DETECTION OF IOT BOTNETS USING DECISION TREES

by

MEGHANA RAGHAVENDRA

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science in Computer Science

Department of Computer Science

Fort Wayne, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Zesheng Chen, Chair
Department of Computer Science

Dr. Jin Soung Yoo
Department of Computer Science

Dr. Bin Chen
Department of Electrical and Computer Engineering

Approved by:
Dr. Jin Soung Yoo

To my parents,
Raghavendra N and Kalpana N
and to my husband,
Raghunandan Aswathanarayana

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to various people without whose support and encouragement, this work would not have been possible. First, I would like to thank my thesis advisor Dr. Zesheng Chen for his guidance and encouragement at every stage of this research. His immense knowledge and experience have helped me in shaping my research and understanding this field better.

I would like to thank Dr. Jin Soung Yoo and Dr. Bin Chen for being on my thesis defense committee and providing valuable feedback that helped in improving the quality of this work. I would like to acknowledge the funding opportunity from 2020 Purdue Fort Wayne Collaborative Research Grant.

Finally, my appreciation goes out to my family and friends for their encouragement and support during my studies.

TABLE OFCONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

ABSTRACT

International Data Corporation[3] (IDC) data estimates that 152,200 Internet of things (IoT) devices will be connected to the Internet every minute by the year 2025. This rapid expansion in the utilization of IoT devices in everyday life leads to an increase in the attack surface for cybercriminals. IoT devices are frequently compromised and used for the creation of botnets. However, it is difficult to apply the traditional methods to counteract IoT botnets and thus calls for finding effective and efficient methods to mitigate such threats. In this work, the network snapshots of IoT traffic infected with two botnets, i.e., Mirai and Bashlite, are studied. Specifically, the collected datasets include network traffic from 9 different IoT devices such as baby monitor, doorbells, thermostat, web cameras, and security cameras. Each dataset consists of 115 stream aggregation feature statistics like weight, mean, covariance, correlation coefficient, standard deviation, radius, and magnitude with a timeframe decay factor, along with a class label defining the traffic as benign or anomalous.

The goal of the research is to identify a proper machine learning method that can detect IoT botnet traffic accurately and in real-time on IoT edge devices with low computation power, in order to form the first line of defense in an IoT network. The initial step is to identify the most important features that distinguish between benign and anomalous traffic for IoT devices. Specifically, the Input Perturbation Ranking algorithm[12] with XGBoost[26]is applied to find the 9 most important features among the 115 features. These 9 features can be collected in real time and be applied as inputs to any detection method. Next, a supervised predictive machine learning method, i.e., Decision Trees, is proposed for faster and accurate detection of botnet traffic. The advantage of using decision trees over other machine learning methodologies, is that it achieves accurate results with low computation time and power. Unlike deep learning methodologies, decision trees can provide visual representation of the decision making and detection process. This can be easily translated into explicit security policies in the IoT environment. In the experiments conducted, it can be clearly seen that decision trees can detect anomalous traffic with an accuracy of 99.997% and takes 59 seconds for training and 0.068 seconds for prediction, which is much faster than the state-of-art deep-learning based detector, i.e., Kitsune[4]. Moreover, our results show that decision trees have an extremely low false positive rate of 0.019%. Using the 9 most important features, decision trees can further reduce the processing time while maintaining the accuracy. Hence, decision trees with important features are able to accurately and efficiently detect IoT botnets in real time and on a low performance edge device such as Raspberry Pi[9].

1. INTRODUCTION

Over the last decade there has been a significant increase in the presence of Internet of Things(IoT) devices in our daily lives. The Internet of things (IoT) describes the network of physical objects—"things" or objects—that are embedded with sensors, software and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet[1]. These IoT devices, ranging from security cameras to baby monitors, are constantly streaming private data throughout the Internet but without strong security protections, leading to having a large number of devices that can be easily compromised and used as bots in IoT botnets[2]. This calls for a faster and deeper scrutiny in detecting threats and securing such devices. IoT botnets are one of the most common attacks against the IoT devices. Attackers compromise less secure IoT devices and convert them into a network of bots, which are then used to conduct distributed denial of service (DDoS) attacks on targeted devices or websites[2]. Therefore, it is important and imperative to detect the appearance of IoT botnets in real time and based on low computation power IoT edge devices.

Recently there has been a rise in applying machine learning algorithms to detect IoT botnet traffic in the network in real-time. Different machine learning algorithms have been explored to improve the detection performance[4][5][6][7]. One the most popular methods used is an unsupervised deep learning method called autoencoders. An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner, where the output is the same as the input during training to obtain the characteristics of the input data in the network[8]. Specifically, Kitsune[4], the state-of-art detector against IoT botnets, uses the benign network as both input and output to the autoencoders during training. When evaluating a packet, this packet is fed into the input of Kitsune, and if the output is different from the input, the packet will be labelled as anomalous. Autoencoders have been shown to be able to accurately detect anomalous traffic in IoT[4,6]. The advantage of using autoencoders is that there is no need for a previously labelled dataset during the training period. While this is a huge advantage to detect network traffic attacks, the time and memory requirements to run autoencoders such as Kitsune are still significant and cannot fit for low computation power IoT edge devices, such as Raspberry Pi[9,10]. As a result, it is still a research question: what is the proper machine learning

method that can detect IoT botnet traffic accurately and at the same time can be run in low computation power IoT edge devices in real time, in order to form the first line of defense in an IoT network.

To answer this question, we begin with studying two botnets – MIRAI and BASHLITE. Specifically, the MIRAI botnet is a collection of IoT bots that attacked several high-profile targets with massive DDoS attacks[28]. The Mirai botnet came into mainstream during 2016 mainly due to the increased usage of IoT devices, which had insecure default passwords configured and lacked embedded security measures. The botnet would first target vulnerable IoT devices, mainly IP cameras, and convert them into bots. Then this network of hundred thousand bots would be used to send concentrated DDoS attacks on specified targets like game servers, e-commerce sites, and telecoms. BASHLITE or GAFGYT is a well-known malware that infects Linux based IoT devices to create bots to launch DDoS attacks[29]. Specifically, it converts linux based IoT devices into bots by brute forcing their telnet access using default credentials. Once the bots are created, it is able to launch DDoS attacks with a bandwidth up to 400Gbps. Bashlite was first discovered in 2015 and is considered a predecessor to Mirai botnets. We obtained the datasets of both MIRAI and BASHLITE from UC Irvine machine learning repository[27]. The datasets contain both benign and anomalous network traffic for 9 different IoT devices and consist of 115 features extracted from network traffic.

Next, we attempt to identify the most important features among 115 features, by using the important feature ranking (IFR) algorithm[12]. Specifically, we applied the IFR method with the XGBoost machine learning algorithm, which is a power tool in machine learning[26], to each of 9 datasets (i.e., 9 IoT devices). The IFR method is able to rank input features based on their effects on the predications of XGBoost in a dataset. We observed that some features ranking high across all 9 datasets. By applying some simple calculations, we are able to find 9 most important features that can essentially be used to differentiate between benign and anomalous traffic. As shown in this thesis, when these 9 most important features are applied to a machine learning, the processing time, for both training and prediction, is much less than that with the original 115 features. Moreover, the detection accuracy is still very high and is over 99.9%.

In this paper, we propose to use decision trees machine learning to detect IoT botnet attacks in real-time and to use in low computation power IoT edge devices as the first line of defense. Specifically, we use balanced trained decision tree classifier with varying depths for an accurate detection of IoT botnets. A decision tree is a predictive supervised machine learning algorithm that utilizes the input features of a packet to predict if the packet is benign or anomalous[11]. The advantages of using decision trees over other machine learning methods include:

- **Fast detection time and low computation power -** Decision trees require very low computation power and are very fast with significantly lower training and prediction times (in milliseconds or seconds). Moreover, comparing with other machine learning methods, decision trees demand less memory to store data or model information. Hence, such a machine learning method can be easily run in IoT edge devices such as Raspberry Pi.
- **Security policy making -** Decision trees can be used to visually and explicitly represent decisions and decision-making. In a decision tree, decision nodes represent the clear conditions to distinguish between benign and anomalous traffic, which can be easily translated into explicit security policies.
- **Tradeoff between accuracy and computation power by varying tree depths–** Decision trees provide the user a flexibility to choose between accuracy and computational power. The user can choose to have a low tree depth with slightly lesser accuracy and higher false positive rate, but with significantly better prediction time and lower computational power. The higher depth the tree is allowed to grow, the better the accuracy. Therefore, the user can choose between these two tradeoffs based on his computing needs or performance requirements. This flexibility is not seen with other machine learning methods, such as autoencoders, where the user is unable to control how big the autoencoder can grow or how much computation power will be used.

From our experiments, we have found that the decision tree is able to match (and sometimes be greater than) the accuracy of unsupervised learning methods, i.e., Kitsune, while taking much less time for both training and prediction. Specifically, using 115 features for both

Kitsune and decision trees with the maximum depth, averagely Kitsune is with 99.459% accuracy and takes 602 seconds for training and 3454 seconds for prediction, whereas decision tree is with 99.997% accuracy and takes 59 seconds for training and 0.068 seconds for prediction. Our results show that decision trees have a lower false positive rate as compared to Kitsune. Moreover, we demonstrate the tradeoff between accuracy and computation power by varying the depth of the decision trees and point out that the decision trees with 9 most important features can further reduce processing time while keeping the similar decision accuracy. Furthermore, we show that decision trees can be easily extended to classify among benign traffic, Mirai botnet traffic and Bashlite botnet traffic.

The rest of the paper is divided into the following sections: Section 2 discusses the literature survey conducted in the domain of anomaly detection, Section 3 provides a detailed look into our datasets, whereas Section 4 provides a description of how the important feature ranking works, Section 5 details the working of our methodology using the decision tree algorithm and Section 6 looks into the experiment results of our algorithms and compares them to an established baseline (i.e., Kitsune). Finally, Section 7 presents our conclusion and discusses the future work.

2. LITERATURE SURVEY

Developing new network intrusion detection techniques has always been a dynamic and predominant research field. Due to the advancement of technology, there is a constant rise in the number and types of network attacks. Human detection methodologies have long since been made obsolete due to the sheer volume of network attacks seen every day. Hence developing new machine-based NIDS (network intrusion detection systems) has become a very relevant research area. Despite the exceptional progress and a large body of work, there are still several opportunities to improve the state-of-the-art in detecting and thwarting network-based attacks[13][14]. Usage of machine learning for implementing NIDSs has been comprehensively researched in the past[15] and they are usually categorized based on the underlying computational methodologies and the detection modes.

There are two types of network intrusion detection techniques, signature based and anomaly-based techniques. Signature based techniques, while being efficient due to frequent updating, are unable to detect new attacks. The focus of this paper is therefore going to be on anomaly-based detection techniques.

One of the highly researched computation methodologies for network intrusion detection is the anomaly detection technique. This technique has been thoroughly researched by various researchers and each have contributed to improve the accuracy of these detections [4][5][6][7].

Anomaly detection generally conforms to the following three steps[15]:

- *Parameterization Stage*: Instances of the target system are represented in a well-defined pre-determined form.
- *Training Stage*: The normal behavior of the target system is illustrated in the form of a model. See Table 2.1 for the different types of models that can be built.
- *Detection Stage*: The parameters from stage 1 are compared with the generated model and any deviations from the normal behavior (usually above a certain threshold level) is tagged as anomalous behavior.

Table 2.1 further summarizes the previous research conducted on the anomaly detection of network intrusions.

Table 2.1*: Past Research on Network Intrusion Detection using Machine Learning*

Paper Name	**Detection Techniques**	**ML/DL technique used**	**Detection Mode**	**Type of Traffic detected**	**Dataset Used**
Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection[4]	Anomaly detection	Autoencoder, ANN based unsupervised learning	Online	Network attacks	Custom dataset
ZeroWall: Detecting Zero Day Web Attacks through Encoder-Decoder Recurrent Neural Networks[5]	Anomaly detection	Encoder-decoder RNN, unsupervised learning	Online detection, Offline periodic retraining	Web attacks, Zero-day web attacks	Custom dataset
N-BaIoT—Network Based Detection of IoT Botnet Attacks Using Deep Autoencoders[6]	Anomaly detection	Autoencoder	Offline	IoT traffic/botnet attacks	Custom dataset
Evaluating and Improving Adversarial Robustness of Machine Learning Based Network Intrusion Detectors[16]	Evasive attack methodology	GAN	Online	Network attacks	Kitsune dataset, CICIDS2017
Unsupervised Anomaly Detection via Variational Autoencoder for Seasonal KPIs in Web Applications[17]	Anomaly detection	Variational Autoencoder	Offline	Key performance indicators of Web applications	Custom dataset
Robust and Unsupervised KPI Anomaly Detection Based on Conditional Variational Autoencoder[18]	Anomaly detection	Conditional Variational Autoencoder	Offline	Key performance indicators of Web applications	Custom dataset

A Deep Learning Approach for Network Intrusion Detection System[7]	Anomaly detection	Self-taught learning, Sparse autoencoder	Offline	Network attacks	NSL-KDD dataset
Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection[19]	Anomaly detection	Autoencoders	Offline	Network attacks	KDDCUP dataset
Machine Learning DDoS Detection for Consumer Internet of Things Devices[20]	Anomaly detection	Random forest, KNN, SVM, decision tree, ANN	Offline	IoT botnet attacks	Custom dataset
A Deep Learning Approach to Network Intrusion Detection[21]	Anomaly detection	Stacked non-symmetric deep autoencoder and Random forest	Offline	Network attacks	KDDCUP, NSL-KDD
Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic[22]	Anomaly detection	LSTM, RNN	Offline	Network attacks	CICIDS2017
Towards an Effective Zero Day Attack Detection Using Outlier Based Deep Learning Techniques[23]	Anomaly detection	Autoencoder, ANN	Offline	Network attacks, zero-day attacks	CICIDS2017

The key aspects in choosing the detection methodologies are based on the accuracy of the methodology and the cost involved in running the entire operation. Due to abundance of resource and bandwidth availability, we do not delve further into the cost aspect in this paper.
However, accuracy of the detection methodology is an important factor while selecting and implementing network intrusion detection techniques.

Machine learning gives a system the ability to evolve its model as new information is processed. Hence usage of machine learning for anomaly detection[4][5][6][7] is a popular technique.

In the N-BaIoT and Kitsune papers[4][6], we see that machine learning techniques such as autoencoders are used to detect anomalous behavior in the IoT (Internet of Things) network.
In [4] the authors have proposed the use of ensemble autoencoders to differentiate between normal and anomalous behavior without supervision in an efficient manner. The main difference between Kitsune and its predecessors that also use autoencoders is that this model is able to do the detection while online with performance that is comparable to offline detections. The authors emphasize the lightweight and scalability of the Kitsune models over other similar models[24].

Network intrusion detection systems can also be divided based on the mode in which they operate: online or offline modes. Most of the detection methodologies explored work in offline mode[6][7][19][20][21] and perform with greater efficiency as compared to online detection systems. The online detection system like Kitsune[4] and ZeroWall[5] show that it is possible to have efficient online detection systems. The main advantage of an offline system is that they are able to run over the training data in batches many times before coming up with an efficient model and do not have the constraint of runtime memory. But the drawback of offline system is that they require frequent updating of their training data for the model to remain relevant when new attacks are being discovered constantly. However, the online detection system has a constant supply of new data that is naturally able to overcome the above problem, but they run into the challenges of real-time processing with neural networks which can lead to a reduction in the runtime efficiency. While offline systems are evaluated based on their detection accuracy, online systems have to show efficiency in both detection performance and runtime performance which makes it an interesting and complex challenge.

Looking at Table 2.1, we see that most of the papers tend to use custom datasets while testing the effectiveness of their detection methodologies. Usage of custom datasets instead of standardized datasets leads to the absence of an established baseline. Benchmark datasets like KDDCUP[39], NSL-KDD[40], CICIDS2017[41] have been collected while considering several factors like findability, accessibility, interoperability and reusability[42]. These datasets have been collected and compiled specifically for the purpose of anomaly detection and have been repeatedly used by various research leading to a comprehensive understanding of its features and characteristics. Using such datasets to test the efficiency of new anomaly detection techniques

helps the reader to get a better understanding of the performance. Usage of custom dataset require the reader to first understand the behavior and characteristics of the dataset and then understand how efficient the algorithm is on that dataset. This, usually, is a tedious process due to the large amounts of network traffic collected and can also skew the performance results of the detection technique. In this paper we are focusing on the Kitsune and N-BaIoT papers, both of which use a custom dataset for training and testing purposes.

The papers chosen for the literature survey use two main types of datasets for anomaly detection – IoT traffic and normal Network traffic. Different preprocessing and machine learning techniques are required to detect attacks in these datasets. Looking into a detailed view of each of the papers, we can see that most of the papers are using autoencoders for the detection purpose. This is due to the ease with which autoencoders are able to extract features and build more accurate models.

First let us look at a paper that compares the traditional machine learning algorithms and the neural networks and see how the accuracies vary.

Machine Learning DDoS Detection for Consumer Internet of Things Devices paper[20] talks about anomaly detection for IoT specific network behavior using different machine learning methods – K-nearest neighbors, Random Forest, Decision tree, Support Vector Machines and Neural network with a 4 layer fully connected feed forward layers. The purpose of the paper is to detect DDoS (Distributed Denial of Service) attacks with high accuracy and low cost for protocol agnostic and flow based IoT traffic. The accuracy results for the different machine algorithms vary between 0.991 to 0.999. The important factor to note here is that due to the flooding nature of DoS packets, there are attacks packets 15 times more than normal packets. This would mean that even if the algorithms predicted all packets to be malicious, it would result in a baseline accuracy of 0.93. The main contribution of the paper depicts the usage of IoT specific features. Using Gini score as a metric to differentiate the stateful and stateless features, the author shows that usage of a few stateless features along with the IoT specific features increases the accuracy and f1 score of detection by 0.5. The authors identify the necessity to study results of the study

using a more balanced distribution of normal and attack traffic, larger dataset, additional features and usage of complex machine learning techniques.

Now let us look into the papers that use autoencoders in varying ways to detect IoT and traditional network attacks.

Kitsune[4] is an unsupervised online network intrusion detection system that uses an ensemble of autoencoders to detect anomalies. The authors of this paper propose their core algorithm KitNet that works in tandem with feature extractors and mappers to detect network attacks with performance equal to that of offline detectors. The paper[4] compares the performance of Kitsune with other standardized algorithms like Suricata (signature based detection), Isolation Forest[35] and Gaussian Mixture Models[36](anomaly based batch detection), Incremental Gaussian Mixture Model[37] and pcStream[38](anomaly based online detection) for nine different types of network attacks that include OS Scan, Fuzzing, Video Injection, ARP MiTM, Active Wiretap, SSDP Flood, SYN DoS, SSL Renegotiation and Mirai botnet. The Kitsune outperforms the online anomaly detection algorithms and holds its own against the offline anomaly detection algorithm as well. The greatest advantage of Kitsune is its runtime performance that is enhanced by using ensemble of autoencoders instead of a single encoder that reduces the number of operations used to process an instance. The paper also clearly shows the tradeoff between the runtime performance and detection performance for the Kitsune model. Depending on the user's requirement, the value of *m* (user defined parameter) can be adjusted to increase the detection performance or the runtime performance. Though the Kitsune solution looks superior to its predecessors, it is not without drawbacks. One of the main drawbacks of the Kitsune algorithm is its susceptibility to adversarial attacks. Since the algorithm assumes all the input network traffic to be benign during the training mode, it is possible for a compromised system to be able to train the algorithm to acknowledge the attacks as normal behavior. Another drawback is that a Denial-of-Service attack on the Feature Extractor can overwhelm the system to store a large number of instances and hence destabilize the NIDS. The authors of the paper[4] run the algorithms on a custom dataset that they have created. While this is acceptable, it is not easy to baseline the evaluation criteria for its performance. It would have been more useful and relevant if the performance statistics were based on standard datasets.

A Deep Learning Approach for Network Intrusion Detection System[7] paper proposes a deep learning-based approach called Self Taught Learning (STL) on NSL-KDD benchmark dataset for network intrusion. The STL consists of using a sparse autoencoder for unsupervised learning of the features and this learnt representation is used for classification which is done using SoftMax regression. The main advantage of this paper is that it is using a benchmark dataset. This evaluates the performance of the STL against traditional SoftMax regression algorithm with STL showing better performance for 2-class than 5 and 23-class detections. The authors propose usage of more advanced techniques in place of the sparse autoencoder like Stacked Autoencoder for unsupervised feature learning, and NB-Tree, Random Tree, or J48 for classification. The authors plan to implement a real-time version of the same along with on-the-go feature learning on raw network traffic headers. The disadvantage of this model is that it performs well only for datasets with a low number of classes. Higher the number of classes, the performance of the model is worse than the traditional SoftMax regression.

Deep Autoencoding Gaussian Mixture Model[19] (DAGMM) for unsupervised anomaly detection utilizes a deep autoencoder that performs dimensionality reduction for input samples, prepares their low dimensional representations from both the reduced space and the reconstruction error features and feeds these representations into the next stage which is the estimation network that predicts the likelihood within the GMM framework. This paper also utilizes a benchmark dataset KDDCUP for estimating the model's performance. The authors have used traditional machine learning algorithms like support vector machines, Deep structured energy-based model and Deep clustering network model along with variations of DAGMM model. The authors have considered precision, recall and F1 score for the accuracy metrics with the DAGMM model showing high precision and F1 score as compared to the other models. The DAGMM shows 14% increase in performance compared to the previous models on the benchmark datasets. Usage of KDDCUP dataset by the authors really highlights the improvements of their models with other models using the same dataset, hence making it easy for other researchers to build upon it. Another advantage of the model is that it is not specific to certain datasets or traffic. The authors have shown that the model works with network intrusion datasets, arrhythmia and thyroid datasets.

A Deep Learning Approach to Network Intrusion Detection[21] proposes an unsupervised feature learning with non-symmetric data dimensionality reduction technique using autoencoders. This helps in providing better classification results using the random forest classifier. The main goal of this paper is to utilize deep and shallow learning techniques to improve detection accuracy while reducing training time and analytical overheads. Usage of NSS-KDD benchmark dataset for evaluation highlights the improvements made by this model over the existing approaches. The authors use a variety of network attacks like DoS, Probe, R2L and U2R for the detection. The model shows promising improvements for granular and detailed datasets while decreasing the training time by 78% when compared to Deep Belief Networks (DBN). The future work identified by the authors is an improvement to the model to handle zero day attacks. The strength of the paper is that it is able to significantly reduce the training time while increasing the accuracy of anomaly detection. Another important factor is that the current approach reduces the false alarm rate when compared to DBNs. The only drawback is that the method works well for detailed and large datasets but performs moderately for simple datasets with low number of classes.

N-BaIoT—Network Based Detection of IoT Botnet Attacks Using Deep Autoencoders[6] proposes an anomaly detection methodology that extracts behavior snapshots of the network and utilizes deep autoencoders to detect the anomalous IoT traffic. The authors compare the results of their methodology against One class SVM, Local Outlier Factor and Isolation Forest showing that their methodology achieves a better true positive rate, decreased false positive rate and utilizes less time for detection as compared to the traditional approaches. The advantage of this paper is that it demonstrates a 100% true positive rate using its model and an almost 0% false positive rate for both the BASHLITE and MIRAI IoT attacks. The disadvantage is that the paper utilizes a custom dataset procured by the authors which reduces the ability to compare with previously estimated accuracy results. The authors plan to evaluate transfer learning techniques for their future works.

After looking into papers that utilize autoencoders and unsupervised learning for anomaly detection, let us now focus on the usage of new technologies like RNN (Recurrent Neural Networks), LTSM-RNN, etc.

ZeroWall: Detecting Zero Day Web Attacks through Encoder-Decoder Recurrent Neural Networks[5] is an unsupervised approach for the detection of Zero-day Web attacks hidden in Web requests. The paper proposes a methodology that utilizes the encoder-decoder RNN to capture the features of benign requests and prepare a self-translation model. When an attack request passes through this model, it cannot be translated back to the original benign request and hence can be declared as an attack. This approach successfully detects the zero-day attacks missed by traditional Web Application Firewalls (WAF) and outperforms it by achieving a high F1 score of 98%. The strength of this approach is that it can be used as a augmentation on top of the existing WAFs and is immediately usable in real world scenarios. It is also the first paper to translate the usage of encoder-decoder RNNs for detection of zero-day web attacks. Another advantage of this paper is that it utilizes real world traffic. While the paper achieves such great results, it is not without drawbacks. One of the drawbacks is that this approach focuses on contextual and collective anomaly detection which is unable to be used for evaluation of this paper's model. Another drawback is that too small a dataset penalizes the performance metrics of the model. Since the model is used in real world scenario and is unsupervised, it is susceptible to poisoning attacks. Future work could be focused on evading such poisoning attacks.

Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic[22] paper talks about sequence modeling using long short-term memory recurrent neural networks (LSTM-RNN) on the CICISS2017 benchmark dataset. The paper borrows the concept from natural language processing literature and applies it to anomaly detection. The paper uses a simple frequency-based method of outlier detection for its baseline. The paper shows that in most cases, the simple frequency-based method works better than the LSTM methodology. This could be because the LSTM methods are more suited to capturing the beginning of an attack as compared to detecting all the flows related to the attacks. The paper claims that the methodology they have proposed is a steppingstone for further research in the field. The drawbacks of the paper are that it neither identifies the good features to be used nor the best aggregated sequences that could lead to improved detection accuracy.

We have looked into various works related to the machine learning techniques that can be used for anomaly detection. Let us now delve into the feature selection mechanism for such

datasets. Feature selection is the process of reducing the number of input features to a machine learning algorithm such that the model developed is using the most important features in its decision-making process. Such methods help in reducing computation cost and sometime can enable better performance. Supervised feature selection methodologies can be divided into 3 types - Filter methods, Wrapper methods and Embedded methods[43].

Filter models predict based on measures of the overall traits of the training data such as distance, consistency, dependency, information, and correlation. Relief [45], Fisher score [46] and Information Gain based methods[47] are among the most representative algorithms of the filter model[44]. Wrapper selection methods utilize a classification algorithm during the feature evaluation step that leads to better performance . Filter methods are independent of classification algorithms and hence are computationally less intensive but perform much poorer compared to wrapper methodologies. The embedded model was proposed to bridge the above gap between the filter and wrapper models. It includes the statistical criteria, same as the filter model to select feature subsets and then chooses the subset with highest classification accuracy similar to wrapper models. Hence the embedded model achieves both high performance accuracy and low computation time[44].

3. DATASET DESCRIPTION

The dataset used in our paper, consists of 115 features extracted from raw network traffic data flowing through 9 different IoT devices[27]. The dataset consists of benign traffic and anomalous MIRAI and BASHLITE traffic packets. The benign traffic is assigned the class 0 while malicious traffic is assigned the class 1. This leads to the decision tree to conduct a binary classification on our datasets. In the case of multi class decision tree classification, we have assigned class 0 for benign traffic, class 1 for MIRAI botnet traffic and class 2 for BASHLITE botnet traffic.

The 9 datasets are collected from different IoT devices like webcam, doorbells, thermostat, security cameras and baby monitor. Each device has its own functionality and different network traffic pattern which are extracted into 23 features from 5 different time windows of the most recent 100ms, 500ms, 1.5sec, 10sec and 1minute[6]. The features are extracted from individual network packets with packets being considered separately(flow context not present). The datasets are described in Table 3.1 in more detail.

Table 3.1*: Description of the 9 datasets*

No.	Dataset Name	Dataset Type	Benign Samples	Malicious Samples	Mirai Botnet Data	Bashlite Botnet Data
1	Danmini Doorbell	Doorbell	49548	968750	✓	✓
2	Ecobee Thermostat	Thermostat	13113	822763	✓	✓
3	Ennio Doorbell	Doorbell	39100	316400	X	✓
4	Philips B120N10 Baby Monitor	Baby Monitor	175240	923437	✓	✓
5	Provision PT737E Security Camera	Security Camera	62154	766106	✓	✓
6	Provision PT838 Security Camera	Security Camera	98514	738377	✓	✓
7	Samsung SNH1011N Webcam	Webcam	52150	323072	X	✓
8	SimpleHome XCS71002WHT Security Camera	Security Camera	46585	816471	✓	✓
9	SimpleHome XCS71003WHT Security Camera	Security Camera	19528	831298	✓	✓

The 23 features that are extracted from network traffic are aggregated as listed below in Table 3.2.

Table 3.2*: Features extracted from network traffic for each dataset*[6]

Feature Type	Packet Details	Aggregation Method	Number of Features Extracted
Packet Size (outbound packets only)	Source IP	Mean, Variance	8
	Source MAC		
	Channel		
	Socket		
Packet Size(both inbound and outbound packets)	Channel	Magnitude, Radius, Covariance, Correlation-coefficient	8
	Socket		
Packet Count	Source IP	Weight	4
	Source MAC		
Packet Jitter	Channel	Mean, Variance, Weight	3

The aggregation methods[27] used for extracting these features are explained in detail below:

- Mean - Average of the size of the packets with particular Source IP, Source MAC, Channel and socket. It is also used to calculate the average of the packet jitter over a particular channel.
- Variance - It measures how spread out the packet size and packet jitter values are. It is calculated as average squared deviation of each number from the mean of a dataset.
- Magnitude - It measures the root squared sum of the means of the inbound and outbound packet streams.
- Radius - It calculates the root squared sum of the variances of the inbound and outbound packet streams.
- Covariance - It computes the mean value of the product of the deviations of the inbound and outbound packet streams from their respective means.
- Correlation coefficient - It calculates the linear correlation between the inbound and outbound packet streams.
- Weight - It computes the number of elements observed in the recent time window.

Looking into the dataset, we can see that each feature is named using certain symbols like H, HH, MI etc., along with numbers and the aggregation methodology. The dataset has been consolidated such that each network feature extracted and aggregated is named in the format of "FeatureName_LTimeWindow_AggregationMethod". The Table 3.3 below shows the meaning of each feature and their symbolization[6]. The time windows are 100ms(L0.01), 500ms(L0.1), 1.5sec(L1), 10sec(L3) and 1min(L5).

Table 3.3*: Features from Stream Aggregation*

Feature Name	Packet Detail	Meaning	Example
H	Source IP	Statistical summary of the recent traffic from this packet's host IP	H_L0.01_variance
MI_dir	Source MAC-IP	Statistical summary of the recent traffic from this packet's host MAC-IP	MI_dir_L1_weight
HH	Channel	Statistical summary of the recent traffic going from this packet's host (Source IP) to the packet's destination host (Destination IP).	HH_jit_L5_mean
HH_jit	Channel Jitter	Statistical summary of the jitter of the traffic going from this packet's host (Source IP) to the packet's destination host (Destination IP).	HH_L3_magnitude
HpHp	Socket	Statistical summary of the recent traffic going from this packet's host and port (IP) to the packet's destination host and port.	HpHp_L5_covariance

4. IMPORTANT FEATURE RANKING

For supervised learning predictive models, finding important features is essential in making accurate predictions. For classification tree-based models, the important features can be determined by evaluating the number and weight of splits, the given feature was involved in[12]. Using important feature ranking can result in higher accuracy while reducing the computation time.

For our datasets, we are able to determine the best and important features to differentiate between normal and anomalous traffic by using the Input Perturbation Ranking (IPR) method with XGBoost algorithm. Once the IPR algorithm is run for each dataset, we rank the most important and common features across all the 9 datasets to find the 9 most important features. These features help us in understanding how the anomalous traffic deviates from the normal behavior. They can also be used to reduce the computation power for creating a decision tree classifier with just 9 features instead of 115.

The following steps are followed to find the 9 most important features (depicted in Figure 4.4):

- **Input Perturbation Ranking -** IPR is a feature importance algorithm that calculates the loss of a model when each of the input features to the model is perturbed by the algorithm[12]. It means that if an important feature value is changed, then the model suffers a very visible change as well. When a feature is perturbed, it becomes useless and is equivalent to be removed from the evaluation. Thus, if a feature is more important, such perturbation will lead to more losses and less accurate results for the model. The feature importance ranking is displayed to user in the form of a table consisting of the feature name, its importance value and the error or loss. The higher the loss, more important is the feature. Each feature has an importance value which is calculated using the formula:

$$Feature\ Importance = \frac{Feature\ Loss}{Maximum\ Loss}$$

Equation 4-1: *Feature Importance*

Higher the importance value, more important is the feature. We fit all our datasets using the XGBoost algorithm[26] as it leverages the advantages of random forest model and gradient boosting to strengthen the model and provide prediction errors ten times lower than random forest models. XGBoost also has a better performance time when compared to the traditional random forest approach and hence it is able to process larger sized datasets faster and more accurately. Once we have the trained model, we predict the target probabilities and perturb the features. By running the IPR on each of these datasets, we are able to collect 30 most important features based on the importance values from the 115 features available.

- **Feature Ranking per dataset -** Each of the important features retrieved in the previous step are ranked from 1 to 30 with 1 being the most important and 30 being the least, as shown in Table 4.1.

Table 4.1*: Feature Ranking per dataset*

	1	2	3	4	5	6	7	8	9
	Danmini Doorbell	Ecobee Thermostat	Ennio Doorbell	Philips B120N10 Baby Monitor	Provision PT737E Security Camera	Provision PT838 Security Camera	Samsung SNH1011N Webcam	SimpleHome XCS71002WHT Security Camera	SimpleHome XCS71003WHT Security Camera
1	MI_dir_L0.1_weight	HH_jit_L5_mean	HH_jit_L5_mean	HH_jit_L5_mean	HH_jit_L5_mean	HH_jit_L0.01_mean	MI_dir_L0.01_mean	HH_jit_L5_mean	HH_jit_L5_mean
2	HH_jit_L0.01_mean	HH_jit_L0.01_mean	HH_jit_L0.01_mean	HH_jit_L0.01_mean	HH_jit_L0.01_mean	MI_dir_L0.1_weight	HH_jit_L5_mean	HH_jit_L0.01_mean	HH_jit_L0.01_mean
3	HH_jit_L5_mean	MI_dir_L5_weight	MI_dir_L0.1_weight	MI_dir_L0.1_weight	MI_dir_L0.1_weight	HH_jit_L5_mean	HH_jit_L0.1_mean	MI_dir_L0.01_weight	MI_dir_L0.1_weight
4	MI_dir_L0.01_mean	MI_dir_L0.1_weight	HH_jit_L0.1_mean	HH_L0.01_radius	HH_jit_L0.01_variance	HH_L0.01_radius	MI_dir_L0.01_variance	MI_dir_L0.1_weight	HH_jit_L0.01_variance
5	HpHp_L0.01_covariance	HH_jit_L1_variance	MI_dir_L0.01_mean	HpHp_L0.01_radius	HpHp_L0.01_weight	MI_dir_L0.01_mean	MI_dir_L0.01_weight	HH_L0.01_radius	MI_dir_L1_weight
6	MI_dir_L0.01_variance	MI_dir_L0.01_mean	MI_dir_L1_mean	MI_dir_L0.01_weight	MI_dir_L0.01_weight	HH_jit_L0.01_variance	HH_L0.1_weight	HH_L0.1_weight	HpHp_L0.01_weight
7	HH_L0.01_covariance	HH_jit_L0.1_variance	HH_L0.01_mean	HpHp_L0.01_covariance	HH_jit_L0.1_variance	HH_jit_L0.1_variance	MI_dir_L0.1_weight	HH_jit_L0.01_variance	MI_dir_L1_variance
8	HH_jit_L1_variance	HpHp_L0.01_covariance	HH_jit_L1_variance	HH_L0.01_weight	MI_dir_L0.01_mean	MI_dir_L0.01_weight	HH_jit_L0.1_variance	HpHp_L0.01_covariance	MI_dir_L0.1_variance
9	MI_dir_L0.01_weight	HH_jit_L3_variance	HH_jit_L0.1_variance	HH_jit_L0.01_variance	HpHp_L0.01_std	HpHp_L0.01_std	HpHp_L0.01_weight	MI_dir_L0.1_variance	HH_L0.01_radius
10	HpHp_L5_std	HH_L0.01_radius	HH_jit_L5_variance	MI_dir_L0.01_variance	HH_L0.01_covariance	HH_L0.1_mean	HH_jit_L0.01_variance	MI_dir_L0.01_mean	HpHp_L0.01_std
11	HH_L0.01_radius	HH_L0.1_weight	HH_jit_L0.01_variance	HH_L5_magnitude	HH_L0.1_weight	MI_dir_L3_variance	MI_dir_L5_weight	MI_dir_L0.01_variance	HpHp_L0.01_covariance
12	HH_L0.1_weight	HpHp_L0.01_std	HpHp_L0.01_weight	HH_jit_L0.1_mean	HH_L0.01_radius	HH_L0.01_covariance	MI_dir_L0.1_mean	HpHp_L0.01_std	MI_dir_L0.01_variance
13	HH_L1_radius	MI_dir_L0.1_variance	HH_L0.1_magnitude	HpHp_L5_magnitude	HH_L0.1_mean	HH_jit_L1_variance	MI_dir_L1_weight	HpHp_L0.01_weight	MI_dir_L0.1_mean
14	HH_jit_L0.01_variance	HH_jit_L0.01_variance	MI_dir_L1_variance	HH_L0.01_mean	HH_L0.01_weight	HH_L5_weight	HH_L5_mean	HH_jit_L0.1_variance	MI_dir_L5_mean
15	MI_dir_L3_weight	HH_L0.01_magnitude	HH_L0.01_radius	HH_L0.01_std	MI_dir_L0.01_variance	HH_L0.01_mean	HH_jit_L1_variance	HH_jit_L5_variance	HH_L0.1_weight
16	HpHp_L0.01_radius	MI_dir_L0.1_mean	MI_dir_L0.1_mean	MI_dir_L1_mean	HH_L1_weight	HH_jit_L0.1_mean	HH_L0.01_radius	HpHp_L0.1_weight	HH_jit_L0.1_variance
17	HH_L5_weight	HpHp_L0.01_radius	MI_dir_L5_mean	HH_L0.01_magnitude	HH_L5_radius	MI_dir_L1_mean	MI_dir_L0.1_variance	MI_dir_L1_mean	MI_dir_L0.01_weight
18	HpHp_L0.1_covariance	MI_dir_L1_mean	HH_jit_L1_mean	MI_dir_L1_weight	HH_jit_L1_variance	MI_dir_L5_variance	HpHp_L0.01_magnitude	HH_jit_L1_mean	HH_L1_mean
19	HH_L1_weight	HH_jit_L1_mean	HH_L1_magnitude	HH_jit_L1_variance	MI_dir_L1_weight	HH_L1_weight	HH_jit_L0.01_mean	HH_jit_L1_variance	MI_dir_L5_weight
20	MI_dir_L3_variance	MI_dir_L3_variance	HpHp_L0.1_radius	MI_dir_L0.01_mean	HH_L0.01_magnitude	MI_dir_L3_weight	HH_L0.01_std	MI_dir_L1_weight	MI_dir_L0.01_mean
21	MI_dir_L5_weight	HpHp_L3_mean	MI_dir_L0.01_weight	HH_jit_L0.1_variance	MI_dir_L1_variance	HH_L1_radius	HH_L0.1_magnitude	HH_L5_weight	MI_dir_L3_weight
22	HpHp_L5_radius	H_L0.01_variance	MI_dir_L5_variance	HpHp_L1_weight	MI_dir_L5_weight	HH_L0.1_covariance	MI_dir_L1_variance	HH_L0.01_weight	HH_L0.1_radius
23	MI_dir_L1_mean	MI_dir_L0.01_variance	HH_L0.1_weight	MI_dir_L0.1_mean	HpHp_L0.1_weight	HpHp_L5_weight	HH_L1_radius	HpHp_L1_radius	H_L0.01_weight
24	HpHp_L3_radius	MI_dir_L5_mean	MI_dir_L0.01_variance	HH_L0.1_weight	HpHp_L0.1_magnitude	MI_dir_L0.1_mean	HpHp_L0.01_std	H_L0.01_variance	HH_L0.01_std
25	MI_dir_L5_mean	HH_L5_weight	HH_L1_radius	HH_L0.01_covariance	HpHp_L5_magnitude	MI_dir_L5_mean	HH_L0.1_std	HpHp_L1_weight	HH_L0.01_covariance
26	HH_L5_std	HH_L3_weight	HH_L1_std	MI_dir_L3_weight	MI_dir_L0.1_mean	HH_L3_mean	HpHp_L0.1_radius	HpHp_L5_weight	MI_dir_L1_mean
27	MI_dir_L3_mean	HH_L5_radius	HH_L0.01_magnitude	MI_dir_L5_mean	HH_L0.1_std	MI_dir_L5_weight	HH_L0.01_magnitude	MI_dir_L0.1_mean	MI_dir_L5_variance
28	HH_jit_L5_variance	HH_L0.01_covariance	HpHp_L0.1_weight	HH_L1_radius	HH_L3_weight	HpHp_L0.01_weight	HpHp_L0.1_weight	HH_L5_std	H_L0.01_variance
29	HH_L3_radius	HpHp_L1_pcc	HH_L5_radius	HH_L1_weight	H_L0.01_variance	HH_L0.1_radius	HH_jit_L1_mean	HH_L0.1_std	HH_L0.01_magnitude
30	H_L0.01_weight	HH_L1_radius	HH_L0.1_mean	HH_L5_mean	HH_L5_weight	HH_L0.1_pcc	HpHp_L5_weight	HpHp_L0.01_mean	MI_dir_L3_variance

- **Feature Ranking Aggregation across datasets -** After collecting the most important 30 features for all 9 datasets, we filter the features that are common across all the 9 datasets (shown in Table 4.2). Once these common features have been filtered, we add the ranking of this feature across the 9 datasets and get a final ranking score. For example, if a feature A, has rank 1 in dataset 1, rank 4 in dataset 2 and so on, we add the rank number to get an aggregated rank score.

Table 4.2*: Sample of features that are present across different datasets*

FEATURES	No of Datasets its present in
MI_dir_L0.01_mean	9
HH_jit_L0.01_mean	9
MI_dir_L0.01_variance	9
HH_jit_L0.01_variance	9
HH_jit_L1_variance	9
HH_jit_L5_mean	9
MI_dir_L0.1_weight	9
MI_dir_L0.01_weight	9
HH_L0.01_radius	9
MI_dir_L1_weight	8
HH_jit_L0.1_variance	8
HH_L0.1_weight	8
MI_dir_L5_mean	7
HH_L1_radius	7
MI_dir_L0.1_mean	7
MI_dir_L5_weight	6
HH_jit_L0.1_mean	6
MI_dir_L1_mean	6
HpHp_L0.01_std	6
HpHp_L0.01_weight	5

- **Most Important Feature Ranking -** After aggregating the rank score for all the common features across 9 datasets, we rank their importance based on the lowest ranking score. The one with the lowest ranking score is the most important feature.

Using the above defined methodology, we have resulted in retrieving 9 most important features as shown in Table 4.3.

Table 4.3*: 9 Most Important Features*

Features	Important Feature Ranking
HH_jit_L5_mean	**14**
MI_dir_L0.1_weight	**31**
HH_jit_L0.01_mean	**35**
MI_dir_L0.01_mean	**76**
HH_jit_L0.01_variance	**87**
HH_L0.01_radius	**91**
MI_dir_L0.01_weight	**91**
HH_jit_L1_variance	**130**
MI_dir_L0.01_variance	**134**

The 3 most important features are:

- **HH_jit_L5_mean -** This feature defines the average of the packet jitter between the packet's host and destination over a particular channel in the time window of 1 minute (L5). Looking into the datasets and the values this feature holds, we can see that for normal traffic, the jitter is a nominal amount while for anomalous traffic, the

jitter tends to either be very high (in millions) or extremely low (between 0 and 1). This feature helps in understanding how packet jitter variations are able to influence the distinction of traffic as normal or anomalous.

- **MI_dir_L0.1_weight -**It describes the weight of the packets seen during the time window of 500ms(L0.1) originating from a particular Source MAC-IP. Weight describes how many of such packets were observed during a particular time period. This feature when plotted against the classes (Figure 4.1), shows that for anomalous values, it is very high whereas for normal values it tends to cluster around 0 to 2 (log scaled). The high weight for anomalous traffic is in line with the generic botnet activity of producing high volume of packets. Hence this is an important feature in distinguishing between normal and anomalous traffic.

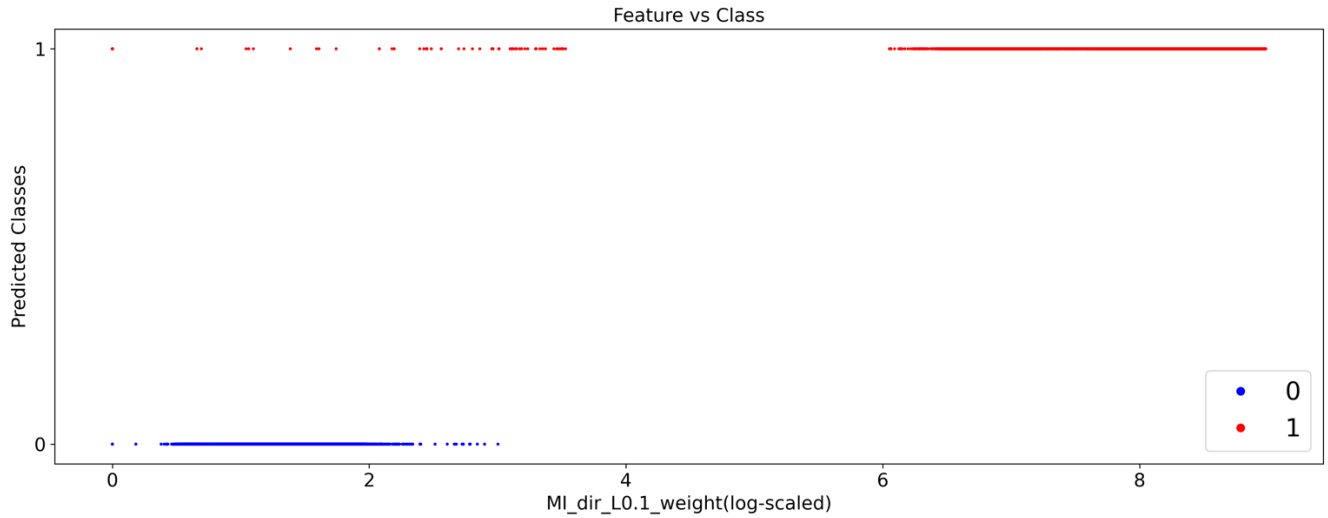


Figure 4.1: *Log-scaled Scatter Plot of feature MI_dir_L0.1_weight against the classes for Danmini Doorbell dataset*

- **HH_jit_L0.01_mean -** This feature defines the average of the packet jitter between the packet's host and destination over a particular channel in the time window of 100ms(L0.01). This feature is used as the root node of the decision trees for 5 of our datasets showing the importance of this feature. The decision tree for Danmini

Doorbell datasets as shown in figure 4.2, classifies all samples having values below 773.419 as normal and anything above as anomalous.

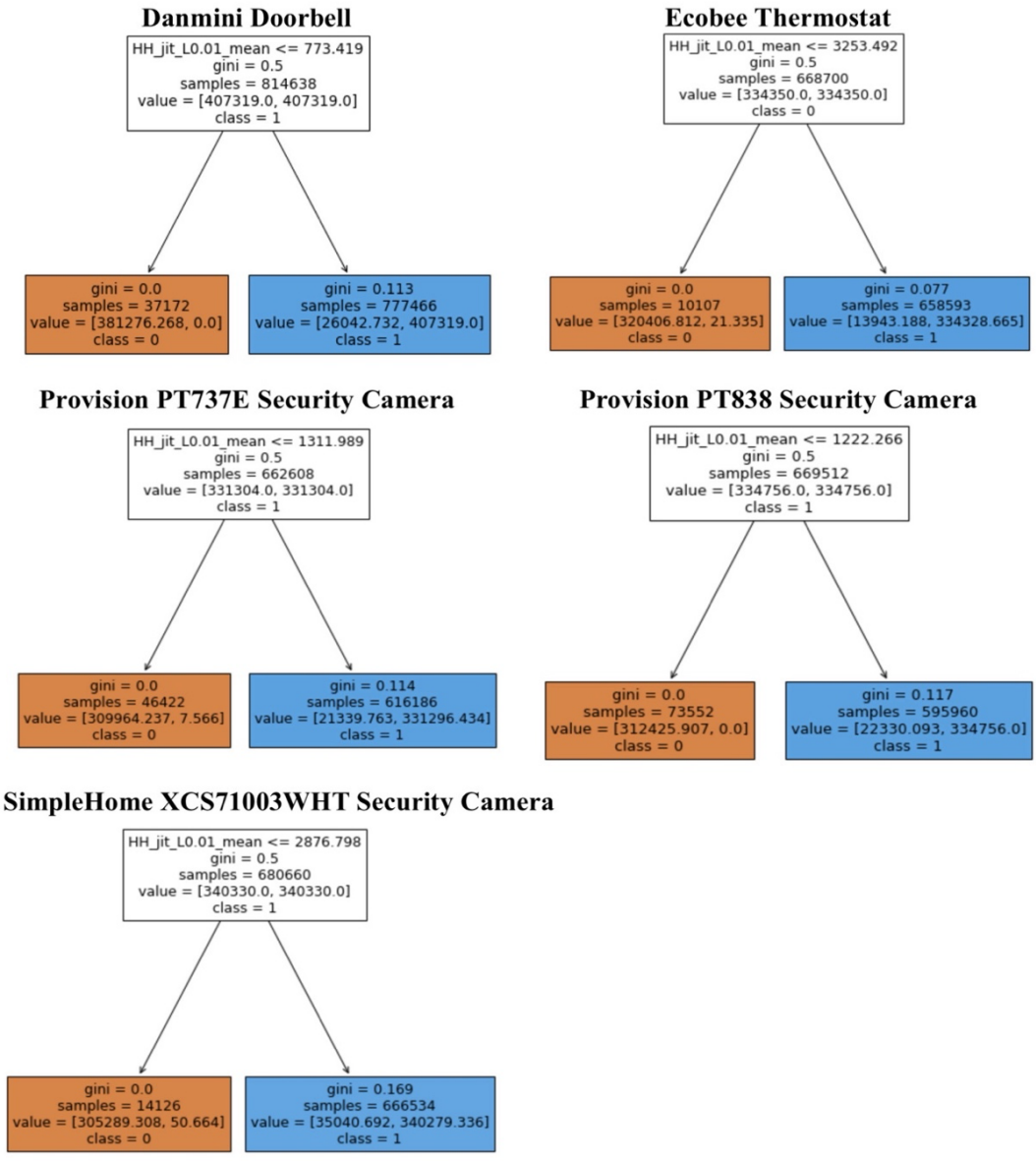


Figure 4.2: *Depicts the use of HH_jit_L0.1_mean as root node for Decision tree classifier in various datasets*

From the 9 datasets, we separate the 9 features for each dataset and run decision tree classifier on each of them. The performance accuracy is over 99.9% for all datasets and the processing times are in the form of a few seconds. This shows that the above 9 features can be used as the main features to detect any malicious traffic with very high accuracy and very small

processing time. This helps in detecting important features that are common across all datasets helping us to understand network features that might be a common factor in differentiating between normal and anomalous IoT traffic.

5. DECISION TREE CLASSIFIER

The model we are proposing in this thesis is based on the usage of Decision Trees to detect botnet traffic in various different types of IoT devices. We are using Decision trees which is a supervised predictive model as they take less computation time while providing accurate results when compared to other supervised or unsupervised machine learning methods. The usage of minimal computation resources and time helps in using decision trees on IoT edge devices as a first line of defense. The decision tree model also provides very clear and specific rules based on which the model classifies the samples as normal or anomalous. These clear and concise rules can be easily converted into security policies and applied on network security devices to prevent malicious traffic from infiltrating the network. Another advantage that decision tree provides over unsupervised learning methods like autoencoders is that we can have a tradeoff between computation time and power with accuracy. By varying the height to which a decision tree model can grow, the user can have a tradeoff between the computation time and power with accuracy. The higher the tree grows, the better the accuracy but it requires more computation time and power. In the case of small IoT devices such as doorbells or thermostats with very low computational power, running our methodology with a low tree depth can help with detection without greatly affecting its functionality. In devices like security cameras that come with more powerful processors, the tree depth can be increased proportionally to ensure more efficient detection performance. Hence, the user is given the flexibility to choose according to the needs of the network environment. This flexibility is not readily controllable with other unsupervised machine learning algorithms like autoencoders.

Decision tree is a predictive supervised machine learning algorithm that helps in classification or regression problems. The decision tree algorithm used for classification, creates a training model that is able to predict the class of target variable by learning simple rules from a prior training dataset. The decision tree simulates the tree structure and has the following terminology:

- Root Node: The first node at the top of the decision tree contains the entire sample and further gets divided into two (binary classification) or more (multi class classification) sub-nodes.

- Decision Nodes: When the sub-node is split into further sub-nodes, the parent sub-node is called the decision node which contains the rule based on which the classification happened.
- Leaf Node: The nodes at the bottom of the tree structure upon which further division is not possible.
- Tree depth: This specifies the number of levels a decision tree can have from the root node to the leaf node.

The above-mentioned elements of the decision tree are explained with an example in Figure 5.1.

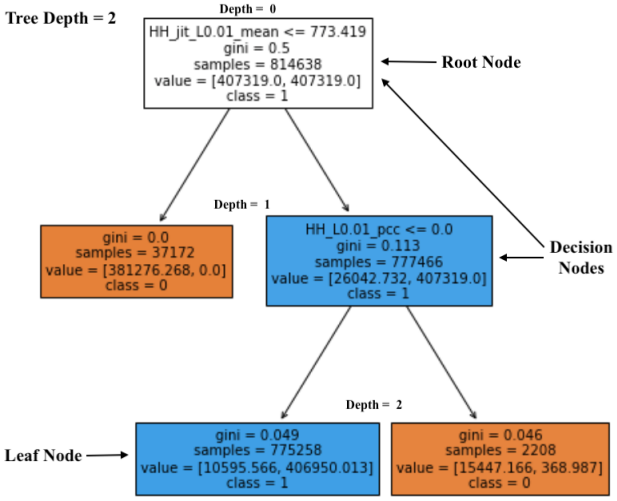


Figure 5.1: *Decision Tree elements explained with an example*

The crux of the decision tree split depends on choosing the feature that best separates the noise from the information to differentiate between classes. To measure the informativeness of a feature and use it as the decision node there are two methods - Entropy and Gini Index.

- **Entropy -** It is used to measure the impurity or the randomness of a dataset. Lower the entropy value, better is the discrimination capability of the algorithm.

$$Entropy = -\sum_i p_i * \log_2(p_i)$$

Equation 5-1: *Entropy*

where,

p_i is the probability of a sample being classified to a particular class $i \in \{1 \ldots n\}$.

- **Gini Index -** The Gini Index[30]is used to decide the important features based on which the tree will be split into sub-nodes. The lower down the tree we go, the better the classification. Gini index measures the probability of a sample being classified in the wrong class. The formula for Gini index is:

$$Gini\ Index = 1 - \sum_{i=1}^{n} (p_i)^2$$

Equation 5-2: *Gini Index*

where,

p_i is the probability of a sample being classified to a particular class $i \in \{1 \ldots n\}$.

n is the number of classes.

The Gini index varies between 0 and 1-1/n, with 0 showing that all samples belong to the class they were classified into and 1-1/n shows that the samples are evenly distributed between the classes. Hence, a low value of Gini index in the final nodes of the tree imply greater discrimination. In our methodology, we have used the Gini Index as a measure for splitting the decision tree nodes.

The first step in our proposed methodology (Figure 5.2) is to prepare the datasets into training and testing sets with an 80-20 distribution. Each training dataset is going to be run through the decision tree classifier and a model is generated. Since our datasets contain a higher number of anomalous traffic as compared to normal traffic, using plain decision trees can make the trees very skewed giving inaccurate results. Hence, we are going to use weighted decision trees to train the model. By using class weights that are inversely proportional to the length of a class sample, we can make sure that the tree is going to be balanced and impartial.

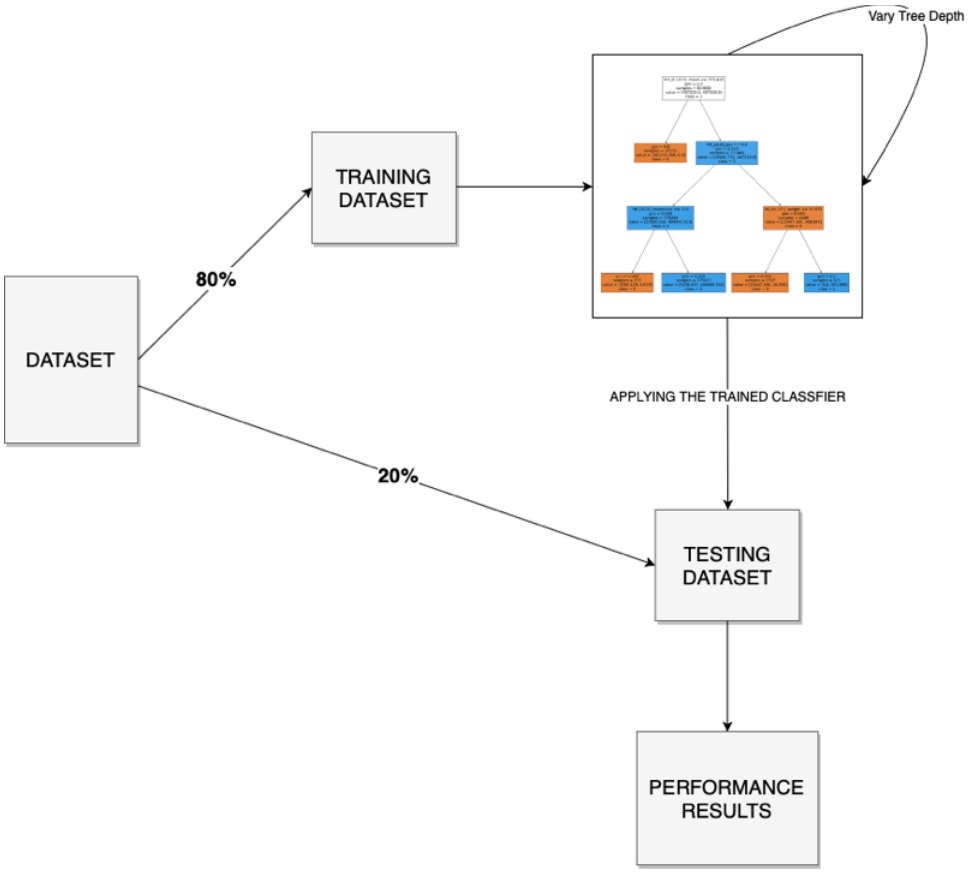


Figure 5.2*: Illustration of our Decision Tree algorithm methodology*

The training datasets are run with the maximum tree depth, meaning that the decision tree is allowed to grow to its full length, till no more nodes can be split into sub-nodes. Next, we are going to vary the tree depths and evaluate the tradeoff between performance and processing time. Preliminary analysis for the values of tree depth shows that a depth of 3 has results that are comparable to the maximum depth tree and hence we limit the experiments to using 1,2,3 and maximum depth for the tree depth values. For each of the decision training model that are built, we are going to calculate the total time taken for training.

The second step is to run the respective testing datasets over the trained model to predict the classes. The prediction times are calculated as well and combined with the training time to find the total time taken to run samples through our methodology. This will help in comparing the processing efficiency of our methodology over the baseline. Along with carrying out binary classification where benign traffic is assigned class 0 and anomalous traffic is assigned class 1,

we also carry out multi class classification by assigning benign traffic as class 0, Mirai botnet malicious traffic as class 1 and Bashlite botnet malicious traffic as class 2. The advantage of using decision trees is that, on one model we can do binary or multi class classification whereas Kitsune baseline would require two different models to be able to differentiate between benign and Mirai traffic and benign and Bashlite traffic. We also are able to combine the datasets together to form one dataset and run it through the decision tree classifier. This is not possible using autoencoders as combining the datasets will remove the patterns of specific devices and hence make the autoencoder model unusable.

Finally, for calculating and measuring the performance of our method, we use metrics such as accuracy, precision, recall, F1-measure, true and false positive rates. Our methodology provides an average of 99% accuracy for all 9 datasets. We will discuss more of these results in Section 6.

6. EVALUATION

In this section, we are going to look at the evaluation of our methodology in terms of packet detection and runtime performance. We are going to look at the experimental setup and then discuss the results generated.

6.1 Experimental Setup

In our evaluations, we compare our methodology with the Kitsune algorithm and other supervised learning methods. Though Kitsune is an unsupervised online learning methodology, we compare our methodology with it in an effort to show the difference in runtime while maintaining accuracy. Decision Tree which is a supervised learning method has a detection performance similar to an unsupervised deep learning method with significantly lesser runtime. Comparing with Kitsune shows using a simpler methodology as a first line of defense is more efficient than using a computation heavy methodology especially in case of IoT edge devices.

We use the same set of training and testing datasets for both methodologies so that the baseline is accurate and impartial. Since Kitsune is an unsupervised machine learning methodology, the only change in the training dataset would be to remove the malicious samples. Table 3.1 describes the benign and malicious sample count for each of the datasets used in the experiment.

6.2 Evaluation Metrics

The output of the decision tree classification are the predicted class values which can be compared against the actual class values and the performance of our algorithm can be measured. The detection performance can be measured using the True-Positive(TP), True-Negative(TN), False-Positive(FP) and False-Negative(FN) rates. The TP, TN, FP and FN values can be found using a confusion matrix developed by comparing the actual values with the predicted values as shown in Figure.

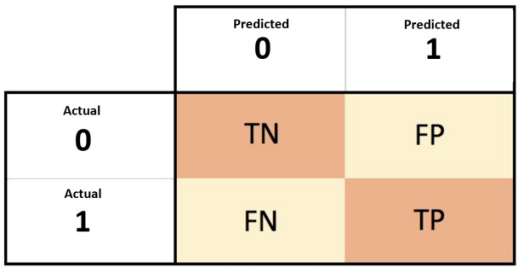


Figure 6.1: *Confusion matrix with true-positive, true-negative, false-positive and false-negative values depicted*

Using the TP, TN, FP and FN values , we can find their respective rates using formulae shown below.

$$TPR = \frac{TP}{TP + FN}$$

$$TNR = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{TN + FP}$$

$$FNR = \frac{FN}{TP + FN}$$

Equation 6-1: *True-positive, True-Negative, False-Positive and False-Negative Rates formulae*

These measures help us to understand how many anomalous packets were detected accurately and how many were mis-classified. The false-positive rate depicts the probability of a true negative value, being classified as a positive value. It is very important to have a low false-positive rate for any anomaly detection algorithm so that false alarms can be avoided.

We also use metrics such as accuracy, precision, recall and f1-score to understand how well the anomaly detection happens.

- **Accuracy -** calculates the ratio of correctly predicted samples to the total number of samples. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation 6-2: *Accuracy*

- **Precision -** measures how many values are accurately predicted as positive out of the total positively predicted values. It is defined by the following formula:

$$Precision = \frac{TP}{TP + FP}$$

Equation 6-3: *Precision*

- **Recall** or **Sensitivity** - also known as true-positive rate (TPR) specifies how many samples are actually positive out of the total samples predicted as positive. It is defined as:

$$Recall = \frac{TP}{TP + FN}$$

Equation 6-4: *Recall or Sensitivity*

- **F1-Score -** is the weighted average of both precision and recall. This measure helps in better understanding of the prediction accuracy in case of uneven class distribution. It is defined as:

$$F1 - Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

Equation 6-5: *F1-Score*

Area under the receiver operating characteristic curve (AUC) and receiver operating characteristic (ROC) is used to measure if the algorithm is classifying the samples correctly or if it is randomly guessing the labels. ROC curve is plotted with TPR as y-axis and FPR as x-axis. It

is a probability curve that plots TPR against FPR at various threshold levels. Figure 6.2 depicts the ROC and AUC plot. AUC represents the measure of separability. Higher the AUC, better is the model at predicting the classes accurately. AUC values vary between 0 and 1 with AUC=0 meaning that the classifier inaccurately predicts the positives as negatives and vice versa, AUC=1 meaning that the classifier accurately differentiates between the positive and negative samples.

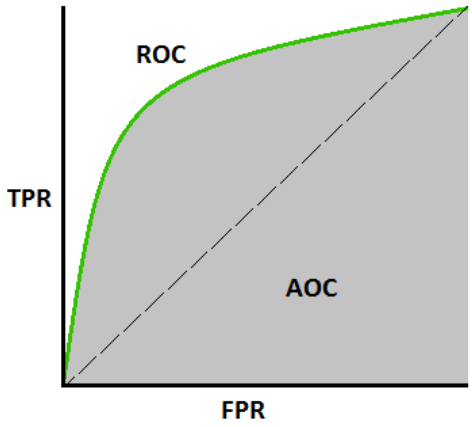


Figure 6.2: *ROC curve*

6.3 Decision Tree Classification Evaluation

6.3.1 Binary Classification

One of the greatest advantages of using decision trees is the very fast runtime performance exhibited by the algorithm. We can see from Table 6.1, the runtimes for decision trees with different depths. To be fairly compared with the baseline, Kitsune, in this section, all 115 features are applied to both binary and multi-class decision tree classification, instead of the 9 important features.

Table 6.1*: Runtimes for varying tree depths*

Dataset	Tree Depth	Training Time (seconds)	Prediction Time (seconds)
Danmini Doorbell	1	8.412	0.097
	2	15.818	0.082
	3	23.369	0.083
	MaxDepth	77.163	0.095
Ecobee Thermostat	1	6.419	0.073
	2	14.026	0.069
	3	21.496	0.067
	MaxDepth	55.728	0.064
Ennio Doorbell	1	1.209	0.026
	2	2.228	0.025
	3	3.233	0.026
	MaxDepth	13.428	0.029
Philips B120N10 Baby Monitor	1	11.341	0.087
	2	22.723	0.088
	3	32.934	0.086
	MaxDepth	121.201	0.103
Provision PT737E Security Camera	1	6.217	0.065
	2	13.474	0.065
	3	19.217	0.064
	MaxDepth	62.954	0.072
Provision PT838 Security Camera	1	6.628	0.068
	2	12.844	0.068
	3	18.393	0.069
	MaxDepth	70.122	0.079
Samsung SNH1011N Webcam	1	1.361	0.025
	2	2.487	0.024
	3	3.607	0.024
	MaxDepth	12.413	0.026
SimpleHome XCS71002WHT Security Camera	1	7.349	0.065
	2	14.825	0.061
	3	22.236	0.061
	MaxDepth	48.896	0.071
SimpleHome XCS71003WHT Security Camera	1	7.023	0.061
	2	14.023	0.062
	3	20.945	0.062
	MaxDepth	76.269	0.072

From the experiment, we can clearly see that the runtime is very less when compared to the Kitsune baseline as depicted in the Table 6.2. In this table, we are comparing a decision tree with maximum depth to the Kitsune autoencoder baseline. Table 6.2 also depicts that the decision tree algorithm has better detection rates as shown by the higher true-positive rate and accuracy metrics and very low false-positive rates when compared to Kitsune. For a few datasets, Kitsune shows 100% true-positive rate and recall, the false-positive rates are higher in the range of 3-10%. This shows that while Kitsune is extremely good at detecting the normal traffic patterns, it falters a bit with raising higher number of false-alarms due to higher false-positive

rate. Comparing our methodology with this we can clearly see that our false-positive rates are extremely low in the range of 0.0 to 0.03% making our algorithm much better at detection of anomalous traffic. Comparing the computation time, our methodology takes milliseconds to predict values, whereas the Kitsune baseline varies from 20 minutes up to 2 hours. Since the accuracy is same for methodologies at 99%, we can clearly see a huge advantage for our methodology in terms of runtime performance especially on IoT edge devices with low computation power.

Table 6.2*: Comparing Decision Tree with max depth results with Kitsune Baseline*

Dataset	Decision Trees								Kitsune - Autoencoders							
	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Danmini Doorbell	77.163	0.095	99.999	0.02	99.998	99.999	99.999	99.999	1029.25	5601.15	100	3.873	99.813	99.804	100	99.902
Ecobee Thermostat	55.728	0.064	99.998	0.038	99.997	99.999	99.998	99.998	318.609	3244.13	100	9.923	99.845	99.843	100	99.921
Ennio Doorbell	13.428	0.029	99.998	0.038	99.994	99.995	99.998	99.997	403.141	2303.06	100	6.686	99.26	99.175	100	99.586
Philips B120N10 Baby Monitor	121.201	0.103	99.998	0.011	99.996	99.998	99.998	99.998	833.776	6770.37	99.995	5.733	99.088	99.933	99.995	99.461
Provision PT737E Security Camera	62.954	0.072	99.997	0.032	99.995	99.997	99.997	99.997	707.744	3221.28	99.999	5.561	99.578	99.546	99.999	99.772
Provision PT838 Security Camera	70.122	0.079	99.999	0.005	99.998	99.999	99.999	99.999	593.537	3188.11	99.999	6.677	99.202	99.105	99.999	99.55
Samsung SNH1011N Webcam	12.413	0.026	99.995	0	99.996	100	99.995	99.998	627.169	1448.75	99.997	6.436	99.095	98.962	99.997	99.477
SimpleHome XCS71002WHT Security Camera	48.896	0.071	99.999	0	99.999	100	99.999	99.999	593.46	2678.47	99.994	10.222	99.435	99.411	99.994	99.702
SimpleHome XCS71003WHT Security Camera	76.269	0.072	99.999	0.026	99.998	99.999	99.999	99.999	312.68	2634.75	100	8.173	99.813	99.809	100	99.904
AVERAGE	**59.797**	**0.068**	**99.998**	**0.019**	**99.997**	**99.998**	**99.998**	**99.998**	**602.152**	**3454.45**	**99.998**	**7.032**	**99.459**	**99.51**	**99.998**	**99.697**

Looking at the nodes of Decision trees in Figure 6.3 depicting the various tree depths for Danmini Doorbell dataset, we can say that certain features are important in making decisions and they can be used to add security policies. Figure 6.4 and Figure 6.5 shows how the confusion matrix and AUC for Danmini Doorbell dataset improves as the tree depth grows. From the confusion matrix, we can clearly see that as the tree depth increases, the false positives decrease significantly. For the tree with maximum depth, the false-positive is very low and the AUC is 1 showing that the algorithm performs very well in distinguishing the normal and malicious traffic.

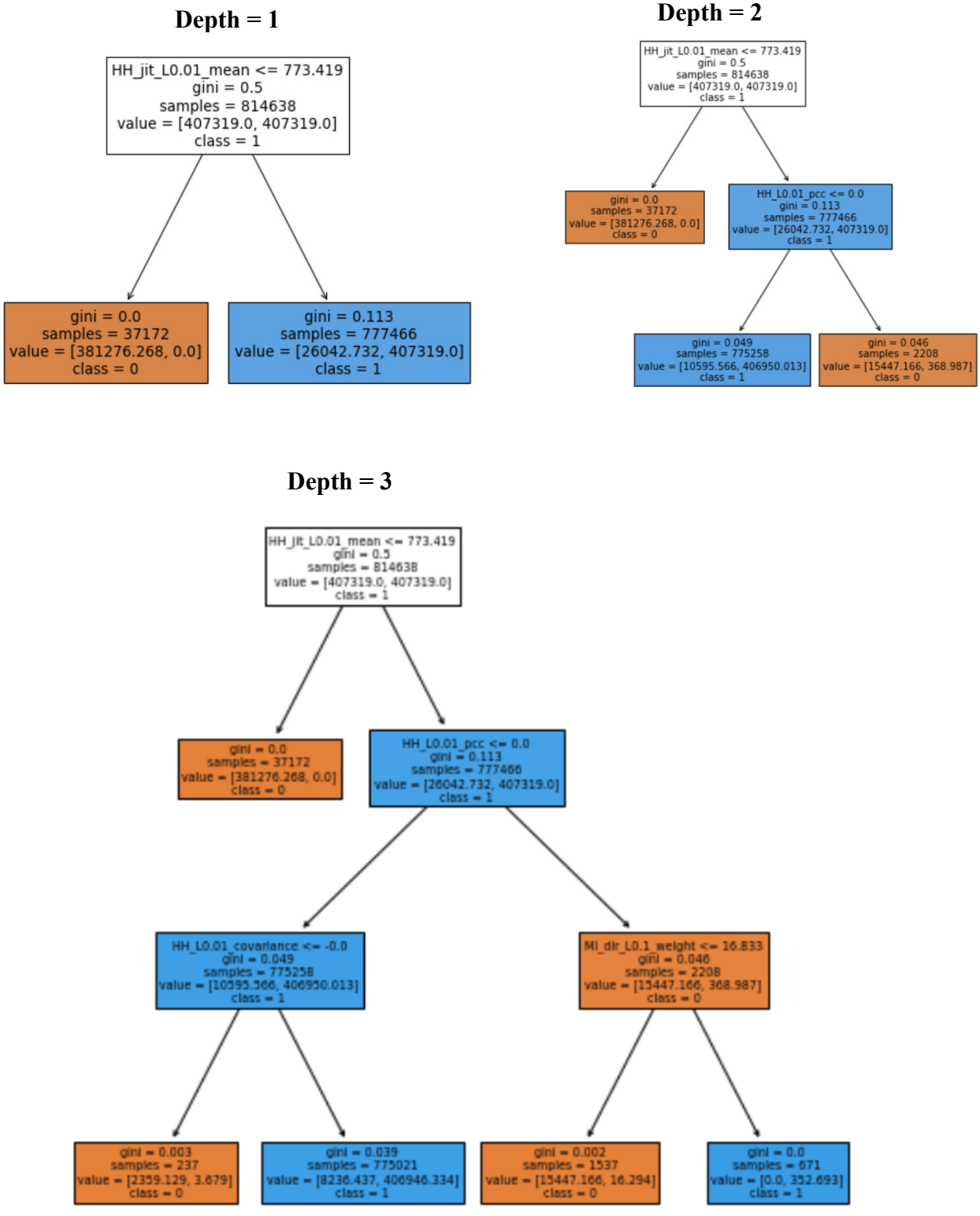


Figure 6.3*: Decision Tree with varying tree depths for Danmini Doorbell dataset*

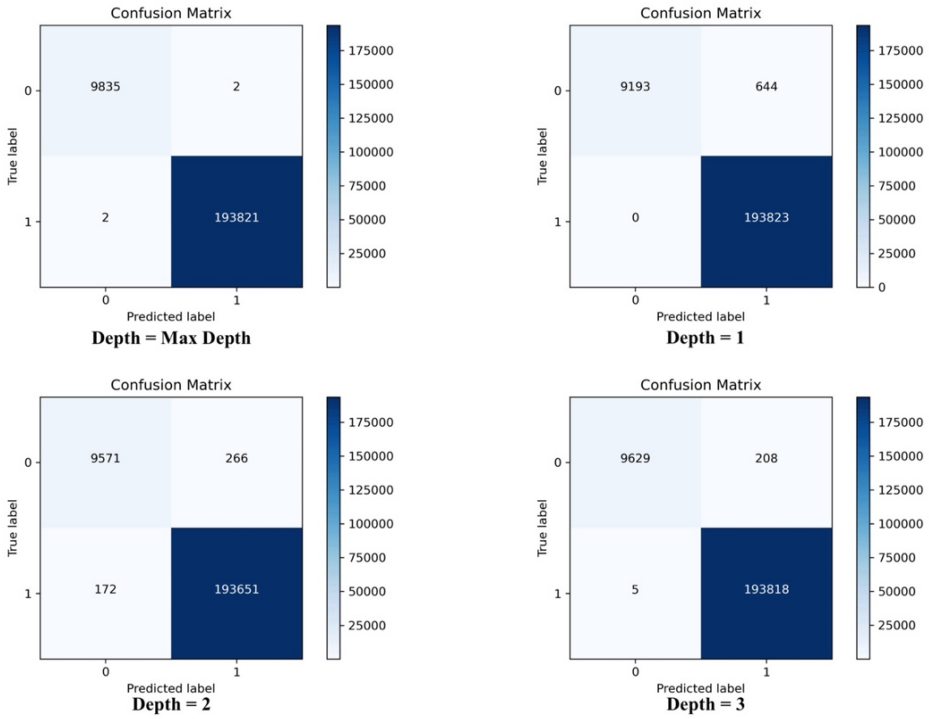


Figure 6.4*: Confusion matrix depicts improved detection performance as the depth of Decision Tree grows for Danmini Doorbell dataset*

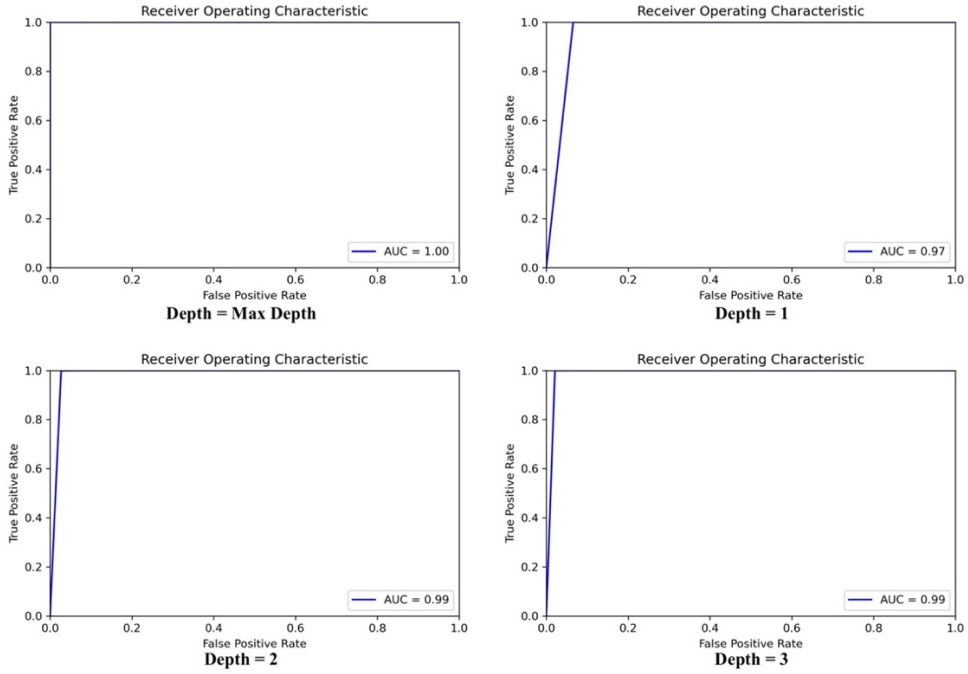


Figure 6.5: *AUC depicts improved detection performance as the depth of Decision Tree grows for Danmini Doorbell dataset*

6.3.2 Multi-class Classification

As mentioned in Section 4, we will be training a new model to differentiate between the Mirai and Bashlite botnet traffic from the benign traffic. Using the multi-class classification decision tree, we are able to conduct the experiment with 99% detection accuracy. This shows that decision tree is capable of differentiating between two types of IoT botnet traffic well. The runtime and detection performance are also measured as shown in Table 6.3. In Table 6.3, we are comparing a decision tree with maximum depth to the Kitsune autoencoder baseline. Figure 6.6 depicts the decision tree of varying depth formed for the multi-class classification of Danmini Doorbell dataset along with the confusion matrix for the same. We can see that in the tree with depth as 1, the decision tree classifies everything into only two classes as it is unable to accommodate the third class due to the limitation of the tree depth. But for depths 2 and 3, we can clearly see that all 3 classes are depicted in the decision tree and the false-positive rate drops with increase in tree depth. Figure 6.7 depicts the confusion matrix for Multi-class Decision Tree Classification for Danmini Doorbell with varying tree depths.

Table 6.3*: Multi-class Decision Tree with max depth depicting runtime and detection performance*

Dataset	Training Time (seconds)	Prediction Time (seconds)	Accuracy (%)	Weighted Precision (%)	Weighted Recall (%)	Weighted F1-Score (%)
Danmini Doorbell	60.525	0.08	99.997	99.997	99.997	99.997
Ecobee Thermostat	84.901	0.065	99.995	99.995	99.995	99.995
Philips B120N10 Baby Monitor	92.068	0.091	99.998	99.998	99.998	99.998
Provision PT737E Security Camera	39.661	0.071	99.996	99.996	99.996	99.996
Provision PT838 Security Camera	44.864	0.069	99.995	99.995	99.995	99.995
SimpleHome XCS71002WHT Security Camera	82.065	0.076	99.991	99.991	99.991	99.991
SimpleHome XCS71003WHT Security Camera	70.49	0.082	99.999	99.999	99.999	99.999
AVERAGE	**67.796**	**0.076**	**99.996**	**99.996**	**99.996**	**99.996**

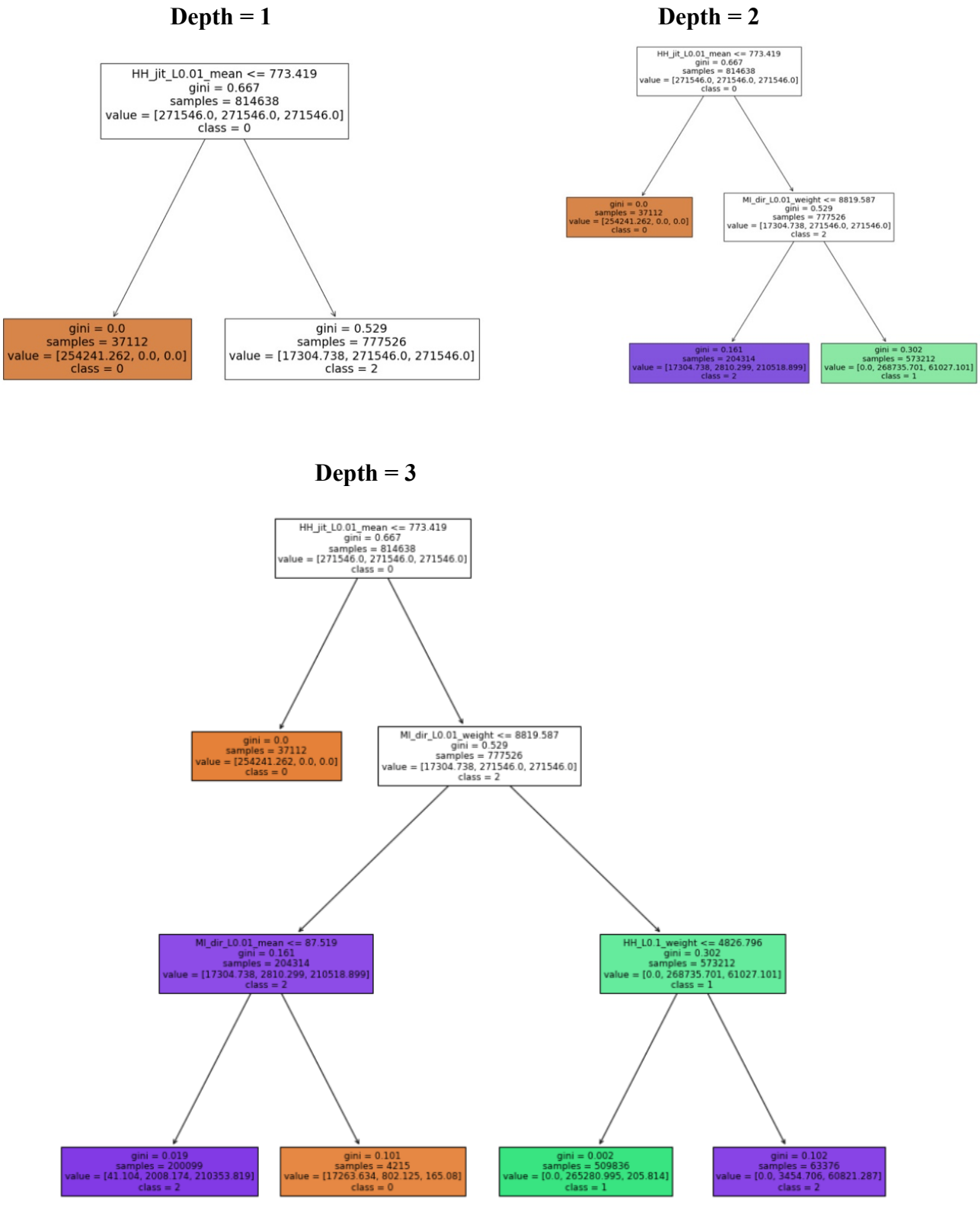


Figure 6.6*: Multi-class Decision Tree Classification for Danmini Doorbell with varying tree depths*

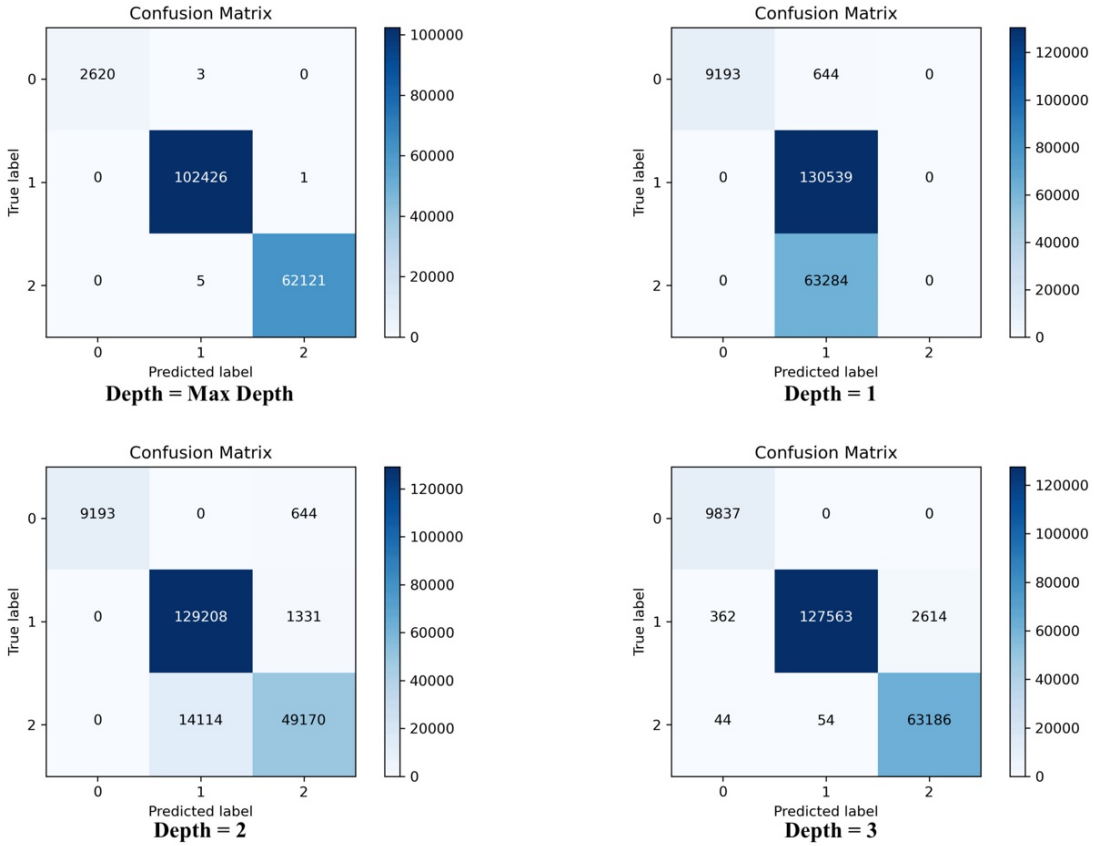


Figure 6.7: *Confusion matrix for Multi-class Decision Tree Classification for Danmini Doorbell with varying tree depths*

6.3.3 Combined datasets

We combine the 9 datasets into one single dataset and run it through a binary decision tree classifier. The botnet traffic from different datasets is mixed together and are tested against our methodology. This experiment is clearly able to distinguish between benign and malicious traffic with an accuracy of over 99% for decision trees with various depths(shown in Figure 6.8). Figures 6.9 and 6.10 depict how the prediction accuracy increases as the tree depth increases. For the decision tree with maximum depth, the training time for 5,650,084 samples is 1024.18 seconds and prediction time for 1,412,522 samples is 4.97 seconds(Table 6.4). We can clearly say that decision trees work well for botnet traffic infecting any type of IoT device.

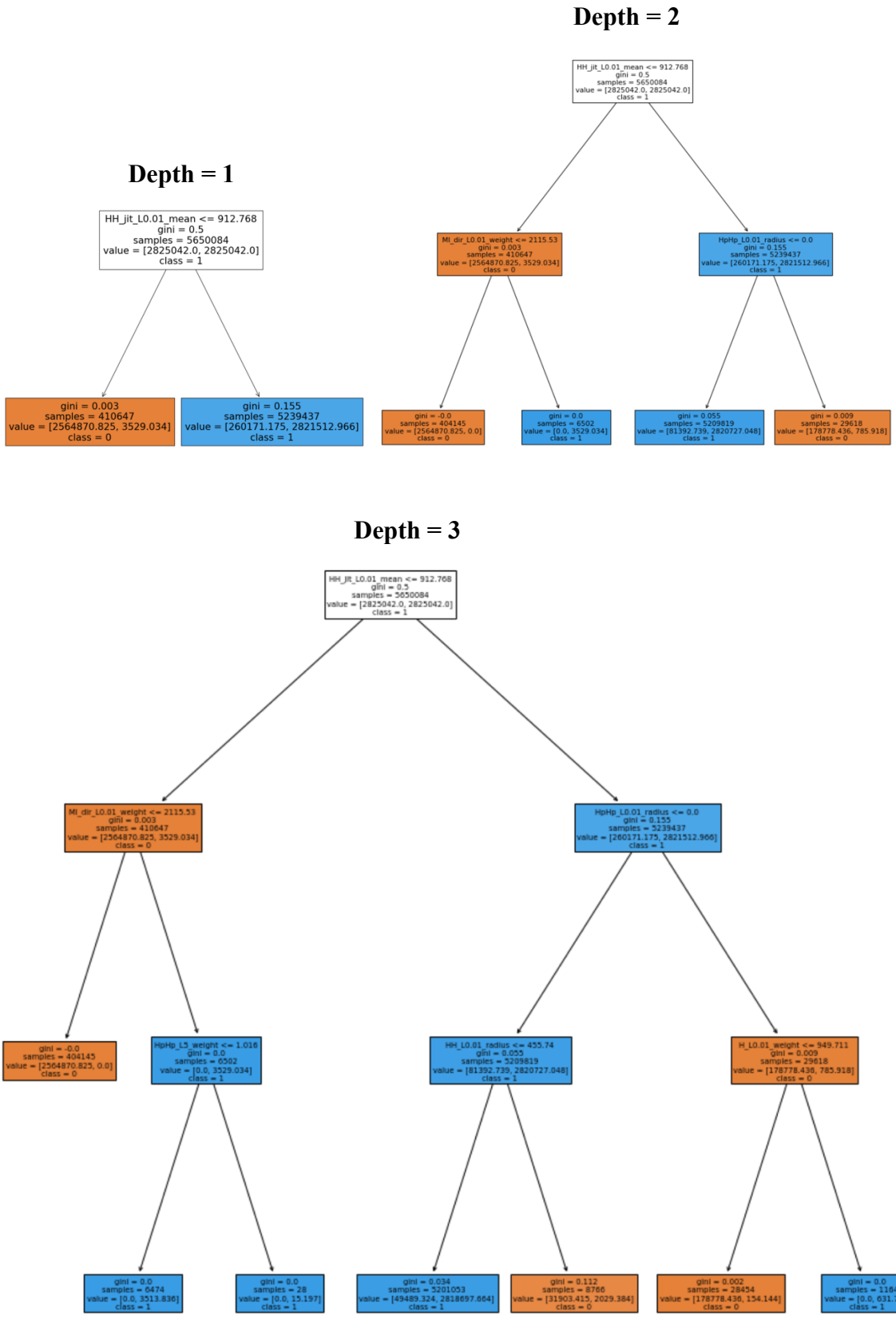


Figure 6.8: *Decision Tree Classification for combined dataset with varying tree depths*

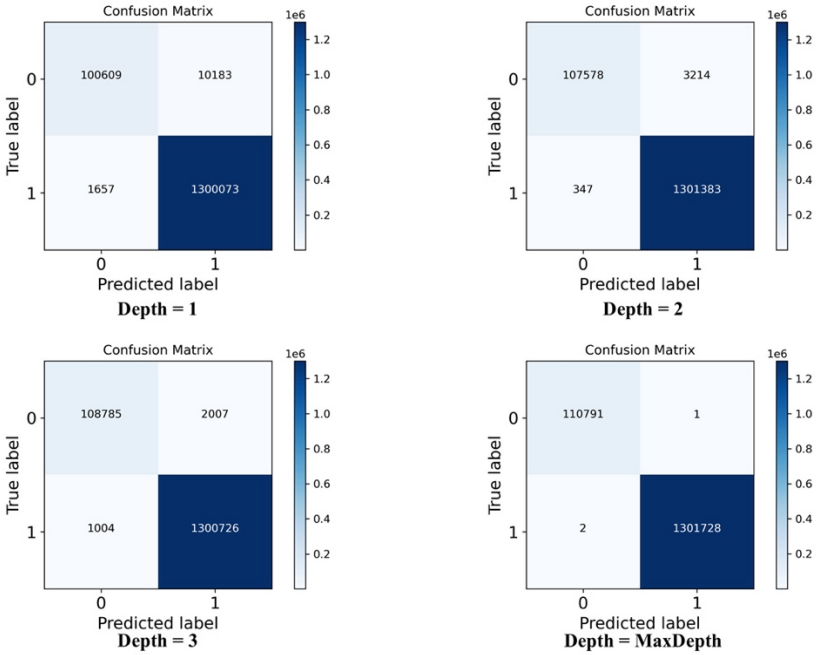


Figure 6.9: *Confusion matrix for Decision Tree Classification for combined dataset with varying tree depths*

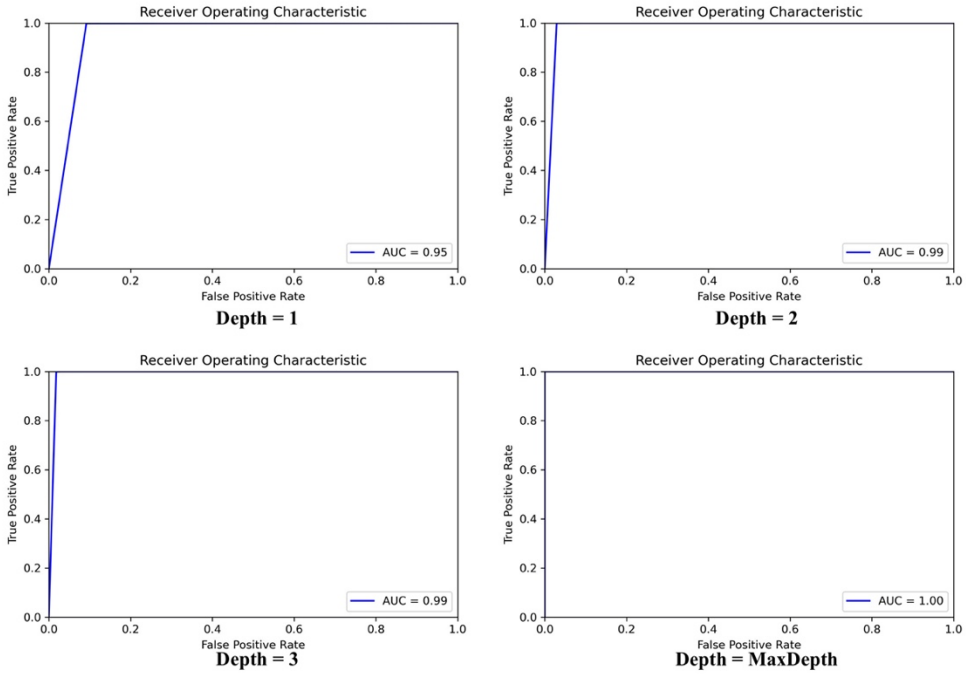


Figure 6.10: *AUC depicts improved detection performance as the depth of Decision Tree grows for combined dataset*

For the combined datasets, we also compare our methodology with other supervised learning methodologies like Logistic Regression[50], Naïve Bayes[51], Random Forest Classifier[52], AdaBoost Classifier[53], XGB Classifier[26], SGD Classifier[54] and ANN[55]. We have not shown the comparison with KNeighbors Classifier[56] as it is an extremely slow learning methodology which calculates the distance(Euclidean or other methods) for each of its predicted samples with its k-nearest neighbors. This would take a lot of runtime for 1 million testing samples and hence would be out of scope for our usage[48][49].Comparing our algorithm with above-mentioned supervised learning algorithms, we can see that our methodology performs extremely well with respect to low computation time while maintain a high accuracy(Table 6.4). Though Random Forest, XGBoost and AdaBoost methodologies also boast of a 100% accuracy, the running times are high showing that these methodologies might not be well suited for IoT edge devices that need to respond quickly. We also compare with an ANN model with 2 hidden layers using 'relu' activation and the output layer with 'sigmoid' activation for binary classification. The model configuration includes 'adam' function for optimization, 'binary_crossentropy' function for loss measurement and the model is trained for 150 epochs. The runtime for this ANN model is more than 4 hours while the prediction time is a comparatively low 20 seconds. The model has high accuracy of 98%, but the false positive rate is very high at 15%. Compared to our methodology, ANN takes more runtime and computation power and would be unsuitable for small IoT devices like doorbells or thermostats.

Table 6.4: *Comparison of Decision Tree methodology performance with other Supervise learning methods*

ML Algorithms	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Decision Tree	1024.186	4.971	100	0.001	100	100	100	100
Logistic Regression	1835.866	3.889	57.766	30.093	58.718	95.754	57.766	72.06
Naïve Bayes	177.89	6.597	99.982	99.619	92.17	92.183	99.982	95.924
Random Forest Classifier	3049.799	11.268	100	0	100	100	100	100
AdaBoost Classifier	12826.466	66.409	100	0.003	100	100	100	100
XGBoost Classifier	3754.705	6.521	100	0.001	100	100	100	100
SGD Classifier	4493.248	4.02	58.843	27.811	59.89	96.133	58.843	73.002
ANN	17604.109	20.63	100	15.041	98.82	98.736	100	99.364

6.4 Important Feature Ranking Evaluation

Using the important feature ranking algorithm as explained in Section 4, we arrive at 9 unique features that are ranked highly across all 9 datasets. When these 9 features are isolated for each dataset and the decision tree is run, we see that the detection performance is higher than 99.9% showing that these features are key in distinguishing benign samples from malicious samples. These features also show that they can distinguish IoT botnet traffic irrespective of the type of IoT device being infected. Table 6.5 shows the training and testing times for datasets with only these 9 features and their detection performance metrics. We can clearly see that the false-positive rate is negligible showing that the algorithm works very well in accurately classifying the samples. Figure 6.11 shows the confusion matrices for all the 9 datasets.

Table 6.5*: Detection and Runtime performance using 9 most important features with Decision Trees*

Dataset	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Danmini Doorbell	5.231	0.014	99.999	0.03	99.998	99.998	99.999	99.999
Ecobee Thermostat	5.749	0.014	99.999	0.038	99.998	99.999	99.999	99.999
Ennio Doorbell	1.42	0.005	100	0.025	99.997	99.997	100	99.998
Philips B120N10 Baby Monitor	9.566	0.021	99.999	0.009	99.998	99.998	99.999	99.999
Provision PT737E Security Camera	2.991	0.012	99.999	0	99.999	100	99.999	99.999
Provision PT838 Security Camera	4.755	0.014	99.998	0.01	99.997	99.999	99.998	99.998
Samsung SNH1011N Webcam	0.946	0.006	100	0	100	100	100	100
SimpleHome XCS71002WHT Security Camera	3.907	0.012	99.999	0	99.999	100	99.999	100
SimpleHome XCS71003WHT Security Camera	4.078	0.011	100	0.026	99.999	99.999	100	100
AVERAGE	**4.294**	**0.012**	**99.999**	**0.015**	**99.998**	**99.999**	**99.999**	**99.999**

The above experiment has shown that these 9 features can run decision tree classifier with average training time of 4.294 seconds and average prediction time of 0.012 seconds, while maintaining an average accuracy of 99.998%. Hence, using datasets filtered with only these 9 features while classifying can help in faster detection on IoT edge devices with low computation power and memory.

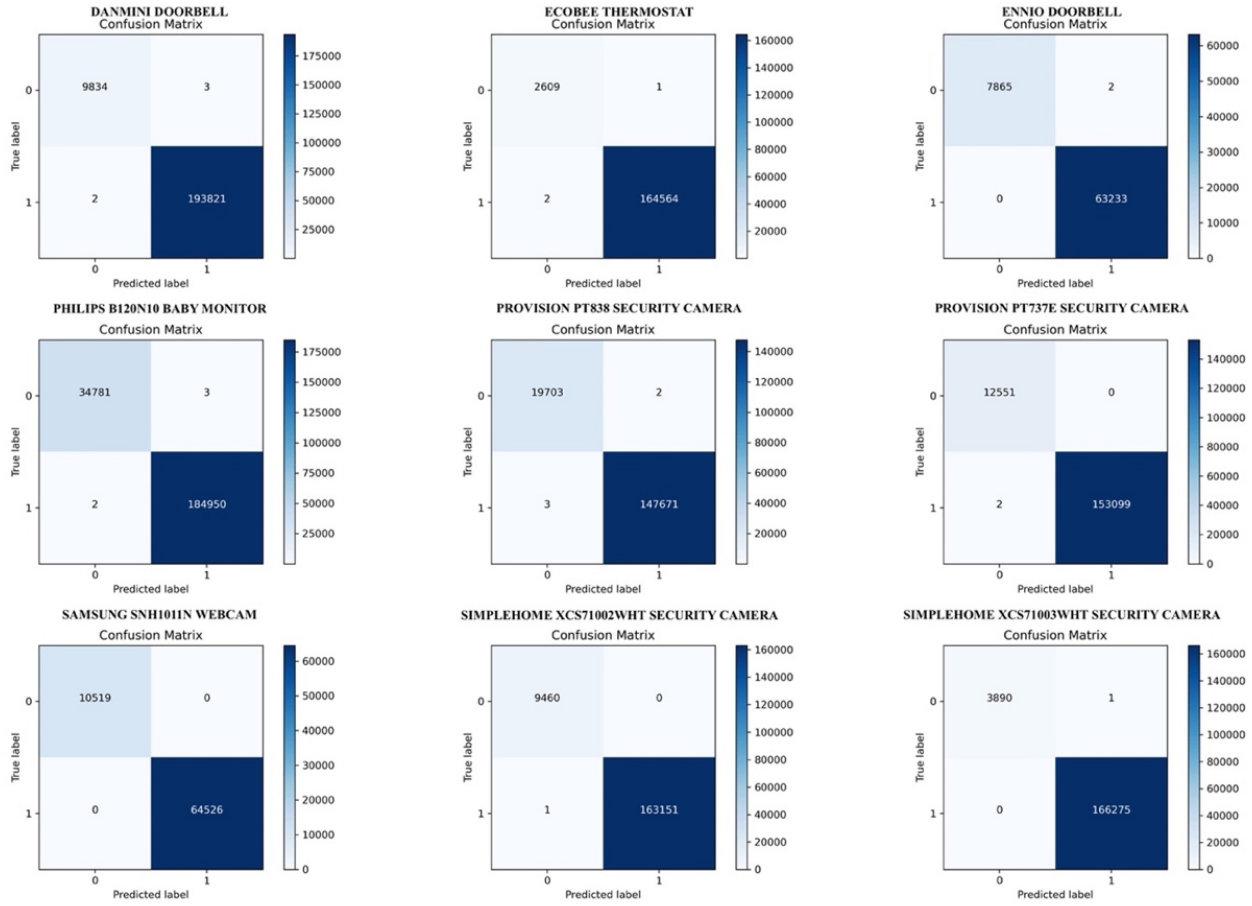
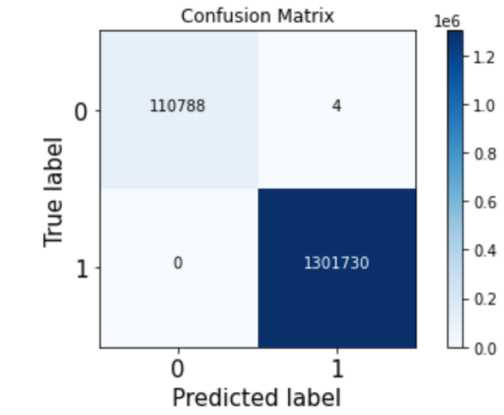


Figure 6.11*: Confusion matrix for datasets using the using 9 most important features with Decision Trees*

For the combined dataset mentioned in Section 6.3.3, we separate the 9 most important features from it and run the decision tree classifier with maximum depth. We find that the training time is 46.93 seconds, and the prediction time is 0.078 seconds which is a great improvement when compared with running the combined dataset with all 115 features(see Section 6.3.3). We also see that the accuracy is maintained the same while the runtime is reduced showing the usefulness of feature importance ranking(Figure 6.12).




```
True Positive Rate =  100.0
True Negetive Rate =  99.996
False Positive Rate =  0.004
False Negetive Rate =  0.0
Accuracy =  100.0
Precision =  100.0
Recall =  100.0
F1-Score =  100.0
```

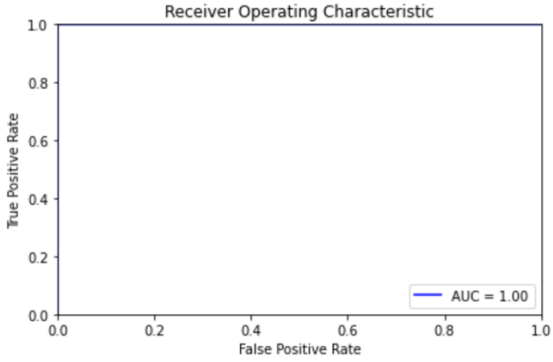


Figure 6.12: *Confusion Matrix with evaluation metrics and AUC curve for combined dataset*

7 CONCLUSION AND FUTURE WORK

Our methodology has been designed to act as an initial line of defense for IoT edge devices having low memory and computation powers. Our decision tree algorithm is a supervised machine learning method that is able to perform classification with high accuracy, low false-positive rate and in minimal amount of time. In this paper, we have discussed the usage of decision trees and the ability to control the tradeoff between performance and computation time using tree depths. We have also evaluated the detection and runtime performance of our methodology against an established baseline in detail which showcases the efficiency of our algorithm. An important contribution of our research is the feature ranking algorithm which produces 9 most important features. These features when segregated from network traffic, can help in identifying normal and malicious traffic with high accuracy and less runtime on devices with low computation power. Hence, we can say that our methodology is efficient and cost-effective especially as a first line of defense in the IoT environment.

Our work establishes the need for better ways to efficiently detect anomalous traffic at the edge of networks and raises more questions. An extension to our work can be in the form of extending the study about the important features found using the Important Feature Ranking algorithm. These important features can be studied in depth to figure out if they can be manipulated by adversaries to make their anomalous traffic look similar to normal traffic. These features can also be used to test with other IoT botnets to see if they are able to maintain the same standard of detection and runtime performances. This can help in (1) better understanding of what network properties make up anomalous traffic and (2) more efficient detection and runtime performances of other anomaly detection algorithms.

REFERENCES

[1] Wikipedia, Internet of Things [Online]
Available: https://en.wikipedia.org/wiki/Internet_of_things

[2] Radware, A Quick History of IoT Botnets [Online]
Available: https://blog.radware.com/uncategorized/2018/03/history-of-iot-botnets/

[3] Jovana Letic, Internet of Things Statistics for 2020 – Taking things apart [Online]
Available: https://dataprot.net/statistics/iot-statistics/

[4] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici and Asaf Shabtai, Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection, Network and Distributed Systems Security Symposium (NDSS) 2018

[5] Ruming Tang, Zheng Yang, Zeyan Li, Weibin Meng, Haixin Wang, Qi Li, Yongqian Sun, Dan Pei∗, Tao Wei‖, Yanfei Xu and Yan Liu, ZeroWall: Detecting Zero-Day Web Attacks through Encoder-Decoder Recurrent Neural Networks, IEEE INFOCOM 2020 - IEEE Conference on Computer Communications

[6] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Dominik Breitenbacher, Asaf Shabtai, and Yuval Elovici, N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders, IEEE Pervasive Computing (Volume: 17 , Issue: 3 , Jul.-Sep. 2018)

[7] Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Mansoor Alam, A Deep Learning Approach for Network Intrusion Detection System, IEEE Transactions on Emerging Topics in Computational Intelligence (Volume: 2 , Issue: 1 , Feb. 2018)

[8] Wikipedia, Autoencoder [Online]
Available: https://en.wikipedia.org/wiki/Autoencoder

[9] Raspberry Pi [Online]
Available: https://www.raspberrypi.org

[10] Sean Bryson, Five Components of IoT Edge Devices [Online]
Available: https://www.cisco.com/c/en/us/solutions/internet-of-things/iot-edge-devices.html

[11] Wikipedia, Decision Tree Learning [Online]
Available: https://en.wikipedia.org/wiki/Decision_tree_learning

[12] Jeff Heaton, Feature Importance in Supervised Training, Predictive Analytics and Futurism Newsletter, Issue 17, April 2018

[13] A. Sundaram, "An introduction to intrusion detection," *Crossroads*, vol. 2, no. 4, pp. 3–7, April 1996.

[14] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, Network Anomaly Detection: Methods, Systems and Tools, IEEE Communications Surveys & Tutorials (Volume: 16, Issue: 1, First Quarter 2014)

[15] Garcia-Teodoro et. al. Anomaly-based network intrusion detection: Techniques, systems and challenges. computers & security, 28(1):18–28, 2009.

[16] Dongqi Han, Zhiliang Wang,Ying Zhong, Wenqi Chen, Jiahai Yang,Shuqiang Lu, Xingang Shi,and Xia Yin, Evaluating and Improving Adversarial Robustness of Machine Learning-Based Network Intrusion Detectors, arXiv:2005.07519

[17] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, HonglinQiao, Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications, WWW 2018: Proceedings of the 2018 World Wide Web Conference

[18] Zeyan Li,Wenxiao Chen, Dan Pei, Robust and Unsupervised KPI Anomaly Detection Based on Conditional Variational Autoencoder, 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)

[19] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng Cristian Lumezanu, Daeki Cho, Haifeng Chen, Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection, ICLR 2018

[20] Rohan Doshi, Noah Apthorpe, Nick Feamster, Machine Learning DDoS Detection for Consumer Internet of Things Devices, 2018 IEEE Security and Privacy Workshops (SPW)

[21] Nathan Shone, Tran Nguyen Ngoc, Vu DinhPhai, Qi Shi, A Deep Learning Approach to Network Intrusion Detection, IEEE Transactions on Emerging Topics in Computational Intelligence (Volume: 2, Issue: 1, Feb. 2018)

[22] Benjamin J. Radford, Bartley D. Richardson and Shawn E. Davis, Sequence Aggregation Rules for Anomaly Detection in Computer Network Traffic, 2018 ASA Symposium on Data Science and Statistics

[23] Hanan Hindy, Robert C. Atkinson, C. Tachtatzis, J. Colin, Ethan Bayne, X. Bellekens, Towards an Effective Zero-Day Attack Detection Using Outlier-Based Deep Learning Techniques, 2020 Computer Science, ArXiV

[24] Miao Xie, Jiankun Hu, Song Han, and Hsiao-Hwa Chen. Scalable hypergrid k-nn-based online anomaly detection in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems, 24(8):1661–1670, 2013.

[25] George Saif, A Guide to Decision Trees for Machine Learning and Data Science [Online] Available:https://towardsdatascience.com/a-guide-to-decision-trees-for-machine-learning-and-data-science-fe2607241956

[26] XGBoost Documentation [Online]
Available: https://xgboost.readthedocs.io/en/latest/

[27]Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Dominik Breitenbacher, Asaf Shabtai and Yuval Elovici, detection_of_IoT_botnet_attacks_N_BaIoT Data Set, March 18,[Online]
Available:https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT#

[28] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Michalis Kallitsis, Damian Menscher, Zakir Durumeric, Deepak Kumar, Chad Seaman, J. Alex Halderman, Luca Invernizzi, Chaz Lever, Zane Ma, Joshua Mason, Nick Sullivan, Kurt Thomas, Yi Zhou, Understanding the Mirai Botnet

[29] K. Angrishi, "Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets," 2017

[30] Wikipedia, Gini Impurity [Online]
Available: https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity

[31] Yusuke Sugiyama and Kunio Goto. Design and implementation of a network emulator using virtual network stack. In 7th International Symposium on Operations Research and Its Applications (ISORA08),
pages 351–358, 2008.

[32] Eric Leblond and Giuseppe Longo. Suricata idps and its interaction with linux kernel.

[33] Borja Merino. Instant Traffic Analysis with Tshark How-to. PacktPublishing Ltd, 2013.

[34] The Packet++ Project, GitHub [Online]
Available: https://github.com/seladb/PcapPlusPlus

[35] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference , pages 413–422. IEEE, 2008.

[36] Douglas Reynolds. Gaussian mixture models. Encyclopedia of biometrics, pages 827–832, 2015.

[37] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. In Proceedings of the ACM/IEEE international conference on Human-robot interaction, pages 255–262.ACM, 2007.

[38] Yisroel Mirsky, Tal Halpern, Rishabh Upadhyay, Sivan Toledo, and Yuval Elovici. Enhanced situation space mining for data streams. In Proceedings of the Symposium on Applied Computing, pages 842–849.ACM, 2017.

[39] KDD Cup 1999 Data, The Third International Knowledge Discovery and Data Mining Tools Competition

[40] NSL-KDD dataset, Canadian Institute for Cybersecurity

[41] Intrusion Detection Evaluation Dataset (CIC-IDS2017), Canadian Institute for Cybersecurity

[42] Mark D. Wilkinson, Michel Dumontier, Barend Mons, The FAIR Guiding Principles for scientific data management and stewardship, Scientific Data 3

[43] B. Xue, M. Zhang, W. N. Browne and X. Yao, "A Survey on Evolutionary Computation Approaches to Feature Selection," in IEEE Transactions on Evolutionary Computation, vol. 20, no. 4, pp. 606-626, Aug. 2016

[44] Tang, J., Alelyani, S., & Liu, H. (2014). Feature selection for classification: A review. In Data Classification: Algorithms and Applications (pp. 37-64). CRC Press. [Online] Available: https://doi.org/10.1201/b17320

[45] M. R. Sikonja and I. Kononenko. Theoretical and empirical analysis of Relief and ReliefF. Machine Learning, 53:23–69, 2003

[46] R.O. Duda, P.E. Hart, and D.G. Stork. Pattern Classification. John Wiley & Sons, New York, 2nd edition, 2001.

[47] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 1226–1238, 2005.

[48] Stack Overflow, KNN classifier taking too much time even on gpu [Online] Available: https://stackoverflow.com/questions/51693501/knn-classifier-taking-too-much-time-even-on-gpu

[49] Stack Overflow, Why does test takes longer than training? [Online] Available: https://stackoverflow.com/questions/53133458/why-does-test-takes-longer-than-training

[50] Wikipedia, Logistic Regression [Online] Available: https://en.wikipedia.org/wiki/Logistic_regression

[51] Wikipedia, Naïve Bayes Classifier [Online] Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[52] Wikipedia, Random Forest Classifier [Online] Available: https://en.wikipedia.org/wiki/Random_forest

[53] Wikipedia, AdaBoost Classifier [Online]
Available: https://en.wikipedia.org/wiki/AdaBoost

[54] Scikit Learn, SGD Classifier [Online] Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[55] Wikipedia, ANN [Online]
Available: https://en.wikipedia.org/wiki/Artificial_neural_network

[56] Wikipedia, k-nearest Neighbors Algorithm [Online]
Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm