5-2018

Comparison of Machine Learning Algorithms and Their Ensembles for Botnet Detection

Songhui Ryu

Purdue University

COMPARISON OF MACHINE LEARNING ALGORITHMS AND THEIR ENSEMBLES FOR BOTNET DETECTION

by

Songhui Ryu

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Department of Computer and Information Technology

West Lafayette, Indiana

May 2018

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

ACKNOWLEDGMENTS

I wish to gratefully acknowledge my thesis advisor, Dr. John Springer for his insightful comments and guidance. He was always open to help me and willing to solve my doubts whenever I struggled with my research question or writing. He consistently allowed this paper to be my own work, but encouraged me to proceed in the right the direction whenever he thought I needed it. I would also love to acknowledge my thesis committee members, Dr. Eric Matson and Dr. Baijian Yang. They helped me to constantly stick to my research providing motivation and opportunities to attend conferences to meet other researchers in the same area. Also, I would like to thank to my ITaP coworkers, Alex Younts and Preston Smith who gave me the opportunity to work as a research assistant. By working with them, I was able to kept myself motivated and learning the computing environment for big data analysis. Finally, I appreciate all the support and encouragement that my family and my friends have been giving to me. Their trust in me made my thesis and achievement possible. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AdaBoost	Adaptive Boosting
DT	Decision Tree
FN	False Negative
FP	False Positive
GNB	Gaussian Naive Bayes
IDS	Intrusion Detection System
MCC	Matthews Correlation Coefficient
ML	Machine Learning
NN	Neural Networks
TN	True Negative
TP	True Positive

GLOSSARY

Botnet – A botnet is a network of compromised devices used by a botnet owner to perform various malicious tasks. The devices – often personal computers – called “Bots” are under the control of a human “Botmaster.”

Ensemble method - An ensemble method is an approach that makes a set of classifiers into an ensemble by combining the prediction from each classifier possibly with weights. It is regarded as one of the methods for improving the accuracy.

Machine learning algorithm - A machine learning algorithm refers to an algorithm that is used to generate a statistical model from input data for various purposes from new incoming data. In practice, the former phase is called training, and the latter is called testing.

ABSTRACT

Author: Ryu, Songhui. M.S.
Institution: Purdue University
Degree Received: May 2018
Title: Comparison of Machine Learning Algorithms and Their Ensembles for Botnet Detection
Major Professor: John Springer

A Botnet is a network of compromised devices controlled by a botmaster often for nefarious purposes. Analyzing network traffic to detect Botnet traffic has historically been an effective approach for systems monitoring for network intrusion. Although such system have been applying various machine learning techniques, little investigation into a comparison of machine algorithms and their ensembles has been undertaken. In this study, three popular classification machine learning algorithms – Naive Bayes, Decision tree, and Neural network – as well as the ensemble methods known to strengthen said classifiers are evaluated for enhanced results related to Botnet detection. This evaluation is conducted with the CTU-13 public dataset, measuring the training time and accuracy scores of each classifier.

CHAPTER 1. INTRODUCTION

This chapter provides an overview of the research study. It introduces the research questions and covers the research significance, assumptions, limitations, and delimitations which define the extent of the study.

1.1 Research Question

For botnet detection, which machine learning algorithm and related ensemble method for classification are the most accurate?

A botnet, which is a network of compromised devices, is an ongoing threat to cybersecurity. Controlled by a hacker referred to as a botmaster, the botnet is used to execute Denial of Service attacks, send spam emails, steal personal information, etc. Because the botmaster communicates with his botnet via a Command & Control (C&C) server, the network traffic that the botnet generates can be traced.

Machine learning algorithms have been used to detect botnet traffic from the ongoing flow of network. Even though there are some previous studies about botnet detection using machine learning, the accuracy of ensemble methods for botnet detection is still in question. While ensemble methods were designed to strengthen machine learning algorithms, are they indeed effective and efficient on botnet detection as well? This can be evaluated by comparing the accuracy of each algorithm and its ensembles on botnet traffic dataset.

1.2 Scope

In terms of cybersecurity, network traffic is one of the main types of data that researchers want to investigate as most of the cybersecurity threats – including Denial of

Service, spam emails, malware, or worms – are executed remotely through the internet (Salem, Hershkop, & Stolfo, 2008).

One of the challenges in network traffic analysis is that the amount of the data to be processed is enormous. This big data, however, can be a benefit for machine learning, which requires significant amounts of data input for its training. The author explored which machine learning algorithms would provide the most accurate botnet detection results out of network traffic. The traffic for the evaluation should resemble real-world traffic and be labeled for the classification while the machine learning algorithms for classification were selected based on their popularity for anomaly detection (Salem et al., 2008). Furthermore, to measure the accuracy of the trained model, The F1 score and the Matthews correlation coefficient (MCC) score were used. The F1 score is well known to compare the difference between two different data and to find their similarity. The MCC does the same work; however, it is well known to be more accurate for skewed data.

For the study, the CTU-13 dataset – a public botnet traffic dataset generated by Garcia et al. (2014) – was used. This dataset provides a set of refined real botnet traffic with each network flow labeled.

1.3 Significance

Silva, Silva, Pinto, and Salles (2013) indicated botnet has been growing as a significant threat since the first botnet, EggDrop, was reported in 1993. For example, Chandrasekar et al. (2017) reported that the *Necurs* botnet was one of the most active distributors of malware in 2016. Observing just one day on November 24, 2016, Necurs sent five spam runs that generated more than 2.3 million spam emails including JavaScript downloaders, VBS, and .wsf attachments. Also, according to the same report (Chandrasekar et al., 2017) the *Mirai* botnet drove the largest DDoS attack ever recorded in 2016 on the French hosting company OVH peaking at 1Tbps. The Mirai botnet mostly targeted IoT devices, such as home routers, DVRs, and internet-connected cameras. As Gartner predicted that there will be more than 20 billion IoT devices in the world by 2020 (van der Meulen, 2017), it is important that botnets such as Mirai are addressed.

The speed of its growth is also rapid. In the most recent report from Spamhaus (*Spamhaus Botnet Threat Report 2017*, 2018), the number of Control and Command server (C&C), which botmasters use to communicate with bots, hosted by Amazon in 2017 increased 6 times against that of 2016. All the other botent-hosting Internet Service Providers within top 10 rank in the report also increased in the number of C&Cs by at least 3 times against the previous year.

Obviously, because there is no solution to stop hackers from attacking a network or a host, prevention methods that prohibit the attack before the attacker starts the task have been discussed. Even though there are previous studies where the researchers make use of different machine learning algorithms to detect botnet detection (Livadas, Walsh, Lapsley, & Strayer, 2006; Lu, Rammidi, & Ghorbani, 2011; Sangkatsanee, Wattanapongsakorn, & Charnsripinyo, 2011; Strayer, Lapsely, Walsh, & Livadas, 2008), still the accuracy and performance of ensemble methods for botnet detection have not been discussed yet. In this regard, this study evaluated the several popular machine learning algorithms for classification along with their ensembles. This would be a help future researcher to decide which algorithms they want to choose during their preparation of Intrusion Detection System.

1.4 Assumptions

The study required a collection of data that includes botnet traffic and a library that provides reliable machine learning algorithms. Regarding with those, the following assumptions had been made.

1. The CTU-13 dataset shows similar patterns, characteristics, and the types of traffics with the real-world network traffic.

2. The normal and background traffic in the CTU-13 dataset do not carry malicious traffic such as traffic of another botnet.

3. The Scikit-learn library provides algorithms that work in the same way or similarly with other machine learning libraries, such as Tensorflow, Caffe, etc. Therefore, the evaluation results from the same algorithm with the same dataset will be similar regardless of libraries.

1.5 Limitations

The limitations associated with the study are:

1. As the packet traces as pcap files are enormous and need to be aggregated into flows, such as NetFlow.

2. While Scikit-learn library only handles numeric data type such as integer and float, the data includes string types as well. For example, TCP, UDP, ICMP, etc. are values of the Protocol feature, and SR_A, INT, FA_R, etc. are values of the Flags feature. Therefore, a proper preprocessing of the data is essential.

1.6 Delimitation

The delimitations of this research include:

1. For the evaluation, this study used the NetFlow data that was aggregated out of pcap files by Garcia et al. (2014)

2. Therefore, this study did not cover feature extraction by using already refined data from the previous study.
3. This study took into account 3 datasets among 13 total datasets that were chosen because of their relatively higher ratio of botnet traffic.
4. This study focused on evaluating which ML algorithm and ensemble methods would be effective for botnet detection and did not consider deploying the evaluation process into an actual intrusion detection system.

1.7 Summary

This chapter provided the scope, significance, research question, assumptions, limitations, delimitations, definitions, and other background information for the research project.

CHAPTER 2. REVIEW OF LITERATURE

This chapter provides a review of the literature relevant to botnet detection and machine learning technologies for classification.

2.1 Botnet

Botnet history dates back at least 1993 when EggDrop had emerged with the new concept that the victim device is connected to an IRC channel to listen for malicious commands from a Botmaster (Silva et al., 2013). Traditionally, botnets have been featuring centralized architectures where a botmaster uses a Control and Command (C&C) server to communicate with the bots as shown in Figure 2.1. The advantage of using a C&C server is that it enables quick communication and easy monitoring. However, a centralized architecture also means that the C&C server itself can be a single point of failure (Micro, 2006; Wang, Sparks, & Zou, 2010). For the protocol, Internet Relay Chat (IRC) has been used popularly because of its flexibility. The IRC protocol supports not only group multicast but also unicast between two members, which enables a botmaster to carry out an attack to a specific group in the botnet (Grizzard, Sharma, Nunnery, Kang, & Dagon, 2007). However, despite those benefits of the IRC, IRC is vulnerable to interruption and easy to detect because it is not popular in corporate networks. For this reason, HyperText Transfer Protocol (HTTP), which is one of the most common traffic in networks, has become popular for the C&C communication (Micro, 2010). Botnet also has a decentralized architecture based on peer-to-peer (P2P) protocol as shown in Figure 2.1. Because there is no central server for a botnet, it is more difficult to destroy a P2P botnet because detecting a number of bots does not guarantee they make up the entire botnet (Grizzard et al., 2007).

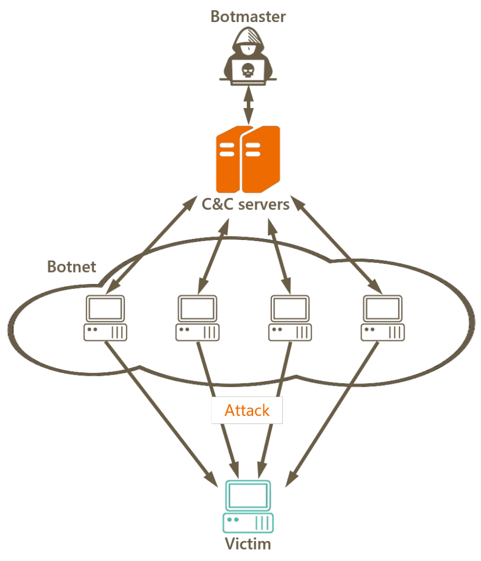


Figure 2.1. A centralized botnet architecture.

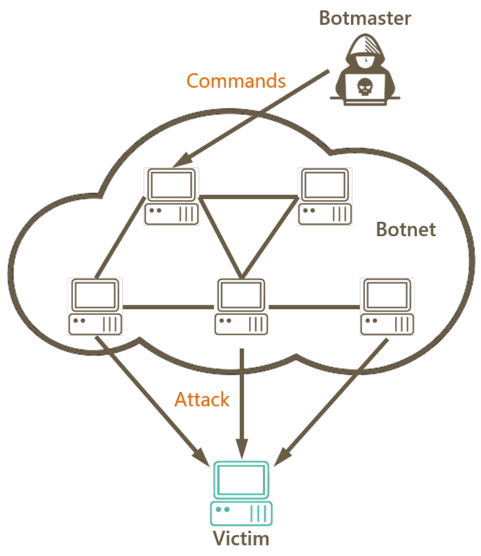


Figure 2.2. A decentralized P2P botnet architecture.

2.2 Machine Learning for Classification

"An Intrusion detection system (IDS) aids the network to resist external attacks by providing a wall of defense to confront the attacks of computer systems on Internet (Tsai, Hsu, Lin, & Lin, 2009).” Moreover, “studies make use of either a single machine learning techniques or a combination of multiple machine learning techniques, in the form of classifiers that are used to determine whether the incoming traffic is benign or malicious” (Tsai et al., 2009). In the previous research – including Livadas et al. (2006); Lu et al. (2011); Sangkatsanee et al. (2011); Strayer et al. (2008) – where they used supervised machine learning algorithms, three algorithms (naive Bayes, decision tree, and (artificial) neural networks) were most frequently adopted.

2.2.1 Naive Bayes

Naive Bayes method is a simple and intuitive classification technique based on the Bayes’ theorem that describes the probability of an event from prior knowledge of the condition that potentially related to the event (McCallum, Nigam, et al., 1998). Naive Bayes algorithm assumes that each feature contributes independently to the probability of an event. Specifically in machine learning, the naive Bayes classifier calculates all the probabilities of all classes (values) for a target feature and selects the one with the highest probability. Furthermore, Gaussian naive Bayes (GNB) assumes that the values associated with each class of each feature follow a Gaussian distribution. Even though these two assumptions in naive Bayes and Gaussian naive Bayes are unlikely to happen in real network traffic environment, it shows relatively better results than other models like logistic regression. Additionally, this algorithm is less computationally intense and generates the mining model quickly. Due to its simplicity and fastness, it is popularly used for SPAM filtering and other real-time detections. For example, Metsis, Androutsopoulos, and Paliouras (2006) evaluated accuracies of different types of naive Bayes for SPAM filtering.

2.2.2 Artificial Neural Networks (Multi-layer Perceptron)

Neural networks (NN), analogous to the human brain, refer to large connections of simple units called neurons. Consisting of three layers (the input layer, hidden layer(s) and the output layer0, neural networks take each record and passes its features into input layer, and then the model makes decisions calculating weights of hidden neurons to get the single highest value at the output layer. A feed-forward neural network where the output of one layer is used as input to the next layer does iterate for the same data to compare the output to true value so that it adjusts the weights in the hidden neurons with its error term. Recurrent neural networks, however, adopt feedback loops between neurons, which more resembles human brains (Nielsen, 2015). According to Tsai et al. (2009), a back-propagation neural network works by feeding back errors of misclassified terms to the network so that they are not repeated in the further iterations.

2.2.3 Decision Tree

As another popular classification method, a decision tree (DT) generates a tree-like model of decisions based on decision rules inferred from the data. Unlikely other machine learning algorithms, a decision tree is easy to interpret with a tree visualization. Also, it works for both categorical and numerical variables as well since it doesn’t require an assumption about the data distribution and classifier structure. Over fitting and data loss when categorizing numeric variables, however, are the most practical difficulties in a decision tree classifier (Quinlan, 1987). According to Rokach and Maimon (2014), a decision tree is built by splitting the training data into sub-data samples based on the most significantly differentiating feature. When a new incoming data arrives, the attribute of the data is checked all the way down from the root of the tree, eventually ending up to a leaf node that represents the classification of the data.

2.3 Ensemble Methods

For a system that utilize machine learning technology for either classification, regression and clustering, using multiple training models is not so uncommon (Mendes-Moreira, Soares, Jorge, & Sousa, 2012). By using ensemble methods, one can make classifiers more powerful by combining different classifier into one or iteratively training a classifier. The typical ways of combining two or more classifier are majority voting and weighted voting (Dietterich et al., 2000). Also, sub-sampling approaches called boosting and bagging have been studied in previous researches(Breiman, 1996; Freund & Schapire, 1995). In this section, those three types of ensemble methods are explained.

2.3.1 Voting

Voting is the simplest way to form an ensemble. Voting classifiers consist of multiple models of different types. In the training step, all the models are trained separately with whole training data and it averages the posterior probabilities that are calculated by each model in the recognition step. Panda and Patra (2009) explains that by combining outputs of several classifiers, the risk of selecting a poorly performing classifier can be reduced. The voting method can be weighted where weighted voting lets each classifier hold different voting power. According to previous research (Ekbal & Saha, 2011), where the researchers constructed a weighted vote-based classifier ensemble for Named Entity Recognition, the Genetic Algorithm ensemble for classifying 4 deferent language group in India outperforms all the other individual classifiers (Maximum Entropy, Conditional Random Field, and Support Vector Machine) when it comes to F1 measure. Another finding of this research is that an increase in the number of classifiers may not always increase the overall performance of their system. For example, when they evaluated the 80 best performing ME-based classifiers, the accuracy is the same with the

140 best-performing ME classifiers and with all 152 ME classifiers. This is the same case when they increased the training size from 100K to 312K. The accuracy measure increases as the training size becomes larger, but the rate of improvement decreases gradually (Ekbal & Saha, 2011).

2.3.2 Bagging

The bagging method (Breiman, 1996), also called bootstrap aggregation because it uses bootstrap sampling (Efron & Tibshirani, 1994), randomly breaks down the original training data to make several sub-training datasets and trains a classifier from each of those training subsamples. The predictions are then combined via averaging for regression or majority voting for classification. In the paper, Quinlan et al. (1996) evaluated boosting and bagging on C4.5 decision tree with a collection of datasets from the UCI Machine Learning Repository. In the experiment, Bagged C4.5 and Boosted C4.5 generally showed the markedly lower error rates than those of C4.5. Interestingly, even though boosting shows a reduced error rate by 15% over C4.5 and 10% over bagging and additionally outperforms bagging in 20 of the 27 data sets, boosting shows more erratic outcomes. For example, it led to a 36% increase in error on the iris dataset and 26% on colic. According to Freund and Schapire (1995), that could be because of overfitting explaining a large number of trials leads to the classifier to become very complex. The solution that Quinlan et al. (1996) tried was stopping the ensemble learner when any classifier shows zero error. With this approach, C4.5 appeared that it only needs three boosted trials to achieve the objective.

2.3.3 Boosting

Boosting methods, which were suggested by Kearns and Valiant (1994) to create a single strong learner from several weak learners, were strengthened by Schapire (1990).

Boosting also manipulates the training data like bagging, but it maintains a set of weights on training data. Especially with methods such as AdaBoost, weighted errors of each model update weights on the training data, giving more weight on the data with the lower accuracy and less on the data with the higher (Vezhnevets & Vezhnevets, 2005). Zhou, Wu, and Tang (2002) explained that a boosting learner combines the predictions from multiple classifiers of homogeneous type, and the results are averaged out for regression or voted for classification.

2.3.4 Random forest

Along with the machine learning algorithms and ensemble methods, the author also considered random forests algorithm to test. According to Breiman (2001), the random forests classifier is a type of ensemble methods that built with multiple decision trees that are independently bootstrap-sampled. As a bagging of decision tree, random forest is explained as "These averaging techniques improve the performance of single tree models by creating multiple trees and, randomly selecting a subset of variables at each node. This reduces variance more than in single trees" (Elith, Leathwick, & Hastie, 2008). Yeh, Chi, and Lin (2014) describe the advantages of random forests indicating that the variables can be both continuous and categorical. Also, a random forests classifier is recommended by Yeh et al. (2014) because it is robust against overfitting by averaging trees during the run which also results in "low-bias and low-variation but highly accurate classification and predictions." The same author also suggested that random forests are preferred over Support Vector Machine or Neural network because of the strengths explained above.

2.4 Related Works

In the paper by Livadas et al. (2006), machine learning algorithms for classification, i.e. J48 decision tree, naive Bayes, and Bayesian network were employed to identify C&C traffic of IRC-based botnets. By capturing real-life network traffics from Dartmouth's wireless campus network, and also generating botnet testbed traffic from the *Kaiten* bot, they demonstrated that only the naive Bayes classifiers achieved low false negative rate (FNR) identifying 35 out of the 38 botnet flows with an FNR of 7.89%. On the other hand, the J48 and the Bayesian networks classifiers performed poorly possibly because of overfitting.

García, Zunino, and Campo (2014) conducted a survey on network-based botnet detection methods. There several suggested botnet detection methods were compared. In the following sections, the tree systems are introduced along with the analysis by García et al.

2.4.1 BotSniffer

The BotSniffer (Gu, Zhang, & Lee, 2008) processed the sniffed network packets by grouping hosts that connect to the same destination considering port numbers as well and then separating the groups into time windows. The first approach is to examine if there are hosts that had triggered attacks including SPAM sending and port scanning. One group is marked as a possible botnet if more than a half of the hosts in the group shows the attack's fingerprints. The second approach focuses more on IRC protocol responses, by looking for hosts that answered similar IRF responses using the F1 score as a similarity function. IRC responses are clustered on the basis of this similarity measure. In the case where the biggest cluster is more than half the size of a group, this group is marked as a possible botnet.

García et al. (2014) stated that:

> This paper is one of the most cited papers in the botnet detection field. It presents several behavioral techniques to detect botnets that accomplish good results. However, much new data and botnets have been found since its publication. The dataset used for validation may be too scarce for generalizing the technique. Only one real IRC botnet was captured. The rest of the dataset is composed of one IRC text log and five custom-compiled LAN botnets.

García et al. (2014) also pointed out that the possible bias by whitelist filtering done to the dataset during preprocessing stage was not analyzed.

2.4.2 BotMiner

The BotMiner detection framework (Gu, Perdisci, Zhang, Lee, et al., 2008) has three phase of analysis. At the first phase, it groups hosts with similar activity patterns derived from flow information. The IP addresses, ports, and the network profile as well as statistical measures including number of flows per hour and average bytes per packets were used to make similar host groups along with X-means clustering method. The second phase groups hosts regarding with similar attacking patterns. The Snort IDS was used to identify attacks. At the third phase, the similarities from the previous 2 phases are utilized to score similarities between hosts. But García et al. (2014) pointed out the fact that the dataset is not published and no explanation provided how the dataset was verified. Also, the design – where a list of well-known hosts should be maintained – could be error-prone and time-consuming.

But for encouraging results, García et al. (2014) stated that:

> Unlike other proposals, this work uses one novel idea to differentiate between botnets and manual attacks: botnets act maliciously and always communicate in a similar way, but the manual attacks only act maliciously.

2.4.3 BotHunter

The BotHunter framework (Gu, Porras, Yegneswaran, Fong, & Lee, 2007) utilized a state-based infection sequence model. By using the modified Snort IDS added with two proprietary detection plug-ins, “it looks for evidence of botnet life cycle phases to drive a bot dialogue correlation analysis” (García et al., 2014). Each host gets the infection score derived from IDS warning and a host is labeled as bot when the score meets certain thresholds. In addition to this local host infection, when there is attack propagation or evidence of outward bot, an infection is reported. Because the Snort IDS is statical, detections by the BotHunter is also static. For this reason, some known botnet servers can be embedded into the Snort configuration, and the sequence of bytes in the binary download and the Snort fingerprints could be used.

Gu et al. (2007) pointed out that the accuracy metrics is not complete as the proposal reports either TPR or FP and TN for each experiment they conducted. Also, the BotHunter neither uses the traffic from a host nor differentiates between botnets and manual (or automatic) attacks.

García et al. (2014) also indicated that:

> However, as the model is based on the life cycle of botnets, it is very probable that it could work fine for this situation. The method has two major advances. First, it seems capable of analyzing, detecting and reporting botnets in real time. Second, it is the only proposal that was published as a product.

2.5 Summary

This chapter provided a review of the literature relevant to botnet detection and machine learning algorithms for classification. The next chapter provides the methodology to be used in the research project.

CHAPTER 3. METHODOLOGY

3.1 Research Overview

In this thesis, the author evaluated the three most popular machine learning algorithms for botnet classification – Gaussian naive Bayes (GNB), neural networks (NN) and decision tree (DT) – based on the previous research by Salem et al. (2008). the well-known ensemble methods (voting, bagging, and boosting) were also measured to help address the research question: For botnet detection, which machine learning algorithm and related ensemble method for classification are the most accurate? To compare these classification models, the training time and two different measures (F1 score and MCC score) were calculated. Furthermore, the random forest (RF) classifier, which is one of the popular algorithm based one the bagging method, was also evaluated.

With the already aggregated NetFlow data, CTU-13, the experiment was conducted on the Rice cluster, which is a part of Purdue Community Clusters. “Rice is optimized for Purdue’s communities running traditional, tightly-coupled science and engineering applications” (*ITaP Research Computing*, 2017). On the cluster, the evaluation system was constructed with all ML classifiers, their ensembles, and accuracy measurement methods.

Once the evaluation program was set, the training data was input to the system to generate classification models and ensembles. The test data, then, was input to generate the prediction outcome. The running time was measured during those training and test processes. Figure 3.1 describe the overview of the experiment.

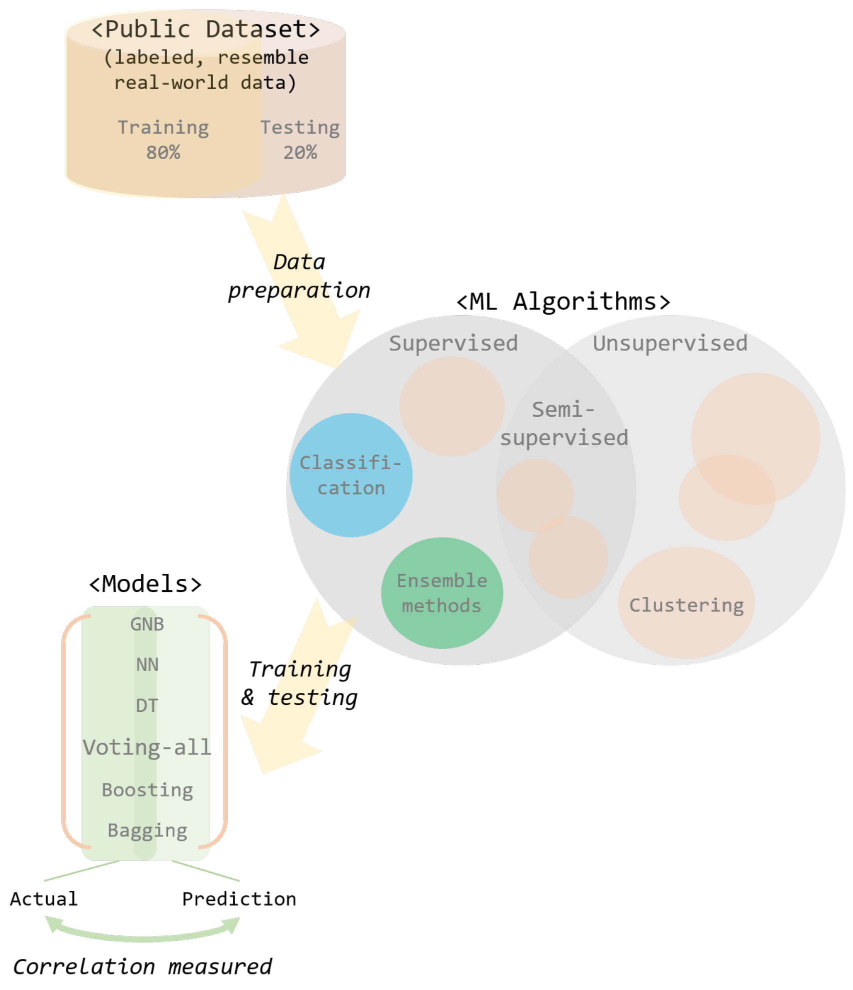


Figure 3.1. Overview of the experiment.

3.2 Dataset

Finding an appropriate network traffic dataset for machine learning is challenging since, because supervised machine learning classifiers require the data to be labeled unless the target feature is already in the dataset, labeling each network traffic can be very onerous. Even further, the data is preferred to resemble a real-world network traffic to be cleanly captured at a well-administered lab. Specifically, Sommer and Paxson (2010) discussed the importance of getting a *perfect* data which is not deficient in certain statistical characteristics. For this reason, even though Honeynet project, CAIDA and other similar types of projects have presented an enormous number of network traffic dataset, researchers have generated their own datasets like those of Livadas et al. (2006); Saad et al. (2011); Shiravi, Shiravi, Tavallaee, and Ghorbani (2012). For example, Shiravi et al. (2012) set up a distributed DoS attack using an IRC botnet to generate the dataset which is suitable for intrusion detection.

The CTU-13 dataset is also one of these cases. Garcia et al. (2014) created the CTU-13 dataset where the data is labeled as botnet, normal, or background. Garcia et al. (2014) mentioned that even though there had been several botnet datasets downloadable, such as Dainotti, King, Papale, Pescape, et al. (2012); Saad et al. (2011); Shiravi et al. (2012); Sony and Cho (2000), they are either a) not representative of the real-world traffic, b) not labeled, or c) not suitable for every detection algorithms that the authors wanted to compare. For these reasons, the CTU-13 dataset was generated with several fundamental design goals as follow.

Garcia et al. (2014) set the design goals (Garcia et al., 2014):

- "Must have real botnets attacks and not simulations.
- Must have unknown traffic from a large network.
- Must have ground-truth labels for training and evaluating the methods.
- Must include different types of botnets.
- Must have several bots infected at the same time to capture synchronization patterns.
- Must have NetFlow files to protect the privacy of the users."

The CUT-13 dataset was captured in the CTU University, the Czech Republic in 2011. On top of a Linux Debian host, they constructed a set of virtual machines with the Microsoft Windows XP SP2 operating system and bridged each virtual machine into the University network. After infecting the virtual machines with a particular botnet, they captured the traffic both on the Linux host and on one of the University router using *tcpdump*. Once they gathered all pcap files for the 13 different infection scenarios, the captures were aggregated into the NetFlow file standards in the form of CSV file. Each flow was labeled with Botnet, Normal, or Background where two former labels meant the network traffics were generated from their testbed and the last label meant the traffic came from the University networks.

For the evaluation, the author chose three datasets out of 13 datasets in the CTU-13: Scenarios 4, 10 and 11. The reason for selecting these datasets is that they feature the same bot, Rbot, but with different ratios of botnet traffic: 0.15%, 8.11%, and 7.6%, respectively.

3.3 Accuracy Metrics

To measure the accuracy of a classifier, taking into account the confusion matrix is the most common way; in particular, precision and recall are often used. Precision pertains the percentage of correctly predicted event from the pool of total predicted events while recall concerns the percentage of correctly predicted event from the pool of actual events.

$$Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN} \tag{3.1}$$

3.3.1 F1 score

Taking both precision and recall into account, the F1 score gives a more balanced view compared to using only precision or recall. The F1 score can be between 0 and 1. A

F1 score of 0 means there are no true positives, 1 means there are neither false negatives or false positives, and undefined when there are only true negatives in the prediction.

$$F1 = 2 * \frac{P * R}{P + R}, \text{ where P is Precision, R is Recall} \tag{3.2}$$

which can also be represented as:

$$F1 = \frac{2TP}{2TP + FN + FP} \tag{3.3}$$

3.3.2 Matthews Correlation Coefficient(MCC)

The Matthews correlation coefficient, also known as the phi coefficient, was introduced by Matthews (1975). As another measure of the quality of binary classification, the MCC incorporates True Negative as well, unlike the F1 score. The range of the MCC lie between -1 to +1 where +1 means a perfect prediction, 0 no better than a random prediction, and -1 perfect disagreement between true values and predictions.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.4}$$

According to Boughorbel, Jarray, and El-Anbari (2017):

> Most standard machine learning algorithms work well with balanced training data but they face challenges when the dataset classes are imbalanced. In such situation, classification methods tend to be biased towards the majority class. These algorithms are inefficient in this case mainly because they seek to maximize a measure of performance such as accuracy which is no longer a proper measure for imbalanced data.

Taking True Negatives into consideration, the MCC is regarded more robust to the data imbalance. In the same study, Boughorbel et al. (2017) showed that MCC and Area Under ROC Curve (AUC) are more robust to data imbalance than the F1 score and *accuracy*. In the recent studies by Chicco (2017); Powers (2011), MCC was claimed to be the most informative score in a context of confusion matrix and to be the best measurement for a binary classification.

3.4 Machine Learning Library

There are multiple machine learning libraries available, such as Theano, TensorFlow, Scikit-learn, Caffe, etc. In this thesis, Scikit-learn was used because it comes as a service consisting of all algorithms to be used.

The reason why Scikit-learn was chosen rather than the other popular machine learning libraries like SparkML and Torch, which can also parallelize the classifier to take advantage of using *big data*, was a lack of stable implementation of ensemble methods at the time of this thesis.

Scikit-learn is well known for its high-level functions that allow users to take care of the parameters of functions, or configuration rather than their implementation itself (Pedregosa et al., 2011). For example, to build a Linear Support Vector Machine with Scikit-learn, one can call *LinearSVC()* training with the researcher's dataset and then test iteratively changing its parameters such as class weights, the maximum number of iterations to be run, etc. For the classification, the following methods were used.

- *naive_bayes.GaussianNB()*: GaussianNB() works by assuming a Gaussian distribution of data. In this reason, the data should be normalized beforehand.
- *neural_network.MLPClassifier()*: This multi-layer perceptron classifier has parameters including *hidden_layer_sizes* defining number of hidden layers and neurons and *solver* for weight optimization.

- *tree.DecisionTreeClassifer()*: This builds a decision tree having *max_depth, min_samples_split*, etc. as parameters.

For the ensembles, *ensemble.AdaboostClassifier()*, *ensemble.BaggingClassifier()*, *ensemble.VotingClassifier()* were used. The first two methods take a single type of base estimator to make an ensemble, and *number of estimators* can be set to define the maximum number of estimators at which the ensemble method is terminated. In case of a perfect fit, the learning procedure stops early. VotingClassifier() takes a list of estimators to make a voting ensemble.

Scikit-learn also provides various accuracy metrics for classification, regression, clustering, etc. In this study, *metrics.f1_score() and metrics.matthews_corrcoef()* were used.

3.5 Data preparation and evaluation

Table 3.1 provides the list of features in the CTU-13 dataset. After capturing PCAP files, Garcia et al. carefully selected features while aggregating the traffic to NetFlow standards. While the previous work used WEKA tool for the classification which can handle categorical data, a proper data preprocessing was needed for this thesis. Because Scikit-learn only expects continuous input, all categorical features should be encoded with *sklearn.preprocessing.LabelEncoder*.

After encoding, data standardization was conducted. While Scikit-learn provides functionality to convert the data into standard normally distributed data which is Gaussian with zero mean and unit variance, scaling the data to a certain range is an alternative preprocessing.

Table 3.1. Data features in the CTU-13 dataset (Garcia et al., 2014)

Feature	**Type**
Start time	Numerical
End time	Numerical
Duration	Numerical
Protocol	Categorical
Src IP address	Categorical
Src port number	Categorical
Direction	Categorical
Dst IP address	Categorical
Dst port number	Categorical
Flags	Categorical
Type of services	Categorical
Number of packets	Numerical
Number of bytes	Numerical
Number of flows	Numerical
Label	Categorical

To do so, *sklearn.preprocessing.MinMaxScaler* offers “scaling features to lie between a given minimum and maximum,” which is considered to provide “robustness to very small standard deviations of features and preserving zero entries in sparse data” (Pedregosa et al., 2011). For more details about data preprocessing, please refer to Appendix A.

Once preprocessing was done, the data was randomly split into the training data and the test data with the ratio of 8 to 2. For each classifier, the input parameters were selected to the optimum after a few empirical tests. For each scenario, the average time and accuracy of 5 times of the each classifier were recorded as the final results.

3.6 Deliverables

From this research, the following were delivered to future researchers.

- A comparative study of ML algorithms and ensembles used to detect botnet traffic based on a literature survey.
- A report on the authors approach towards the research questions and the way to implement the evaluable to be scalable.

3.7 Summary

This chapter described methodology for the research including a description of dataset, accuracy metrics, machine learning library, data preprocessing, and deliverables.

CHAPTER 4. RESULTS

This chapter provides the result of the experiment. It includes the comparison of accuracies and training time.

4.1 Results

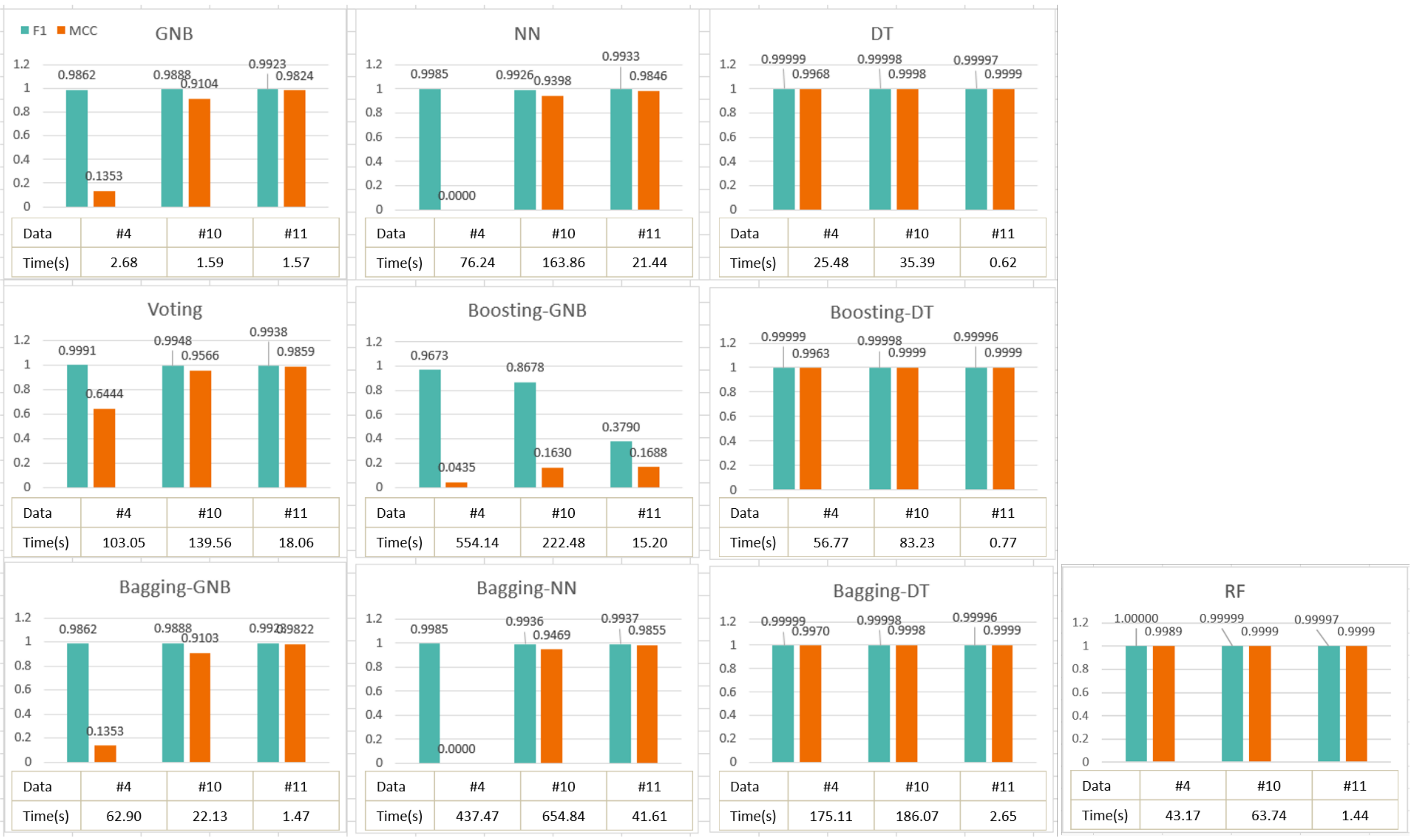
To evaluate the classification algorithms along with ensemble methods for the CTU-13 dataset, Scikit-learn on a single core of Intel Xeon-E5 with 64GB of memory was used. The results are described in Table 4.1, Table 4.2, and Figure 4.1.

For every data and algorithms, F1 scores were higher than MCC scores. This is because the F1 score does not consider the true negatives. For this reason, MCC is preferred for a binary classification. In the following discussion, accuracy refers to MCC score and S denotes scenario of the dataset.

Among individual algorithms, all showed decent accuracies over 0.91 on S10 and S11. But for S4, GNB and NN showed poor results of 0.13 and 0.00 respectively. In terms of the training time, GNB run much faster than other algorithms on S4 and S10. For S11,

Table 4.1. Time consumed for model training (sec)

Method	**Scenario 4**	**Scenario 10**	**Scenario 11**
GNB	2.68	1.59	1.57
NN	76.24	163.86	21.44
DT	25.48	35.39	0.62
Voting	103.05	139.56	18.06
Boosting-GNB	554.14	222.48	15.20
Boosting DT	56.77	83.23	0.77
Bagging-GNB	62.90	22.13	1.47
Bagging-NN	437.37	654.84	41.61
Bagging-DT	175.11	186.07	2.65
RF	43.17	63.74	1.44



GNB

Data	#4	#10	#11
Time(s)	2.68	1.59	1.57

NN

Data	#4	#10	#11
Time(s)	76.24	163.86	21.44

DT

Data	#4	#10	#11
Time(s)	25.48	35.39	0.62

Voting

Data	#4	#10	#11
Time(s)	103.05	139.56	18.06

Boosting-GNB

Data	#4	#10	#11
Time(s)	554.14	222.48	15.20

Boosting-DT

Data	#4	#10	#11
Time(s)	56.77	83.23	0.77

Bagging-GNB

Data	#4	#10	#11
Time(s)	62.90	22.13	1.47

Bagging-NN

Data	#4	#10	#11
Time(s)	437.47	654.84	41.61

Bagging-DT

Data	#4	#10	#11
Time(s)	175.11	186.07	2.65

RF

Data	#4	#10	#11
Time(s)	43.17	63.74	1.44

Figure 4.1. Experiment results

Table 4.2. Evaluation result

Method	**Scenario 4**		**Scenario 10**		**Scenario 11**	
	F1	**MCC**	**F1**	**MCC**	**F1**	**MCC**
GNB	0.986159	0.135260	0.988762	0.910358	0.992302	0.982357
NN	0.998489	0.000000	0.992646	0.939776	0.993299	0.984639
DT	0.999990	0.996779	0.999982	0.999849	0.999971	0.999935
Voting	0.999117	0.644421	0.994763	0.956586	0.993841	0.985878
Boosing-GNB	0.967339	0.043543	0.867776	0.162963	0.378982	0.168781
Boosing-DT	0.999989	0.996285	0.999983	0.999857	0.999963	0.999916
Bagging-GNB	0.986170	0.135319	0.988758	0.910333	0.992253	0.982245
Bagging-NN	0.998489	0.000000	0.993613	0.946912	0.993670	0.985486
Bagging-DT	0.999991	0.996955	0.999981	0.999836	0.999955	0.999897
RF	0.999997	0.998930	0.999988	0.999896	0.999972	0.999935

DT run faster taking 0.62 seconds. Regarding S10, even though NN showed high accuracy as DT, it took 3-4 times longer than DT did.

The only structure difference among the three datasets is the ratio of botnet traffic. Even though S4 is the largest dataset, it only brings one Rbot having 0.15% of botnet traffic ratio, which means the dataset is highly scarce. In contrast, S10 has 8.11% of botnet traffic and S11 has 7.6% which can be considered fairly large enough.

Overall, NN and DT showed relatively higher accuracy than GNB, and NN was the slowest classifier among all. This pattern appeared the same on the result of voting. For the voting, it showed relatively higher accuracies on S10 and S11, recording 0.96 and 0.99, respectively while it recorded only 0.64 on S4.

Boosting method did not significantly help either GNB or DT. Especially for GNB, the accuracy significantly dropped down on all datasets showing less than 0.17. Especially it recorded a MCC score of 0.04 on S4 where 0 means no better than random prediction. The training time was also huge compared to the GNB classifier recording 554.14 seconds on S4 for example. When it comes to boosting DT, The accuracy results were not largely different from those of DT classifier. But, of course, it took longer time than DT taking 2-3 times longer than DT did.

Each bagging algorithm seemed very similar to using a single classifier only for each dataset. The results of using s bagging classifier showed the same pattern where DT and NN were relatively more accurate than GNB while GNB run the fastest. But the training time of bagging was much larger than that of using a classifier without ensemble methods except when bagging was applied to GNB against S4.

While the ensemble methods offered by Scikit-learn were not significantly beneficial on each algorithm, random forest appeared highly effective in terms of both accuracy and training time. It showed very decent accuracies in MCC scoring more than 0.998 for every dataset, recording moderate training time compared to other ensembles. Especially noteworthy is that it took less time than using NN alone.

Figure 4.2, 4.3, and 4.4 provide graphical views to compare the classifiers against each scenario based on their time consumption and MCC score.

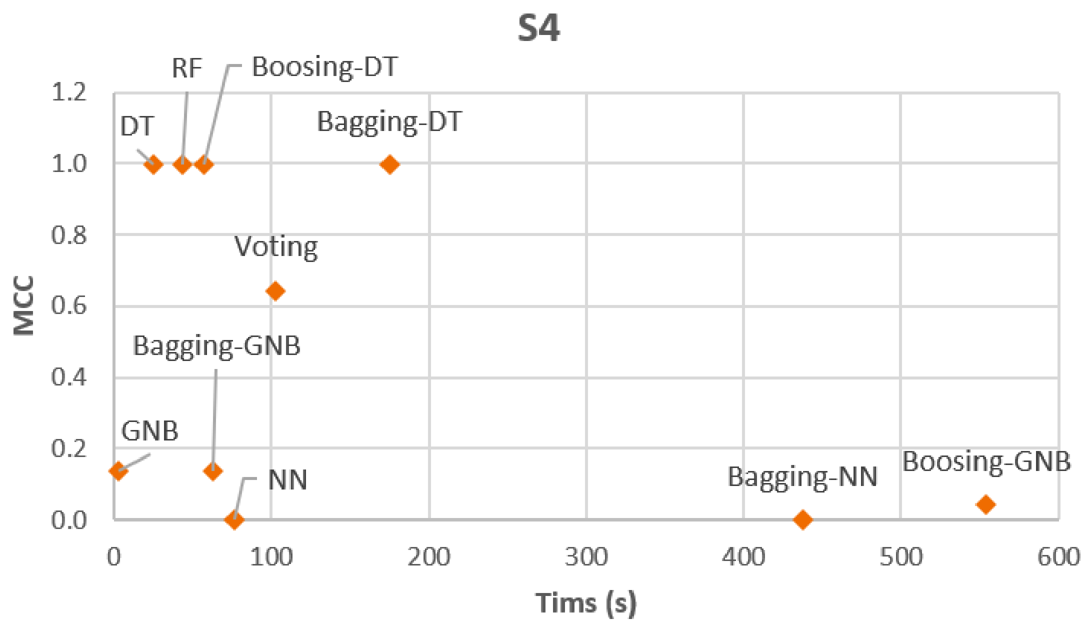


Figure 4.2. Time and MCC evaluation against S4

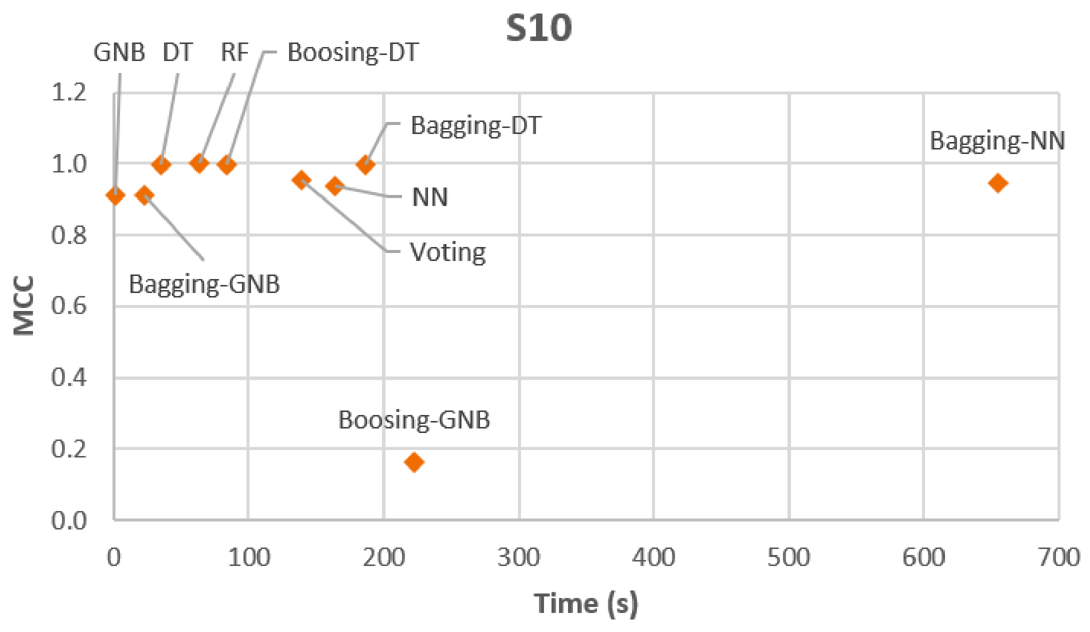


Figure 4.3. Time and MCC evaluation against S10

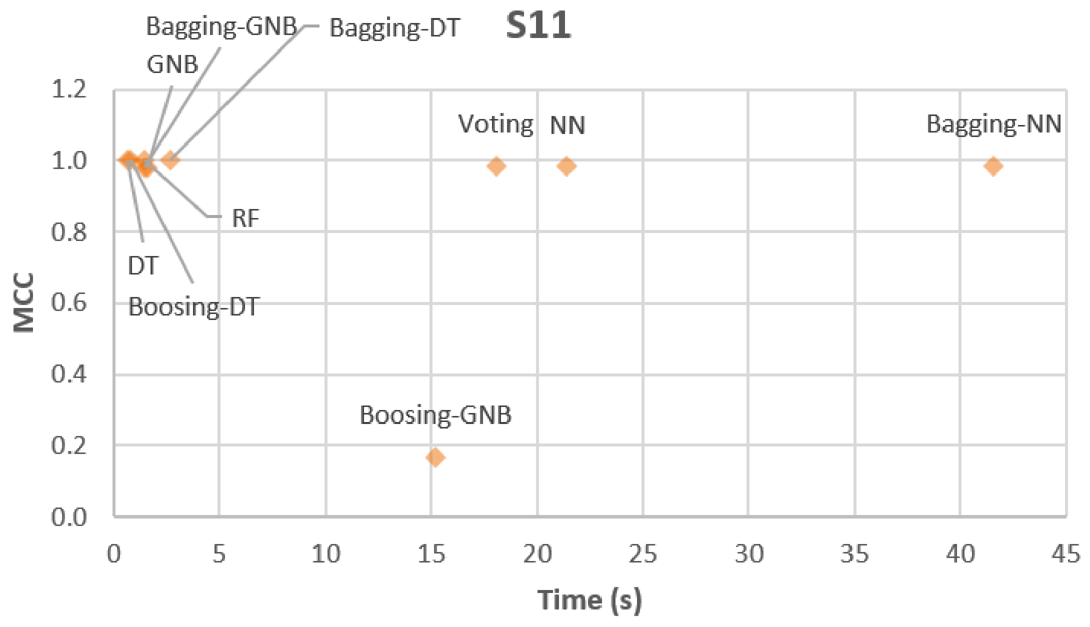


Figure 4.4. Time and MCC evaluation against S11

For the S4 data, DT, RF, Boosting DT, and Bagging DT give the predictions of high quality as shown in Figure 4.2. On the other hand, Bagging DT and Boosting DT took a longer time than decision tree alone or random forest. Compared to those classifiers, bagging NN or boosting GNB turned out to be the worst with poor predictions and enormous time consumption. Also, differently from S10 and S11, the neural networks classifiers worked poorly giving a low MCC score.

Figure 4.3 shows slightly different results from those of S4. For S10, most classifiers worked well showing high MCC scores with the exception of Boosting GNB. Considering time consumption together, GNB, Bagging GNB, DT, RF and Boosing DT seem reasonable. Similarly with the case of S4, the decision tree classifier and the random forest classifier gave better MCC that the Gaussian naive Bayes classifier. Unlike S4, Bagging NN provide accurate prediction, but it took too long which makes considering bagging neural networks undesirable.

Because the size of S11 is very small compared to the previous 2 datasets, all classifiers ended in 45 seconds for Scenario 11 (Figure 4.4). Similarly with the case of S10, Boosting GNB gave the poor prediction and the time bagging neural networks took was longer than any other classifiers. This time, the Gaussian naive Bayes and decision tree, and random forest classifiers provided fair accuracies in a very short time.

4.2 Summary

This chapter described the result of the experiment in terms of training time and accuracy.

CHAPTER 5. CONCLUSION

In this chapter, more detailed discussion on the evaluation results and future work are described.

5.1 Discussion

After the experiment, to detect botnet traffic out of all network traffic, decision tree without any ensemble method or random forest would be the most reliable approaches. They run much faster than NN alone and with better accuracy. Even though GNB ran the fastest, the accuracy varied on the dataset. Unlike the common expectation, adopting ensemble methods on machine learning algorithms for botnet detection in a hope of enhancing the accuracy is not preferable because it does not give remarkably more accurate result while consuming much more time.

Voting showed the same pattern with using other classifiers without ensemble methods. This is on the ground that voting works by averaging out each outcome from the models.

When it comes to boosting, it showed minimally better accuracy compared to using a sole algorithm primarily because of the nature of boosting. In boosting, it turns weak models with slightly better prediction than random into a strong one. In this regard, it obviously did not make DT strong as it already had a good accuracy. On the other hand, boosting GNB showed the results that deviated with those of other algorithms. For S4, the MCC score was near zero which means the prediction is not better than random. As for S10 and S11, it only showed around 0.16 of MCC which is the opposite result of using a sole GNB. According to Ting and Zheng (2003), the similar drop-down appeared in a specific dataset, Tic-tac-toe. Because Naive Bayes is very stable classifier carrying a strong bias, in the boosting process the sub-classifiers may not be diverse enough (Ting & Zheng, 2003). This does warrant additional inquiry.

While training a bagging model, multiple sub-datasets sampled out from the original dataset make their own classifier. Then predictions from those classifiers are voted. This dataset, however, might not take the benefit from sampling because the data is too sparse or skewed.

Random forest, as a combination of decision tree, performs implicit feature selection taking feature importance into consideration. Also making multiple sub-decision trees with part of features and data rows, it can run extremely faster than other methods and even can be easily parallelized. Considering parallelization is tough to be implemented in ensemble methods, Random forest seems like an excellent choice.

5.2 Comparison with the benchmarks

The results of the evaluation were also compared to those of the previous research by Garcia et al. (2014). In the previous research, the authors separated the entire CTU-13 dataset into two groups considering the following criteria (Garcia et al., 2014):

- The "training and cross-validation datasets should be approximately 80% of the dataset.
- The testing dataset should be approximately 20% of the dataset.
- None of the botnet families used in the training and cross-validation dataset should be used in the testing dataset."

Meeting those criteria, they separated the dataset into training, testing, and cross-validation carefully considering features, such as the duration in minutes, the number of clusters, the number of NetFlow and the number of aggregated NetFlows of the scenarios. Scenarios 1, 2, 6, 8, and 9 were selected for training and cross-validation and the others were for testing. Among 20 different botnet detection methods analyzed in the research, the BClus (Garcıa, 2014) and the BotHunter (Gu et al., 2007) methods were considered for further comparison with this thesis as they have utilized machine learning approaches as well as rule-based approaches.

Even though the authors of the research did not adopt MCC score, they measured F1 scores. The score of Scenario 8 could not be computed because the BotHunter algorithm could not detect a single TP against the dataset (Garcia et al., 2014). Compared to the result of this thesis, the F1 scores are far below for all of the Scenarios except Scenario 11. According to Garcia et al. (2014), the data separation was meant to ensure that the methods can generalize and detect new behaviors and to avoid the bias toward the majority class of Background. Thus, the evaluation utilizing three machine learning algorithms and their ensembles offer better detection accuracy compared to the previous research.

5.3 Future work

The error metrics from the thesis looks more prominent than those of the benchmarks from Garcia et al. (2014), but still the gap between those two research can be narrowed. Firstly, the different use of the dataset can be resolved. while the previous research used the entire CTU-13 dataset to split the training data and the test data, the author utilized 3 out of 13 datasets. Secondly, the metric can also be expanded in scope. In the Garcia et al. (2014), they suggested the new error metric to resolve the semantic gap between the traditional error metrics and the practical application.

Specifically Garcia et al. (2014) stated:

The error metrics usually used by researchers to analyze their results (e.g. FPR, FMeasure) were historically designed from a statistical point of view, and they are really good to measure differences and to compare most methods. But the needs of a network administrator that is going to use a detection method are slightly different. These error metrics should have a meaning that can be translated to the network.

To resolve the problem, they established the following principles for a proper error metric (Garcia et al., 2014):

- “Errors should account for IP addresses instead of NetFlows.
- To detect a botnet IP address (TP) early is better than later.
- To miss a botnet IP address (FN) early is worst than later.
- The value of detecting a normal IP address (TN) is not affected by time.
- The value of missing a normal IP address (FP) is not affected by time.”

Considering the principles together would give a more reliable comparison of the thesis to the benchmark study.

Also, even though the CTU-13 dataset were generated for the use of machine learning technologies for botnet detection, it is still the best to use the actual datasets to which the detection system would be applied. In this reason, capturing network packets of the targeted network and evaluating the algorithms against the network data would provide the more practical results.

5.4 Summary

In this study, three popular machine learning algorithms – Gaussian naive Bayes, neural networks, decision tree – were tested against part of the CTU-13 dataset featuring one or more Rbots. Furthermore, the ensemble methods – voting, boosting, and bagging – were also compared to measure how significantly beneficial the ensemble methods would be for botnet detection. Along with these, Random forest. Based on this study, decision tree without any ensemble methods or random forest would be the most reliable approaches to detect botnet traffic out of all network traffic.

REFERENCES

Boughorbel, S., Jarray, F., & El-Anbari, M. (2017). Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, *12*(6), e0177678.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Chandrasekar, K., Cleary, G., Cox, O., Lau, H., Nahorney, B., et al. (2017, April). Internet security threat report. *Technical Report*, *22*.

Chicco, D. (2017). Ten quick tips for machine learning in computational biology. *BioData mining*, *10*(1), 35.

Dainotti, A., King, A., Papale, F., Pescape, A., et al. (2012). Analysis of a/0 stealth scan from a botnet. In *Proceedings of the 2012 acm conference on internet measurement conference* (pp. 1–14).

Dietterich, T. G., et al. (2000). Ensemble methods in machine learning. *Multiple classifier systems*, *1857*, 1–15.

Efron, B., & Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Ekbal, A., & Saha, S. (2011). Weighted vote-based classifier ensemble for named entity recognition: A genetic algorithm-based approach. *ACM Transactions on Asian Language Information Processing (TALIP)*, *10*(2), 9.

Elith, J., Leathwick, J. R., & Hastie, T. (2008). A working guide to boosted regression trees. *Journal of Animal Ecology*, *77*(4), 802–813.

Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (pp. 23–37).

Garcıa, S. (2014). *Identifying, modeling and detecting botnet behaviors in the network.* Unpublished doctoral dissertation, Universidad Nacional del Centro de la Provincia de Buenos Aires.

Garcia, S., Grill, M., Stiborek, J., & Zunino, A. (2014). An empirical comparison of botnet detection methods. *computers & security*, *45*, 100–123.

García, S., Zunino, A., & Campo, M. (2014). Survey on network-based botnet detection methods. *Security and Communication Networks*, *7*(5), 878–903.

Grizzard, J. B., Sharma, V., Nunnery, C., Kang, B. B., & Dagon, D. (2007). Peer-to-peer botnets: Overview and case study. *HotBots*, *7*, 1–1.

Gu, G., Perdisci, R., Zhang, J., Lee, W., et al. (2008). Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Usenix security symposium* (Vol. 5, pp. 139–154).

Gu, G., Porras, P. A., Yegneswaran, V., Fong, M. W., & Lee, W. (2007). Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Usenix security symposium* (Vol. 7, pp. 1–16).

Gu, G., Zhang, J., & Lee, W. (2008). Botsniffer: Detecting botnet command and control channels in network traffic. In *Ndss* (Vol. 8, pp. 1–18).

Itap research computing. (2017). Retrieved 2017-10-19, from `https://www.rcac.purdue.edu/compute/rice`

Kearns, M., & Valiant, L. (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, *41*(1), 67–95.

Livadas, C., Walsh, R., Lapsley, D., & Strayer, W. T. (2006). Using machine learning techniques to identify botnet traffic. In *Local computer networks, proceedings 2006 31st ieee conference on* (pp. 967–974).

Lu, W., Rammidi, G., & Ghorbani, A. A. (2011). Clustering botnet communication traffic based on n-gram feature selection. *Computer Communications*, *34*(3), 502–514.

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, *405*(2), 442–451.

McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *Aaai-98 workshop on learning for text categorization* (Vol. 752, pp. 41–48).

Mendes-Moreira, J., Soares, C., Jorge, A. M., & Sousa, J. F. D. (2012). Ensemble approaches for regression: A survey. *ACM Computing Surveys (CSUR)*, *45*(1), 10.

Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes? In *Ceas* (Vol. 17, pp. 28–69).

Micro, T. (2006). Taxonomy of botnet threats. *A Trend Micro White Paper*.

Micro, T. (2010). The botnet chronicles: A journey to infamy. *A Trend Micro White Paper*.

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press. Retrieved 2017-06-07, from `http://neuralnetworksanddeeplearning.com`

Panda, M., & Patra, M. R. (2009). Ensemble voting system for anomaly based network intrusion detection. *International journal of recent trends in engineering*, *2*(5).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Powers, D. M. (2011). Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation.

Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies*, *27*(3), 221–234.

Quinlan, J. R., et al. (1996). Bagging, boosting, and c4. 5. In *Aaai/iaai, vol. 1* (pp. 725–730).

Rokach, L., & Maimon, O. (2014). *Data mining with decision trees: theory and applications*. World scientific.

Saad, S., Traore, I., Ghorbani, A., Sayed, B., Zhao, D., Lu, W., … Hakimian, P. (2011). Detecting p2p botnets through network behavior analysis and machine learning. In *Privacy, security and trust (pst), 2011 ninth annual international conference on* (pp. 174–180).

Salem, M. B., Hershkop, S., & Stolfo, S. J. (2008). A survey of insider attack detection research. *Insider Attack and Cyber Security*, 69–90.

Sangkatsanee, P., Wattanapongsakorn, N., & Charnsripinyo, C. (2011). Practical real-time intrusion detection using machine learning approaches. *Computer Communications*, *34*(18), 2227–2235.

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, *5*(2), 197–227.

Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, *31*(3), 357–374.

Silva, S. S., Silva, R. M., Pinto, R. C., & Salles, R. M. (2013). Botnets: A survey. *Computer Networks*, *57*(2), 378–403.

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

Sony, C., & Cho, K. (2000). Traffic data repository at the wide project. In *Proceedings of usenix 2000 annual technical conference: Freenix track* (pp. 263–270).

Spamhaus botnet threat report 2017. (2018). Retrieved 2018-04-01, from `https://www.spamhaus.org/news/article/772/spamhaus-botnet-threat-report-2017`

Strayer, W. T., Lapsely, D., Walsh, R., & Livadas, C. (2008). Botnet detection based on network behavior. In *Botnet detection* (pp. 1–24). Springer.

Ting, K. M., & Zheng, Z. (2003). A study of adaboost with naive bayesian classifiers: Weakness and improvement. *Computational Intelligence*, *19*(2), 186–200.

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications*, *36*(10), 11994–12000.

van der Meulen, R. (2017, Feburary). *Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016.* Retrieved 2017-11-08, from `https://www.gartner.com/newsroom/id/3598917`

Vezhnevets, A., & Vezhnevets, V. (2005). Modest adaboost-teaching adaboost to generalize better. In *Graphicon* (Vol. 12, pp. 987–997).

Wang, P., Sparks, S., & Zou, C. C. (2010). An advanced hybrid peer-to-peer botnet. *IEEE Transactions on Dependable and Secure Computing*, *7*(2), 113–127.

Yeh, C.-C., Chi, D.-J., & Lin, Y.-R. (2014). Going-concern prediction using hybrid random forests and rough set approach. *Information Sciences*, *254*, 98–110.

Zhou, Z.-H., Wu, J., & Tang, W. (2002). Ensembling neural networks: many could be better than all. *Artificial intelligence*, *137*(1-2), 239–263.

APPENDIX A. DATA PREPARATION

In this appendix, the methods taken to pre-process the CTU-13 data is introduced.

Because the CTU-13 dataset contains categorical values for the most of the feature, it needed to be transformed into numeric data that Scikit-learn can process. In this regard, `sklearn.preprocessing` offers `LabelEncoder` that encodes labels with value between 0 and `n_classes-1`. In addition, `sklearn.preprocessing.MinMaxScaler` transforms features by scaling each feature to a given range. To split the dataset into training and test set, both masking and using `model_selection.train_test_split` could be applied. After practice, masking approach was mainly used.

```
def data_prep(df):
    from sklearn import preprocessing, decomposition, model_selection

    categorical = ['Prot', 'Src_IP', 'Src_Port', \
                   'Dst_IP', 'Dst_Port', 'Flags', 'Tos']

    #Categorical
    le = preprocessing.LabelEncoder()
    for col in categorical:
        df[col] = le.fit_transform(df[col])

    #Normalize
    scaler = preprocessing.MinMaxScaler()
    toNormalize = ['Durat', 'Packets', 'Bytes', 'Flows']
    for col in toNormalize:
        df[col] = scaler.fit_transform( df[col] )

    df['Target'] = 1
```

```
df.loc [ df['Label'] == 'Botnet', 'Target' ] = 2


msk = np.random.rand( len(df) ) < 0.8
trainDF = df[msk]
testDF = df[~msk]
#X = df[ cols[:-1] ]
#y = df[ 'Target' ]
#return model_selection.train_test_split(X, y, train_size=0.6, random_state=42)


return trainDF[cols[:-1]], trainDF['Target'], testDF[cols[:-1]], testDF['Target']
```