Hindawi
Security and Communication Networks
Volume 2019, Article ID 4656480, 10 pages
https://doi.org/10.1155/2019/4656480

Research Article

Optimizing Computer Worm Detection Using Ensembles

Nelson Ochieng,[1] **Waweru Mwangi**,[2] **and Ismail Ateya**[1]

[1]*Strathmore University, Kenya*
[2]*Jomo Kenyatta University of Agriculture and Technology, Kenya*

Correspondence should be addressed to Nelson Ochieng; nochieng@strathmore.edu

Received 13 December 2018; Revised 26 February 2019; Accepted 3 March 2019; Published 11 April 2019

Guest Editor: Pelin Angin

The scope of this research is computer worm detection. Computer worm has been defined as a process that can cause a possibly evolved copy of it to execute on a remote computer. It does not require human intervention to propagate neither does it attach itself to an existing computer file. It spreads very rapidly. Modern computer worm authors obfuscate the code to make it difficult to detect the computer worm. This research proposes to use machine learning methodology for the detection of computer worms. More specifically, ensembles are used. The research deviates from existing detection approaches by using dark space network traffic attributed to an actual worm attack to train and validate the machine learning algorithms. It is also obtained that the various ensembles perform comparatively well. Each of them is therefore a candidate for the final model. The algorithms also perform just as well as similar studies reported in the literature.

1. Introduction

Malware includes computer virus, Trojan horse, spyware, ad-ware, computer worms among many others. In survey by [1], a malware event occurs in organizations every 3 minutes and attacks many sectors with alarming losses to intellectual property, compromised customer records and even destruction of data. This research has as its scope computer worm detection in a network. Reference [2] defines a computer worm as a "process that can cause a (possibly evolved) copy of it to execute on a remote computational machine". Worms self-propagate across computer networks by exploiting security or policy flaws in widely used network services. Unlike computer viruses, computer worms do not require user intervention to propagate nor do they piggy-back on existing files. Their spread is very rapid [3, 4] with the ability to infect as many as 359,000 computers in under 14 hours, or even faster. Computer worms therefore present unique challenges to security researchers hence motivating this study.

Defense against computer worm attacks may be through prevention of worm attacks, detection of worms, containment of worm spread and removal of worm infections. Prevention is not always wholly possible because of the inherent vulnerabilities found in all software. Detection is therefore the better approach.

A number of computer worm detection approaches have been explored in the research environment. Content-based fingerprinting captures a worm's characteristics by deriving the most representative content-sequence as the worm's signature. Anomaly-detection leverages the fact that worms are likely to exhibit anomalous behavior such as port-scanning and failed connection attempts, which are distinct from normal behavior. Behavioral foot-printing makes use of the fact that each worm exhibits a definite communication pattern as it propagates between hosts in a network and these patterns can be used to uniquely identify a worm. Intelligent detection approaches that use machine learning have also been proposed. While each of these approaches has its strengths, a number of weaknesses have also been noted. For example, content-signature schemes, while an established dimension to detect worms, fail to detect novel worms and are expensive on the system. In anomaly detection, profiling normal network behavior is impossible and establishing detection threshold is also difficult. Behavioral foot-printing is prone to behavior camouflaging attacks. Approaches that leverage machine learning have generated high false positive

and false negative rates. This has been partly because of poor characterization of worm traffic and also because of the lack of sound datasets for training and validation of the algorithms.

This paper presents an approach that attempts to provide better performance. The feature set used for the machine learning algorithms are selected network packet header fields as reported in an earlier paper by the authors [5]. The rest of the paper is organized as follows. Section 2 reviews existing literature on computer worm detection using machine learning. Section 3 discusses the methodology for the research. Section 4 discusses the results. The paper concludes with a summary in Section 5.

2. Related Work

A number of approaches for computer worm detection have been reviewed in the literature. These include content-based signature schemes, anomaly-detection schemes, and behavioral-signature detection schemes, a summary and analysis of which has been presented by the authors in an earlier paper [6]. For this present work, only approaches that utilize machine learning are emphasized. Reference [7] was one of the seminal works in using machine learning techniques for malware detection. It used static program binary properties and achieved a detection rate of 97.76%. Reference [8] used n-grams extracted from the executable to form training examples. They apply several learning methods such as Nearest Neighbors, Naïve Bayes, Support Vector Machines, Decision Trees, and Boosting. Boosted Decision Trees performed the best with an Area under Curve (AUC) of 0.996. Win32 Portable Executables (PE) as a feature is used by [9–11]. Paper [12] achieves a True Positive Rate of 98.5% and a False Positive Rate of 0.025 using Windows Application Programming Interface (API) calls as the features. Other feature types used include Operation Code (OPcode) [13], sequence of instructions that capture program control flow information [14], and binary images [15]. Reference [15] obtains an accuracy of 98%. Paper [16] uses a restricted Boltzmann machine, a neural network, to create a new set of features from existing ones. These are then used to train a one-side perceptron (OSP) algorithm. The work gets very close to obtaining a zero false positive classifier. Paper [17] uses Logistic Model Trees, Naïve Bayes, Support Vector Machines (SVM), and k Nearest Neighbors (kNN) and obtains an accuracy of 98.3% with the Linear Model Tree algorithm. A deep neural network that uses byte entropy histogram, PE import features and PE metadata features is deployed by [17] and achieves a detection rate of 95% and a false positive rate of 0.1%. Reference [18] also uses deep learning.

Most of the reviewed works utilize a single parameter for the detection. The present work will utilize many features as reported by the authors in [5].

3. Methods

The main aim of this work is to investigate various machine learning ensembles on computer worm detection using unidirectional network traffic to a dark space. The methodology adopted follows the standard procedure in machine learning: (1) collecting data, (2) exploring and preparing the data, and (3) training a model on the data and evaluating model performance.

3.1. Dataset. The datasets used for the experiments were obtained from the University San Diego California Center for Applied Data Analysis (USCD CAIDA). The center operates a network telescope that consists of a globally rooted /8 network that monitors large segments of lightly used address space. There is little legitimate traffic in this address space; hence, it provides a monitoring point for anomalous traffic that represents almost 1/256th of all IPv4 destination addresses on the Internet.

Two sets of datasets were requested and obtained from this telescope. The first is the Three Days of Conficker Datasets [19] containing data for 3 days between November 2008 and January 2009 during which Conficker worm attack [20] was active. This dataset contains 68 compressed packet capture (pcap) files each containing one hour of traces. The pcap files only contain packet headers with the payload having been removed to preserve privacy. The destination IP addresses have also been masked for the same reason. The other dataset is the Two Days in November 2008 dataset [21] with traces for the 12th and 19th November 2008, containing two typical days of background radiation just prior to the detection of Conficker which has been used to differentiate between Conficker-infected traffic and clean traffic.

The datasets were processed using the CAIDA Corsaro software suite [22], a software suite for performing large-scale analysis of trace data. The raw pcap datasets were aggregated into the FlowTuple format. This format retains only selected fields from captured packets instead of the whole packet, enabling a more efficient data storage, processing and analysis. The 8 fields are source IP address, destination IP address, source port, destination port, protocol, Time to Live, TCP flags, and IP packet length. An additional field, value, indicates the number of packets in the interval whose header fields match this FlowTuple key.

The instances in the Three Days of Conficker dataset have been further filtered to retain only instances that have a high likelihood of being attributable to Conficker worm attack of the year 2008. Reference [20] focuses on Conficker's TCP scanning behavior (searching for victims to exploit) and indicates that it engages in three types of observable network scanning via TCP port 445 or 139 (where the vulnerable Microsoft software Windows Server service runs) for additional victims. The vulnerability allowed attackers to execute arbitrary code via a crafted RPC request that triggers a buffer overflow. These include local network scanning where Conficker determines the broadcast domain from network interface settings, scans hosts nearby other infected hosts and random scanning. Other distinguishing characteristics include TTL within reasonable distance from Windows default TTL of 128, incremental source port in the Windows default range of 1024-5000, 2 or 1 TCP SYN packets per connection attempt instead of the usual 3 TCP SYN packets per connection attempt due to TCP's retransmit behavior.

This dataset solves the privacy challenge by removing the payload and also masking out the first octet of the destination IP address. It is also a more recent dataset than the KDD dataset that has been the one available for network security researchers. However, it only includes unidirectional traffic to the network telescope and therefore does not allow the researchers to include features of computer worms that would be available in bidirectional traffic and would deliver a more complete training for the classifiers.

3.2. Features. This section presents an analysis of the features to be used for detection and their contribution towards the detection capability of the learning algorithms. These features were obtained after performing feature selection experiments whose results were reported in [5]. The best features for the classification task were there identified as Time to Live (TTL), Internet Protocol (IP) packet length, value or number of packets in the packet capture interval whose header fields match the Flow Tuple key, well known destination ports or destination ports within the range 0-1024, and IP packet source country China. The above features are IP packet header fields. TTL is used to avoid looping in the network. Every packet is sent with some TTL value set, which tells the network how many network routers (hops) this packet can cross. At each hop, its value is decremented by one and when the value reaches zero, the packet is discarded. Different operating systems have default TTL ranges and since computer worms target vulnerabilities in particular operating systems, they will usually be associated with TTL within certain ranges. For example, Conficker worm packets have TTL within reasonable distance from Windows default TTL of 128. Packet length indicates the size of the packet. Particular computer worms are associated with particular packet length sizes. For example, the packet length for Conficker worm is around 62 bytes. The value feature referred to the number of packets with a unique packet header signature sequence. A number of flow tuples with a particular key would be suspicious. China originates most of the malicious software packets. Computer worms target well known ports where popular services run for maximum impact. Conficker worm, for example, targets port 445 or 139.

3.3. Ensembles. Various machine learning ensembles were explored and their detection capabilities investigated. Ensemble methods try to construct a set of learners and combine them. The ensemble methods investigated included averaging technique, GradientBoostingClassifier, AdaBoost, Bagging, Voting, Stacking, Random Forests, and ExtraTreesClassifier. The base classifiers used included SVM, Multilayer perceptrons, kNN, NB, Logistic Regression, and Decision Trees. Python programming language was used for the classification experiments and more especially the Scikit-learn library [22]. These ensemble techniques are described as follows.

3.3.1. ExtraTreesClassifier. This builds an ensemble of unpruned decision trees according to the classical top-down procedure [23].

The Extra-Trees splitting procedure for numerical attributes is given in Algorithm 1. It has two parameters: K, the number of attributes randomly selected at each node and nmin, the minimum sample size for splitting a node. It is used several times with the (full) original learning sample to generate an ensemble model (we denote by M the number of trees of this ensemble). The predictions of the trees are aggregated to yield the final prediction, by majority vote in classification problems.

3.3.2. Random Forests. Paper [24] defines a random forest as a classifier consisting of a collection of tree-structured classifiers h(x, θk), k = 1,..., where the θk are independent, identically distributed random vectors and each tree casts a unit vote for the most popular class at input x. This is as shown in Algorithm 2.

3.3.3. AdaBoost. Reference [25] explains AdaBoost as taking as input a training set $(x_1, y_1)\ldots(x_m, y_m)$ where each x_i belongs to some domain or instance space X, and each label y_i is in some label set Y. AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds t = 1...T. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example i on round t is denoted Dt (i). Initially, all weights are set equally, but on each round the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. AdaBoost is shown in Algorithm 3.

3.3.4. Bagging. The name Bagging came from the abbreviation of Bootstrap AGGregatING [26]. The two key ingredients of Bagging are bootstrap and aggregation. Bagging applies boostrap sampling to obtain the data subsets for training the base learners. Given a training data set containing m number of training examples, a sample of m training examples will be generated by sampling with replacement. Each of these datasets is used to train a model. The outputs of the models are combined by averaging (in regression) or voting (in classification) to create a single output. Algorithm 4 shows Bagging.

3.3.5. Gradient Boosting. Gradient Boosting [27] is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. Gradient Boosting is shown in Algorithm 5.

3.3.6. Voting. Voting is the most popular and fundamental combination method for nominal outputs. In majority voting, every classifier votes for one class label, and the final output class label is the one that receives more than half of the votes; if none of the class labels receives more than half of

Split a node(S)
Input: the local learning subset S corresponding to the node we want to split
Output: a split $[a < a_c]$ or nothing
(i) If **Stop split**(S) is TRUE then return nothing.
(ii) Otherwise select K attributes $\{a_1, \ldots, a_K\}$ among all non-constant (in S) candidate attributes;
(iii) Draw K splits $\{s_1, \ldots, s_K\}$, where s_i = **Pick a random split**(S, a_i), $\forall i = 1, \ldots, K$;
(iv) Return a split s^* such that Score(s^*, S) = $\max_{i=1,\ldots,K}$ Score(s_i, S).
Pick a random split(S,a)
Inputs: a subset S and an attribute a
Output: a split
(i) Let $a_{S_{\max}}$ and $a_{S_{\min}}$ denote the maximal and minimal value of a in S;
(ii) Draw a random cut-point a_c uniformly in $[a_{S_{\min}}, a_{S_{\max}}]$;
(iii) Return the split $[a < a_c]$.
Stop split(S)
Input: a subset S
Output: a boolean
(i) If $|S| < n_{\min}$, then return TRUE;
(ii) If all attributes are constant in S, then return TRUE;
(iii) If the output is constant in S, then return TRUE;
(iv) Otherwise, return FALSE.

Algorithm 1: Extra Trees Algorithm.

Input: Learning set- S, Ensemble size B, Proportion of attributes considered f
Output: Ensemble E
(1) E = φ
(2) **for** i = 1 to B **do**
(3) S^i=Boostrap Sample(S)
(4) C^i=Build Random Tree Model (S^i, f)
(5) E = E ∪ {C^i}
(6) **return** E

Algorithm 2: Random forest.

the votes, a rejection option will be given and the combined classifier makes no prediction.

3.3.7. Stacking. Reference [28] explains that Stacked Generalization is a method for combining heterogeneous base models, that is, models learned with different learning algorithms such as the nearest neighbor method, DTs, NB, among others. The base models are not combined with a fixed scheme such as voting, but rather an additional model called meta model is learned and used for combining base models. First, the meta learning dataset is generated using predictions of the base models and then, using the meta learning set, the meta model is learned which can combine predictions of base models into a final prediction.

Input: Learning set- S, Ensemble size B.
Output: Ensemble E
(1) E = φ
(2) W=Assign Equal Weights (S)
(3) **for** i = 1 to B **do**
(4) C^i=Construct-Models (S, W)
(5) E_{rr}=Apply Model (C^i, S)
(6) **if** (E_{rr} = 0) ∪ (E_{rr} ≥ 0.5) **then**
(7) Terminate Model Generation
(8) **return** E
(9) **for** j = 1 to Number Of Examples (S) **do**
(10) if Correctly Classified (S_j, C^i) **then**
(11) $W_j = W_j \; E_{rr} / 1 - E_{rr}$
(12) W = Normalize Weights W
(13) E = E ∪ {C^i}
(14) **return** E

Algorithm 3: AdaBoost.

3.4. Ensemble Experiments. Ensemble experiments started with a comparison of the base classifiers. The base classifiers performed as shown in Figure 1.

Multilayer perceptron performed poorest and was therefore considered for elimination.

To build an ensemble of various models, the experiments started by benching a set of Scikit-learn classifiers on the

Input: Learning set- S, Ensemble size B.
Output: Ensemble E
(1) E = φ
(2) **for** i = 1 to B **do**
(3) S=Boostrap Sample(S)
(4) C=Construct-Base Model (S^i)
(5) E = E ∪ {C^i}
(6) **return** E

Algorithm 4: Bagging.

Inputs:
(i) input data (x, y) Ni=1
(ii) number of iterations M
(iii) choice of the loss-function (y, f)
(iv) choice of the base-learner model h (x, θ)
Algorithm:
(**1**) initialize f_0 with a constant
(2) for t = 1 to M do
(3) compute the negative gradient gt(x)
(4) fit a new base-learner function h(x, θt)
(5) find the best gradient descent step-size ρt : ρt = arg minρ N i=1 yi, ft−1(xi) + ρh(xi, θt)
(6) update the function estimate: ft ⟵ ft−1 + ρth(x, θt)
(**7**) **end for**

Algorithm 5: Gradient boosting.

dataset. The considered models performed as shown in Table 1 .

It is evident that the base classifiers performed almost equally well in terms of accuracy.

A way to understand what is going on in an ensemble when the task is classification is to inspect the Receiver Operator Curve (ROC). This curve shows the tradeoff between precision and recall or the rate of true positives versus true negatives. Typically, different base classifiers make different tradeoffs. An ensemble can adjust these. The ROC curve obtained for the various classifiers and how they compare to the ensemble averaging technique is shown in Figure 2.

Random Forest reported the results in Table 2 and Figures 3 and 4.

A Cohen Kappa score of 0.932 was obtained for Random Forest.

Experiments with ExtraTreesClassifier gave the results shown in Table 3 and Figures 5 and 6.

AdaBoost gave the results reported in Table 4 and Figure 7.

Bagging gave the results reported in Table 5 and Figure 8.

Voting reported the results as shown in Table 6 and Figure 9.

4. Discussion of Results

Before the construction of classifier ensembles, it was found out that errors were significantly correlated for the different

Table 1: Benchmarking models.

Model	Score
Logistic Regression	0.943
Decision Tree	0.975
Support Vector Machines	0.990
Naïve Bayes	0.985
K Nearest Neighbors	0.954

Table 2: Random Forest.

Accuracy	Precision	Recall	F1
0.896	0.907	0.888	0.895

Table 3: ExtraTreesClassifier scores.

AUC	Accuracy	Precision	Recall	F1
0.972	0.887	0.889	0.884	0.889

Table 4: AdaBoost scores.

AUC	Accuracy	Precision	Recall	F1
0.993	0.915	0.861	1.0	0.925

Table 5: Bagging scores.

AUC	Accuracy	Precision	Recall	F1
0.995	0.911	0.873	0.971	0.919

Table 6: Voting scores.

Accuracy	Precision	Recall	F1
0.919	0.872	0.992	0.927

classifiers, which is to be expected for models that perform well. Yet most correlations were in the 50-80% span, showing decent room for improvement could be realized by ensembles.

When ROC curves were plotted for the averaging ensemble technique and the base algorithms, the ensemble technique outperformed Logistic Regression, Decision Tree, and kNN. This was as shown in Figure 2 where the curve for the ensemble technique approached the left topmost corner the most. The ensemble technique performed almost as well as the NB and SVM classifiers. Trying to improve the ensemble by removing the worst offender (Logistic Regression in Figure 2) gave a truncated ensemble ROC-AUC score of 0.990, a further improvement.

Table 7 summarizes the performance of the ensemble classifiers.

The highest ROC-AUC score was achieved by GradientBoosting (0.997) while the lowest was achieved by Random Forest (0.970). The figures were however all high and not very different from one another indicating that all ensemble techniques perform well. Voting was removed from the comparison as it was slow, especially with more base classifiers integrated. Some of the ensemble classifiers however did not generalize well. These were ExtraTreesClassifier and

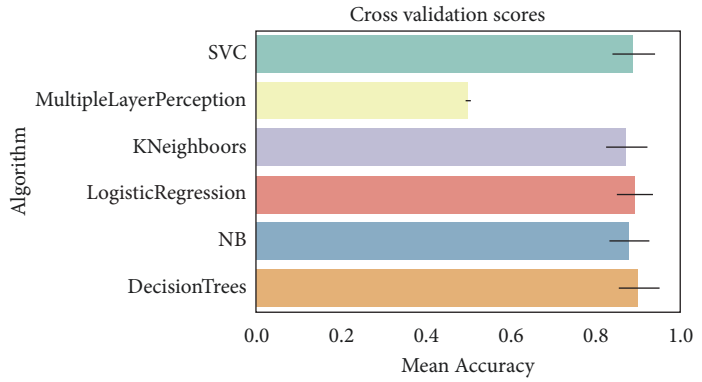

Figure 1: Comparison of base classifiers.

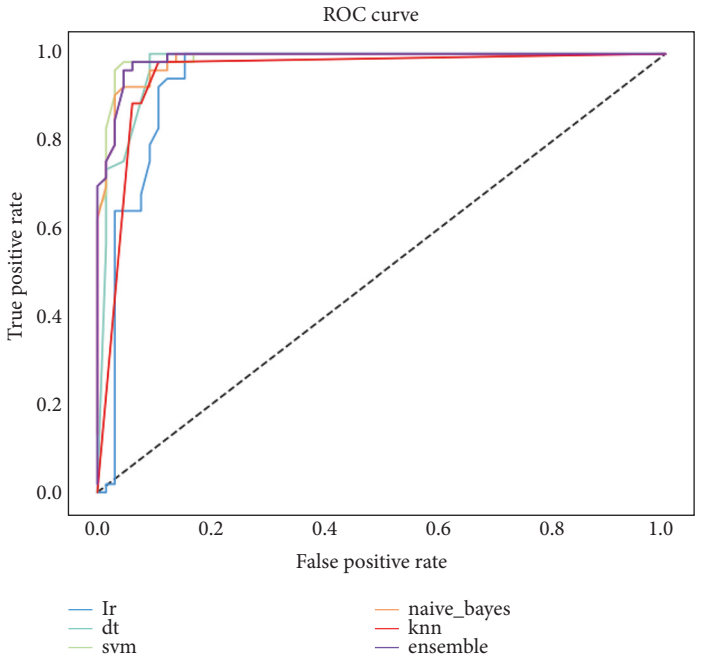

Figure 2: ROC comparison of simple average and base classifiers.

Table 7: Performance comparison of the ensemble classifiers.

Algorithm	AUC
Gradient Boosting	0.997
Random Forest	0.970
Extra Trees Classifier	0.972
AdaBoost	0.993
Bagging	0.995
Stacking	0.986
Simple Averaging	0.987

Random Forest. The rest of the ensemble techniques investigated generalized well including the slower voting ensemble technique.

It was evident that ensemble techniques improved obtained scores higher than some base learners though the performance difference was not significant as would be expected.

5. Conclusion

The study addressed the problem of detecting computer worms in networks. The main problem to be solved was that existing detection schemes fail to detect sophisticated computer worms that use code obfuscation techniques. In addition, many existing schemes use single parameter for the detection leading to a poorer characterization of the threat model hence a high rate of false positives and false negatives. The datasets used in many approaches is also outdated. The study aimed to develop a behavioral machine learning model to detect computer worms. The datasets used for the experiments were obtained from the University San Diego California Center for Applied Data Analysis (USCD CAIDA).

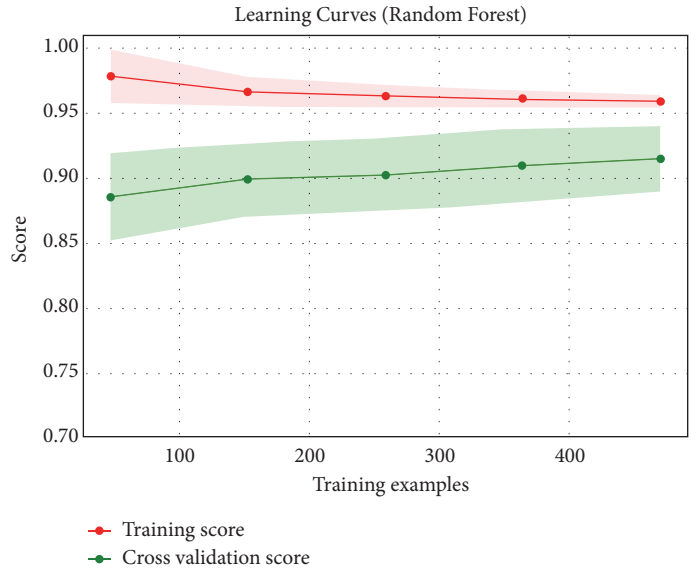

Figure 3: Learning curve for Random Forest.

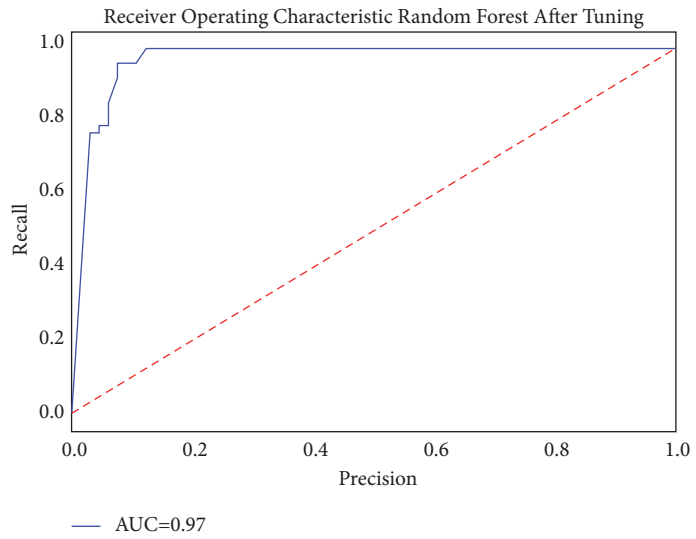

Figure 4: ROC for Random Forest.

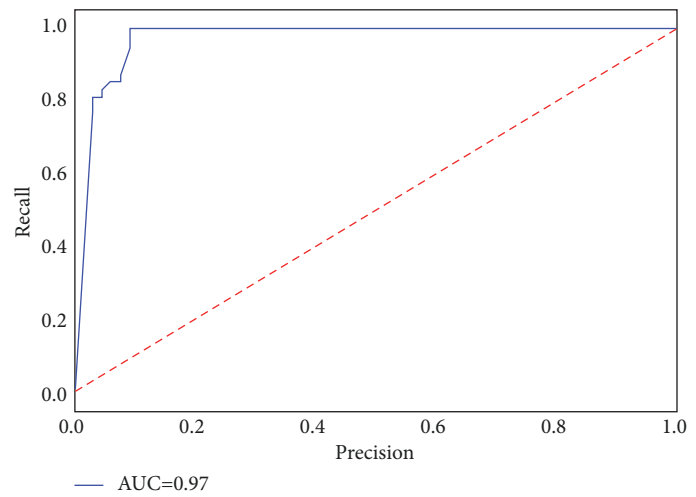

Figure 5: ROC for ExtraTreesClassifier.

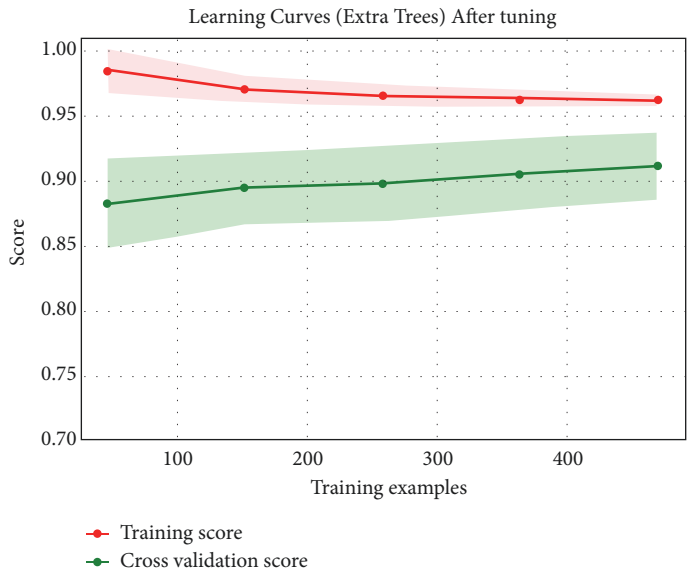

Figure 6: Learning curves for ExtraTreesClassifier.

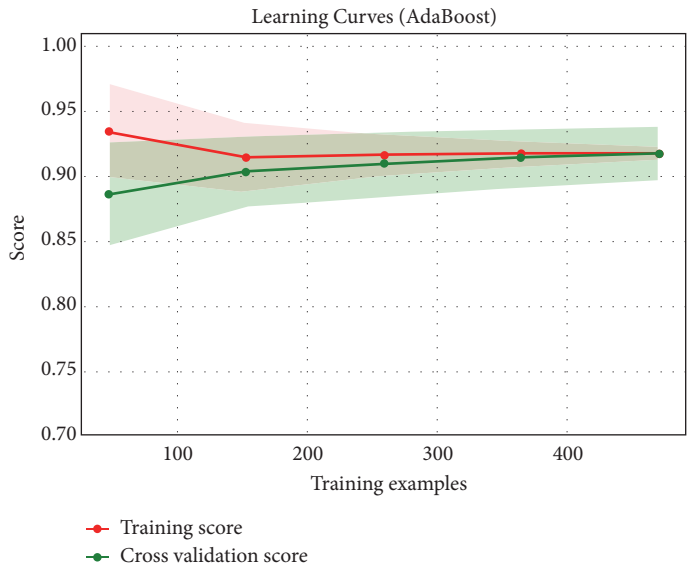

Figure 7: AdaBoost learning curve.

The results were promising in terms of accuracy and generalization to new datasets. There were no marked differences between the classifiers, especially when the datasets were standardized.

It is apparent that the particular classifier used may not be the determinant in classification in machine learning experiments but rather the choice of features. While this is largely consistent with other similar studies, it should be further confirmed by future research.

It is true that not all computer worms can be detected by a single method. In future, it is recommended that a combination of different detection approaches be combined to be able to detect as many types of computer worms as possible. Also, the span of features used for detection should be expanded to include even more features for the detection. The contribution of each feature to the detection ability should be documented.

Unsupervised learning has not been investigated in this research. Unlabeled traffic datasets are available to security researchers and practitioners. The cost of labeling them is high. This makes unsupervised learning useful for threat detection. The manual effort of labeling new network traffic can make use of clustering and decrease the number of labeled objects needed for the usage of supervised learning.

Data Availability

The packet capture (pcap) data used to support the findings of this study are provided by the UCSD, Center

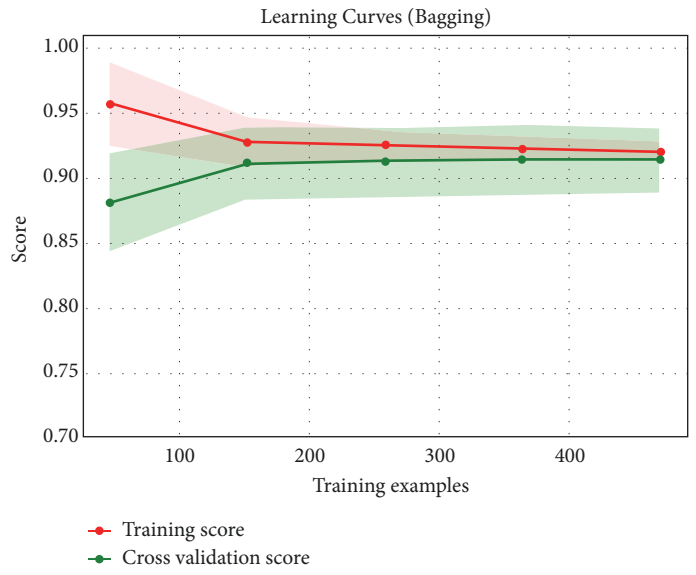

Figure 8: Bagging learning curve.

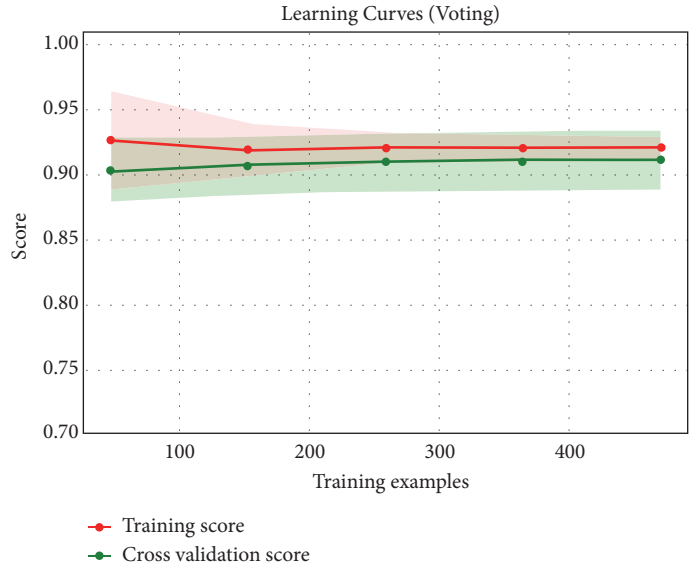

Figure 9: Learning curve for voting.

for Applied Internet Data Analysis. Two datasets were used: 1. the CAIDA UCSD Network Telescope "Two Days in November 2008" Dataset and 2. the CAIDA UCSD Network Telescope "Three Days of Conficker". They may be released upon application to IMPACT Cyber Trust, who can be contacted at the website address https://www.impactcybertrust.org/dataset_view?idDataset=382 and https://www.impactcybertrust.org/dataset_view?idDataset=383. The Corsaro tool that was used to process the pcap files is available as an open source tool at the Center for Applied Internet Data Analysis (CAIDA) website at https://www.caida.org/tools/measurement/corsaro/.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] FireEye, The need for speed, https://www2.fireeye.com/ismg-incident-response-survey.html, 2018.

[2] D. Ellis, "Worm anatomy and model," in *Proceedings of the ACM Workshop on Rapid Malcode (WORM '03)*, pp. 42–50, ACM, October 2003.

[3] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security & Privacy*, vol. 1, no. 4, pp. 33–39, 2003.

[4] D. Moore, C. Shannon, and K. Claffy, "Code-Red: a case study on the spread and victims of an internet worm," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment (IMW '02)*, pp. 273–284, Marseille, France, November 2002.

[5] O. Nelson, W. Mwangi, and I. Ateya, "A Hybrid Filter/Wrapper Method for Feature Selection for Computer Worm Detection

using Darknet Traffic," *International Journal of Computer Applications*, vol. 180, no. 44, pp. 12–17, 2018.

[6] N. Ochieng, W. Mwangi, and I. Ateya, "A Tour of the Computer Worm Detection Space," *International Journal of Computer Applications*, vol. 104, no. 1, pp. 29–33, 2014.

[7] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 38–49, May 2001.

[8] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *Journal of Machine Learning Research*, vol. 7, pp. 2721–2744, 2006.

[9] Z. Markel, "Machine learning based malware detection," https://apps.dtic.mil/dtic/tr/fulltext/u2/a619747.pdf, 2015.

[10] P. Vinod, V. Laxmi, M. S. Gaur, and G. Chauhan, "Detecting malicious files using non-signature-based methods," *International Journal of Information and Computer Security*, vol. 6, no. 3, pp. 199–240, 2014.

[11] J. Bai, J. Wang, and G. Zou, "A malware detection scheme based on mining format information," *The Scientific World Journal*, vol. 2014, Article ID 260905, 11 pages, 2014.

[12] M. Alazah, S. Venkatranan, and P. Watters, "Zero-day malware detection based on supervised learning algorithms of api call signatures," in *Proceedings of the 9th Australasian Data Mining Conference*, Data Mining and Analysis, 2011.

[13] A. Mamoun, H. Shamsul, A. Jemal et al., "A hybrid wrapper-filter approach for malware detection," *Journal of Networks*, vol. 9, no. 11, 2014.

[14] S. Muazzam, W. Morgan, and L. Joohan, "Detecting Internet worms using data mining techniques," *Journal of Systematics, Cybernetics and Informatics*, 2009.

[15] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware Images: visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ACM, 2011.

[16] R. Benchea and D. T. Gavrilut, *Combining Restricted Boltzmann Machine adn One Side Perceptron for Malware Detection*, Springer International Publishing, 2014.

[17] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proceedings of the 10th International Conference on Malicious and Unwanted Software, (MALWARE '15)*, pp. 11–20, USA, October 2015.

[18] W. Huang and J. W. Stokes, "MtNet: a multi-task neural network for dynamic malware classification," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2016.

[19] The CAIDA UCSD Network Telescope, "Three Days of Conficker," http://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml.

[20] E. Aben, "Conficker/Conflicker/Downadup as seen from the UCSD Network Telescope," Technical Report, CAIDA, February 2009, https://www.caida.org/research/security/ms08-067/conficker.xml.

[21] The CAIDA UCSD Network Telescope, "Two Days in November 2008," Dataset, http://www.caida.org/data/passive/telescope-2days-2008-dataset.xml.

[22] F. Pedregosa, G. Varoquaux, and A. Gramfort, "Scikit-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[24] S. Dzeroski, P. Panov, and B. Zenko, *Ensemble Methods in Machine Learning*, Encyclopedia of Complexity and Systems Science, 2009.

[25] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal of Japanese Society For Artificial Intelligence*, vol. 14, no. 771–780, p. 1612, 1999.

[26] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[27] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, vol. 7, article 21, 2013.

[28] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.