Machine Learning with Applications 9 (2022) 100351

Contents lists available at ScienceDirect

Machine Learning with Applications

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

Jasmine: A new Active Learning approach to combat cybercrime

Jan Klein [a,*], Sandjai Bhulai [b], Mark Hoogendoorn [c], Rob van der Mei [a]

[a] Department of Stochastics, Centrum Wiskunde & Informatica, Science Park 123, 1098XG, Amsterdam, The Netherlands
[b] Department of Mathematics, Vrije Universiteit, De Boelelaan 1111, 1081HV, Amsterdam, The Netherlands
[c] Department of Computer Science, Vrije Universiteit, De Boelelaan 1111, 1081HV, Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Active Learning
Dynamic query function
Network intrusion detection
Human oracle
Partially labeled

ABSTRACT

One of the reasons that the deployment of network intrusion detection methods falls short is the lack of realistic labeled datasets, which makes it challenging to develop and compare techniques. It is caused by the large amounts of effort that it takes for a cyber expert to classify network connections. This has raised the need for methods that learn from both labeled and unlabeled data which observations are best to present to the human expert. Hence, Active Learning (AL) methods are of interest.

In this paper, we propose a new hybrid AL method called Jasmine. Firstly, it uses the uncertainty score and anomaly score to determine how suitable each observation is for querying, i.e., how likely it is to enhance classification. Secondly, Jasmine introduces dynamic updating. This allows the model to adjust the balance between querying uncertain, anomalous and randomly selected observations. To this end, Jasmine is able to learn the best query strategy during the labeling process. This is in contrast to the other AL methods in cybersecurity that all have static, predetermined query functions. We show that dynamic updating, and therefore Jasmine, is able to consistently obtain good and more robust results than querying only uncertainties, only anomalies or a fixed combination of the two.

1. Introduction

The fight against cybercrime has become a priority for many countries. This is with good reason, because the average cost of a single cyberattack in Europe is around 50 thousand euros, as estimated by Forrester Consulting and Hiscox (Consultancy.eu, 2020). Most common attacks were mostly targeted on companies and even governmental institutes. For example, 68% of Dutch firms reported at least one cyber incident in 2019. Therefore, the amount of money that the surveyed European companies invested in cybersecurity has increased by 39% from 1.3 million to 1.8 million euros. In academia, the awareness for research in the field of cybersecurity has also grown. For instance, Mouloua, Ferraro, Mouloua, Matthews, and Copeland (2019) analyzed the articles published in the Proceedings of the Human Factors and Ergonomics Society (HFES) from 1980 to 2018. They showed that 73% of articles related to cybersecurity published in that almost 40-year span were written in the last nine years.

Since most cyberattacks are aimed at companies and countries, it is important to know how networks of computers can be protected. A Network Intrusion Detection System (NIDS) is software designed to detect unusual, malicious events in a computer network. There are several types of systems with each having their own set of challenges for which several solutions have been proposed (Sommer & Paxson, 2010; Sultana, Chilamkurti, Peng, & Alhadad, 2019; Zamani & Movahedi, 2013). However, there are some broader challenges in cybersecurity research. Most importantly, Xin et al. (2018) and Yang, Ren, Zhu, and Zhang (2018) argue that not much consideration is given to deployment efficiency. This means that little is practically done with published research, because of time complexity of the techniques and the efficiency of detection in actual networks. The latter is due to the lack of realistic datasets, since it takes a lot of time and effort for a human to classify network connections correctly. Moreover, cyber analysts have to label many redundant connections just to construct a representative dataset. This loads the experts with tedious work and leads to an underuse of their capabilities. Therefore, it would be beneficial to only present 'informative' network connections to the cyber analyst. This can be realized in Active Learning (AL), in which the model chooses from which unlabeled data instances it wants to learn and then queries their labels (Kumar & Gupta, 2020; Settles, 2009).

Several AL methods have been proposed in intrusion detection research. Many of them focus on querying *uncertain* data, i.e., requesting the label of observations about which the model is not sure how to classify them (Görnitz, Kloft, Rieck, & Brefeld, 2009; Guerra Torres, Catania, & Veas, 2019; Li & Guo, 2007). Adding these observations with their correct label is expected to enhance classification performance

* Corresponding author.
E-mail addresses: j.g.klein@cwi.nl (J. Klein), s.bhulai@vu.nl (S. Bhulai), m.hoogendoorn@vu.nl (M. Hoogendoorn), r.d.van.der.mei@cwi.nl (R. van der Mei).

https://doi.org/10.1016/j.mlwa.2022.100351
Received 6 September 2021; Received in revised form 12 November 2021; Accepted 2 June 2022
Available online 8 June 2022

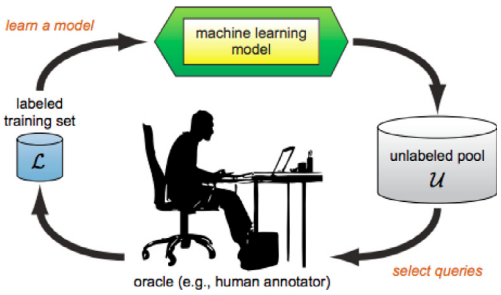


Fig. 1. Illustration of Active Learning framework (Settles, 2009).

more quickly than randomly selecting observations. Other studies consider different query strategies or combine several query approaches to make the AL procedure more robust (Yin, Wang, & Fan, 2018). Stokes, Platt, Kravis, and Shilman (2008) query both uncertain and *anomalous* instances. The latter are observations that behave vastly differently than expected, which could indicate malicious activities. Although combining query types increases prediction performance, the optimal balance of the types depends on, for example, the dataset. Moreover, the balance has to be determined beforehand and is still up for debate.

In our research, we propose a novel AL method called Jasmine that introduces *α-dynamic updating*. This allows our method to adjust the balance between querying different types of observations such that the right types are proposed to the human expert at the right time. This makes sense, because the structure of the labeled set (on which the classifier is trained) changes during the labeling procedure. Hence, Jasmine is able to learn the best query strategy during the process. The types of instances that Jasmine considers potentially informative are uncertain, anomalous and randomly selected observations. Jasmine is able to dynamically change the balance between the three types to ensure that the most informative observations to the intrusion detection model are queried. This sets our method apart from existing AL methods, because (to our knowledge) Jasmine is the only method that allows for this. Our contributions are:

- We propose a new AL method called Jasmine that introduces dynamic updating of the balance between querying uncertain, anomalous and randomly selected unlabeled data.
- We present the mathematical formulation of Jasmine and explain why it can find a good balance given the current labeling state.
- We apply Jasmine to two commonly used network intrusion detection datasets and use them in different experimental settings. We show that Jasmine obtains good results and is more robust than static query approaches. Moreover, it performs even better in the case of highly imbalanced data. Therefore, Jasmine is more reliable to use, because it can adapt itself to different situations.

The rest of the paper is organized as follows. In Section 2, we explain the AL paradigm and what methods have been proposed in intrusion detection. Section 3 introduces the mathematical notation used in the AL framework. In Section 4, we propose our method Jasmine and explain how it works. The experiments that we execute to validate our method are discussed in Section 5. Section 6 subsequently shows the results of these experiments and their interpretations. Finally, in Section 7, we draw conclusions about this study and make suggestions for further research.

2. Related work

Active Learning is a subfield of machine learning in which the premise is that only a small part of the data is labeled, while the labels for the rest of the data are not specified. The AL procedure is illustrated in Fig. 1. Firstly, an ML model is trained on the labeled set and applied to the unlabeled observations to obtain output predictions. Then, interesting observations from the unlabeled part are selected and an oracle is asked to provide the actual labels. Subsequently, these query observations are added to the labeled set and the procedure continues. An important advantage of AL is that the model needs less train data to learn better (Kumar & Gupta, 2020; Settles, 2009). For instance, it has been used to improve performance in the diagnosis and treatment of diseases (Budd, Robinson, & Kainz, 2021), in the personalization and hybridization of recommender systems (Elahi, Ricci, & Rubens, 2016), and in community detection (Gadde, Gad, Avestimehr, & Ortega, 2016). Moreover, AL is especially beneficial in domains in which it is laborious to label data. Examples of such fields are speech recognition, information extraction (person names from texts, annotation of genes, etc.), classification of files (relevant or irrelevant documents, images, etc.) (Settles, 2009) and network intrusion detection. The diversity among these examples shows the broad application capability of AL.

2.1. Query strategies

There are multiple ways to query the oracle, but we focus on *pool-based sampling*. Here, the decision to query certain observations is based on some informativeness measure that is calculated for all instances in the unlabeled pool (or a subset thereof). This approach to querying has been applied to many real-world problems, including many of the application domains mentioned before. It works well with a human oracle and when it is relatively easy to compute the informativeness of all observations at once. A commonly used measure for this is the uncertainty score. This score is used to query observations about which the model is the least certain on how to predict their label (Lewis & Gale, 1994). This is a simple informativeness measure, because no new models have to be trained and only the output of the classifier is required to determine the uncertainty of each observation.

2.2. Active Learning in network intrusion detection

Which specific query strategy and ML technique are used depends on the AL application. In network intrusion detection, several AL methods have been proposed. These approaches mostly rely on uncertainty sampling. To illustrate this, Li and Guo (2007) use Transductive Confidence Machines for K-Nearest Neighbors for supervised intrusion detection with uncertainty sampling and query by committee; Görnitz et al. (2009) use a Support Vector Domain Description (SVDD) for anomaly detection with uncertainty sampling as the AL component; and Guerra Torres et al. (2019) make use of Random Forests for prediction and query uncertain observations. These studies show that a method with AL obtains better results than one without it, or that the proposed query strategy performs better than randomly presenting observations to the oracle. Though, the diversity in considered query approaches is low since uncertainty sampling is often chosen. Both Gu and Zydek (2014) and Yang et al. (2018) present overview of multiple other query strategies for intrusion detection, but the authors ultimately choose for uncertainty sampling too. Yet, literature does show that this common query approach performs well.

However, there is room for significant improvement. For example, what happens when more than one informativeness measure is considered for query selection? There are studies that incorporate two measures to improve classification performance. Stokes et al. (2008) propose to combine uncertainty sampling with querying anomalous data, thus presenting instances that behave vastly differently than expected to the oracle. Yin et al. (2018) combine the uncertainty informativeness measure with density information as the query approach. More specifically, the distance of an observation to its nearest neighbor is calculated and instances residing in high density areas are more likely to be queried. These two studies show that using multiple informativeness measures further improves performance, because more characteristics of the data are used to determine which unlabeled observations would improve predictions the most if they were labeled.

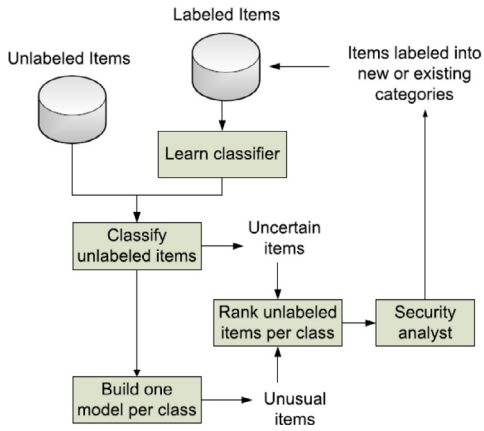


Fig. 2. Schematic representation of an ALADIN iteration (Stokes et al., 2008).

The common factor in the aforementioned research is that all proposed query functions are *static* in nature. From the start, it is exactly known how the set of query observations is constructed and this approach cannot be changed. This means the contribution of each informativeness measure in the selection of the query observations has to be fixed beforehand. However, the optimal balance of these contributions depends on the overall structure of the data and also on the current state of the labeling process. Therefore, we consider a *dynamic* query approach to address these problems. To this end, the balance can be adapted during the procedure to best fit the data. More specifically, the method learns the distribution of query types that is expected to increase prediction performance the most given the current state.

2.3. ALADIN

The AL methodology for network intrusion detection that we use as a starting point was developed by Stokes et al. (2008) and is called ALADIN. It was chosen because it combines two important informativeness measures: the *(i)* uncertainty score and *(ii)* anomaly score. On the one hand, by querying selected anomalies, new classes of network traffic can be found within the data. On the other hand, by querying uncertainties, the accuracy of the classifier should increase in the next time step when the correct classes have been provided by the human expert. ALADIN combines both Pelleg's algorithm for anomaly detection (Pelleg & Moore, 2004) and Almgren's algorithm for intrusion detection (Almgren & Jonsson, 2004). In the first algorithm, an anomaly detection model is constructed for each data class, so one for the benign class and one for the malicious class in the binary case. How unlikely an unlabeled observation is according to the model of its (predicted) class, determines how anomalous it is. The less likely an instance is, the more interesting it is for querying. In the second algorithm, observations that lie close to the decision boundary of the trained classifier are deemed uncertain, and hence, interesting for querying. The description of ALADIN is illustrated in Fig. 2. On the 1999 KDD-Cup dataset (Stolfo et al., 1999), Stokes et al. (2008) show that ALADIN achieved high accuracy in predicting known traffic classes. Moreover, it was able to detect previously unknown categories quickly. This shows that incorporating anomalous and uncertain observations in the query set led to favorable results.

ALADIN uses simple machine learning techniques such as logistic regression and naive Bayes to be able to scale well. However, the authors mention that the benign class in the KDD-Cup dataset is very diverse, meaning that this class may not be easily predicted with the logistic regression classifier. In our research, we consider more advanced machine learning techniques that are better able to capture the diversity of network traffic. More importantly, in ALADIN, half of the queried observations are anomalous, while the other half is uncertain. This fixed 50/50 split is not motivated by the authors. Hence, model performance can improve when the proportion between querying anomalies and uncertainties is changed depending on the considered dataset and within the process. This leads to our α-dynamic updating.

3. Preliminaries

Before we provide the mathematical formulation of Jasmine, we introduce some notation to describe the AL framework in general. Firstly, we assume to have a dataset $\mathbf{X} \in \mathbb{R}^{M\times K}$, with $M \in \mathbb{N}$ the number of observations and $K \in \mathbb{N}$ the number of features or attributes. Let $\mathbf{x}_i \in \mathbb{R}^K$ be the feature vector of observation $i \in \{1, \dots, M\}$ and let $y_i \in \{0, 1, *\}$ be its corresponding response value. Here, '0' represents the benign class and '1' the malicious class. The symbol '*' means that the class or label is missing. In AL, it is assumed that only a (possibly small) part of the data is labeled, while the rest is unlabeled. The set of labeled observations $\mathcal{L}(t)$ and the set of unlabeled observations $\mathcal{U}(t)$ depend on the iteration or time step $t = 1, \dots, T$, where $T \in \mathbb{N}$ is a predetermined maximum number of iterations. Since more labels become available when the iteration procedure progresses, the vector of response values of all the instances $\mathbf{y}(t) = (y_1(t), \dots, y_M(t))$ is dependent on time. Now, for all iterations it holds that $\mathcal{L}(t)$ and $\mathcal{U}(t)$ are disjoint and their union is the complete dataset $(\mathbf{X}, \mathbf{y}(t))$ with labels up to time t. Let $L(t) := |\mathcal{L}(t)|$ be the number of labeled observations and $U(t) := |\mathcal{U}(t)|$ the number of unlabeled instances at time t. Note that $L(t+1) > L(t)$, while $U(t+1) < U(t)$, because every iteration the human expert adds labels to previously unlabeled observations. $\mathcal{L}(0)$ contains the instances that are labeled from the start, while $\mathcal{U}(0)$ consists of all initially unlabeled observations. In iteration t, a supervised machine learning technique $f_t : \mathbb{R}^K \to [0, 1]$ is trained on $\mathcal{L}(t-1)$. This classifier f_t is then applied to $\mathcal{U}(t-1)$ to obtain predictions for the actual classes of the unlabeled observations. Then, a query function ψ determines which instances from $\mathcal{U}(t-1)$ are selected to be queried to the human expert. Let $\mathcal{Q}(t) \subseteq \mathcal{U}(t-1)$ be the constructed set of query observations. Usually, this set has fixed size $Q := |\mathcal{Q}(t)|$. Then, the expert provides the labels $y_q(t) \in \{0, 1\}$ for the observations $q \in \mathcal{Q}(t)$. Note that in the previous iteration their labels were still unknown: $y_q(t-1) =$ '*'.

4. Methods

In this section, we introduce our Active Learning method Jasmine. Its key component is α-dynamic updating, which allows the model to dynamically adjust the balance between querying anomalous, uncertain and random observations during the procedure. The adjustment of the balance comes in two flavors. Firstly, the initial proportions of the three types of query observations are determined. Based on the available initially labeled data $\mathcal{L}(0)$, it can be beneficial to start with querying 60% anomalous, 15% uncertain and 25% random observations, for example. Secondly, the balance between the types can be changed during the labeling process, because it may be better to query increasingly more anomalous instances when more and more labels are provided by the oracle. This leads to dynamically updating the query fractions. Moreover, we also consider querying random observations. This may seem counter-intuitive in the AL setting, because its premise is not to bother the human expert with labeling redundant observations. However, when $\mathcal{L}(0)$ is not a good representation of the entire dataset, querying some random observations could be beneficial.

The complete Jasmine procedure is illustrated in Fig. 3. It consists of two consecutive phases. In Phase 2, the actual Active Learning component of Jasmine is executed given the results of Phase 1. More specifically, the classifier is trained on $\mathcal{L}(t)$ and applied to $\mathcal{U}(t)$ to obtain malicious probabilities (Section 4.3). Then, the informativeness measures of the unlabeled observations are calculated (Section 4.4) and the query sample is constructed based on these measures (Section 4.5).

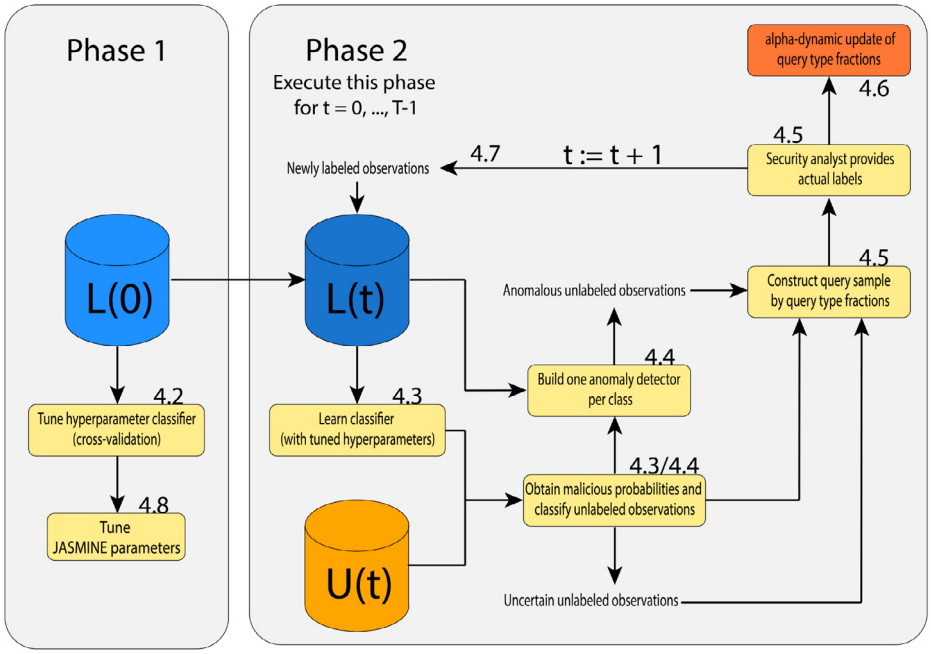


Fig. 3. Schematic representation of complete Jasmine procedure. The numbers of the form '4.x' denote in which subsection the corresponding component is explained.

Next, the actual labels are provided by the human oracle. Then, the crucial part of Jasmine is performed by updating the query fractions based on the results obtained during the iteration (Section 4.6). Consequently, the query set of the next iteration should have a better balance of anomalous, uncertain and random observations. Finally, the labeled query observations are added to $\mathcal{L}(t+1)$ (Section 4.7) and the procedure continues. In Phase 1, good values of the hyperparameters of the classifier are determined (Section 4.2). Also, good values of the Jasmine-specific parameters are chosen in this phase (Section 4.8). Since tuning of the Jasmine parameters is a reduced version of the second phase, we explain the latter first in this section.

4.1. Classification and anomaly detection techniques

The supervised machine learning technique used as the classifier in Jasmine is the Gradient Boosting Machine (GBM), which was designed by Friedman (2001). As described by Caruana, Karampatziakis, and Yessenalina (2008), Ogutu, Piepho, and Schulz-Streeck (2011) and Natekin and Knoll (2013) one of the main merits of the GBM is its flexibility and robustness with respect to the number of hyperparameters. It can easily be customized for different practical purposes. Another advantage of the GBM is its interpretability. Since the technique is an ensemble of multiple simple models, it can be easily understood.

The anomaly detection method used in Jasmine is the Isolation Forest (IF), which was developed by Liu, Ting, and Zhou (2008). Just as the GBM, it is a tree-based ensemble machine learning technique. Most anomaly detection models try to construct a representation of normal behavior, but the IF aims to isolate the anomalous observations. The authors use two important properties of anomalies: *(i)* there are few of them, and *(ii)* they have distinctly different characteristics than the majority of observations. Because these properties hold, it is relatively easy to isolate them from the rest. The main advantage of the IF technique is that it is fast and relatively easy to interpret (Liu, Ting, & Zhou, 2012).

4.2. Tuning GBM hyperparameters

The GBM has hyperparameters that are determined beforehand. In Jasmine, good values are found via hyperparameter tuning (Claesen & De Moor, 2015). This is done on the set $\mathcal{L}(0)$, because by definition the observations in this set are the only ones with labels from the start. We use k-fold cross validation such that each observation is both utilized for training and evaluating the model. This is desirable, because $\mathcal{L}(0)$ is usually rather small, so we want to effectively use each provided label. During tuning, also the computation time is taken into account, because the GBM is retrained each AL iteration and training times quickly stack. We want to find hyperparameters that yield good performance in both predictive ability and computation time. To this end, assume there are H hyperparameter combinations yielding a sequence of performance metrics $h_1, \dots, h_H$ and a sequence of computation times $t_1, \dots, t_H$. Let $h_* := \max\{h_1, \dots, h_H\}$ be the best performance and let $j_* := \arg\max\{h_1, \dots, h_H\}$ be the combination yielding this performance. If there are several combinations resulting in the best performance metric, then j_* is the one with the smallest computation time t_{j_*}. Also, if there is a combination yielding a performance of almost h_*, but with a much smaller computation time than t_{j_*}, then we prefer to go for that combination. In that case, combination j is chosen as the optimal one whenever

$$d_j := \frac{h_{j^*} - h_j}{t_{j^*} - t_j} < \varepsilon,$$

where $\varepsilon > 0$ is some predefined threshold. If several j satisfy this requirement, then the one with the smallest d_j is chosen.

4.3. Training, evaluating and predicting

The classification is done by means of a GBM, which we described in Section 4.1. The hyperparameter values for this technique are determined by tuning, as described in Section 4.2. In iteration t, the GBM is trained with k-fold cross validation on $\mathcal{L}(t-1)$, yielding the classifier f_t and some threshold probability $\theta \in (0, 1)$. This θ represents the border between predicting an instance as 0 (benign) or as 1 (malicious), and is the value that maximizes some performance metric on $\mathcal{L}(t-1)$. In Jasmine, we are interested in how confident f_t is in its predictions. The closer the predicted probability is to θ, the less certain the model is. Intuitively, a value of 0.5 can be seen as the least certain probability for a binary classifier, because it is precisely between 0 and 1. Therefore, θ is transformed to be 0.5 and the probabilities are changed accordingly.

We provide more reasoning why this is done in Section 4.4. The function $\varphi_\theta : [0,1] \to [0,1]$, defined as

$$\varphi_\theta(y) = \frac{(1-\theta)y}{(1-2\theta)y+\theta},$$

is applied to all predicted probabilities to transform them. We chose this function, because it has the following necessary and desirable properties: *(i)* φ is continuously differentiable, *(ii)* $\varphi_\theta(0) = 0$, *(iii)* $\varphi_\theta(\theta) = 0.5$, *(iv)* $\varphi_\theta(1) = 1$, and *(v)* $\varphi'_\theta(y) \geq 0$. Note that the probabilities do not change when θ is already 0.5: $\varphi_{\theta=0.5}(y) = y$ for all $y \in [0,1]$.

After this, f_t is applied to the unlabeled set $\mathcal{U}(t-1)$ to obtain predicted probabilities $\hat{y}_u(t) \in [0,1]$ (which have been transformed by φ_θ) for each $u \in \mathcal{U}(t-1)$. Consequently, the predicted class $\hat{c}_u(t)$ is 0 if $\hat{y}_u(t) < 0.5$ and 1 if $\hat{y}_u(t) \geq 0.5$.

4.4. Calculating certainty score and anomaly score

In the next step, the measures needed for the query function ψ^{Jas} to determine which unlabeled observations to present to the expert are calculated. These measures are the certainty score and anomaly score.

The certainty score $z_u(t) \in [0,1]$ is defined as

$$z_u(t) := 2\left|\hat{y}_u(t) - \frac{1}{2}\right|, \tag{1}$$

for each $u \in \mathcal{U}(t-1)$. The lower this score, the more uncertain the trained model f_t is about the predicted label of u. Eq. (1) is the commonly used definition for the certainty score in AL methods, such as in ALADIN. It also shows why we transformed the raw predicted probabilities by φ_θ. Now, the distance from $\frac{1}{2}$ can be at most 0.5 in both the benign direction (corresponding to $\hat{y}_u(t) \downarrow 0$) and the malicious direction ($\hat{y}_u(t) \uparrow 1$), making $z_u(t)$ a symmetric score.

The IF technique that we described in Section 4.1 is used to determine the anomaly score $a_u(t)$ for each $u \in \mathcal{U}(t-1)$. Firstly, the class-specific observation sets are defined for each $c \in \{0,1\}$ by

$$\begin{aligned}\mathcal{L}^{(c)}(t-1) &:= \{l \in \mathcal{L}(t-1) : y_l = c\},\\ \mathcal{U}^{(c)}(t) &:= \{u \in \mathcal{U}(t-1) : \hat{c}_u(t) = c\}.\end{aligned} \tag{2}$$

Now, one IF is trained on the set $\mathcal{L}^{(0)}(t-1) \cup \mathcal{U}^{(0)}(t)$ and one on the set $\mathcal{L}^{(1)}(t-1) \cup \mathcal{U}^{(1)}(t)$, yielding a benign IF $I_t^{(0)} : \mathbb{R}^K \to [0,1]$ and a malicious IF $I_t^{(1)} : \mathbb{R}^K \to [0,1]$, respectively. Then, the anomaly score of $u \in \mathcal{U}(t-1)$ is defined as

$$a_u(t) := I_t^{(\hat{c}_u(t))}(\mathbf{x}_u). \tag{3}$$

This is the output value that the appropriate IF produces when the feature vector $\mathbf{x}_u$ is fed into it.

4.5. Constructing query sample $\mathcal{Q}(t)$

An important component of Jasmine is the way in which the query function ψ^{Jas} determines how the query sample $\mathcal{Q}(t)$ is constructed. There are three types of query observations: anomalies, uncertainties and random instances. Which part of $\mathcal{Q}(t)$ should be allocated to which type is given by the anomaly fraction $\alpha_a(t)$, uncertainty fraction $\alpha_z(t)$ and randomness fraction $\alpha_r(t)$. For each class $c \in \{0,1\}$, the unlabeled observations in $\mathcal{U}^{(c)}(t)$ (see (2)) are sorted from most anomalous to least anomalous. Then, the top $\frac{1}{2}\alpha_a(t) \cdot Q$ (rounded to the nearest integer) observations are taken as the anomalous query instances for predicted class c. The uncertainties are selected in a similar way: the observations in $\mathcal{U}^{(c)}(t)$ are sorted from least certain to most certain for each class c. Then, the top $\frac{1}{2}\alpha_z(t) \cdot Q$ instances are selected for the uncertain query observations for predicted class c. The random query observations are selected in a simple way: a sample of size $\alpha_r(t) \cdot Q$ is taken from $\mathcal{U}(t)$ (without the already selected anomalous and uncertain observations).

There are some important technicalities in constructing $\mathcal{Q}(t)$. First of all, it is possible that there are not enough observations for a specific class. Then, observations from the other class are added to reach the number of required anomalies or uncertainties. Secondly, there can be an overlap between the most anomalous and least certain instances: some observations can be both anomalous and uncertain, and hence, we select them for both query types.

Ultimately, when the query observations have been determined, the query set $\mathcal{Q}(t)$ is shown to the human expert and they provide the correct label y_q for each $q \in \mathcal{Q}(t)$.

4.6. α-Dynamic update

4.6.1. Constructing update parameters

The query set $\mathcal{Q}(t)$ obtained in Section 4.5 can be decomposed in $\mathcal{Q}_a(t)$, $\mathcal{Q}_z(t)$ and $\mathcal{Q}_r(t)$, which are the sets of anomalous, uncertain and random queries, respectively. Also, let $Q_a(t)$, $Q_z(t)$ and $Q_r(t)$ be their respective sizes. Furthermore, let $y_q \in \{0,1\}$ be the real label of query observation q, $\hat{y}_q(t) \in [0,1]$ its predicted malicious probability and $\hat{c}_q(t) \in \{0,1\}$ its predicted label. In practice, the real label is available since the cyber expert provides it. We want to construct a metric for the anomalous queries and one for the uncertain queries that describes how much information the observations of those types can potentially add to the model. We do this by looking at the false negatives (FNs) and false positives (FPs) in $\mathcal{Q}(t)$. The reasoning is that when there are many FNs and FPs, then the model was bad at predicting the real classes of those unlabeled observations. Hence, adding these observations to the labeled set $\mathcal{L}(t)$ could yield a lot of information for the classifier trained in the next iteration. In short, we consider the fraction of FPs and FNs in $\mathcal{Q}(t)$, but also take a look at how convincing they are. If the model is fairly certain that an observation is malicious while it is in fact benign, then this FP obtains a larger weight than if the model is not that sure. Consequently, we define the anomaly information metric $\delta_a^\beta(t)$ as follows:

$$\delta_a^\beta(t) := \frac{\sum_{q \in \mathcal{Q}_a(t)} |\hat{y}_q(t) - y_q| \cdot \left(\beta \cdot \mathbf{1}_{\{\hat{c}_q=0, y_q=1\}} + \mathbf{1}_{\{\hat{c}_q=1, y_q=0\}}\right)}{Q_a(t) + (\beta-1) \cdot |\{q \in \mathcal{Q}_a(t) : y_q = 1\}|}. \tag{4}$$

Note that the first indicator function in (4) corresponds to an FN: the real label is 1, but the predicted label is 0. Equivalently, the second indicator function corresponds to an FP. If the query observation q is an FN, it gets weighted by some parameter $\beta > 0$. This enables us to put more ($\beta > 1$) or less ($\beta < 1$) emphasis on the FNs compared to the FPs. Thus, if q is an FN, then it obtains value $\beta|\hat{y}_q(t) - y_q|$; if it is an FP, it obtains $|\hat{y}_q(t) - y_q|$. The farther the predicted probability $\hat{y}_q(t)$ is from 0 or 1 (thus the less certain the model is about the prediction), the larger the assigned value for q becomes. The denominator ensures that the value of $\delta_a^\beta(t)$ is at most 1. When there are no FPs and FNs, then the value of the metric is 0. Consequently, $\delta_a^\beta(t) \in [0,1]$. The larger $\delta_a^\beta(t)$ is, the more information the anomalies convey, because we expect that incorrectly predicted observations add relevant information to the model if they are added to the train set with the correct labels.

Similarly, we define the uncertainty information metric

$$\delta_z^\beta(t) := \frac{\sum_{q \in \mathcal{Q}_z(t)} |\hat{y}_q(t) - y_q| \cdot \left(\beta \cdot \mathbf{1}_{\{\hat{c}_q=0, y_q=1\}} + \mathbf{1}_{\{\hat{c}_q=1, y_q=0\}}\right)}{Q_z(t) + (\beta-1) \cdot |\{q \in \mathcal{Q}_z(t) : y_q = 1\}|}, \tag{5}$$

as a measure on how much information the uncertain observations in the query set add on average. Also, $\delta_z^\beta(t) \in [0,1]$.

Next, the difference between the information metrics that we defined in (4) and (5) describes whether anomalies or uncertainties could add more information to the model. We define this difference $\Delta(t)$ as

$$\Delta(t) := \delta_a^\beta(t) - \delta_z^\beta(t) \in [-1,1].$$

It is used to determine how the query fractions of anomalies and uncertainties are updated. If $\Delta(t) > 0$, then the queried anomalies could add more information on average, and hence, preferably, more anomalies are selected the next iteration. If $\Delta(t) < 0$, then the uncertainties could add more information, and so, more uncertainties are selected.

Now, we want the possibility to put more or less emphasis on bigger or smaller values of $\Delta(t)$. Hence, we introduce a non-linearity governed by $\gamma > 0$ to obtain $\Delta^\gamma(t)$. To this end, we define the update factor as

$$\Delta^\gamma(t) := \text{sgn}(\Delta(t)) \cdot |\Delta(t)|^{1/\gamma}. \tag{6}$$

When $\gamma = 1$, then (6) reduces to the linear case $\Delta^\gamma(t) = \Delta(t)$. If $0 < \gamma < 1$, then $|\Delta^\gamma(t)| \leq |\Delta(t)|$, so the update factor is relatively smaller. On the other hand, if $\gamma > 1$, then $|\Delta^\gamma(t)| \geq |\Delta(t)|$ and the factor is relatively larger.

4.6.2. Defining query fractions

The update factor $\Delta^\gamma(t)$ is used to determine whether more anomalies or uncertainties should be queried in the next iteration. The updates $\alpha_a(t+1)$ and $\alpha_z(t+1)$ have the forms

$$\alpha_a(t+1) = \lambda_{t+1}\left(\alpha_a(t) + w_a^{(1)}(t) \cdot \max\{0, \Delta^\gamma(t)\} + w_a^{(2)}(t) \cdot \min\{0, \Delta^\gamma(t)\}\right), \tag{7}$$

and

$$\alpha_z(t+1) = \lambda_{t+1}\left(\alpha_z(t) + w_z^{(1)}(t) \cdot \max\{0, -\Delta^\gamma(t)\} + w_z^{(2)}(t) \cdot \min\{0, -\Delta^\gamma(t)\}\right), \tag{8}$$

respectively. Here, the w variables are non-negative constants that we need to ensure that the fractions stay within the correct bounds for time step t. Moreover, λ_{t+1} denotes a linear function which guarantees that the updated fractions are also within the bounds for time step $t+1$. Let us examine (7) in a bit more detail: the new fraction $\alpha_a(t+1)$ is based on the old value $\alpha_a(t)$ plus some value when $\Delta^\gamma(t) > 0$ or minus some value when $\Delta^\gamma(t) < 0$. In other words, when the anomalies add more information on average than the uncertainties, then $\alpha_a(t+1)$ becomes larger. Otherwise, $\alpha_a(t+1)$ becomes smaller. The update dynamics are the other way around for (8). We provide the explicit mathematical definitions of the w variables and λ_{t+1} in Appendix. It is important to mention that they depend on the hyperparameters $\alpha_a^{(0)}$ and $\alpha_z^{(0)}$, which are the initial query fractions of anomalies and uncertainties, respectively.

The fraction of random observations that we want to query relates to the number of available labeled observations $L(t)$. If $L(0)$ is small, then $\mathcal{L}(0)$ is less likely to be a good representation of the complete labeled dataset $(\mathbf{X}, \mathbf{y})$. In that case, we want to query relatively more random instances at the start to be able to obtain a representative train set for the GBM classifier to learn from. As $L(t)$ increases, we want to query less and less random observations and let Active Learning take over in the sense of querying anomalies and uncertainties. Therefore, let $\alpha_r(t)$ be an exponentially decreasing function in t. More specifically,

$$\alpha_r(t) = \alpha_r^{\max} \cdot 2^{-\tau \cdot L(t)}, \tag{9}$$

with $\tau > 0$ the decrease speed and $L(t) = L(0) + Q \cdot t$. Note that this function is determined beforehand. Hence, how the fraction of randomly queried observations changes, is fixed during the Jasmine procedure. The value of $\alpha^{\max}$ is directly determined by the hyperparameters $\alpha_a^{(0)}$ and $\alpha_z^{(0)}$.

It is important to see that executing the α-dynamic update step does not take much computation time. No additional machine learning models have to be trained and only relatively simple operations are performed to determine the values of the query fractions for the next time step.

4.7. Final iteration updates

The last steps that Jasmine has to perform are rather simple, the labeled query set $\mathcal{Q}(t)$ is added to the current labeled set $\mathcal{L}(t-1)$ and removed from the current unlabeled set $\mathcal{U}(t-1)$. More specifically, $\mathcal{L}(t) := \mathcal{L}(t-1) \cup \mathcal{Q}(t)$ and $\mathcal{U}(t) := \mathcal{U}(t-1) \setminus \mathcal{Q}(t)$. The complete Jasmine procedure is summarized in Algorithm 1.

Algorithm 1 Jasmine procedure

Require: Labeled set $\mathcal{L}(0)$, unlabeled set $\mathcal{U}(0)$, number of iterations T, query function ψ^{Jas}

```
Tune parameters of GBM on L(0) (Section 4.2)
Tune Jasmine-specific hyperparameters (α_a^(0), β, γ, τ) on L(0) (Section 4.8) and determine α_z^(0) by normalization
for t = 1 to T do
    Train GBM f_t with tuned parameters on L(t − 1)
    Apply f_t to U(t − 1) to obtain predictions ŷ_u(t)
    Assign each u ∈ U(t − 1) to its most likely class
    Calculate each z_u(t) as defined in (1)
    Construct one IF for each class (Section 4.4).
    Compute each a_u(t) as defined in (3)
    Compose Q(t) using query function ψ^Jas as described in Section 4.5
    Obtain actual classes y_q of q ∈ Q(t) by human expert
    Use y_q and predictions ŷ_q(t) to determine δ_a^β(t), δ_z^β(t) and Δ^γ(t) with (4), (5) and (6), respectively.
    Update query fractions to obtain α_a(t+1) and α_z(t+1) (Section 4.6)
    L(t) ⇐ L(t − 1) ∪ Q(t) and U(t) ⇐ U(t − 1) \ Q(t)
end for
return
```

4.8. Tuning jasmine hyperparameters

Chronologically speaking, tuning of the Jasmine-specific hyperparameters takes place in Phase 1 before the actual Active Learning procedure in Phase 2, as we illustrated in Fig. 3. To be more specific, *Jasmine tuning* occurs after tuning the hyperparameters for the GBM, and thus, directly after Section 4.2. Recall that the hyperparameters are *(i)* $\alpha_a^{(0)}$ and $\alpha_z^{(0)}$, the starting fractions of querying anomalous and uncertain observations, respectively; *(ii)* β, the parameter assigning a certain weight to an FN compared to an FP, given in (4) and (5); *(iii)* γ, the update magnitude, given in (6); and *(iv)* τ, the decrease speed in querying random observations, given in (9). Using normalization, $\alpha_a^{(0)}$ and τ completely determine $\alpha_z^{(0)}$, so the latter does not have to be tuned.

To determine appropriate values for these hyperparameters, the initially labeled set $\mathcal{L}(0)$ is randomly partitioned into the sets $\mathcal{L}_J(0)$, $\mathcal{U}_J(0)$ and $\mathcal{E}_J$. During Jasmine tuning, $\mathcal{L}_J(0)$ is taken as the initially labeled set, $\mathcal{U}_J(0)$ as the initially unlabeled set and $\mathcal{E}_J$ as the evaluation set. Let $\mathcal{J}$ be the set with hyperparameter values that we want to consider for tuning. Thus, an element $j \in \mathcal{J}$ is a four-dimensional vector of the form $(\alpha_a^{(0)}, \beta, \gamma, \tau)$. Let Q_J be the number of unlabeled observations that should be queried in each iteration. Ideally, the value of Q_J is close to Q, but we do want to perform some iterations before $\mathcal{L}_J(t)$ reaches the size of $\mathcal{L}(0)$. The number of iterations in Jasmine tuning is given by $T_J := \lceil \frac{U_J(0)}{Q_J} \rceil - 1$, where $U_J(0) := |\mathcal{U}_J(0)|$. The minus one is because the last unlabeled observations are the least informative, and hence, not of interest to query.

Let $j \in \mathcal{J}$ be some hyperparameter combination. Then a GBM model is trained on $\mathcal{L}_J(0)$, similar to what we described in Section 4.3. This model is then applied to the evaluation set $\mathcal{E}_J$ resulting in some performance metric $p_j(0)$. Then the rest of the AL procedure, as we described in Section 4.4 to 4.7, is executed. After all iterations are performed, the sequence of performance measures $\{p_j(t-1)\}_{\{t=1,\ldots,T_J\}}$ is obtained. We use this sequence to determine which hyperparameter combination works best, since such a sequence is obtained for each $j \in \mathcal{J}$.

Because stochasticity is involved, we repeat Jasmine tuning S_J times. Each simulation, the set $\mathcal{L}(0)$ is randomly divided in the sets $\mathcal{L}_J(0)$, $\mathcal{U}_J(0)$ and $\mathcal{E}_J$. Also, each simulation yields a sequence of performance metrics for each hyperparameter combination. Then the combination that yields the best performance over the simulations is taken as (relatively) optimal for $\alpha_a^{(0)}$, β, γ and τ.

5. Experimental setup

We conducted several experiments to determine whether our AL method Jasmine performed better than ALADIN and to decide if α-dynamic updating yielded significant improvements over baseline query functions. In this section, firstly, we discuss the datasets on which the experiments were performed. These sets are fully labeled, so the labels for the observations in the 'unlabeled' set were hidden until they were queried by Jasmine. After that, we explain the steps taken to execute the procedures. These include which hyperparameters were tuned over which ranges and how the different AL methods were evaluated.

5.1. Data

Yavanoglu and Aydos (2017) and Ferrag, Maglaras, Moschoyiannis, and Janicke (2020) provided overviews of publicly available security-related datasets commonly used in intrusion detection research. The sets that were discussed span from 1999 to 2018, showing that even old network datasets are still used to benchmark intrusion detection techniques. However, this is mostly due to the lack of public data, as discussed in Section 1. Here, firstly, we used the *NSL-KDD* dataset for assessing the considered AL methods, since it is the most used for evaluation in the field of cybersecurity. Secondly, we considered the *UNSW-NB15* dataset. This set is cited often too, and is much more recent than the popular NSL-KDD data. Moreover, it has several more realistic aspects, which are discussed later. Henceforth, these datasets were used to assess the performance of the discussed AL methods and to compare the obtained results for different data.

5.1.1. NSL-KDD

The NSL-KDD dataset was developed by Tavallaee, Bagheri, Lu, and Ghorbani (2009) and is an improvement on the KDD-Cup-99 dataset. The latter was prepared by Stolfo, Fan, Lee, Prodromidis, and Chan (2000) and consists of a train set and a test set. These two sets were constructed such that they do not have the same underlying distribution. Each observation in KDD-Cup-99 is made up of a 41-dimensional feature vector and an output label with the attack type. There are four global types of actual attacks and a 'normal' type, indicating a benign connection. Within the attack types, there can be several distinct attack scenarios. The test set contains scenarios from the train set as well as new scenarios. As mentioned before, the NSL-KDD dataset is a revision of the KDD-Cup-99 data and addresses many of the problems. What these problems are and how the data was revised is described by Tavallaee et al. (2009). Finally, the total number of train observations in NSL-KDD is 125,972 and the number of test instances is 22,544. We removed the categorical features Protocol_type, Service, Flag, Difficulty_level, because Jasmine is based on numerical techniques. Moreover, since Jasmine expects binary output labels, all attack types obtained value 1 (the malicious class), while the normal type obtained value 0 (the benign class). 46.5% of the train instances are malicious, while 56.9% are malicious in the test set.

5.1.2. NSL-KDD-rand

Besides considering the NSL-KDD data as provided by Tavallaee et al. (2009), we also considered the dataset we call *NSL-KDD-rand*. We constructed this set by combining the train and test set of the NSL-KDD dataset. Now, 48.1% of all observations are malicious. During the experiments, each time a new train and test set were chosen. These new sets are expected to have the same underlying distribution, in contrast to the provided train and test set of the NSL-KDD data.

5.1.3. UNSW-NB15

The UNSW-NB15 dataset was constructed by Moustafa and Slay (2015), partially to address some problems of the NSL-KDD data. The UNSW-NB15 dataset contains 2,540,047 observations with each network connection consisting of a 47-dimensional feature vector and two output attributes. The first output is the specific attack type and the second is a binary value indicating whether the observation is benign (0) or malicious (1). Nine attack types are present in the dataset, but we only used the second output attribute, since Jasmine expects a binary output label. We reduced the number of predictive features from 47 to 36 by removing srcip, sport, dstip, dsport, proto, state, service, stcpb, dtcpb, Stime and Ltime, because they directly determine the output label, are categorical or have no predictive use. Furthermore, there are several missing values in UNSW-NB15 which we chose from context to be 0. Finally, 12.6% of the observations correspond to attacks. This lower fraction of malicious traffic is one of the reasons why this dataset is closer to reality than the NSL-KDD data. More information about UNSW-NB15 is given by Moustafa and Slay (2015).

Although the attack balance is more realistic, there are still relatively many malicious observations. Therefore, we also constructed a dataset in which the attacks were downsampled to 1.0% to see how Jasmine performs on highly unbalanced data.

5.2. Experiments

5.2.1. Query functions

We considered several query functions in the experiments. First of all, we regarded Jasmine with its characteristic α-dynamic query function ψ^{Jas} (as described in Section 4.5) as the main focus of this research (jas.main). Furthermore, we examined some simpler query functions for the Jasmine procedure: only querying anomalies (jas.anom), only querying uncertainties (jas.uncert), and only querying random observations (jas.rand). For these three query functions, α-dynamic updating is not involved. Note that the uncertainty query approach in jas.uncert corresponds to many of the studies discussed in Section 2. Also, the query strategies in jas.anom and jas.rand have ties to related work. Naturally, we also considered the full ALADIN procedure of Stokes et al. (2008) (ala.main), just as the incomplete Jasmine procedure with ALADIN's query function of querying anomalies and uncertainties in a fixed 50/50 split (jas.basic). Consequently, we compared Jasmine to five different AL methods.

5.2.2. Global parameters

The global parameters are the variables defined before any computation took place. These include the initial size of the labeled set $L(0)$, the initial size of the unlabeled set $U(0)$, the size of the evaluation set E and the query set size Q. Moreover, N is the maximum number of observations that were to be queried to the human expert during the process. Together with Q, the parameter N determines the total number of iterations: $T := \lfloor N/Q \rfloor$. Finally, since Jasmine is an inherently stochastic procedure, we repeated the experiments S times. This was done to analyze how our method behaved on average and in what range its performances resided. Note that, for each repetition, the sets $\mathcal{L}(0)$ and $\mathcal{U}(0)$ were newly constructed. For the NSL-KDD-rand and UNSW-NB15 datasets, the evaluation set $\mathcal{E}$ was also freshly sampled. This was not necessary for the NSL-KDD data, since $\mathcal{E}$ is a provided fixed set.

The values chosen for the global parameters for the experiments are presented in Table 1. As the table shows, two different values $L(0)$ were considered, because we were interested in how the initially labeled set size influenced the performance of the AL methods. Furthermore, we chose Q to be 40, as we deemed this a good balance between allowing for the query fractions to update not too erratically (needing Q to be large) and allowing for relatively small updates to the classifier (needing Q to be small). Also, we chose N to be 15,000 because we saw from exploratory studies that the models do not drastically change anymore when more labels were provided. Finally, for the NSL-KDD-rand and UNSW-NB15 datasets, we chose E to be 5,000, to obtain a representative test set without using too many observations only for evaluation.

Table 1
Values for global parameters.

Data	$L(0)$	$U(0)$	E	Q	N	S
NSL-KDD	125	125,848	22,544	40	15,000	30
NSL-KDD	250	125,723	22,544	40	15,000	30
NSL-KDD-rand	125	146,392	5,000	40	15,000	30
NSL-KDD-rand	250	146,267	5,000	40	15,000	30
UNSW-NB15	125	2,534,922	5,000	40	15,000	30
UNSW-NB15	250	2,534,797	5,000	40	15,000	30

Table 2
Tuning ranges for hyperparameters in `h2o.gbm` (csr = 'col_sample_rate').

Distribution Bernoulli	Histogram_type RoundRobin	Learn_rate_annealing {0.95, 0.99, 0.999}
max_depth {6, 12, 24}	sample_rate {0.60, 0.78, 1.0}	ntrees {250, 500, 1,000}
nbins {10, 16, 25}	csr {0.84, 0.92, 1.0}	learn_rate {0.02, 0.05, 0.125}
min_rows {6, 8, 10}	csr_per_tree {0.40, 0.64, 1.0}	csr_change_per_level {0.94, 1.0, 1.06}

Table 3
Tuning ranges for Jasmine parameters.

$\alpha_a^{(0)}$	β	γ	τ	S_J
$\left\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\}$	$\left\{\frac{1}{2}, 1, 2\right\}$	$\left\{\frac{1}{2}, 1, 2\right\}$	$\left\{\frac{1}{800}, \frac{1}{400}, \frac{1}{200}, \frac{1}{100}\right\}$	4

5.2.3. Hyperparameter tuning GBM

As mentioned in Section 4.2, good values for the GBM were found by tuning on the initially labeled set $\mathcal{L}(0)$ with k-fold cross validation. We used the study by Tama and Rhee (2019) and exploratory research to determine which hyperparameters to tune and over what range to tune them. The parameters that were not selected for tuning obtained their default settings as given by the `h2o.gbm` function of the `H2O.ai` package, which we used in the `R` programming language. The values or tuning ranges of the parameters are shown in Table 2. Since the number of possible hyperparameter combinations is large (more than 177 thousand), tuning was performed using a random search over all combinations for a maximum of 4 h. The combination with the best trade-off between performance metric and computation time was chosen. We considered the F_1 score as the performance metric and we chose the threshold ε to be 10^{-4}.

5.2.4. Jasmine tuning

The relevant hyperparameters for the Jasmine tuning phase, as we described in Section 4.8, are the initial anomaly query fraction $\alpha_a^{(0)}$, the FN weight factor β, the update magnitude γ and the decrease speed in querying random instances τ. The ranges that the parameters were tuned over are shown in Table 3, yielding 108 possible combinations. We chose the ranges for $\alpha_a^{(0)}$, β and γ to be symmetric around the 'unity value'. For $\alpha_a^{(0)}$, this is $\frac{1}{2}$, since then the initial number of anomalous observations was equal to the number of uncertain instances. For β, this is 1, because then the FNs and FPs were weighed equally. For γ, this is also 1, since then $\Delta^{(\gamma)}(t)$ reduced to the linear difference $\Delta(t)$.

During Jasmine tuning, we wanted to choose Q_J as close to Q as possible, but also perform at least three iterations. Hence, we defined the tuning query size as $Q_J := \min\{U_J(0)/4, Q\}$. The initially unlabeled set size was divided by 4 (= 3 + 1), since the last iteration was not performed. This is because we deem the last unlabeled observations the least informative. Each Jasmine parameter combination j yielded a performance metric for every time step. This produced the sequence $\{p_j(t)\}_{\{t=0,\dots,T_J-1\}}$. Again, this metric was the F_1 score. After the iterations were performed, the area underneath the $(t, p_j(t))_{\{t=0,\dots,T_J-1\}}$-'curve' was calculated. This is an example of a *learning curve*, which is commonly used in the AL paradigm to assess the quality of a method (Kumar & Gupta, 2020; Settles, 2009). The larger this area, the better parameter combination j is. As there is stochasticity in the techniques used in Jasmine, the tuning phase was repeated S_J times. The combination with the largest average area was chosen as the hyperparameter setting for Jasmine in the actual AL procedure. Note that the Jasmine-specific parameters were tuned on $\mathcal{L}(0)$, and so, Jasmine did not get an unfair advantage by seeing more data in advance than the other AL methods, which did not need to execute Jasmine tuning.

5.2.5. Evaluation of AL methods

To evaluate the six different AL methods, we utilized the evaluation set $\mathcal{E}$ that was set aside each simulation. The quality of the predictions of an AL method on $\mathcal{E}$ was determined by the performance metric $p(t)$ (in this case the F_1 score) for every iteration t. After some reference value t_{ref} of iterations were performed ($0 \leq t_{\text{ref}} \leq T$), a sequence of performance metrics $\{p(t)\}_{\{t=0,\dots,t_{\text{ref}}\}}$ was obtained. Similar to Jasmine tuning, we took the area $A(t_{\text{ref}})$ underneath the $(t, p(t))_{\{t=0,\dots,t_{\text{ref}}\}}$-learning curve as a measure of performance. Briefly said, the higher $A(t_{\text{ref}})$, the better the method is up to the iteration step t_{ref}. However, since there is stochasticity involved, we repeated each complete AL procedure S times. During simulation s, $\mathcal{L}(0)$ and $\mathcal{U}(0)$ were randomly chosen (for NSL-KDD-rand and UNSW-NB15 also $\mathcal{E}$ was randomly sampled) and all six procedures were provided the same initial sets. Consequently, this led to the vector $(A_s^{(1)}(t_{\text{ref}}), \dots, A_s^{(6)}(t_{\text{ref}}))$ of paired area metrics. Next, we statistically compared the area metrics of the Jasmine α-dynamic method with the metrics of the other methods. This was done by the Wilcoxon signed-rank test (with significance threshold of 0.05) to determine whether

$$H_0^{(m)}(t_{\text{ref}}) : \underset{s=1,\dots,S}{\text{median}} \left\{A_s^{(1)}(t_{\text{ref}})\right\} < \underset{s=1,\dots,S}{\text{median}} \left\{A_s^{(m)}(t_{\text{ref}})\right\} \tag{10}$$

could be rejected for method $m = 2, \dots, 6$. When this was the case, then α-dynamic querying performed significantly better than the other considered query functions and AL techniques.

6. Results

In this section, the results of our research are presented. They were obtained by performing the steps explained in Section 5 on the NSL-KDD, NSL-KDD-rand and UNSW-NB15 data. First of all, the learning curve of F_1 scores is shown for each of the six AL methods that we considered. This curve gives an insight in how well the classifier of a specific method performed on the evaluation set throughout the AL process. Secondly, the p-values of the Wilcoxon signed-rank test are presented for predetermined specific iteration steps. Thirdly, the dynamics of the query fractions $\alpha_a(\cdot)$, $\alpha_z(\cdot)$ and $\alpha_r(\cdot)$ are shown to illustrate how Jasmine adjusted the balance between querying anomalous, uncertain and random observations. Finally, we discuss the implications of these results per dataset.

6.1. Results on NSL-KDD

6.1.1. Learning curves

Fig. 4 shows the average learning curves for each of the six AL methods on the fixed evaluation set NSL-KDD-$\mathcal{E}$ for initially labeled set sizes $L(0) = 125$ and $L(0) = 250$. Each simulation yielded a learning curve, hence we took the average of the curves over all simulations.

Table 4
p-values Wilcoxon test jas.main vs. ... with $L(0) = 125$.

t_{ref}	$L(t_{\text{ref}})$	jas.basic	jas.rand	jas.anom	jas.uncert	ala.main
9	485	*0.992*	0.000172	*1.00*	0.0155	0.894
16	765	*0.991*	$1.52 \cdot 10^{-5}$	*1.00*	0.0249	0.381
22	1005	*0.975*	$5.96 \cdot 10^{-7}$	*1.00*	0.00983	0.0571
34	1485	0.855	$5.00 \cdot 10^{-7}$	*0.971*	0.000729	0.000128
47	2005	0.786	$5.00 \cdot 10^{-7}$	0.388	0.000932	$4.99 \cdot 10^{-7}$
122	5005	0.0131	$2.99 \cdot 10^{-6}$	$1.28 \cdot 10^{-7}$	0.131	$1.30 \cdot 10^{-8}$
247	10005	$3.46 \cdot 10^{-6}$	$3.96 \cdot 10^{-5}$	$1.86 \cdot 10^{-9}$	0.908	$1.57 \cdot 10^{-7}$
372	15005	$7.10 \cdot 10^{-7}$	0.000190	$1.86 \cdot 10^{-9}$	*0.975*	$2.86 \cdot 10^{-7}$

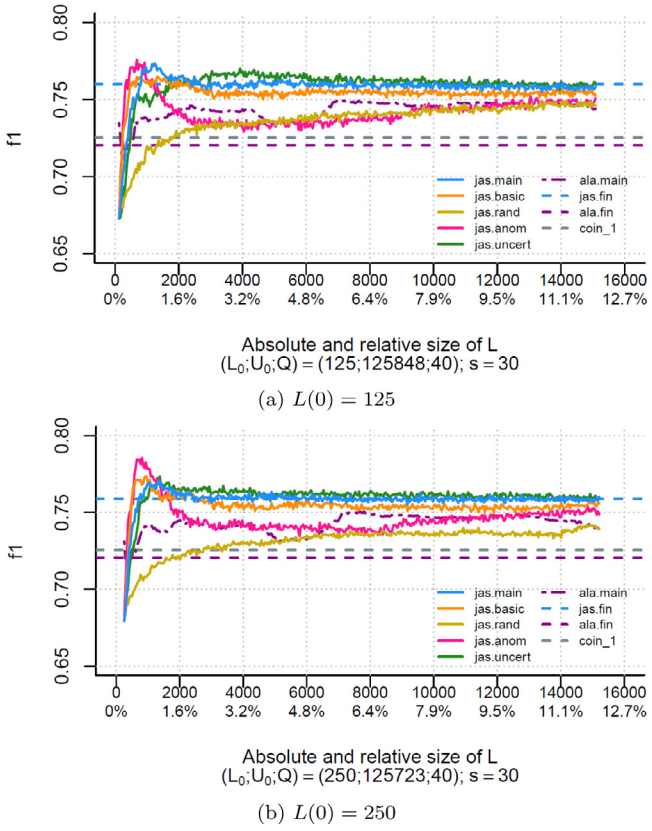


Fig. 4. Learning curves on NSL-KDD-$\mathcal{E}$ for different initial sizes $L(0)$.

The blue dashed line (jas.fin) is the average final performance on $\mathcal{E}$ of the GBM trained on the complete train set with all labels available. This performance metric was $\overline{F_1} \approx 0.760$ for Fig. 4(a) and $\overline{F_1} \approx 0.759$ for Fig. 4(b). The value of the line is approximately the same for both plots, because the evaluation set $\mathcal{E}$ is fixed and independent of $\mathcal{L}(0)$. However, each GBM was tuned differently, leading to small differences. Since all labels of the NSL-KDD dataset are technically available, we included the final performance to show how quickly the Jasmine-based AL methods reached this value. The purple dashed line (ala.fin) is the final performance on $\mathcal{E}$ of the logistic regression classifier of the ALADIN procedure ($F_1 \approx 0.720$). This final performance was constant during the simulations and for both settings of $L(0)$, since the classifier of ALADIN is deterministic. Finally, the gray dashed line (coin_1) is the expected performance on $\mathcal{E}$ ($F_1 \approx 0.725$) of the best dummy classifier, which classifies each evaluation observation as malicious.

6.1.2. Statistical tests

To determine whether Jasmine performed significantly better than the other five AL methods, we determined the *p*-value of the test in (10) for each method $m = 2, \ldots, 6$ and for different values of t_{ref}. Tables 4 and 5 show the results for the experiments with $L(0) = 125$ and $L(0) = 250$, respectively. A black value indicates that Jasmine (jas.main) performed significantly better than the method in the corresponding column for the labeled set size $L(t_{\text{ref}})$. A value in italic, however, means that Jasmine performed significantly worse (by interchanging the sides of (10)). A gray value means that the test was indecisive and could not conclude whether Jasmine was better or worse.

6.1.3. α-dynamic updating

Finally, Fig. 5 presents the α-dynamic updating procedure of Jasmine for $L(0) = 125$ and $L(0) = 250$. Similar to the figure with the learning curves, every simulation resulted in a α-dynamic curve for each of the query fractions $\alpha_a(\cdot)$ (a.anom), $\alpha_z(\cdot)$ (a.uncert) and $\alpha_r(\cdot)$ (a.rand). We took the average of the query fractions to obtain the average α-dynamic curves. A shaded region indicates the interval in which 80% of the observed fractions with matching color resided, and therefore, shows the spread of the values. Since $S = 30$ simulations were performed, each region contains the 24 'middle' fraction values.

6.1.4. Implication of results on NSL-KDD

The first thing that we observe in Fig. 4 is that the average F_1 score rapidly increased in the first iterations for the Jasmine (jas.main), jas.anom, jas.uncert and jas.basic procedures. This means that by using the corresponding query approaches, valuable unlabeled observations were queried to the oracle, because the GBMs were able to make better predictions on the evaluation set NSL-KDD-$\mathcal{E}$. This increase is most notable for jas.anom and then especially for $L(0) = 250$. This makes sense, because NSL-KDD-$\mathcal{E}$ contains new anomalous attack scenarios that are not found in the train set. Remarkably, the performance of jas.anom went higher than the final (average) F_1 score. For several iteration steps, also the other three methods obtained scores higher than the final performance for both settings of $L(0)$. This means that a carefully constructed smaller dataset led to better predictions on the evaluation set than the complete train set did.

Even though jas.anom performed better than the other methods at the start of the iterations, its effectiveness decreased when more anomalous observations were added to the labeled set. The decrease of effectiveness is also visible in Tables 4 and 5: for small values of t_{ref} querying only anomalies performed better than Jasmine, but later Jasmine obtained significantly better results. It could be that the labeled set became more and more abnormal when more anomalous observations were added, resulting in impaired training of the GBM classifier in later iterations after it had improved before.

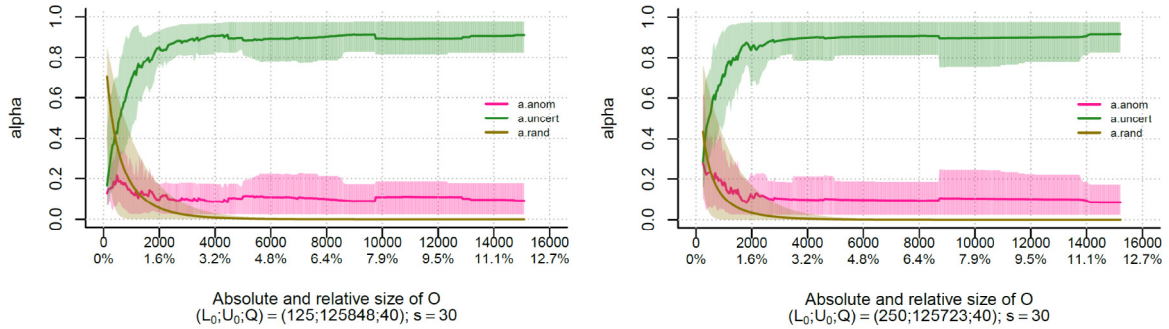
Furthermore, it is clear that Jasmine performed better than only querying uncertainties, as the *p*-values show. It appears that also querying anomalies is important. However, when time progresses, its performance is not significantly better anymore and eventually becomes significantly worse. It should be stressed, though, that the learning curves show that both jas.main and jas.uncert have converged to the final score and only differ a little.

Only querying random observations, as done by jas.rand, performed significantly worse than Jasmine for all reference iterations. This shows that specifically choosing anomalous or uncertain observations works better than only querying random observations.

Also interesting to note is that Jasmine was on par with or worse than jas.basic at the start of the iteration procedure. However, when the

Table 5
p-values Wilcoxon test jas.main vs. ... with $L(0) = 250$.

t_{ref}	$L(t_{\text{ref}})$	jas.basic	jas.rand	jas.anom	jas.uncert	ala.main
9	610	0.924	0.000128	*0.994*	0.0790	0.516
16	890	*0.958*	$2.57 \cdot 10^{-6}$	*1.00*	0.118	0.0502
22	1130	*0.963*	$4.99 \cdot 10^{-7}$	*1.00*	0.122	0.00530
34	1610	0.915	$1.28 \cdot 10^{-7}$	*0.999*	0.388	$3.14 \cdot 10^{-5}$
47	2130	0.897	$1.02 \cdot 10^{-7}$	*0.967*	0.556	$1.38 \cdot 10^{-6}$
122	5130	0.122	$4.00 \cdot 10^{-8}$	0.00233	*0.997*	$9.31 \cdot 10^{-9}$
247	10130	$9.43 \cdot 10^{-5}$	$4.16 \cdot 10^{-7}$	$8.20 \cdot 10^{-8}$	*1.00*	$8.20 \cdot 10^{-8}$
372	15130	$1.89 \cdot 10^{-6}$	$1.90 \cdot 10^{-6}$	$2.79 \cdot 10^{-9}$	*1.00*	$1.62 \cdot 10^{-6}$



(a) $L(0) = 125$ (b) $L(0) = 250$

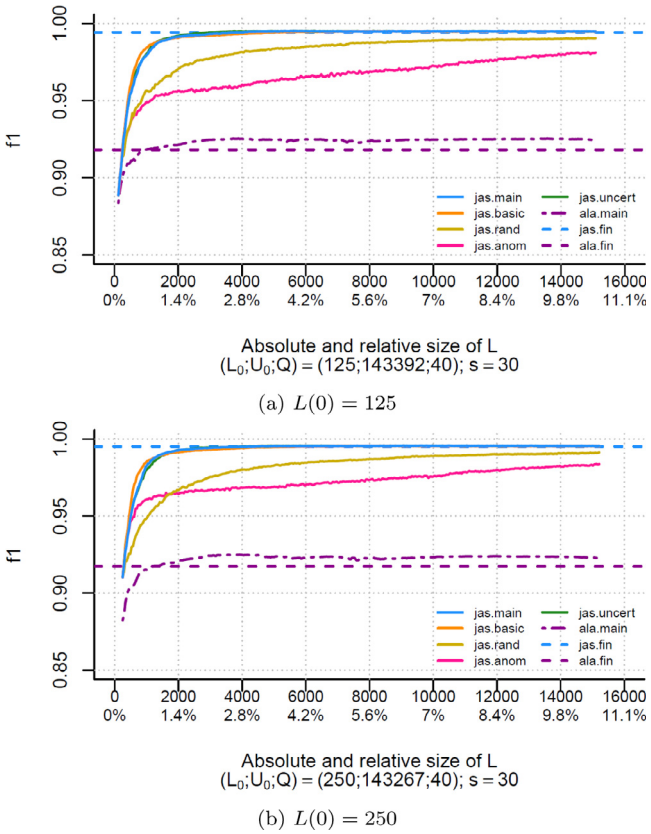
Fig. 5. Progress of query fractions for different initial sizes.

size of $\mathcal{L}(\cdot)$ grew, Jasmine became significantly better, as the p-values in both Tables 4 and 5 show. This means that dynamically adjusting the query balance in a later stadium has an advantage over querying anomalies and uncertainties in a fixed 50/50 fashion. Combining this with the progress of the query fractions in Fig. 5 and with the fact that jas.anom's performance worsens over time shows that Jasmine found the right balance between querying uncertainties and anomalies. Nonetheless, since anomalies appeared to be better at the start, it is curious why Jasmine did not query more anomalies in that stage. Hence, the information metrics as defined in (4) and (5) could have a preference for uncertain over anomalous observations.

Lastly, Jasmine performed fairly quickly significantly better than ALADIN as the p-values show. This is partly due to the simpler ML techniques in the latter, as it took Jasmine less effort to obtain better results than ALADIN than it took to perform better than jas.basic.

In general, there do not seem to be large differences between the experiments with $L(0) = 125$ and with $L(0) = 250$. However, the α-dynamic curves in Fig. 5 show that the average initial random query fraction $\alpha_r(0)$ was noticeably bigger for $L(0) = 125$ than for $L(0) = 250$, since the brown curve (a.rand) starts higher in the former. This makes sense, because $\mathcal{L}(0)$ was randomly constructed, and hence, it became less essential to query random instances when we chose $L(0)$ larger.

Considering the computation times, the procedures incorporating anomaly detection (jas.main, jas.basic and jas.anom) took significantly longer (on average 2.8 h for $L(0) = 125$ and 3.3 h for $L(0) = 250$) than those without the Isolation Forest (jas.rand, jas.uncert and ala.main) did (respectively 1.5, 1.8 and 2.0 h on average for $L(0) = 125$ and 2.0, 2.3 and 2.1 h for $L(0) = 250$). The longer computation times for $L(0) = 250$ compared to $L(0) = 125$ is presumably because of the hyperparameters chosen for the GBM, e.g., more trees are constructed, thus training takes longer. This is supported by the negligible difference in times for ALADIN, which does not incorporate a GBM. It is important to note that jas.main also performs Jasmine tuning, and therefore, requires additional computation time.



(a) $L(0) = 125$

(b) $L(0) = 250$

Fig. 6. Learning curves on NSL-KDD-rand with random evaluation set.

6.2. Results on NSL-KDD-rand

6.2.1. Learning curves

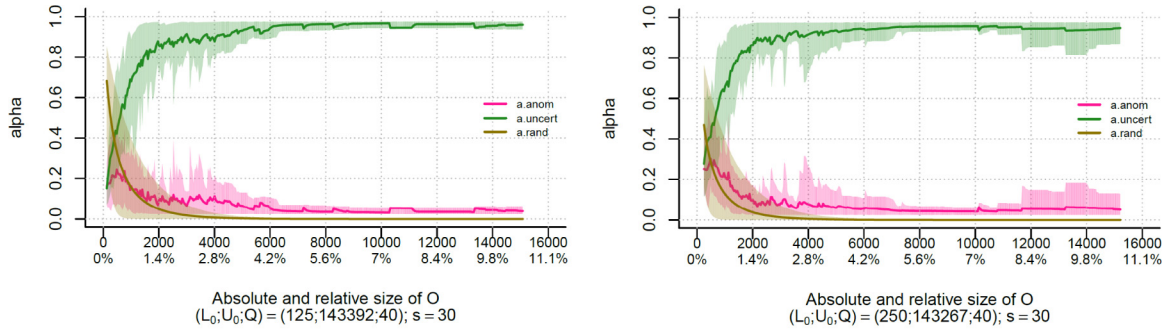
Similar to the results presented in Section 6.1, Fig. 6 presents the average learning curves for each of the six AL methods and two constant reference lines. In contrast to the results on the NSL-KDD dataset, the evaluation set $\mathcal{E}$ was chosen anew for each simulation. More specifically, at the start of each repetition the complete NSL-KDD-rand dataset was randomly partitioned in $\mathcal{L}(0)$, $\mathcal{U}(0)$ and $\mathcal{E}$. The blue

Table 6
p-values Wilcoxon test jas.main vs. ... with $L(0) = 125$.

t_{ref}	$L(t_{\text{ref}})$	jas.basic	jas.rand	jas.anom	jas.uncert	ala.main
9	485	0.798	0.000365	0.0192	0.657	$5.96 \cdot 10^{-7}$
16	765	0.886	$5.96 \cdot 10^{-7}$	0.000333	0.657	$1.86 \cdot 10^{-9}$
22	1005	0.904	$1.30 \cdot 10^{-8}$	$1.52 \cdot 10^{-5}$	0.642	$1.86 \cdot 10^{-9}$
34	1485	0.894	$9.31 \cdot 10^{-10}$	$8.20 \cdot 10^{-8}$	0.672	$9.31 \cdot 10^{-10}$
47	2005	0.831	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.708	$9.31 \cdot 10^{-10}$
122	5005	0.492	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.836	$9.31 \cdot 10^{-10}$
247	10005	0.428	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.846	$9.31 \cdot 10^{-10}$
372	15005	0.365	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.841	$9.31 \cdot 10^{-10}$

Table 7
p-values Wilcoxon test jas.main vs. ... with $L(0) = 250$.

t_{ref}	$L(t_{\text{ref}})$	jas.basic	jas.rand	jas.anom	jas.uncert	ala.main
9	610	*0.970*	$4.16 \cdot 10^{-7}$	0.313	0.930	$2.79 \cdot 10^{-9}$
16	890	*0.985*	$1.28 \cdot 10^{-7}$	0.0261	0.701	$1.86 \cdot 10^{-9}$
22	1130	*0.975*	$1.77 \cdot 10^{-8}$	$8.59 \cdot 10^{-4}$	0.548	$9.31 \cdot 10^{-10}$
34	1610	*0.963*	$9.31 \cdot 10^{-10}$	$5.96 \cdot 10^{-7}$	0.516	$9.31 \cdot 10^{-10}$
47	2130	0.945	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.564	$9.31 \cdot 10^{-10}$
122	5130	0.635	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.650	$9.31 \cdot 10^{-10}$
247	10130	0.444	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.687	$9.31 \cdot 10^{-10}$
372	15130	0.444	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.650	$9.31 \cdot 10^{-10}$



(a) $L(0) = 125$ (b) $L(0) = 250$

Fig. 7. Progress of query fractions for different initial sizes.

dashed line (jas.fin) is the average final performance of the simulation-specific GBMs on their corresponding evaluation sets. This metric was $\overline{F_1} \approx 0.994$ for Fig. 6(a) and $\overline{F_1} \approx 0.995$ for Fig. 6(b). This time, the average final performance of ALADIN represented by the purple dashed line (ala.fin) was no longer necessarily constant, but $\overline{F_1} \approx 0.918$ for initial set size $L(0) = 125$ and $\overline{F_1} \approx 0.917$ for $L(0) = 250$. Lastly, the expected performance of the best dummy classifier is not visible in the figures, since it was $\overline{F_1} \approx 0.650$ for $L(0) = 125$ and $\overline{F_1} \approx 0.651$ for $L(0) = 250$.

6.2.2. Statistical tests

Tables 6 and 7 show the *p*-values of the Wilcoxon signed-rank test with null hypothesis as given in (10) for different values of t_{ref}.

6.2.3. α-dynamic updating

The dynamics of the α-updating procedure of Jasmine are shown in Fig. 7 for $L(0) = 125$ and $L(0) = 250$. As described before, the curves represent the average query fractions $\alpha_a(\cdot)$ (a.anom), $\alpha_z(\cdot)$ (a.uncert) and $\alpha_r(\cdot)$ (a.rand) throughout the iteration process.

6.2.4. Implication of results on NSL-KDD-rand

At first glance, the results on the randomly selected evaluation sets of NSL-KDD-rand are much better for every considered AL method. This seems reasonable, because the fixed evaluation set for the NSL-KDD data contains unseen cyberattacks on which the classifier could not train. In the case of NSL-KDD-rand, the evaluation set was randomly chosen from the complete dataset, so we expected it to have the same structure as the train set, making it easier for the classifier to learn. This was specifically the case for the GBM, because it rapidly obtained an almost perfect F_1 score on the evaluation set.

The figures also show that the three learning curves for Jasmine, jas.basic and jas.uncert increased in a similar fashion at the start of the procedure. This was not the case for jas.anom, indicating that it was mostly important to query uncertain observations. This was also reflected in the *p*-values, as shown by Tables 6 and 7. Jasmine quickly performed significantly better than jas.anom, but it had more difficulty in obtaining significantly better results than jas.basic and jas.uncert. The latter was even significantly better than Jasmine. However, it should be noted that the F_1 scores were already near perfect for these three query approaches. Moreover, the dynamics of the query fractions in Fig. 7 show that there was a clear preference for querying more uncertainties than anomalous observations.

Furthermore, the learning curves show that Jasmine obtained better results than jas.rand and ALADIN. Both Tables 6 and 7 indicate that Jasmine was significantly better for all considered reference values.

6.3. Results on UNSW-NB15

6.3.1. Learning curves

Fig. 8 shows the average learning curves for every one of the six AL methods and two constant reference lines. The evaluation set $\mathcal{E}$ was chosen anew for each simulation, equivalent to what we discussed in Section 6.2. The blue dashed line (jas.fin) is the average final performance of the simulation-specific GBMs on their corresponding evaluation sets. For Fig. 8(a), this metric was $\overline{F_1} \approx 0.974$, while it was $\overline{F_1} \approx 0.973$ for Fig. 8(b). The average final performance of ALADIN

Table 8
p-values Wilcoxon test jas.main vs. ... with $L(0) = 125$.

t_{ref}	$L(t_{\text{ref}})$	jas.basic	jas.rand	jas.anom	jas.uncert	ala.main
9	485	***0.994***	$4.95 \cdot 10^{-5}$	0.0213	***0.994***	0.0701
16	765	***0.990***	$1.90 \cdot 10^{-6}$	$3.96 \cdot 10^{-5}$	***0.997***	$6.87 \cdot 10^{-5}$
22	1005	***0.985***	$9.96 \cdot 10^{-7}$	$4.00 \cdot 10^{-6}$	***0.992***	$3.46 \cdot 10^{-6}$
34	1485	***0.965***	$1.93 \cdot 10^{-7}$	$1.28 \cdot 10^{-7}$	***0.971***	$1.30 \cdot 10^{-8}$
47	2005	0.918	$4.00 \cdot 10^{-8}$	$6.52 \cdot 10^{-9}$	0.897	$9.31 \cdot 10^{-9}$
122	5005	0.815	$9.31 \cdot 10^{-9}$	$1.86 \cdot 10^{-9}$	0.612	$2.79 \cdot 10^{-9}$
247	10005	0.271	$5.12 \cdot 10^{-8}$	$1.86 \cdot 10^{-9}$	0.191	$1.86 \cdot 10^{-9}$
372	15005	0.122	$1.28 \cdot 10^{-7}$	$1.86 \cdot 10^{-9}$	0.131	$1.86 \cdot 10^{-9}$

Table 9
p-values Wilcoxon test jas.main vs. ... with $L(0) = 250$.

t_{ref}	$L(t_{\text{ref}})$	jas.basic	jas.rand	jas.anom	jas.uncert	ala.main
9	610	0.715	$3.46 \cdot 10^{-7}$	$7.99 \cdot 10^{-6}$	***0.989***	0.149
16	890	0.735	$5.96 \cdot 10^{-7}$	$1.90 \cdot 10^{-6}$	***0.998***	0.00128
22	1130	0.761	$4.16 \cdot 10^{-7}$	$2.36 \cdot 10^{-7}$	***0.997***	0.000230
34	1610	0.650	$9.31 \cdot 10^{-9}$	$1.77 \cdot 10^{-8}$	***0.998***	$1.52 \cdot 10^{-5}$
47	2130	0.350	$2.79 \cdot 10^{-9}$	$9.31 \cdot 10^{-10}$	***0.990***	$2.57 \cdot 10^{-6}$
122	5130	0.185	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.786	$9.31 \cdot 10^{-10}$
247	10130	0.0481	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.306	$9.31 \cdot 10^{-10}$
372	15130	0.0275	$9.31 \cdot 10^{-10}$	$9.31 \cdot 10^{-10}$	0.180	$9.31 \cdot 10^{-10}$

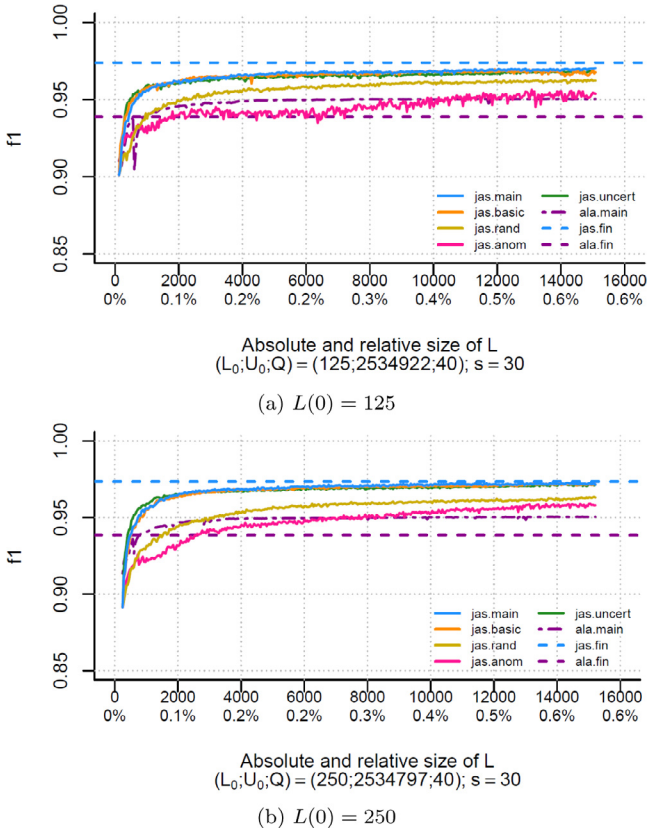


Fig. 8. Learning curves on UNSW-NB15 with random evaluation set for different $L(0)$.

indicated by the purple dashed line (ala.fin) was $\overline{F_1} \approx 0.939$ for both initial set sizes. Lastly, the figures do no show the expected performance of the best dummy classifier, since it was $\overline{F_1} \approx 0.225$ for $L(0) = 125$ and $\overline{F_1} \approx 0.222$ for $L(0) = 250$. These remarkably lower baseline values are because relatively far fewer positive observations are present in UNSW-NB15 compared to NSL-KDD.

6.3.2. Statistical tests

Tables 8 and 9 present the *p*-values of the Wilcoxon signed-rank test with null hypothesis as given in (10) for different values of t_{ref}.

6.3.3. α-dynamic updating

Fig. 9 shows the α-dynamic updating procedure of Jasmine for $L(0) = 125$ and $L(0) = 250$. Equivalent to the results in Sections 6.1 and 6.2, the curves indicate the average query fractions $\alpha_a(\cdot)$ (a.anom), $\alpha_z(\cdot)$ (a.uncert) and $\alpha_r(\cdot)$ (a.rand) throughout the iteration process.

6.3.4. Implication of results on UNSW-NB15

There are no striking differences between the two plots in Fig. 8. However, it seems that the average learning curves for initial set size $L(0) = 125$ are a bit more shaky. Since $L(0)$ is the only different global parameter between the two plots, the change in behavior is presumably due to how the GBM parameters and Jasmine-specific parameters were tuned. It probably also had to do with the imbalance of this dataset. Approximately 12.6% of the data is related to cyberattacks, so on average about 16 observations were malicious in $\mathcal{L}(0)$. For tuning purposes, this initial set was split in training, validation and test sets, making it possible that only one malicious observation ended up in any of those sets. Consequently, the tuned parameters possibly led to less stable behavior of the GBM and the α-dynamic update procedure. Hence, balancing techniques of the training data such as under- or oversampling could be considered to improve stability.

Furthermore, the learning curves in Fig. 8 are similar to those for NSL-KDD-rand. This is also true for the *p*-values presented in Tables 8 and 9 and the dynamics of the average α-update curves shown in Fig. 9.

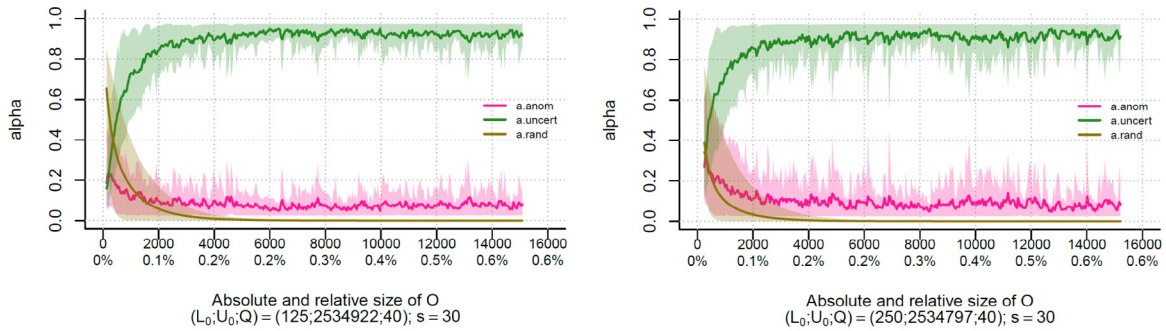
Additionally, the precise effect of the attack imbalance was studied by randomly downsampling the malicious observations in UNSW-NB15 from 12.6% to 1.0%. Fig. 10 shows the results of the experiments on this highly unbalanced dataset. The learning curves (averaged over 20 runs) show that all procedures struggled at the start of the labeling process: performance was much lower than in Fig. 8. However, jas.main, jas.basic and jas.uncert improve very quickly. Moreover, Jasmine is usually the best performing procedure, and for the uncommon instances it is not, it appears to be second best.

7. Discussion

In this final section of the paper, we firstly draw conclusions based on the implications of the results discussed in Section 6. Secondly, we consider further directions for AL in the field of network intrusion detection.

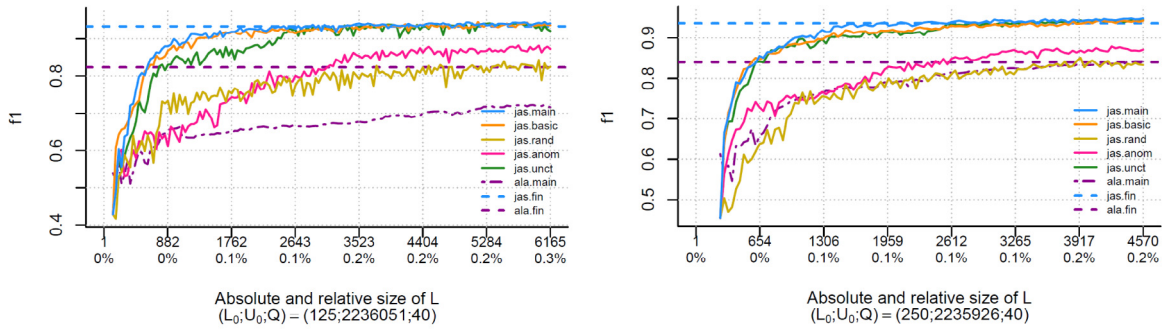
7.1. Conclusion

The goal of this research was to propose our hybrid Active Learning method Jasmine for network intrusion detection. It consists of



(a) $L(0) = 125$ (b) $L(0) = 250$

Fig. 9. Progress of query fractions for different initial size $L(0)$.



(a) $L(0) = 125$ (b) $L(0) = 250$

Fig. 10. Progress of query fractions for different initial size $L(0)$.

α-dynamic querying, which means Jasmine is able to dynamically adjust the balance between querying anomalous, uncertain and random observations. Consequently, only the potentially most interesting observations are presented to the human expert. This sets our method apart from other AL approaches that have a static query function.

On the datasets that we considered, Jasmine performed significantly better than ALADIN, the benchmark AL method used in this research. This means for practitioners that it is beneficial to choose Jasmine over ALADIN for their AL problems. However, we should note that Jasmine needs preprocessing time, which ALADIN and other static AL methods do not need. This is because of GBM tuning and Jasmine tuning. Furthermore, we observed that Jasmine performed more robustly with respect to the datasets. We compared the characteristic α-dynamic query function of Jasmine with other query functions: querying only anomalies, only uncertainties, only random observations and querying anomalous and uncertain observations in a fixed 50/50 fashion. We noticed that Jasmine did not always outperform the other approaches, but its performance was less influenced by the considered data, and hence, more robust. On the NSL-KDD dataset, only querying anomalies performed better when the labeling process had just started, but Jasmine took over in the long run. On the NSL-KDD-rand and UNSW-NB15 data, only querying uncertainties or querying in a 50/50 fashion reigned supreme, but Jasmine followed closely. Only querying anomalies was clearly a bad strategy for these two datasets. These findings suggest that Jasmine is better able to adapt to the provided labeled dataset $\mathcal{L}(\cdot)$. This is particularly interesting in the face of concept drift: NIDSs operate in high non-stationary contexts, which makes it wise to consider an AL method that is able to adjust its query approach dynamically. Lastly, we saw that the performance of Jasmine is not hindered by unbalanced data. Initially it does perform worse, but it quickly improves when the labeling procedure starts and it becomes the best or one of the best performing methods.

However, there is room for improvement in the way α-dynamic querying is performed here. The results on NSL-KDD show that at first querying more anomalous than uncertain observations was beneficial, but this was not reflected in the way that the query fractions were updated. The updating procedure seems to have a bias towards querying uncertainties.

7.2. Future work

Our first suggestion for further research is to reconsider the α-dynamic query process such that the bias towards selecting uncertain observations is eliminated.

Another suggestion is to add unlabeled observations about which the classifier has a high prediction certainty to the labeled set without asking the oracle for its label. Consequently, the labeled set increases much more during an iteration, while the human expert does not have to label more observations. This reduces labeling time drastically and possibly leads to better predictions in an earlier stage of the AL process. Moreover, when more labels become available, the preprocessing steps can be repeated (in a less extensive version) to allow for the GBM to better adjust to the changing train set by retuning its hyperparameters. Also, some Jasmine-specific parameters could be retuned during the process.

Our last suggestion is to consider human uncertainty in the AL method, since it is not always clear whether a connection is benign or malicious. This can be done by asking the oracle for their confidence in each label that they provide or by incorporating a general probability.

These suggestions would increase the deployment efficiency of Jasmine even more. Therefore, we would like to apply our method in a practical setting to see how it performs.

CRediT authorship contribution statement

Jan Klein: Conceptualization, Methodology (main developer), Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization,

Project administration. **Sandjai Bhulai:** Conceptualization, Methodology, Resources, Writing – review & editing. **Mark Hoogendoorn:** Conceptualization, Methodology, Resources, Writing – review & editing. **Rob van der Mei:** Conceptualization, Methodology, Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix

In this section, we provide the mathematical specifications about α-dynamic querying. Before we can give the explicit definitions of the query fractions for the next iteration, we have to derive some general restrictions on the values of the fractions $\alpha_r(t)$, $\alpha_a(t)$ and $\alpha_z(t)$ for all $t \in \{0, \dots, T\}$, where T denotes the maximum number of iterations. First, a fundamental requirement of the fractions is that

$$\alpha_r(t) + \alpha_a(t) + \alpha_z(t) = 1, \tag{A.1}$$

and $\alpha_r(t), \alpha_a(t), \alpha_z(t) \in [0, 1]$. Furthermore, each iteration, we want to query at least one anomaly and one uncertainty, otherwise (4) or (5) is undefined, and consequently, $\Delta^\gamma(t)$ is undefined. Thus, we need $\alpha_a(t)$ and $\alpha_z(t)$ to be at least $\alpha_{a,z}^{\min} := 1/Q$. Then, the number of anomalies and uncertainties in $Q(t)$ is at least $Q \cdot \alpha_{a,z}^{\min} = 1$. This means the upper bounds for $\alpha_a(t)$ and $\alpha_z(t)$ are restricted to be at most $1 - \alpha_{a,z}^{\min}$. The upper bound for $\alpha_r(t)$ is at most $\alpha_r^{\max} := 1 - 2\alpha_{a,z}^{\min}$, since we do allow to query no random observations, and hence, allow $\alpha_r(t)$ to go to 0. The definition of $\alpha_r(t)$ is given in (9). Note that $\alpha_r(t) \in [\alpha_r^{\min}, \alpha_r^{\max}) \subset [0, 1]$ with $\alpha_r^{\min} := \alpha_r(T)$. Now, we can define the upper bound for $\alpha_a(t)$ and $\alpha_z(t)$ as

$$\alpha_{a,z}^{\max}(t) := 1 - \alpha_r(t) - \alpha_{a,z}^{\min}.$$

Note that this upper bound depends on the iteration number t.

By using *(i)* $\alpha_r(t), \alpha_a(t), \alpha_z(t) \in [0, 1]$, *(ii)* (A.1), and *(iii)* the definitions of $\alpha_{a,z}^{\min}$, $\alpha_r^{\min}$, $\alpha_{a,z}^{\max}(t)$ and $\alpha_r^{\max}$, we characterize the w variables introduced in update rules (7) and (8) as follows:

$$\begin{aligned}
w_a^{(1)}(t) &= \alpha_{a,z}^{\max}(t) - \alpha_a(t) \\
w_a^{(2)}(t) &= \alpha_a(t) - \alpha_{a,z}^{\min} \\
w_z^{(1)}(t) &= \alpha_{a,z}^{\max}(t) - \alpha_z(t) \\
w_z^{(2)}(t) &= \alpha_z(t) - \alpha_{a,z}^{\min}.
\end{aligned}$$

Let us explain the specifics of the update of the anomaly fraction $\alpha_a(\cdot)$. We take the old value of $\alpha_a(t)$ as a starting point for $\alpha_a(t+1)$. This fraction can be increased by at most $w_a^{(1)}(t) = \alpha_{a,z}^{\max}(t) - \alpha_a(t)$ to obtain $\alpha_{a,z}^{\max}(t)$ as the new value. This increase $w_a^{(1)}(t)$ is scaled down by $\max\{0, \Delta^\gamma(t)\}$ such that the increment is proportional to the value of $\Delta^\gamma(t)$. Similarly, $\alpha_a(t)$ can be decreased by at most $w_a^{(2)}(t) = \alpha_a(t) - \alpha_{a,z}^{\min}$ to obtain $\alpha_{a,z}^{\min}$. The decrease $w_a^{(2)}(t)$ is then scaled down by $\min\{0, \Delta^\gamma(t)\}$. Hence, the constants ensure that $\alpha_a(t+1)$ lies in the interval $I_{a,z}(t) := [\alpha_{a,z}^{\min}, \alpha_{a,z}^{\max}(t)]$. However, it should lie in the interval $I_{a,z}(t+1) := [\alpha_{a,z}^{\min}, \alpha_{a,z}^{\max}(t+1)]$. This is why we apply the linear transformation $\lambda_{t+1} : I_{a,z}(t) \to I_{a,z}(t+1)$. This function is given by

$$\lambda_{t+1}(\alpha) = \frac{\alpha_{a,z}^{\max}(t+1) - \alpha_{a,z}^{\min}}{\alpha_{a,z}^{\max}(t) - \alpha_{a,z}^{\min}}(\alpha - \alpha_{a,z}^{\min}) + \alpha_{a,z}^{\min}. \tag{A.2}$$

Note that this function is not well-defined whenever $\alpha_{a,z}^{\max}(t) - \alpha_{a,z}^{\min} = 0$. This happens when $\alpha_r(t) = 1 - \alpha_{a,z}^{\min} = \alpha_r^{\max}$. However, the definition of $\alpha_r(t)$ in Eq. (9) shows that $\alpha_r(t)$ is strictly less than $\alpha_r^{\max}$, and hence the denominator in Eq. (A.2) cannot be zero and λ_{t+1} is well-defined.

Finally, for $t \in \{0, \dots, T-1\}$, the systems of equations for the three query fractions are given by

$$\begin{cases} \alpha_r(0) = \alpha_r^{\max} \cdot 2^{-\tau \cdot L(0)} \\ \alpha_r(t+1) = \alpha_r^{\max} \cdot 2^{-\tau \cdot (L(0) + Q \cdot (t+1))}, \end{cases}$$

$$\begin{cases} \alpha_a(0) = \alpha_a^{(0)} \\ \alpha_a(t+1) = \lambda_{t+1}\Big(\alpha_a(t) + (\alpha_{a,z}^{\max}(t) - \alpha_a(t)) \max\{0, \Delta^\gamma(t)\} \\ \qquad\qquad + (\alpha_a(t) - \alpha_{a,z}^{\min}) \min\{0, \Delta^\gamma(t)\} \Big), \end{cases}$$

and

$$\begin{cases} \alpha_z(0) = \alpha_z^{(0)} \\ \alpha_z(t+1) = \lambda_{t+1}\Big(\alpha_z(t) + (\alpha_{a,z}^{\max}(t) - \alpha_z(t)) \max\{0, -\Delta^\gamma(t)\} \\ \qquad\qquad + (\alpha_z(t) - \alpha_{a,z}^{\min}) \min\{0, -\Delta^\gamma(t)\} \Big). \end{cases}$$

References

Almgren, M., & Jonsson, E. (2004). Using active learning in intrusion detection. In *Proceedings. 17th IEEE computer security foundations workshop, 2004* (pp. 88–98). IEEE.

Budd, S., Robinson, E. C., & Kainz, B. (2021). A survey on active learning and human-in-the-loop deep learning for medical image analysis. *Medical Image Analysis*, Article 102062.

Caruana, R., Karampatziakis, N., & Yessenalina, A. (2008). An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on machine learning* (pp. 96–103).

Claesen, M., & De Moor, B. (2015). Hyperparameter search in machine learning. arXiv preprint arXiv:1502.02127.

Consultancy. eu (2020). Cost of cybercrime per incident jumps six-fold to €50, 000. URL https://www.consultancy.eu/news/4409/cost-of-cybercrime-per-incident-jumps-six-fold-to-50000.

Elahi, M., Ricci, F., & Rubens, N. (2016). A survey of active learning in collaborative filtering recommender systems. *Computer Science Review*, *20*, 29–50.

Ferrag, M. A., Maglaras, L., Moschoyiannis, S., & Janicke, H. (2020). Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, *50*, Article 102419.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 1189–1232.

Gadde, A., Gad, E. E., Avestimehr, S., & Ortega, A. (2016). Active learning for community detection in stochastic block models. In *2016 IEEE international symposium on information theory* (pp. 1889–1893). IEEE.

Görnitz, N., Kloft, M., Rieck, K., & Brefeld, U. (2009). Active learning for network intrusion detection. In *Proceedings of the 2nd ACM workshop on security and artificial intelligence* (pp. 47–54).

Gu, Y., & Zydek, D. (2014). Active learning for intrusion detection. In *2014 National wireless research collaboration symposium* (pp. 117–122). IEEE.

Guerra Torres, J. L., Catania, C. A., & Veas, E. (2019). Active learning approach to label network traffic datasets. *Journal of Information Security and Applications*, *49*, Article 102388.

Kumar, P., & Gupta, A. (2020). Active learning query strategies for classification, regression, and clustering: A survey. *Journal of Computer Science and Technology*, *35*(4), 913–945.

Lewis, D. D., & Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *SIGIR'94* (pp. 3–12). Springer.

Li, Y., & Guo, L. (2007). An active learning based TCM-KNN algorithm for supervised network intrusion detection. *Computers & Security*, *26*(7–8), 459–467.

Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In *2008 Eighth IEEE international conference on data mining* (pp. 413–422). IEEE.

Liu, F. T., Ting, K. M., & Zhou, Z. H. (2012). Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *6*(1), 1–39.

Mouloua, S. A., Ferraro, J., Mouloua, M., Matthews, G., & Copeland, R. R. (2019). Trend analysis of cyber security research published in HFES proceedings from 1980 to 2018. In *Proceedings of the human factors and ergonomics society annual meeting, Vol. 63, no. 1* (pp. 1600–1604). Los Angeles, CA: SAGE Publications Sage CA.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military communications and information systems conference* (pp. 1–6). IEEE.

Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, *7*, 21.

Ogutu, J. O., Piepho, H.-P., & Schulz-Streeck, T. (2011). A comparison of random forests, boosting and support vector machines for genomic selection. In *BMC proceedings, Vol. 5, no. S3* (p. S11). Springer.

Pelleg, D., & Moore, A. (2004). Active learning for anomaly and rare-category detection. *Advances in Neural Information Processing Systems*, *17*, 1073–1080.

Settles, B. (2009). *Active learning literature survey*: *Technical report*, University of Wisconsin-Madison Department of Computer Sciences.

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy* (pp. 305–316). IEEE.

Stokes, J. W., Platt, J., Kravis, J., & Shilman, M. (2008). Aladin: Active learning of anomalies to detect intrusions.

Stolfo, S. J., Fan, W., Lee, W., Prodromidis, A., & Chan, P. K. (2000). Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In *Proceedings DARPA information survivability conference and exposition, Vol. 2* (pp. 130–144). IEEE.

Stolfo, S., et al. (1999). KDD cup 1999 dataset. UCI KDD Repository, URL http://kdd.ics.uci.edu.

Sultana, N., Chilamkurti, N., Peng, W., & Alhadad, R. (2019). Survey on SDN based network intrusion detection system using machine learning approaches. *Peer-To-Peer Networking and Applications*, *12*(2), 493–501.

Tama, B. A., & Rhee, K. H. (2019). An in-depth experimental study of anomaly detection using gradient boosted machine. *Neural Computing and Applications*, *31*(4), 955–965.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1–6). IEEE.

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., et al. (2018). Machine learning and deep learning methods for cybersecurity. *IEEE Access*, *6*, 35365–35381.

Yang, K., Ren, J., Zhu, Y., & Zhang, W. (2018). Active learning for wireless IoT intrusion detection. *IEEE Wireless Communications*, *25*(6), 19–25.

Yavanoglu, O., & Aydos, M. (2017). A review on cyber security datasets for machine learning algorithms. In *2017 IEEE international conference on big data* (pp. 2186–2193). IEEE.

Yin, L., Wang, H., & Fan, W. (2018). Active learning based support vector data description method for robust novelty detection. *Knowledge-Based Systems*, *153*, 40–52.

Zamani, M., & Movahedi, M. (2013). Machine learning techniques for intrusion detection. arXiv preprint arXiv:1312.2177.