Plusmine: Dynamic Active Learning with Semi-Supervised Learning for Automatic Classification

Jan Klein
j.g.klein@cwi.nl
CWI, Amsterdam, The Netherlands

Sandjai Bhulai
s.bhulai@vu.nl
Vrije Universiteit, Amsterdam, The Netherlands

Mark Hoogendoorn
m.hoogendoorn@vu.nl
Vrije Universiteit, Amsterdam, The Netherlands

Rob van der Mei
r.d.van.der.mei@cwi.nl
CWI, Amsterdam, The Netherlands

ABSTRACT

A major problem in cybersecurity research is the correct labeling of up-to-date datasets. It relies on the availability of human experts, and is as such very cumbersome. Motivated by this, two techniques have been proposed for efficient labeling: Active Learning (AL) and Semi-Supervised Learning (SeSL). In this paper, we introduce Plusmine: an intrusion detection method that combines the benefits of AL and SeSL to efficiently automate classification. We develop new techniques for both components. Moreover, we empirically show that Plusmine obtains good and more robust results than benchmark methods.

KEYWORDS

active learning, semi-supervised learning, network intrusion detection, automatic labeling, partially labeled

ACM Reference Format:
Jan Klein, Sandjai Bhulai, Mark Hoogendoorn, and Rob van der Mei. 2021. Plusmine: Dynamic Active Learning with Semi-Supervised Learning for Automatic Classification. In 20^{th} *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 14–17 December, 2021, Melbourne, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3486622.3493948

1 INTRODUCTION

Cybercrime has become one of the most influential forms of criminality. Cybersecurity Ventures predicted the global damage to be $6 trillion USD in 2021 and estimated this to increase to $10.5 trillion in the year 2025 [12]. The already rapid growth of cybercrime has been accelerated by the outbreak of the COVID-19 pandemic in early 2020, since the dependency on online communication became even larger. Fortunately, in academia, the interest in cybersecurity research has also grown [13]. Most studies are focused on protecting networks of computers by means of a Network Intrusion Detection System (NIDS) [19]. Although there are several well-known problems of NIDSs, such as either a high false positive rate or high false negative rate [21], Xin et al. state that there are more significant issues [26]. Most notably, there is a lack of realistic, up-to-date cyberdata, which makes it challenging to replicate results of newly developed techniques in actual computer network settings. This lack is caused by two problems: *(i)* issues in security and privacy and *(ii)* difficulties in labeling network observations for training purposes, because connections are diverse, dynamic and with many [20].

To mitigate the second problem, the labeling procedure of a cyberexpert could be optimized such that they only need to classify the connections that are expected to increase overall detection performance the most. After labeling and adding these observations, the model can retrain itself on the increased dataset and select the next batch of interesting connections. This description characterizes *Active Learning* (AL). Although AL is an efficient approach to accelerate learning, it is still restricted by the labeling speed of the cyberanalyst, since the time it takes to correctly classify a network connection can fluctuate drastically [25]. It is, therefore, beneficial if the method itself could exploit the rich set of unlabeled data and could automatically classify specific observations. This is where *Semi-Supervised Learning* (SeSL) comes into play. Each iteration, a subset of observations is specifically selected for Automatic Classification (AC) and is then added to the labeled set. For example, a connection is automatically labeled when the confidence of the model in its prediction exceeds some threshold [24]. SeSL techniques allow the intrusion detection method to quickly expand the labeled pool and increase performance without directly questioning the cyberanalyst.

In this paper, we propose a novel NIDS called *Plusmine* that combines AL and SeSL. Our *Active Semi-Supervised Learning* (ASeSL) method consists of an improved state-of-the-art AL component and a new transductive SeSL approach that considers the expected consequences of labeling observations in the next time step and uses this to determine which connections are the best candidates for AC. This is in contrast to other ASeSL methods for network intrusion detection [10, 11, 17, 24, 29]. Our AC strategy is both powerful and simple. The AL component of Plusmine is an improved version of the Jasmine method [6]. Jasmine incorporates a dynamic query function that allows the model to learn the best query approach during the labeling process. This makes it an adaptable and robust method. However, we make an important improvement to Jasmine to fix the bias it has towards querying a certain type of observations. We apply Plusmine to two popular network intrusion detection datasets that we use in four dataset configurations. We demonstrate

WI '21, 14–17 December 2021, Melbourne, Australia

ACM ISBN 978-1-4503-9115-3/21/12.
https://doi.org/10.1145/3486622.3493948

that Plusmine obtains good, more robust and more reliable results compared to benchmark methods.

2 RELATED WORK

2.1 Active Learning

Active Learning is a type of Machine Learning (ML) in which the learner can interactively query an oracle to classify certain unlabeled observations. The procedure starts by training a classifier on the (small) set of labeled data. Then, the *query function* determines which observations from the unlabeled pool are proposed to the oracle. This expert provides the true labels for the query instances and these are then added to the labeled set. After that, the process repeats itself with the expanded labeled pool. It ends when some predetermined stop criterion is met. An important advantage of AL is that less effort needs to be spent on labeling data [18]. This is especially useful in fields where it is laborious to do so, such as in cybersecurity.

2.1.1 Network Intrusion Detection. Several AL techniques have been proposed in cybersecurity research. A common query strategy is *uncertainty sampling* [4, 9, 27]. Firstly, the model determines its confidence in the prediction of each unlabeled observation. Then, the most uncertain instances are queried to the oracle. Although uncertainty sampling is simple, Yang et al. show that only one third of the labels are required to achieve the same performance as randomly querying [27]. Li and Guo results are even more impressing as one fiftieth of the labels are necessary for the same accuracy [9]. The uncertainty query function is *static*, because the way in which the observations are selected is fixed throughout the labeling process. A query function can also be *dynamic* by using the knowledge it obtains during the AL procedure to adjust the selection approach. Hence, it learns which types of observations are most informative for a certain iteration and is therefore more robust to changes in data. Jasmine is one of the few methods, and state-of-the-art, to propose such a dynamic query function [6]. Klein et al. show that Jasmine obtains more robust results over several datasets than static query functions do. However, as with all AL methods, the only source of labels is the oracle, and hence, the model is limited by the labeling effort. Therefore, our method Plusmine incorporates SeSL to perform Automatic Classification without human intervention.

2.2 Active Semi-Supervised Learning

Semi-Supervised Learning is in between supervised and unsupervised learning. It extends one of the two types by using information from the other [30]. For example, Levatić et al. [8] use tree-based learning that exploits both labelled and unlabelled data to improve performance. In our research, we combine SeSL with AL.

2.2.1 Combining paradigms. There are several ways to combine the AL and SeSL paradigms. On the one hand, they can operate relatively independently. Tomanek and Hahn propose an approach in which highly uncertain observations are queried to the oracle while highly confident ones are labeled automatically [24]. The authors show that their approach reduces the labeling efforts by 60% compared to only AL when the right confidence threshold is chosen. Leng et al. introduce an AL method with an SVM classifier that queries uncertain instances to the oracle, while it uses different SeSL techniques to automatically classify observations about which the model is confident [7]. They show that some ASeSL methods perform better than using only AL, while others perform worse. This demonstrates that the inclusion of SeSL can also have a negative effect. Hady and Schwenker consider the same perspective for AL and SeSL. They remark that both learning paradigms "tackle the same problem, but from different directions" [5].

On the other hand, SeSL and AL can be intertwined more, e.g., SeSL can be made part of the query strategy: its results directly determine which observations are presented to the oracle. Zhu et al. estimate the expected classification error after an observation is queried. This leads to a better query strategy than selecting the most uncertain instances [31]. Here, both learning paradigms are again used to tackle the same problem, but now together from the same direction.

2.2.2 Network Intrusion Detection. Although promising results have been obtained for ASeSL techniques, they have not been explored much in cybersecurity research. Both Mao et al. and Zhang et al. introduce methods that combine AL with co-training [10, 29]. Co-training is used for binary classification when the features can be separated in two uncorrelated sets or *views* that are both sufficient for learning, meaning each view separately is enough for classification. Although both methods obtain good results, Zhang et al. mention it is necessary that the sufficiency and uncorrelatedness assumptions hold, which is not easy to achieve in practice. Meng and Kwok confirm this: they state lots of human efforts are necessary to obtain two uncorrelated, sufficient feature sets [11].

Because of the aforementioned limitations, our method Plusmine does not consider a multi-view approach. In fact, it uses a novel SeSL strategy in which the observations are automatically classified when they are expected to contribute the most to the intrusion detector in the next time step. Although it has similarities with the non-cybersecurity research of Zhu et al. [31], they use the consequences of potentially labeling observations for their query strategy and not for AC, as we do. Besides this, the AL and SeSL paradigms are not intertwined in Plusmine. This makes it easier to analyze the contribution of each component separately.

3 PRELIMINARIES: JASMINE

Before we delve into the details of our method Plusmine, we first explain the workings of Jasmine, which is used as the base for the AL component of Plusmine, and introduce mathematical notation.

Assume we have a feature dataset $\mathbf{X} \in \mathbb{R}^{M \times K}$ consisting of $M \in \mathbb{N}$ observations with $K \in \mathbb{N}$ feature values. Let $y_i(t) \in \{0, 1, *\}$ be the response value or label corresponding to instance $i \in \{1, \ldots, M\}$ at time step $t \in \{0, \ldots, T\}$ with $T \in \mathbb{N}$ the maximum number of iterations. For $y_i(t)$, '0' denotes the benign class, '1' the malicious class, and '$*$' means that the label is (yet) unknown. Let $\mathcal{L}(t)$ be the set of labeled observations and $\mathcal{U}(t)$ the set of unlabeled items. For all iterations, the sets $\mathcal{L}(t)$ and $\mathcal{U}(t)$ are disjoint and their union is the complete dataset $(\mathbf{X}, \mathbf{y}(t))$, with $\mathbf{y}(t) = (y_1(t), \ldots, y_M(t))$.

Following the general AL framework, Jasmine trains a classifier on $\mathcal{L}(t-1)$ during iteration $t \in \{1, \ldots, T\}$. If the classifier has hyperparameters, they are tuned beforehand on $\mathcal{L}(0)$. Also, Jasmine-specific parameters are tuned on this. The trained classifier is applied to $\mathcal{U}(t-1)$ to obtain the malicious probability of each

observation u and its predicted class. This also yields uncertainty scores $z_u(t)$, one of the two *informativeness measures* in Jasmine, for all $u \in \mathcal{U}(t-1)$. An informativeness measure is used to determine which observations are queried. Simultaneously, an anomaly detector is constructed for each class. These detectors yield the anomaly scores $a_u(t)$, the second informativeness measure, for all unlabeled observations. Now, the Jasmine-specific query function constructs the query set $\mathcal{Q}(t) \subset \mathcal{U}(t-1)$ by using the uncertainty and anomaly scores. This set is of fixed size $Q \in \mathbb{N}$ and a mix of uncertain, anomalous and randomly selected observations according to the provided query type fractions $\alpha_z(t)$, $\alpha_a(t)$ and $\alpha_r(t)$, respectively. Then, the oracle provides the actual response values $y_q(t) \in \{0, 1\}$ of the query items $q \in \mathcal{Q}(t)$. Note that the labels of these observations were unknown (thus had value '$*$') in the previous time steps. The queried instances are added to the labeled pool to obtain $\mathcal{L}(t)$ and removed from the unlabeled set to obtain $\mathcal{U}(t)$. Next, the query type fractions are updated. To this end, the *anomaly information metric* $\delta_a^\beta(t)$ and *uncertainty information metric* $\delta_z^\beta(t)$ are calculated. In short, they measure how much information each query type adds on average. When $\delta_a^\beta(t) > \delta_z^\beta(t)$, more anomalies are queried next iteration, while more uncertainties are queried when $\delta_a^\beta(t) < \delta_z^\beta(t)$. This update procedure is called *α-dynamic updating* and is the key component of Jasmine. Finally, the time step is increased and the procedure is repeated. We refer the reader to Klein et al. for a more detailed explanation [6].

4 METHODOLOGY

In this section, we propose Plusmine. Its Active Learning component is based on Jasmine. However, we make some crucial changes to its query approach to eliminate the bias towards querying uncertain observations. After that, the Semi-Supervised Learning component of Plusmine that constitutes Automatic Classification is introduced.

4.1 Improvements over Jasmine

4.1.1 Construction of query set. First of all, the construction of the query set $\mathcal{Q}(t)$ is adjusted. In Jasmine, it is enforced that the number of predicted benign and malicious observations is equal within both the anomalous and uncertain query types (if possible). In Plusmine, this restriction is relaxed such that only at least one observation of each class is necessary (if possible). This is because we do not want to force a 50/50 split in $\mathcal{Q}(t)$ for imbalanced datasets.

4.1.2 α-dynamic updating. Secondly, the unique update procedure of Jasmine called α-dynamic updating is improved. In Plusmine, the information metrics are changed to address the bias towards querying uncertain observations. They become

$$\delta_a^\beta(t) := \frac{\sum_{q \in \mathcal{Q}_a(t)} \left(\tilde{z}_q(t) + \tilde{a}_q(t)\right) \cdot \left(1 + (\beta - 1)\mathbf{1}_{\{y_q(t)=1\}}\right)}{2\left(Q_a(t) + (\beta - 1) \cdot \left|\{q \in \mathcal{Q}_a(t) : y_q(t) = 1\}\right|\right)}, \quad (1)$$

$$\delta_z^\beta(t) := \frac{\sum_{q \in \mathcal{Q}_z(t)} \left(\tilde{z}_q(t) + \tilde{a}_q(t)\right) \cdot \left(1 + (\beta - 1)\mathbf{1}_{\{y_q(t)=1\}}\right)}{2\left(Q_z(t) + (\beta - 1) \cdot \left|\{q \in \mathcal{Q}_z(t) : y_q(t) = 1\}\right|\right)},$$

with $\mathcal{Q}_a(t)$ and $\mathcal{Q}_z(t)$ the subsets of anomalous and uncertain observations in $\mathcal{Q}(t)$ with sizes $Q_a(t)$ and $Q_z(t)$, respectively. Note that $\mathbf{1}_{\{\cdot\}}$ represents the indicator function. To explain the information metrics, consider Equation (1). The term $\tilde{z}_q(t) + \tilde{a}_q(t)$ is the sum of the normalized uncertainty score $\tilde{z}_q(t)$ and normalized anomaly score $\tilde{a}_q(t)$ and indicates how informative observation $q \in \mathcal{Q}_a(t)$ is. The larger the scores, the more information the sum conveys. In Jasmine, the anomaly score was not taken into account, only the uncertainty score. Therefore, the bias towards querying uncertainties is expected to be fixed in Plusmine. Also, the parameter β puts more ($\beta > 1$) or less ($0 < \beta < 1$) emphasis on a malicious observation compared to a benign one, i.e., a false negative weighs respectively more or less than a false positive. Finally, the denominator ensures that $\delta_a^\beta(t), \delta_z^\beta(t) \in [0, 1]$.

4.2 Semi-Supervised Learning

Besides labeling by a human oracle, Plusmine uses transductive SeSL to assign labels to observations. To this end, our method constructs the set of candidate observations $\mathcal{A}(t) := \mathcal{U}(t-1) \setminus \mathcal{Q}(t)$ for Automatic Classification. The instances in the query set $\mathcal{Q}(t)$ are clearly not appropriate, because they obtain the real label by the oracle. Now, $S \in \mathbb{N}$ disjoint subsets of size $B \in \mathbb{N}$ are drawn without replacement from $\mathcal{A}(t)$ to obtain $\mathcal{B}_1(t), \ldots, \mathcal{B}_S(t)$. The observations in these subsets have their predicted classes (as is often done [28]). Then, for all $s \in \{1, \ldots, S\}$, a decision tree is trained on a random subset of $\mathcal{L}(t-1) \cup \mathcal{Q}(t) \cup \mathcal{B}_s(t)$. Note that $\mathcal{L}(t-1)$ and $\mathcal{Q}(t)$ have their real labels, while $\mathcal{B}_s(t)$ has predicted labels. Next, the classifier is validated on the remaining set of observations, which yields the performance metric $V_s(t)$. The set $\mathcal{B}_{\bar{s}}(t)$ with $\bar{s} := \arg\max_{s \in \{1,\ldots,S\}} \{V_s(t)\}$ is selected as the (relatively) optimal subset to be automatically classified (assuming that a larger $V_s(t)$ means better performance). Lastly, both $\mathcal{Q}(t)$ and $\mathcal{B}_{\bar{s}}(t)$ are added to $\mathcal{L}(t-1)$ to obtain $\mathcal{L}(t)$ and removed from $\mathcal{U}(t-1)$ to obtain $\mathcal{U}(t)$. The complete Plusmine method is illustrated in Figure 1.

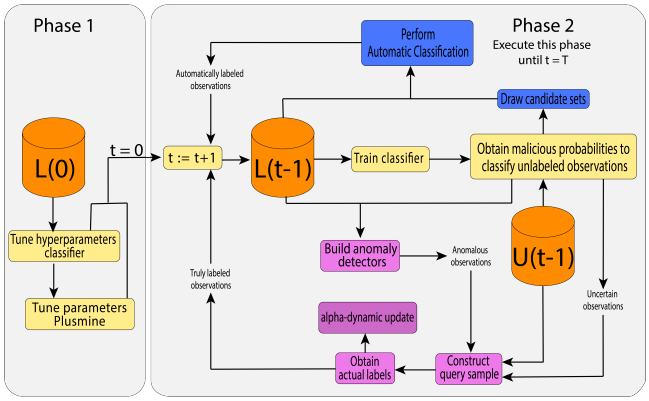


Figure 1: Plusmine methodology with AL component in purple and SeSL component in blue

4.2.1 AC-specific parameters. In general, the larger the number of subsets S is, the higher the likelihood is that a good candidate set for AC is found. However, a larger S means a higher computation time too. The size of each subset B is also a parameter that should be selected beforehand. We consider this a Plusmine-specific hyperparameter and is therefore tuned on $\mathcal{L}(0)$ in the same way the

hyperparameters for the AL component of Plusmine are tuned. This tuning is explained in Section 5.2.4.

5 EXPERIMENTAL SETUP

We conducted several experiments to demonstrate whether Plusmine performs better than benchmark methods.

5.1 Data

Ferrag et al. present an overview of datasets that are commonly used in network intrusion detection research ranging from 1999 to 2018 [2]. From this overview, we selected the popular *NSL-KDD* data and more recent *UNSW-NB15* dataset for our investigations.

5.1.1 NSL-KDD. The NSL-KDD dataset was published in 2009 and is partitioned into a fixed train and evaluation set [23]. Both sets contain the same main attack types, but differ in attack scenarios within the types. We refer to this data as NSLKDDfix.

We were also interested in the workings of the considered methods when the train and evaluation set have (approximately) the same underlying distribution. Therefore, we combined the train and evaluation set of NSLKDDfix into one set. For each experiment, this set was randomly split in a train and evaluation set. We refer to this data as NSLKDDrand.

5.1.2 UNSW-NB15. The UNSW-NB15 data was designed in 2015 by Moustafa and Slay. This dataset is much more recent than NSL-KDD and has more realistic aspects. In fact, it was constructed to address some of the inherent problems of NSL-KDD (for more information, see [14, 15]). We refer to this data as UNSWrand.

Moustafa and Slay also provide a fixed train and evaluation set. that are statistically similar (in contrast to the provided train and evaluation set of NSL-KDD). We refer to this data as UNSWfix.

5.1.3 Preprocessing. Firstly, since the methods require numerical input, the categorical features in all four data settings were removed. In NSL-KDD these are Protocol_type, Service, Flag and Difficulty_level. In UNSW-NB15 these variables are proto, state, service, stcpb and dtcpb. Additionally, the features srcip, sport, dstip, dsport, Stime, Ltime were removed, because they directly determine the output label or have no predictive use. Secondly, the response value was made binary: the benign class was made '0' and the malicious class '1'. Thirdly, all four data settings were standardized and Principal Component Analysis was performed to obtain (linearly) uncorrelated features and to reduce dimensionality [16]. To explain 99% of the variance, 31 Principal Components are necessary for NSLKDDfix and NSLKDDrand, 27 for UNSWrand and 28 for UNSWfix.

5.2 Experiments

5.2.1 AL and ASeSL methods. We considered four main AL and ASeSL techniques: *(i)* Plusmine (plu), *(ii)* a technique (tom) based on the work of Tomanek and Hahn [24], *(iii)* Plusmine-incomplete (pin), and *(iv)* Jasmine (jas) [6]. The technique tom incorporates the query approach of uncertainty sampling and automatically labels highly confident samples. These simple strategies make it a good benchmark. Plusmine-incomplete is Plusmine without the SeSL component (only AL). The difference in performance between plu and pin exactly shows the influence of Automatic Classification. Lastly, the differences in results between pin and jas demonstrate whether the changes made over Jasmine were indeed beneficial.

The intrusion detector is a supervised classifier and was one of two options in each method: Decision Tree (DeT) or Gradient Boosting Machine (GBM). Both learners are tree-based algorithms. DeT is the most basic one, since it consists of only one tree, while GBM integrates multiple boosted decision trees [3]. Hence, we incorporated a fast, but simple and weak predictor; or a slower, but more complex and highly flexible one. Methods that are paired with DeT obtain the suffix '.det' and the ones paired with GBM the suffix '.gbm', e.g., Plusmine with GBM is 'plu.gbm'.

Moreover, the anomaly detector was chosen to be Naive Bayes Classifier (NBC). This is a fast algorithm with no hyperparameters, making the results more robust. The technique is considered in all method settings, except for jas.gbm, since this is the original Jasmine procedure of Klein et al., which uses Isolation Forest.

5.2.2 Global parameters. The global parameters were chosen before the experiments took place. These are the initial size of the labeled set $L(0)$, the initial size of the unlabeled pool $U(0)$, the size of the evaluation set E, the query set size Q, the maximum number of query observations N that are presented to the oracle (labeling budget), and specifically for Plusmine the number of subsets S that are available for AC (see Section 4.2). How many query iterations T were performed is given by: $T := \lfloor \min\{U(0), N\}/Q \rfloor$. Because of stochasticity, each experiment is repeated R times with $\mathcal{L}(0)$ and $\mathcal{U}(0)$ randomly drawn each repetition. The evaluation set $\mathcal{E}$ is also newly constructed for NSLKDDrand and UNSWrand. Hence, the average behavior of the methods can be examined and the range in which the performance resides.

Firstly, we chose $L(0) = 200$, since a small starting size is usually the case in AL. Secondly, $E = 5{,}000$ was selected for NSLKDDrand and UNSWrand, as we deemed it large enough to be representative. The size was already provided for NSLKDDfix and UNSWfix by their authors with $E = 22{,}544$ and $E = 82{,}332$, respectively. Thirdly, $U(0)$ follows directly from $L(0)$ and E. Therefore, we had $U(0) = 143{,}317$ for NSLKDDrand, $U(0) = 125{,}773$ for NSLKDDfix, $U(0) = 2{,}534{,}847$ for UNSWrand and $U(0) = 175{,}141$ for UNSWfix. Fourthly, we selected $Q = 50$, as we see adding 50 new observations sufficient for retraining the classifier. Finally, for Plusmine, $S = 50$ was chosen as a trade-off between computation time and a larger potential performance (see Section 4.2 for details).

Table 1: Hyperparameter GBM ranges (sr = 'sample_rate')

distribution Bernoulli	histogram_type RoundRobin	learn_rate_annealing {0.95, 0.99, 0.999}
max_depth {6, 12, 24}	sr {0.60, 0.78, 1.0}	ntrees {250, 500, 1,000}
nbins {10, 16, 25}	nbins_cats {16, 32, 64}	learn_rate {0.02, 0.05, 0.125}
min_rows {6, 8, 10}	col_sr {0.84, 0.92, 1.0}	col_sr_change_per_level {0.94, 1.0, 1.06}
	col_sr_per_tree {0.40, 0.64, 1.0}	

5.2.3 Hyperparameter tuning Gradient Boosting Machine. GBM can be customized to a high degree, i.e., there are many hyperparameters. In each experiment, good values for these parameters were found by performing k-fold cross validation on $\mathcal{L}(0)$. The research of Tama and Rhee and exploratory studies were used to decide which hyperparameters were tuned and over which range [22]. Table 1 shows the selected hyperparameters and corresponding ranges for the `h2o.gbm` function of the `H2O.ai` package in the `R` programming language. The parameters that are not shown in the table got their default settings. Because the total number of combinations is enormous, random search was performed for 2 hours with the F_1 score as performance measure.

Table 2: Hyperparameter A(SeS)L ranges

$\alpha_a^{(0)}$	β, γ	τ	B
$\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$	$\{\frac{1}{2}, 1, 2\}$	$\{\frac{1}{450}, \frac{1}{150}, \frac{1}{50}\}$	$\{25, 50, 100, 200\}$

5.2.4 Hyperparameter tuning AL and ASeSL. The methods Plusmine, Plusmine-incomplete and Jasmine have AL-specific hyperparameters that had to be tuned. This tuning is part of the procedure. One of the hyperparameters was already introduced in Section 4.1: β, the weight factor in the information metrics. Additionally, we have $\alpha_a^{(0)}$, the initial anomaly query fraction; $\gamma > 0$, the update magnitude in α-dynamic updating; and $\tau > 0$, which is related to the query fraction of random observations. More information about these hyperparameters can be found in [6]. The SeSL component of Plusmine introduces the hyperparameter B, the size of a candidate set for AC. Table 2 shows the ranges over which the hyperparameters were tuned. Now, let $\mathcal{H}$ be the set of all hyperparameter combinations. Thus, $|\mathcal{H}| = 81$ for Plusmine-incomplete and Jasmine, and $|\mathcal{H}| = 324$ for Plusmine.

During tuning, $\mathcal{L}(0)$ is randomly partitioned into the sets $\mathcal{L}_H(0)$, $\mathcal{U}_H(0)$ and $\mathcal{E}_H$ in a 30/50/20-split. As the notation suggests, $\mathcal{L}_H(0)$ is the initially labeled set for tuning with size $L_H(0)$, $\mathcal{U}_H(0)$ is the initially unlabeled set with size $U_H(0)$ and $\mathcal{E}_H$ is the evaluation set. In Plusmine and Plusmine-incomplete, the size of the query set Q_H during tuning is defined as $Q_H := \left\lceil \frac{L_H(0)}{L(0)} \cdot Q \right\rceil$. Thus, the ratio of Q_H to Q is the same as that of $L_H(0)$ to $L(0)$. The number of query iterations in tuning is equal to $t_H := \left\lceil \frac{U_H(0)}{Q_H} \right\rceil - 1$. The minus one is because the last iteration is not performed, since it always contains the leftover observations that are the least informative.

Next, let $h \in \mathcal{H}$ be some hyperparameter combination. Then, the chosen classifier (DeT or GBM) is trained on $\mathcal{L}_H(0)$ and applied to $\mathcal{E}_H$ to obtain the F_1 score $F_1^{(h)}(0)$. Then, the rest of the AL or ASeSL procedure is executed, as described in Section 3 and 4. After this, the sequence of performance scores $(F_1^{(h)}(0), \ldots, F_1^{(h)}(t_H))$ is obtained. Then, the area under the $(t, F_1^{(h)}(t))_{\{t=0,\ldots,t_H\}}$-'curve' is determined to obtain a single quality score. Such a *learning curve* is commonly used in AL research to evaluate the overall performance of a prediction model [18]. The larger the area under the learning curve, the better combination $h \in \mathcal{H}$ is.

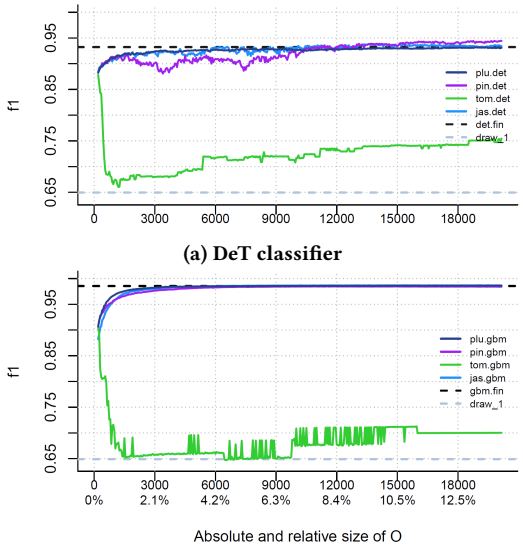
Because of stochasticity, the tuning rounds were repeated at least $R_H = 4$ times. Each repetition, $\mathcal{L}(0)$ was split into the three sets $\mathcal{L}_H(0)$, $\mathcal{U}_H(0)$ and $\mathcal{E}_H$ and the area under the learning curve was obtained. The combination that yielded the largest area averaged over the repetitions provided the (relatively) optimal values for $\alpha_a^{(0)}$, β, γ, τ, and (if applicable) B. In total, the DeT-based methods got 16 hours for this, while the GBM-based procedures got 14 hours, since they already used 2 hours for tuning the GBM classifier. If there was time left (usually for the lighter DeT-based methods), then another repetition was performed.

5.2.5 Assessment of AL and ASeSL methods. For the final assessment of the eight methods, the performance on the separate evaluation set $\mathcal{E}$ was used. In iteration t, the classifier was trained on $\mathcal{L}(t)$ and the performance score $F_1(t)$ was obtained for each method. After some reference iteration t_{ref} ($0 \leq t_{\text{ref}} \leq T$), the area under the $(t, F_1(t))_{\{t=0,\ldots,t_{\text{ref}}\}}$-curve $A(t_{\text{ref}})$ was calculated. Since the experiments were conducted R times, this yielded R reference areas per method. We used these areas to determine whether Plusmine performed significantly better according to a Mann-Whitney U test.

6 RESULTS

We discuss two categories of results: *(i)* average learning curves, and *(ii)* tables of p-values. The first category provides a graphical insight in how the performance of the four main methods (plu, pin, tom and jas) evolved on the evaluation set $\mathcal{E}$. The second category describes the statistical significance of each method for several reference iterations by performing a Mann-Whitney U (MWU) test with significance level 0.05.

6.1 Results on NSL-KDD



(a) DeT classifier

(b) GBM classifier

Figure 2: Learning curves on NSLKDDrand with random evaluation set

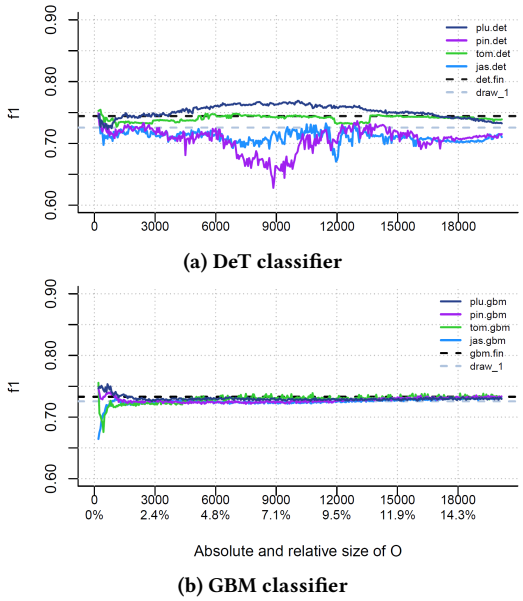
6.1.1 NSLKDDrand. Figure 2 shows the average learning curves of F_1 scores for the four main methods with DeT (Figure 2a) or GBM (Figure 2b) classifier. Each simulation yielded a learning curve

and the figure shows the mean over all runs per method. The horizontal axis is the size of the *truly* labeled set $O(t)$, i.e., the number of observations classified by the oracle. For both classifiers, there is a large discrepancy between tom and the other three methods. Moreover, the learning curves of plu, pin and jas are very close to each other, especially for GBM. For the latter, the three methods quickly reached their final score, which is represented by the black dashed line (.fin). This is the average performance of the classifier trained on the complete train set with all labels available. As the initial performance was already good, there was not much to learn. It seems that Plusmine performed best for both DeT and GBM at the start, which is the most important part. It is also striking that Plusmine-incomplete performed less well than Jasmine. Furthermore, the difference between DeT or GBM is as expected: the F_1 scores for DeT were generally lower than those for GBM. The latter is a more complex technique and was therefore better able to learn structures in the data. Lastly, the gray dashed line is the Dutch Draw baseline [1]. This is the maximum expected score that a classifier that makes random predictions or that predicts only one class can attain. Therefore, a method should at least outperform this baseline.

Table 3: p-values MWU test for NSLKDDrand

t_{ref}	$O(t_{\text{ref}})$	plu.d v. pin.d	plu.d v. tom.d	plu.d v. jas.d	pin.d v. jas.d
4	400	0.50	0.0054	0.75	0.78
10	700	0.52	$3.0 \cdot 10^{-12}$	0.32	0.66
25	1450	0.20	$7.9 \cdot 10^{-15}$	0.12	0.41
63	3350	0.00011	$7.9 \cdot 10^{-15}$	0.15	1.0
156	8000	$1.2 \cdot 10^{-10}$	$7.9 \cdot 10^{-15}$	0.20	1.0
399	20150	0.011	$7.9 \cdot 10^{-15}$	0.92	1.0
t_{ref}	$O(t_{\text{ref}})$	plu.g v. pin.g	plu.g v. tom.g	plu.g v. jas.g	pin.g v. jas.g
4	400	0.30	$7.4 \cdot 10^{-8}$	$1.3 \cdot 10^{-6}$	$4.3 \cdot 10^{-6}$
10	700	0.035	$2.0 \cdot 10^{-9}$	$3.0 \cdot 10^{-7}$	0.00013
25	1450	0.00052	$3.6 \cdot 10^{-11}$	$1.5 \cdot 10^{-7}$	0.046
63	3350	$2.1 \cdot 10^{-5}$	$5.3 \cdot 10^{-13}$	$1.3 \cdot 10^{-7}$	0.48
156	8000	$1.5 \cdot 10^{-5}$	$7.2 \cdot 10^{-12}$	$3.7 \cdot 10^{-5}$	0.92
399	20150	0.00010	$4.6 \cdot 10^{-11}$	0.039	1.0

Table 3 supports what we observed in Figure 2. Note that '.d' represents '.det' and '.g' represents '.gbm'. A green value means that the first method in the corresponding column name performed significantly better than the second method, a red value means it was significantly worse, and a black value means no decisive conclusion could be drawn. Clearly, Plusmine performed significantly better than tom for all reference iterations. The bad performance of the latter was possibly due to the immediate automatic labeling of on average 80% of all unlabeled observations in the first few iterations. Especially for DeT, there were many mistakes in these predictions, making it very difficult for the method to recover from this. Also, plu.gbm was almost always significantly better than pin.gbm and jas.gbm. However, we see in Figure 2b that the differences in performance were small.



(a) DeT classifier

(b) GBM classifier

Figure 3: Learning curves on NSLKDDfix with fixed evaluation set

6.1.2 NSLKDDfix. Although NSLKDDfix is based on the same dataset as NSLKDDrand, the results are vastly different, as Figure 3 shows. Firstly, most surprisingly is the difference between DeT (Figure 3a) and GBM (Figure 3b). The final performance of DeT was higher than that of GBM, even though GBM is a more complex technique. Also, plu.det performed better than plu.gbm after the first ten iterations. It even performed notably better than its final score. Secondly, this time tom was a worthy competitor. Interestingly, still approximately 80% of the observations were automatically labeled and the number of mistakes made was of the same order. Apparently, it has to do with the difference in distribution of train and evaluation set: tom was now better able to generalize, even though there were mistakes in the labeled set. Thirdly, pin was mostly better than or on par with jas, except for a large dip in pin.gbm around $O(t) = 9{,}000$. Lastly, pin.det, jas.det, pin.gbm, tom.gbm and jas.gbm performed worse or much worse than the Dutch Draw baseline at certain iterations. This is undesirable behavior, because it means a better performance could be attained without learning and by predicting everything malicious, which the Dutch Draw classifier does in this specific setting. This means the benefits of AL or ASeSL are not clear here.

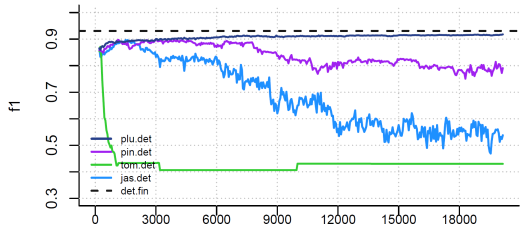
The p-values of the statistical tests in Table 4 suggest that Plusmine in general performed on par or significantly better than the other methods. There is one exception for $t_{\text{ref}} = 4$ and DeT.

6.2 Results on UNSW-NB15

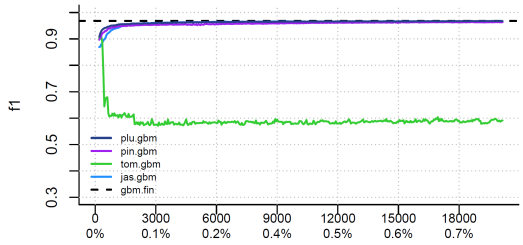
6.2.1 UNSWrand. Figure 4 shows the average learning curves for the UNSWrand setting. Remarkably, the results for GBM (Figure 4b) are very similar to the results on NSLKDDrand. Both datasets have in common that train and evaluation set have approximately the same distribution. However, the results with DeT (Figure 4a) are

Table 4: p-values MWU test for NSLKDDfix

t_{ref}	$O(t_{ref})$	plu.d v. pin.d	plu.d v. tom.d	plu.d v. jas.d	pin.d v. jas.d
4	400	0.12	0.98	0.11	0.37
10	700	0.16	0.89	0.0076	0.052
25	1450	0.0048	0.36	0.00048	0.16
63	3350	0.00052	0.050	$2.5 \cdot 10^{-5}$	0.12
156	8000	$1.1 \cdot 10^{-12}$	0.0042	$4.0 \cdot 10^{-12}$	0.48
399	20150	$7.9 \cdot 10^{-15}$	0.0068	$7.9 \cdot 10^{-15}$	0.74
t_{ref}	$O(t_{ref})$	plu.g v. pin.g	plu.g v. tom.g	plu.g v. jas.g	pin.g v. jas.g
4	400	0.071	0.031	$4.4 \cdot 10^{-7}$	$5.7 \cdot 10^{-7}$
10	700	0.14	0.0068	$4.4 \cdot 10^{-7}$	$8.3 \cdot 10^{-7}$
25	1450	0.24	0.0019	$5.3 \cdot 10^{-6}$	$7.1 \cdot 10^{-5}$
63	3350	0.088	0.0023	$5.9 \cdot 10^{-5}$	0.0051
156	8000	0.039	0.016	0.0011	0.088
399	20150	0.28	0.27	0.076	0.27



(a) DeT as classifier

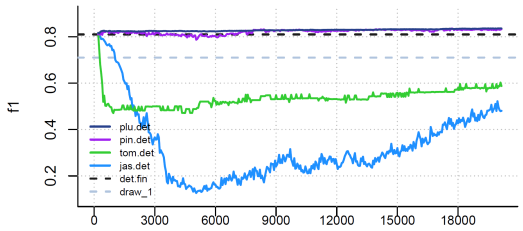


(b) GBM as classifier

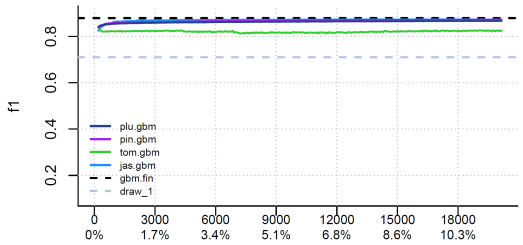
Figure 4: Learning curves on UNSWrand with random evaluation set

Table 5: p-values MWU test for UNSWrand

t_{ref}	$O(t_{ref})$	plu.d v. pin.d	plu.d v. tom.d	plu.d v. jas.d	pin.d v. jas.d
4	400	0.0040	0.022	0.0012	0.14
10	700	0.0033	$6.4 \cdot 10^{-8}$	$1.4 \cdot 10^{-5}$	0.030
25	1450	0.15	$3.5 \cdot 10^{-10}$	0.0057	0.040
63	3350	0.12	$3.0 \cdot 10^{-12}$	0.00032	0.0033
156	8000	0.011	$3.6 \cdot 10^{-13}$	$9.6 \cdot 10^{-12}$	$7.4 \cdot 10^{-8}$
399	20150	$1.2 \cdot 10^{-5}$	$3.2 \cdot 10^{-14}$	$7.9 \cdot 10^{-15}$	$9.4 \cdot 10^{-7}$
t_{ref}	$O(t_{ref})$	plu.g v. pin.g	plu.g v. tom.g	plu.g v. jas.g	pin.g v. jas.g
4	400	0.0020	$1.2 \cdot 10^{-5}$	$4.9 \cdot 10^{-9}$	0.00013
10	700	0.0024	$1.1 \cdot 10^{-8}$	$2.6 \cdot 10^{-8}$	0.00032
25	1450	0.0031	$1.5 \cdot 10^{-7}$	$8.3 \cdot 10^{-7}$	0.0064
63	3350	0.0038	$2.0 \cdot 10^{-7}$	0.00056	0.24
156	8000	0.0027	$8.3 \cdot 10^{-7}$	0.032	0.90
399	20150	0.0019	$1.1 \cdot 10^{-6}$	0.19	0.97



(a) DeT as classifier



(b) GBM as classifier

Figure 5: Learning curves on UNSWfix with fixed evaluation set

notably different for pin.det and jas.det. Both methods started comparably well to Plusmine, but after around 2,000 queried observations jas.det began to worsen, and the same happened for pin.det after approximately 7,500 queries.

Table 5 confirms these observations as Plusmine was significantly better than the other methods for almost all reference values. In contrast to the results on NSL-KDD, it seems that pin is often better than jas, as indicated by the significant p-values in the final column of the tables and the learning curves.

6.2.2 UNSWfix. The learning curves on the UNSWfix data are shown in Figure 5. Again, the GBM classifier (Figure 5b) produced straightforward results: the learning curves of plu.gbm, pin.gbm and jas.gbm were close to each other and their initial performance was already close to the final score. This time, tom.gbm was very stable, but a bit lower than the other three. The results with DeT (Figure 5a) show more contrasting behaviors. plu.det and pin.det almost did not change and performed better than their final score, but tom.det and jas.det were much different. The first had a performance drop at the start, and then tried to recover. As before, most of the unlabeled observations were immediately automatically labeled. jas.det had a fast performance decrease and later on tried to rebound. These results clearly show the benefit of pin over jas.

Finally, the p-values in Table 6 support what the corresponding learning curves with DeT suggest: Plusmine was significantly

Table 6: p-values MWU test for UNSWfix

t_{ref}	$O(t_{\text{ref}})$	plu.d v. pin.d	plu.d v. tom.d	plu.d v. jas.d	pin.d v. jas.d
4	400	0.0057	$3.0 \cdot 10^{-7}$	$4.4 \cdot 10^{-7}$	0.00087
10	700	0.035	$2.4 \cdot 10^{-13}$	$1.2 \cdot 10^{-10}$	$4.1 \cdot 10^{-8}$
25	1450	0.019	$1.6 \cdot 10^{-14}$	$7.9 \cdot 10^{-15}$	$5.5 \cdot 10^{-14}$
63	3350	$7.7 \cdot 10^{-5}$	$3.2 \cdot 10^{-14}$	$7.9 \cdot 10^{-15}$	$7.9 \cdot 10^{-15}$
156	8000	$1.1 \cdot 10^{-5}$	$5.5 \cdot 10^{-14}$	$7.9 \cdot 10^{-15}$	$7.9 \cdot 10^{-15}$
399	20150	0.00060	$5.5 \cdot 10^{-14}$	$7.9 \cdot 10^{-15}$	$7.9 \cdot 10^{-15}$
t_{ref}	$O(t_{\text{ref}})$	plu.g v. pin.g	plu.g v. tom.g	plu.g v. jas.g	pin.g v. jas.g
4	400	0.25	0.0012	0.066	0.17
10	700	0.82	0.00022	0.55	0.34
25	1450	0.99	$2.1 \cdot 10^{-5}$	0.98	0.81
63	3350	1.0	$7.4 \cdot 10^{-6}$	1.0	0.98
156	8000	1.0	$3.8 \cdot 10^{-6}$	1.0	1.0
399	20150	1.0	$5.3 \cdot 10^{-6}$	1.0	0.99

better than the other methods for all reference values, and Plusmine-incomplete always performed better than jas.det. This is in contrast to the p-values for GBM: they seem to not be in favor of Plusmine at all. pin.gbm and jas.gbm firstly performed on par with plu.gbm, but were later on significantly better. Also the final column demonstrates that Jasmine outperforms pin.gbm. However, Figure 5b shows how close the performance scores actually were to each other.

7 CONCLUSION

In this paper, we introduced Plusmine, an NIDS that combines new techniques for both AL and SeSL. Its AL component is an improved version of Jasmine [6]. The results on the UNSW-NB15 data support that this is an improvement in most cases; and when this was not the case, the difference between the learning curves was small. However, this was not the case for the NSL-KDD data. Yet, in general, our query approach in Plusmine performed more reliably than Jasmine, i.e., drops in performance were less severe.

Secondly, the SeSL component of Plusmine had a convincingly positive effect on performance for all dataset configurations. It performed at least on par with the other methods, but often it was significantly better. The rare occassion when this was not the case, the difference between the learning curves was small. Hence, the addition of AC is beneficial. Moreover, our proposed form of AC performed mostly significantly better than the benchmark ASeSL method tom that automatically labels confident observations, as is common practice for SeSL [24].

REFERENCES

[1] E. Van de Bijl, J. Klein, J. Pries, S. Bhulai, M. Hoogendoorn, and R. Van der Mei. 2021. The Dutch Draw: constructing a universal baseline for binary prediction models. *Machine Learning [Under Review]* (2021).

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

[3] J. H. Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[4] Y. Gu and D. Zydek. 2014. Active learning for intrusion detection. In *2014 National Wireless Research Collaboration Symposium*. IEEE, 117–122.

[5] M. F. A. Hady and F. Schwenker. 2010. Combining committee-based semi-supervised learning and active learning. *Journal of Computer Science and Technology* 25, 4 (2010), 681–698.

[6] J. Klein, S. Bhulai, M. Hoogendoorn, and R. Van der Mei. 2021. Jasmine: a New Active Learning Approach to Combat Cybercrime. *Machine Learning with Applications [Under Revision Review]* (2021). https://arxiv.org/abs/2108.06238

[7] Y. Leng, X. Xu, and G. Qi. 2013. Combining active learning and semi-supervised learning to construct SVM classifier. *Knowledge-Based Systems* 44 (2013), 121–131.

[8] J. Levatić, M. Ceci, D. Kocev, and S. Džeroski. 2017. Semi-supervised Classification Trees. *Journal of Intelligent Information Systems* 49, 3 (2017), 461–486.

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

[10] C.-H. Mao, H.-M. Lee, D. Parikh, T. Chen, and S.-Y. Huang. 2009. Semi-supervised co-training and active learning based approach for multi-view intrusion detection. *Proceedings of the 2009 ACM symposium on Applied Computing*, 2042–2048.

[11] Y. Meng and L. F. Kwok. 2012. Intrusion detection using disagreement-based semi-supervised learning: Detection enhancement and false alarm reduction. In *International Symposium on Cyberspace Safety and Security*. Springer, 483–497.

[12] S. Morgan. 2020. *Cybercrime To Cost The World 10.5 Trillion Annually By 2025*. Cybercrime Magazine. Retrieved June 30, 2021 from https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/

[13] S. A. Mouloua, J. Ferraro, M. Mouloua, G. Matthews, and R. R. Copeland. 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. SAGE Publications Sage CA: Los Angeles, CA, 1600–1604.

[14] N. Moustafa and J. Slay. 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 (MilCIS)*. IEEE, 1–6.

[15] N. Moustafa and J. Slay. 2016. The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal: A Global Perspective* 25, 1-3 (2016), 18–31.

[16] K. Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, 11 (1901), 559–572.

[17] Z. Qiu, D. J. Miller, and G. Kesidis. 2017. Flow based botnet detection through semi-supervised active learning. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2387–2391.

[18] B. Settles. 2009. Active learning literature survey. (2009).

[19] P. W. Singer and A. Friedman. 2014. *Cybersecurity: what everyone needs to know*. Oxford University Press.

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

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

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

[23] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 1–6.

[24] K. Tomanek and U. Hahn. 2009. Semi-Supervised Active Learning for Sequence Labeling. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. 1039–1047.

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

[26] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang. 2018. Machine learning and deep learning methods for cybersecurity. *IEEE Access* 6 (2018), 35365–35381.

[27] K. Yang, J. Ren, Y. Zhu, and W. Zhang. 2018. Active Learning for Wireless IoT Intrusion Detection. *IEEE Wireless Communications* 25, 6 (2018), 19–25.

[28] D. Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*. 189–196.

[29] Y. Zhang, J. Wen, X. Wang, and Z. Jiang. 2014. Semi-supervised learning combining co-training with active learning. *Expert Systems with Applications* 41, 5 (2014), 2372–2378.

[30] X. Zhu and A. B. Goldberg. 2009. Introduction to semi-supervised learning. 3, 1 (2009), 1–130.

[31] X. Zhu, J. Lafferty, and Z. Ghahramani. 2003. Combining Active Learning and Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML 2003 workshop on the continuum from labeled to unlabeled data in machine learning and data mining*, Vol. 3.