Hindawi
Security and Communication Networks
Volume 2018, Article ID 8753870, 8 pages
https://doi.org/10.1155/2018/8753870

Research Article

Cost-Sensitive Distributed Machine Learning for NetFlow-Based Botnet Activity Detection

Rafał Kozik, Marek Pawlicki, and Michał Choraś

UTP University of Science and Technology in Bydgoszcz, Poland

Correspondence should be addressed to Marek Pawlicki; marek.pawlicki@biznespoczta.pl

Received 16 May 2018; Revised 8 August 2018; Accepted 29 November 2018; Published 20 December 2018

Academic Editor: Dimitrios Geneiatakis

The recent advancements of malevolent techniques have caused a situation where the traditional signature-based approach to cyberattack detection is rendered ineffective. Currently, new, improved, potent solutions incorporating Big Data technologies, effective distributed machine learning, and algorithms countering data imbalance problem are needed. Therefore, the major contribution of this paper is the proposal of the cost-sensitive distributed machine learning approach for cybersecurity. In particular, we proposed to use and implemented cost-sensitive distributed machine learning by means of distributed Extreme Learning Machines (ELM), distributed Random Forest, and Distributed Random Boosted-Trees to detect botnets. The system's concept and architecture are based on the Big Data processing framework with data mining and machine learning techniques. In practical terms in this paper, as a use case, we consider the problem of botnet detection by means of analysing the data in form of NetFlows. The reported results are promising and show that the proposed system can be considered as a useful tool for the improvement of cybersecurity.

1. Introduction

In this day and age, the more the aspects of one's life that can be smoothly transitioned to the Internet, the more the hazards awaiting for the unprepared users. A multitude of hostile agents lurk in the shadows waiting for the easy prey. A myriad of ways to penetrate users' devices are readily available (and can be even bought in Darknet with so-called Crime as a Service (CaaS) crime model); those threats include trojans, spyware, phishing, and even attacks that require careful orchestration of hundreds, thousands, and even millions of compromised machines, called botnets.

It usually starts with a seemingly benign e-mail, an application downloaded from an unknown source. Sometimes the user is not even aware that his device has downloaded something off the Internet, as it could exploit vulnerabilities in the browser or in the browser's plugins. Nevertheless, the user's device is now part of a mob of machines which respond to the commands of a master, a botmaster. This machine, unbeknownst to the user, can now be part of a DoS attack and could be sending spam or even sharing or stealing data. [1].

Contemporary apprehension procedures, which perform full packet data analysis, also called deep packet inspection (DPI) are rendered inadequate by the sheer volume of traffic at speeds of multiple Gigabits per second. This problem is only going to be amplified by the arrival of massive environments like cloud computing. Furthermore, the storage of full packet data constitutes a risk to users' privacy, as it could contain sensitive information. NetFlow, often abbreviated to simply flow, is a derivative of a data stream shared between two systems. NetFlow records comprise a statistic of traffic between the same IP addresses, same source and destination ports, IP protocols, and IP types of service. For that reason, NetFlow is significantly more efficient and does not infringe on users' privacy [2, 3].

The dynamically changing nature of security breaches caused an increase in the number of dynamically adapting detection algorithms. Anomaly detectors do not use known signatures, but a model of 'normal' behaviour to flag any

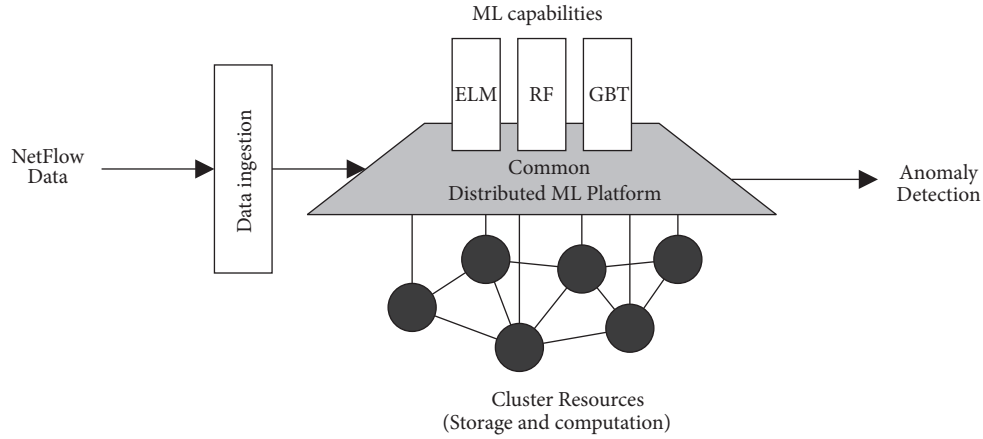

Figure 1: The high-level concept of botnet activity detection platform.

divergences. Those systems are alluring on account of not needing a lot of maintenance from security personnel. Unfortunately, however, many anomaly detectors are not appropriate for real-time environments, as they encounter various setbacks.

In some applications, the training samples must be weighted. One of the reasons behind this may be data imbalance problem. In such cases, the number of training samples belonging to some classes is larger in contrast to other classes.

The problem of data imbalance has recently been deeply studied in the area of machine learning and data mining. In many cases, this problem impacts the machine learning algorithms and as a result deteriorates the effectiveness of the classifier [4]. Typically, in such cases, classifiers will achieve higher predictive accuracy over the majority class, but poorer predictive accuracy over the minority class.

In general, solutions to this problem can be categorised as data-related and algorithm-related. The methods belonging to the data-related category use data oversampling and undersampling techniques, while the algorithm-related ones introduce a modification to training procedures. This group can be further classified into categories using cost-sensitive classification (e.g., assigning a higher cost to majority class) or methods that use different performance metrics (e.g., Kappa metric). Recently cost-sensitive learning has been reported to be the effective solution to class-imbalance in the large-scale setting.

Therefore, in this paper, the solutions incorporating Big Data technologies, effective distributed machine learning, and algorithms countering data imbalance problem are proposed.

The presented paper extends our previous work [5] on the distributed platform for NetFlow data analysis (see Figure 1) and initial works on extreme machine learning classifiers (ELM) for cyber security [6]. The global concept behind the proposed solution is to provide unified and highly scalable environment for botnet activity detection. The data is ingested into the system in the form of NetFlows. In our research, we extensively explore a variety of machine learning solutions that are in the heart of AI-based detection and decision in our platform. In the previous work, a distributed implementation of extreme machine learning classifier (ELM) was used for NetFlow data classification. In this work, we extended the method by introducing two additional cost-sensitive algorithms (Random Forest and Gradient-Boosted Trees) that have been now deployed in the proposed system. Moreover, in contrast to our previous work, we considered scalability challenges and we report the experiments comparing those algorithms from the scalability point of view. Another proven thesis in this paper is that ELM-based approach is still a reasonable choice due to its simplicity and scalability, even though the tree-based techniques allow for achieving slightly better (or comparable) results. To support this thesis we deliberately provide an overview of technical details of all the three methods and approaches, and we also identify some strong and weak sides.

The paper is structured as follows: in Section 2, the related work is presented and discussed. Section 3 gives the details of our proposal of cost-sensitive distributed machine learning for cyber security (botnet detection). Distributed Extreme Learning Machines (ELM), distributed Random Forest, and Distributed Random Boosted-Trees to detect botnets are proposed. Experimental setup, the used datasets, data preprocessing, and results are presented in Sections 4 and 5, respectively. Conclusions are given thereafter.

2. Related Work

Martin Grill, Ivan Nikolaev, Veronica Valeros, and Martin Rehak advocate a NetFlow/IPFIX based DGA-performing malware detector. Domain Generation Algorithms (DGA) are a way for botnets to hide the Command and Control (C&C) botmaster server. When the C&C server is compromised, it loses command over the entire botnet, as antivirus companies and OS vendors blacklist its IP and stop any possible communication at the firewall level. DGA is a form of domain fluxing, where a distinct seed is used to generate a domain name in a specific time frame. The botnet contacts these domain names to listen for commands. To issue commands, the botmaster has to use the same seed and register a specific domain before the botnet tries to contact

it. DGA can use a dictionary based on popular sites like google or trending topics on twitter to create domain names, which makes flagging the possible botnet C&C domains extremely hard. Current botnet detectors experience major performance issues when scaled to larger networks, which render them impractical. Reverse engineering, deep packet inspection, clustering, and other methods absorb too much time and resources. Additionally, users' privacy is at risk. To circumvent these issues, NetFlow data usage is proposed. DGA malware is expected to attempt to try to contact more domains than it does new IP addresses. Because the NetFlows are not unidirectional, additional information has to be used to single out the originator of each transmission. NetFlows with the same IP-port-protocol triples are paired and marked as request and response according to timestamps since a request always comes first. However, this method loses its reliability when scaled to larger networks. In such cases, a service detection algorithm provides a strong feature based on a median number of peers difference. A DNS anomaly detector is implemented, labelling the right tail of the normal distribution as anomalous. This is because DNS requests that are more numerous than the visited IPs are possible C&C botnet connections. Anomaly values are acquired with a fuzzy function. Finally, as the proposed method is susceptible to raising a false positive for DNS resolver service, data from the service detection step is used to tackle this problem [7].

Q. A. Tran, F. Jiang, and J. Hu present a real-time intrusion detection system (IDC) with a hardware-core of High-Frequency Field-Programmable Gate Arrays. The authors examine a batch of IDC's including the recent additions to the field, which are based on computational intelligence, including fuzzy systems, support vector machines, and evolutionary algorithms like differential evolution, genetic algorithms or particle swarm optimisation. While these software-based methods possess the ability to quickly adapt to new threats, their effectiveness in high-volume environments is limited by their detection speed. This translates into the inability to properly address the needs of massive environments in the near future, like cloud computing. The proposed hardware-cored IDS offers a higher detection speed than the software-based counterparts. The method supplies a Field-Programmable Gate Array (FPGA) with an internally evolvable Block Based Neural Network (BBNN). The BBNN in this process is a feed-forward algorithm. The character of the blocks and the weights are determined by a genetic algorithm which seeks a global optimum guided by a specific fitness function. NetFlow data is used as a way to streamline feature extraction, as these can be set to default flow features. Additionally, the NetFlow collector is able to generate real-time data for the FPGA. The procedure itself is as follows: the FPGA performs real-time detection of possible intrusions and adds the record to the database; the BBNN repeatedly retrains itself with the fresh database; the FPGA corrects its configuration building on the structure of the BBNN [8].

K. Flanagan, E. Fallon, A. Awad, and P. Connolly propose the use of a microcluster based outlier detection algorithm (MCOD) as a deviation discovery device, augmented to take into account pattern deviation over time. The procedure employs clustering to cut down on the number of distance calculations it has to perform. The reduced calculation needs to make it suitable for real-time data stream analysis, unlike many other anomaly detection approaches. The distance between a flow and a cluster centroid decides whether it is an anomaly, or not. MCOD is used in a succession of intervals, making the algorithm time-aware. The effects of the anomaly detection are then processed by a polynomial regression to arrive at an approximation of cluster densities over time. In the proposed procedure, all cluster densities are monitored disregarding the cluster edge denoted by the k variable. This approach diminishes the impact of the unlikely supposition that all traffic is distributed equally across the network, allowing for an increased situational awareness. By comparing cluster densities over time two polynomials are generated to represent the cluster activity. Overall, the proposed method flags anomalies in two distinctive ways. The MCOD detects distance-based divergences at the end of every time series. The polynomials created over 3-hour and 24-hour periods, when compared using Frechet distance, reveal any anomalies of actual versus expected behaviour of a cluster [9].

X. Yuan explores deep learning for real-time malware detection. Signature-based approaches form the current industry standard for malware detection, despite obvious shortcomings with detecting obfuscated malware, zero-day exploits, and simply the astounding daily number of new malware releases. To tackle these kinds of problems, anomaly detectors based on machine learning models are implemented. Methods like K-nearest neighbour support vector machines, or decision tree algorithms struggle with high false positive rates. Without sufficient context malware classification is hard to perform accurately. On the other hand, deep learning (DL) algorithms are adequate for making superior decisions, but at a cost. DL needs significantly more time to retrain the detection model, which constitutes a major drawback when new malware strains have to be added frequently. The proposed procedure tries to strike a balance between the accuracy of deep learning and the swiftness of classical machine learning methods by employing a multistage detection algorithm cooperating with the operating system. The first stage involves classical machine learning detection. In case the ML classifies a potential threat, it is carried over to the 'uncertain stage'. In the second stage, a deep learning algorithm decides if the threat is marked as benign or as hostile, and therefore, killed. If new malware is found, the model is retrained with the use of a concept-drift component, which makes sure the model is relevant [10].

In light of the performed analysis, it is evident that the improved effective solutions incorporating Big Data technologies, effective distributed machine learning, and classifiers as well as algorithms countering the data imbalance problem are needed.

Therefore, in the following sections we present cost-sensitive distributed machine learning approach for cyber security.

3. Distributed Machine Learning

3.1. Distributed Extreme Learning Machine. The ELM (Extreme Learning Machine) is one of the methods for data

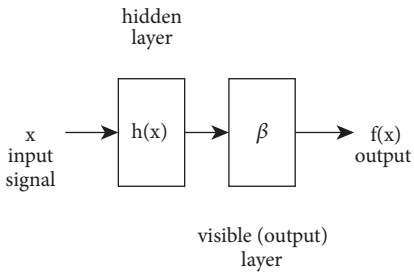

Figure 2: ELM classifier structure and input signal to output mapping.

classification that has been proposed by Huang et al. [11]. The classifier adapts the idea of a single-hidden-layer neural network that is trained without an iterative process. For the ELM classifier a response for a signal x is calculated as a factor of hidden layer response $h(x)$ and weight of the output layer β (see Figure 2).

The neurons in the hidden layer do not have to be neurons in a classical sense as they are not tuned during the learning phase. The hidden layer of neurons used by ELM is applied to map the original feature vectors to the new feature space and can be used in the cyber security domain [6]. Commonly, the h layer performs random and nonlinear projection.

Let $X = \{x_j\}_{j=1}^N$, where $x_j \in \mathbb{R}^d$ is a set of stimuli signals. One can represent the hidden neurons as a layer that is fully connected with d input signals through the set of randomly initialised weights w. Then we can choose arbitrary activation function G and indicate that the response (denoted as H_{ji}) of a i-th neuron (out of L) for j-th stimuli signal (out of N) is given by the following formula:

$$H_{ji} = G\left(w_i, x_j\right) \tag{1}$$

Moreover, let us consider supervised binary classification problem, so that $T = \{t_j\}_{j=1}^N$, where $t_j \in \{-1, +1\}$ indicate response that has to be produced by the network for a signal x_j. Keeping in mind that weights in a hidden layer (indicated as w_i) are not trained, we can use the following formula to train the model by minimising the approximation error in the squared error sense:

$$\min_{\beta \in \mathbb{R}} \left\| H\beta - T \right\|^2 \tag{2}$$

The optimal solution in terms of mean squared error can be achieved using the Moore-Penrose (MP) pseudo-inverse (indicated as $H^\dagger = (H^T H)^{-1} H^T$) of matrix H, such that β can be calculated using formula:

$$\beta = H^\dagger T = \left(H^T H\right)^{-1} H^T T \tag{3}$$

One of the goals of ELM is to achieve good generalisation performance. It is achieved during training phase by minimising both the training error and the norm of the output weight. Therefore, it is common to introduce in formula (2) additional parameter λ controlling the regularisation strength:

$$\min_\beta \frac{\lambda}{2} \left\| H\beta - T \right\|^2 + \frac{1}{2} \left\| \beta \right\|^2 \tag{4}$$

The λ parameter allows for finding the right balance between the model complexity and the effectiveness in order to avoid the problem of overfitting. This usually requires additional tuning or additional optimisation of such grid search with cross-validation. $\widehat{\beta}$ solving this optimisation problem is well known as ridge regression solution:

$$\widehat{\beta} = \left(\lambda I + H^T H\right)^{-1} H^T T \tag{5}$$

Finally, we can easily incorporate additional parameter C_i that will let us make the ELM classifier cost-sensitive:

$$\min_\beta \frac{\lambda}{2} \sum_{i=0}^{N} C_i \left\| H\beta - T \right\|^2 + \frac{1}{2} \left\| \beta \right\|^2 \tag{6}$$

Using the previously introduced matrix notations, formula (6) can be rewritten as

$$\min_\beta \frac{\lambda}{2} \left(H\beta - T\right)^T C \left(H\beta - T\right) + \frac{1}{2} \left\| \beta \right\|^2 \tag{7}$$

Finally, the closed form solution to $\widehat{\beta}$ can be obtained using the following formula:

$$\widehat{\beta} = \left(\lambda I + H^T C H\right)^{-1} H^T C T \tag{8}$$

Additionally, substituting $C = Z^T Z$, $A = ZH$, and $B = ZT$ we can obtain more compact form:

$$\widehat{\beta} = \left(\lambda I + A^T A\right)^{-1} A^T B \tag{9}$$

Although the solution of (9) has the closed form, explicitly computing the inverse of $(\lambda I + A^T A)$ is not the most practical approach. Instead, Singular Value Decomposition (SVD) can be utilised. First, we factorise A in the way that

$$A = UDV^T \tag{10}$$

where U is $N \times L$ matrix of left singular vectors, D is $N \times N$ a diagonal matrix of singular values, and V is the $N \times N$ matrix of right singular vectors. It is easy to show that using the SVD decomposition $\widehat{\beta}$ can be expressed using the following formula:

$$\widehat{\beta} = V \left(D^2 + \lambda\right)^{-1} DU^T B \tag{11}$$

The computation of formula (11) can be effectively distributed using Map-Reduce programming model. The details on implementation can be found in [5].

3.2. Distributed Random Forest. The Random Forest (RF) classifier adapts a modification of the bagging algorithm. The difference is in the process of growing the trees. Commonly the N training samples (each with M input variables) are sampled with replacements to produce B partitions. Each of the B partitions is used to train one of the trees. Each tree is grown (trained) in a classical way by introducing nodes that split data. In case of the Random Forest classifier, the splitting point is selected only for randomly chosen variables (m out of M available). Finally, the prediction score obtained with B trained trees can be calculated using majority vote (12), where I stands for indicator function and D_b indicates b random tree in the ensemble.

$$D(x) = \arg\max_i \sum_{b=1}^{B} I(D_b = i) \tag{12}$$

Another option for classification task is the entropy measure (f_i indicates the frequency of f-th label):

$$\sum_{i=i}^{C} - f_i \log(f_i) \tag{13}$$

There already exists a scalable implementation of the Random Forest classifier in the MLlib Apache Spark library [12]. It uses the distributed computing environment, so that the computation can be parallelised. In practice, the learning process for each decision tree can be performed in parallel. Keeping in mind that each tree is trained only on the subset of data, it leads to effective schema that scales up to a large datasets. More precisely, when the Random Forest is trained in the Apache Spark environment; the algorithm samples (with the replacement) the learning data and assigns it to the decision tree, which is trained on that portion of the data. However, the data samples are not replicated explicitly, but instead it is annotated with the additional records that keep information about probability that given instance belongs to specific data partition used for training.

The training process is coordinated centrally (at so-called master node) using a queue of trees nodes. Therefore, several threes are trained simultaneously. For each node in the queue the algorithm searches for the best split. At this stage cluster resources are engaged (so-called worker nodes). The algorithm terminates when the maximum height of the decision tree is reached or whenever there is no data point that is misclassified. The final output produced by the ensemble is the majority vote of results produced by the decision trees.

One of the drawbacks of the current version of the Apache Spark Random Forest classifier is the fact that it does not handle the cost-sensitive learning (we would like to assign higher importance to data samples indicating an anomaly or cyberattack) and as a result in some cases it may be biased towards the majority class. This issue is important from the perspective of an anomaly detection or intrusion detection systems, as usually these systems should have a high recognition rate but should not overwhelm the administrator with a large number of false alarms.

3.3. Distributed Gradient-Boosted Trees. In contrast to the Random Forest classifier (where many trees can be trained simultaneously) the Boosted-Trees Classifier uses an additive learning approach. In each iteration a single tree is trained and is added to the ensemble in order to fix errors (optimise the objective function) introduced in previous iteration. This can be expressed with the following equation, where x_i indicates i-th data sample, $\widehat{y}_i^{(t)}$ is the response of the ensemble in t-th iteration, and $f_t(x_i)$ is the tree trained in t-th iteration:

$$\widehat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \widehat{y}_i^{(t-1)} + f_t(x_i) \tag{14}$$

The objective function measures the loss and the complexity of the trees comprising the ensemble. In order to handle the arbitrary loss function, common implementations of the GBT algorithms adapt the second-order Taylor expansion. Therefore, the objective function has the following form:

$$obj^{(t)} = \sum_{i=1}^{n} \left[g_i f_k(x_i) + \frac{1}{2} h_i f_k^2(x_i) \right] + C(f_t) \tag{15}$$

where n indicates the number of training samples, g_i the gradient of loss function, h_i the Hessian of loss function, and $C(f_t)$ a function measuring complexity of f_t tree. One of the methods to measure the complexity is to use the following formula:

$$C(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \tag{16}$$

where T indicates the number of leaves in the tree and w_j is the value stored at j-th leaf. γ and λ are the constants. Substituting this into (18), it is easy to show that final optimisation formula is

$$obj^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} \left(H_j + \lambda \right) w_j^2 \right] + \gamma T \tag{17}$$

where G_j and H_j indicate sum of gradients and hessians for all instances belonging to j-th leaf. The optimal w_j^* for such defined objective function is $w_j^* = -G_i/(H_j + \lambda)$ and the minimum value can be calculated as $V_{min} = -(1/2) \sum_{j=0}^{T} (G_j^2/(H_j + \lambda)) + \gamma T$ (the smaller the value is the better the tree is). However, instead of calculating the optimal w^* for fully grown trees, the typical approach is to use these formulas while growing the particular trees in a greedy manner. In other words, we can use V_{min} as node impurity measure to find the best split while building the tree. In particular, given a partial tree, a reasonable heuristic to choose the next splitting node will be the one which will decrease impurity as much as possible.

4. NetFlow Data Acquisition and Preprocessing

In order to evaluate the proposed solutions for cost-sensitive distributed machine learning, we have used datasets provided

by Malware Capture Facility Project [13]. The datasets are fully labelled and each corresponds to different scenarios of malware infections and/or botnet activities. Those scenarios have been detailed in the evaluation section. Moreover, the authors of the dataset have provided an evaluation methodology and a tool to automate it. In principle, the evaluated detection algorithm is trained on a subset of scenarios and tested on the others. It is more real-life strategy than a commonly used approach where part of a specific scenario is used for training and the remaining part for testing.

The used datasets are stored as NetFlows, which capture aggregated network properties. Usually, such data is gathered by routers and later sends the collectors. The information extracted from NetFlows is commonly used for network auditing purposes. Single NetFlow aggregates statistics (e.g., number of bytes send and received) about packets that have been sent by a specific source address to a specific destination address.

Single NetFlow usually is not enough to capture long-term malicious or anomalous behaviour of a specific node. Therefore, additional analysis of the data is required. In the proposed approach we calculate statistical properties of a group of NetFlows that have been collected for a specific source IP address. The calculations are narrowed down to a fix-length time spans called time windows. In particular, for a specific IP address we calculate the following:

(i) Number of NetFlows

(ii) Number of source ports

(iii) Number of protocols used

(iv) Number of destination IP addresses

(v) Number of destination services

(vi) Sum of bytes exchanged between source and destination

(vii) Total number of exchanged packets

Usually, the amount of collected NetFlow data is significantly sizable, even for a medium-sized network monitored for a relatively short period of time. Therefore, to make this analysis feasible and scalable we have proposed to collect the NetFlow data in a HDFS system [14], which is a distributed file system adapted by Apache Spark framework [15]. Exploiting the Map-Reduce programming concept we are able to simultaneously run the time-based calculations on the distributed NetFlow data. In general, the dataset in Apache Spark is arranged into rows of training instances (feature vectors with labels), distributed according to a number of nodes and partitions.

5. Experiments

5.1. Evaluation Methodology. A procedure taking advantage of time-based metrics has been appropriated as a proving method of the performance of the suggested ML approaches (defined in [16]). The creators of this technique have conceived and published an instrument named botnet Detectors Comparer. The aforementioned instrument examines the NetFlow file reinforced with prediction labels generated by other botnet detection methods.

Motivations exist in favour of applying time-based metrics instead of standard performance indicators applied to ML. Essentially, the mean error rates stated with the granularity of a single NetFlow are excessively rose-coloured (e.g., remarkably low rates of false alarms). In contrast, a better overview of the system effectiveness is delivered by IP-based error metrics calculated for a time window (for the most part specific devices in the network are compromised). However, immediate identification of the malevolent IP is preferred over one that is delayed in time. Therefore, a penalty system should be implemented for any delay in the flagging procedure. In order to account for that, the time component is included in the metrics calculation using the correction function:

$$cr(n) = 1 + e^{-\alpha n} \tag{18}$$

where n indicates comparison time window and α is a constant value set to 0.01. This correction function is embedded into classical effectiveness indicators as follows:

$$tTP = \frac{TP * cr}{N_{BOT}} \tag{19}$$

where tTP indicates time-based True Positive number, cr the correction function, and N_{BOT} the number of unique Botnet IP addresses present in the comparison time window. Similarly, we can define

$$tFN = \frac{FN * cr}{N_{BOT}} \tag{20}$$

On the other hand, tFP and tTN do not depend on the time and are defined as follows:

$$tFP = \frac{FP}{N_{NORM}} \tag{21}$$

where N_{NORM} indicates the number of unique normal IP addresses present in the comparison time window. Similarly, the tTN can be defined as

$$tTN = \frac{TN}{N_{NORM}} \tag{22}$$

The procedure for calculating time-based metrics is as follows:

(1) NetFlows are separated into comparison time windows (we have used default time windows of 300s length).

(2) Within the ground-truth NetFlow, labels are examined against the predicted ones and the tTP, tTN, tFP, and tFN values are amassed.

(3) Recall, Precision, Accuracy, Error Rate, and F-measure are estimated at the conclusion of each comparison time window.

(4) Finally, when the whole file with NetFlows is processed, the final error metrics are calculated and produced.

Table 1: Effectiveness comparison (IRC-based botnet scenario).

	Recall	Prec.	Acc.	ER	FM
RF	1.00	0,97	0,99	0,01	0,99
GBT	0.95	0,99	0,97	0,03	0,97
ELM	0,95	0,88	0,95	0,05	0,92

Table 2: Effectiveness comparison (scanning scenario).

	Recall	Prec.	Acc.	ER	FM
RF	1.00	1.00	1.00	0,00	1.00
GBT	1.00	1.00	1.00	0,00	1.00
ELM	1.00	0.92	0.86	0,01	0,99

Table 3: Effectiveness comparison (infection scenario).

	Recall	Prec.	Acc.	ER	FM
RF	0.92	0.99	0.95	0,05	0.96
ELM	0.99	0.89	0.93	0,06	0.94
GBT	0.67	0.99	0.77	0,21	0.80

Table 4: Effectiveness comparison (C&C communication scenario).

	Recall	Prec.	Acc.	ER	FM
RF	0.20	0.94	0.81	0,19	0.33
GBT	0.21	0.85	0.75	0,25	0.33
ELM	0.26	0.47	0.76	0,24	0.33

Each time the algorithm spots a Botnet IP address in the comparison time window correctly, the True Positive counter value is raised. Likewise, a normal IP address evaluated as a non-botnet address increments the True Negative result. Each occurrence of a benign IP classified as a botnet address increments the false positive value. At every instance of a Botnet IP judged as non-botnet the False Negative counter is raised.

5.2. Results. In this section, we have presented the quantitative evaluation of the proposed cost-sensitive classification methods (Recall, Precision, Accuracy, Error Rate, and F-measure). In particular, the Tables 1–4 contain results obtained for several scenarios recorded in the testing dataset and containing the botnet activities.

The first scenario is related to the IRC-based botnet whose main activity focuses on spamming. The results are shown in Table 1. In general, the results obtained with Random Forest and Gradient-Boosted Trees classifiers are comparable. It can be noticed that Random Forest classifier is not the one with the highest precision, but it has the highest recall rate. The Extreme Learning Machine classifier in that scenario achieved the same value of recall as GBT, but it exhibits significantly poorer precision than other classifiers.

The second scenario contains traffic recorded for the botnet that scans the SMPT mail servers for few hours and connects to several remote desktop services. However, the malware does not send any spam. The results are shown in Table 2. In this scenario both RF and GBT achieved similar

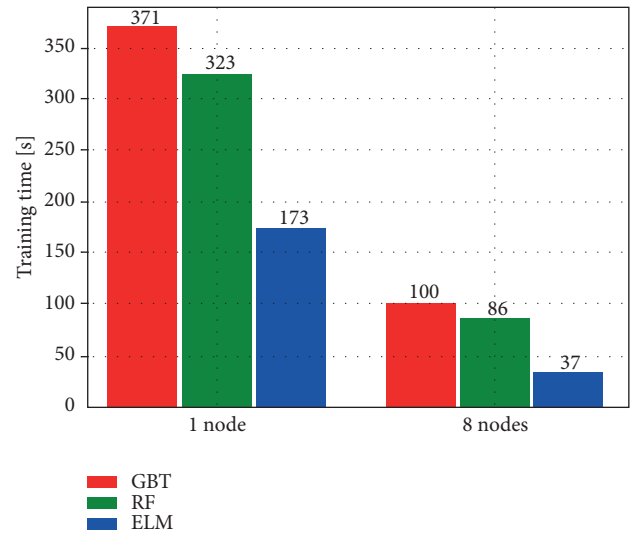

Figure 3: Classifiers training time with respect to number of utilised computing nodes in the Apache Spark System.

effectiveness results. Again, for the ELM classifier, we have reported poorer precision than for other classifiers.

In the third scenario, the traffic records were collected for some hosts that were infected with the Neris malware. After the infection, the host actively starts sending spam e-mails. The results are shown in Table 3. In this scenario, the GBT classifier achieved the worst results among the other classifiers, although the precision of the classifier was similar to the RF classifier (the best one in this scenario). For the GBT classifier we have observed the lowest recall. The error rate for GBT was also significantly higher than the error rate reported for ELM and RF classifiers.

In the fourth scenario, we have evaluated the analysed methods on the traffic records of an infected hosts activity. In this scenario, the infected host contacts various C&C (Command and Control) hosts and receives some encrypted data. The results are shown in Table 4. For all classifiers, we have reported poor results. The highest precision among all classifiers is 94%. We can observe low recall for all classifiers.

6. Computational Performance Evaluation

In this section we have provided an overview of the computational performance evaluation. The presented methods are compared with regard to the time necessary to train their specific models. The experimental results have been presented in Figure 3.

In the presented experiment we have reported the training times for different sizes of the Apache Spark cluster. As it is shown in Figure 3 we measure the training times for 1 and 8 Apache Spark workers, respectively. As it was expected the GBT algorithm turned out to be the most time consuming. The ELM algorithm achieved the shortest learning times. As the experiment showed, increasing the number of worker nodes allowed us to accelerate the training process by a factor of 4.5 in case of ELM and 3.7 in case of RF and GBT. This experiment allows us to draw the conclusion that the

ELM classifier can be considered as a solid choice to solve the anomaly detection problem due to its time efficiency and scalability, even though the tree-based techniques allow achieving a slightly higher effectiveness.

7. Conclusions

The major contribution of this paper is the proposition of NetFlow-based Botnet Activity Detection solutions that leverage cost-sensitive distributed machine learning algorithms. Notably, a method for NetFlow aggregation and analysis is suggested, evaluated, and compared. The proposed approach leverages the Apache Spark framework and Distributed Hadoop File System (HDFS) for efficient and effective time-based calculations. Insights into technical aspects of three classifiers, namely, Extreme Learning Machines, Random Forest, and Gradient-Boosted Trees are provided. The proposed approach is evaluated using benchmark datasets and produces promising results.

The presented research continues our prior work on a scalable and distributed system for NetFlow data analysis. In particular, in this work, we introduced two additional cost-sensitive algorithms (Random Forest and Gradient-Boosted Trees) that have been deployed in the proposed system. We claim that the ELM-based approach is still a reasonable choice due to its simplicity and scalability, even though the tree-based techniques allow achieving slightly better (or comparable) results. In particular, the ELM allows for efficient cluster resources utilisation thanks to Map-Reduce programming model. In contrast to that, GBT requires more iterative approach when trained, because (as it was mentioned in Section 3.3) each new tree corrects the errors from previous iterations. In case of RF classifier, several trees can be trained in parallel. However, the results show that the computation burden does not pay off, as we can observe improvement of 1%-2% (in terms of F-measure metric).

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

We hereby acknowledge our research results presented in [17] that we have used to prepare Related Work section.

References

[1] M. Choraś, R. Kozik, D. Puchalski, and W. Hołubowicz, "Correlation Approach for SQL Injection Attacks Detection," in *Advances in Intelligent and Soft Computing*, A. Herrero et al., Ed., vol. 189, pp. 177–185, Springer, 2012.

[2] S. Abt and H. Baier, "Towards efficient and privacy-preserving network- Based botnet detection using netflow data," in *Proceedings of the 9th International Network Conference, INC 2012*, 2012.

[3] R. Kozik and M. Choraś, "Pattern Extraction Algorithm for NetFlow-Based Botnet Activities Detection," *Security and Communication Networks*, vol. 2017, Article ID 6047053, 10 pages, 2017.

[4] R. Kozik and M. Choraś, "Solution to Data Imbalance Problem in Application Layer Anomaly Detection Systems," in *Hybrid Artificial Intelligent Systems, HAIS*, F. Martinez-Alvarez, A. Troncoso, H. Quintian, and E. Corchado, Eds., vol. 9648, pp. 441–450, Springer, 2016.

[5] R. Kozik, "Distributing extreme learning machines with Apache Spark for NetFlow-based malware activity detection," *Pattern Recognition Letters*, vol. 101, pp. 14–20, 2018.

[6] R. Kozik, M. Choraś, W. Hołubowicz, and R. Renk, "Extreme Learning Machines for Web Layer Anomaly Detection," in *Image Processing and Communications Challenges 8*, vol. 525 of *Advances in Intelligent Systems and Computing*, pp. 226–233, Springer, 2017.

[7] M. Grill, I. Nikolaev, V. Valeros, and M. Rehak, "Detecting DGA malware using NetFlow," in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 1304–1309, Ottawa, ON, Canada, May 2015.

[8] Q. A. Tran, F. Jiang, and J. Hu, "A Real-Time NetFlow-based Intrusion Detection System with Improved BBNN and High-Frequency Field Programmable Gate Arrays," in *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 201–208, Liverpool, UK, June 2012.

[9] K. Flanagan, E. Fallon, A. Awad, and P. Connolly, "Self-configuring NetFlow anomaly detection using cluster density analysis," in *Proceedings of the 2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 421–427, Pyeongchang, Kwangwoon Do, South Korea, 2017.

[10] X. Yuan, "PhD Forum: Deep Learning-Based Real-Time Malware Detection with Multi-Stage Analysis," in *Proceedings of the 2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 1-2, Hong Kong, China, May 2017.

[11] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 42, no. 2, pp. 513–529, 2012.

[12] Apache Spark, "Random Forest Classifier," https://spark.apache.org/docs/2.2.0/ml-classification-regression.html#random-forest-classifier.

[13] "The Malware Capture Facility Project, Project homepage," https://mcfp.weebly.com/.

[14] HDFS (Hadoop Distirbuted File System), "Architecture Guide," https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[15] SparkSQL, "Apache Spark framework –project homepage," http://spark.apache.org.

[16] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Journal of Computers and Security*, vol. 45, pp. 100–123, 2014.

[17] R. Kozik, M. Pawlicki, and M. Chora, "Sparse Autoencoders for Unsupervised Netflow Data Classification," in *Image Processing and Communications Challenges 10. IP&C 2018*, M. Chora and R. Chora, Eds., Advances in Intelligent Systems and Computing, Springer, 2018.