.91E-6	1.91E-6	1.91E-6	1.91E-6			
0.9162	0.0046	S-WRF	1.91E-6	1.91E-6	1.91E-6	1.91E-6	1.91E-6		
0.9124	0.0104	US-trad	1.91E-6	1.91E-6	1.91E-6	1.91E-6	1.91E-6	0.0399	

- $p - value > 0.10$ – No evidence against H_0; H_0 is not rejected
- $0.01 < p - value < 0.05$ – Moderate evidence against H_0; H_1 is accepted at 95% confidence level
- $0.001 < p - value < 0.01$ – Strong evidence against H_0; H_1 is accepted at 99% confidence level
- $p - value < 0.001$ – Very strong evidence against H_0; H_1 is accepted at 99.9% confidence level

Another limitation of this paper is the use of a single data set for all evaluations. The main reason behind this limitation is the lack of publicly available data sets suitable for such evaluations. For example, the recent and well-known UNB2015 and CICIDS2017 network intrusion detection data sets do not contain NIDS alert data. Besides, when we replayed *pcap* network traffic from these data sets to our NIDS with over 50,000 signatures, we observed only 22,631 and 13,709 NIDS alerts for UNB2015 and CICIDS2017, respectively. Unfortunately, such a small amount of NIDS alerts does not allow for a thorough evaluation of the NIDS alert classification methods described in the present work. In contrast, the data set used in this paper contains 11,598,649 NIDS alerts that were aggregated into 1,395,324 data points. This also illustrates the fact that NIDS alert classification is a distinct and challenging research problem requiring data sets specifically designed for this particular problem.

As for future work, we plan to extend the NIDS alert data set to cover a significantly longer time frame (e.g., 1-2 years) to study concept drift issues in NIDS alert classification. Another future research direction is the investigation of attacks against machine learning models in SOC environments. Finally, the current work may be extended by studying the impact of labeling mistakes on the active learning performance.

REFERENCES

[1] https://suricata.io/
[2] https://www.snort.org/
[3] https://www.cisco.com/c/en/us/products/security/ngips/index.html
[4] Risto Vaarandi and Sten Mäses, "How to Build a SOC on a Budget," 2022 IEEE Conference on Cyber Security and Resilience, pp. 171–177.
[5] Risto Vaarandi, "A Stream Clustering Algorithm for Classifying Network IDS Alerts," 2021 IEEE Conference on Cyber Security and Resilience, pp. 14–19.
[6] Egon Kidmose, Matija Stevanovic, Søren Brandbyge, and Jens M. Pedersen, "Featureless Discovery of Correlated and False Intrusion Alerts," IEEE Access, vol. 8, 2020, pp. 108748–108765.
[7] Riyanat Shittu, Alex Healing, Robert Ghanea-Hercock, Robin Bloomfield, and Rajarajan Muttukrishnan, "OutMet: A New Metric for Prioritising Intrusion Alerts using Correlation and Outlier Analysis," 2014 IEEE Conference on Local Computer Networks, pp. 322–330.
[8] Risto Vaarandi and Kārlis Podiņš, "Network IDS Alert Classification with Frequent Itemset Mining and Data Clustering," 2010 International Conference on Network and Service Management, pp. 451–456.
[9] Georgios P. Spathoulas and Sokratis K. Katsikas, "Enhancing IDS performance through comprehensive alert post-processing," Computers and Security, vol. 37, 2013, pp. 176–196.
[10] Tian Wang, Chen Zhang, Zhigang Lu, Dan Du, and Yaopeng Han, "Identifying Truly Suspicious Events and False Alarms Based on Alert Graph," 2019 IEEE International Conference on Big Data, pp. 5929–5936.
[11] Tao Ban, Ndichu Samuel, Takeshi Takahashi, and Daisuke Inoue, "Combat Security Alert Fatigue with AI-Assisted Techniques," 2021 Cyber Security Experimentation and Test Workshop, pp. 9–16.
[12] Iksoo Shin, Yunsoo Choi, Taewoong Kwon, Hyeakro Lee, and Jungsuk Song, "Platform Design and Implementation for Flexible Data Processing and Building ML models of IDS Alerts," 2019 Asia Joint Conference on Information Security, pp. 64–71.
[13] Charles Feng, Shuning Wu, and Ningwei Liu, "A User-Centric Machine Learning Framework for Cyber Security Operations Center," 2017 IEEE International Conference on Intelligence and Security Informatics, pp. 173–175.
[14] Gina C. Tjhai, Steven M. Furnell, Maria Papadaki, and Nathan L. Clarke, "A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm," Computers and Security, vol. 29, 2010, pp. 712–723.
[15] Safaa Al-Mamory and Hongli Zhang, "Intrusion detection alarms reduction using root cause analysis and clustering," Computer Communications, vol. 32, 2009, pp. 419–430.
[16] Jie Ma, Zhi-tang Li, and Wei-ming Li, "Real-Time Alert Stream Clustering and Correlation for Discovering Attack Strategies," 2008 International Conference on Fuzzy Systems and Knowledge Discovery, pp. 379–384.
[17] Burr Settles, "Active Learning Literature Survey," Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 2009.
[18] Thiago N. C. Cardoso, Rodrigo M. Silva, Sérgio Canuto, Mirella M. Moro, and Marcos A. Gonçalves, "Ranked batch-mode active learning," Information Sciences, vol. 379, 2017, pp. 313–337.
[19] Anaël Beaugnon, Pierre Chifflier, and Francis Bach, "ILAB: An Interactive Labelling Strategy for Intrusion Detection," 2017 International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 120-140.
[20] Jorge L. Guerra Torres, Carlos A. Catania, and Eduardo Veas, "Active learning approach to label network traffic datasets," Journal of Information Security and Applications, vol. 49, 2019, pp. 1–13.
[21] Alejandro Guerra-Manzanares and Hayretdin Bahsi, "On the application of active learning for efficient and effective IoT botnet detection," Future Generation Computer Systems, vol. 141, 2023, pp. 40–53.
[22] Hidetoshi Kawaguchi, Yuichi Nakatani, and Shogo Okada, "IDPS Signature Classification Based on Active Learning With Partial Supervision From Network Security Experts," IEEE Access, vol. 10, 2022, pp. 105713–105725.
[23] https://github.com/ristov/nids-alert-data
[24] https://github.com/ristov/nids-alert-proc
[25] Jouni Viinikka, Hervé Debar, Ludovic Mé, Anssi Lehikoinen, and Mika Tarvainen, "Processing intrusion detection alert aggregates with time series modeling," Information Fusion Journal, vol. 10(4), 2009, pp. 312–324.
[26] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro, "TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time," 2019 USENIX Security Symposium, pp. 729–746.

[27] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck, "Dos and Don'ts of Machine Learning in Computer Security," 2022 USENIX Security Symposium, pp. 3971–3988.
[28] Tivadar Danka and Peter Horvath, "modAL: A modular active learning framework for Python," https://github.com/modAL-python/modAL
[29] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[30] Risto Vaarandi, Bernhards Blumbergs, and Emin Çalışkan, "Simple Event Correlator - Best Practices for Creating Scalable Configurations," 2015 IEEE CogSIMA Conference, pp. 96–100.
[31] Bushra A. Alahmadi, Louise Axon, and Ivan Martinovic, "99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms," 2022 USENIX Security Symposium, pp. 2783–2800.
[32] Manfred Vielberth, Fabian Böhm, Ines Fichtinger, and Günther Pernul, "Security Operations Center: A Systematic Study and Open Challenges," IEEE Access, vol. 8, 2020, pp. 227756–227779.
[33] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Brdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco, "The Role of Machine Learning in Cybersecurity, " Digital Threats: Research and Practice, vol. 4(1), 2023, pp. 1–38.
[34] Giovanni Apruzzese, Pavel Laskov, and Aliya Tastemirova, "SoK: The Impact of Unlabelled Data in Cyberthreat Detection," 2022 IEEE European Symposium on Security and Privacy, pp. 20–42.
[35] Kristijan Vidović, Ivan Tomičić, Karlo Slovenec, Miljenko Mikuc, and Ivona Brajdić, "Ranking Network Devices for Alarm Prioritisation: Intrusion Detection Case Study," 2021 International Conference on Software, Telecommunications and Computer Networks, pp. 1–5.
[36] Thijs van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna, "DEEPCASE: Semi-Supervised Contextual Analysis of Security Events," 2022 IEEE Symposium on Security and Privacy, pp. 522–539.
[37] Janez Demšar, "Statistical comparisons of classifiers over multiple data sets", The Journal of Machine learning research, vol. 7, 2006, pp. 1-30.
[38] Nathalie Japkowicz and Mohak Shah, "Evaluating learning algorithms: a classification perspective", Cambridge University Press, 2011.