Contents lists available at GrowingScience

International Journal of Data and Network Science

homepage: www.GrowingScience.com/ijds

Botnet attacks detection in IoT environment using machine learning techniques

Mousa AL-Akhras[a], Abdulmajeed Alshunaybir[b], Hani Omar[c] and Samah Alhazmi[b*]

[a]*King Abdullah II School for Information Technology, The University of Jordan, Amman 11942, Jordan*
[b]*Computer Science Department, College of Computing and Informatics, Saudi Electronic University, Riyadh 11673, Saudi Arabia*
[c]*Faculty of Information Technology, Applied Science Private University, Amman, Jordan. MEU Research Unit, Middle East University, Amman, Jordan*

C H R O N I C L E

Article history:
Received: April 10, 2023
Received in revised format: May 25, 2023
Accepted: July 29, 2023
Available online: July 29, 2023

Keywords:
IoT
Botnet
DDoS
Mirai
Bashlite
IDS
Machine Learning
Noise

A B S T R A C T

IoT devices with weak security designs are a serious threat to organizations. They are the building blocks of Botnets, the platforms that launch organized attacks that are capable of shutting down an entire infrastructure. Researchers have been developing IDS solutions that can counter such threats, often by employing innovation from other disciplines like artificial intelligence and machine learning. One of the issues that may be encountered when machine learning is used is dataset purity. Since they are not captured from perfect environments, datasets may contain data that could affect the machine learning process, negatively. Algorithms already exist for such problems. Repeated Edited Nearest Neighbor (RENN), Encoding Length (Explore), and Decremental Reduction Optimization Procedure 5 (DROP5) algorithm can filter noises out of datasets. They also provide other benefits such as instance reduction which could help reduce larger Botnet datasets, without sacrificing their quality. Three datasets were chosen in this study to construct an IDS: IoTID20, N-BaIoT and MedBIoT. The filtering algorithms, RENN, Explore, and DROP5 were used on them to filter noise and reduce instances. Noise was also injected and filtered again to assess the resilience of these filters. Then feature optimizations were used to shrink the dataset features. Finally, machine learning was applied on the processed dataset and the resulting IDS was evaluated with the standard supervised learning metrics: Accuracy, Precision, Recall, Specificity, F-Score and G-Mean. Results showed that RENN and DROP5 filtering delivered excellent results. DROP5, in particular, managed to reduce the dataset substantially without sacrificing accuracy. However, when noise got injected, the DROP5 accuracy went down and could not keep up. Of the three dataset, N-BaIoT delivers the best accuracy overall across the learning techniques.

1. Introduction

With the Internet moving away from being a luxury to a necessity, more devices and appliances are turning into IoT devices. Having the property of being connected made these devices popular amongst their adopters. It allowed their users to control and communicate with them with ease, without the need to be physically present. But such simplicity must have a price, though. Those devices nowadays are increasingly becoming major contributors to a variety of cyberattacks. In the last few years, several Botnet-based Distributed Denial of Service (DDoS) attacks have been seen hitting major services around the world. The DDoS attack on Dyn (Feingold, 2016), which rendered many high-profile websites such as Twitter and GitHub and CNN inaccessible, was caused by IoT devices hijacked by Botnets. Kaspersky, the company behind several security-focused suites, claimed a jump from 12 million in 2018 to more than 100 million attacks in 2019, on their Botnet honeypots

* Corresponding author.
E-mail address: s.alhazmi@seu.edu.sa (S. Alhazmi)

ISSN 2561-8156 (Online) - ISSN 2561-8148 (Print)

doi: 10.5267/j.ijdns.2023.7.021

(Demeter et al., 2019). Even companies that provide protection against DDoS attacks, like ProxyPipe, were not far from being a target too (Krebs, 2017). Given that the number of connected IoT devices is projected to reach 75 billion by 2025 (Statista, 2018), which is already threefold bigger than 2019's 26 billion, Botnets are not expected to slow down in the foreseeable future.

Having that happening to IoT devices is quite reasonable. The focus on the cost effectiveness side of IoT devices has always led to poor security designs and vulnerable software that are rarely supported (Mafarja et al., 2020). The lack of resources also prevented these devices from integrating sophisticated security solutions. Because of these issues, IoT devices have turned out to be the gift that Botnet and DDoS attackers have been waiting for. i.e., a swarm of devices that can be hijacked with ease and used to commit DDoS, spamming and data theft attacks. Knowing how stiff the competition is in that industry, vendors are not likely to solve the security issues too. So, cutting costs to maintain profits will probably continue to hammer the security of IoT devices.

Researchers have been busy exploring and proposing ways to detect Botnet attacks with Intrusion Detection Systems (IDS). An IDS is security countermeasures that can detect malicious activities either via their signature or through anomalies in their behavior. With the number of innovations going on in the fields of Artificial Intelligence (AI) and Machine Learning (ML), IDSs future only looks to be heavily reliant on them.

Studies conducted on cybersecurity explored a variety of solutions for IDS. They used different ML techniques and optimizations to achieve that goal. However, IoT devices tend to be diverse, and limiting experiments on a single environment or a couple of devices might miss some opportunities. Therefore, increasing the number of datasets increases diversity. Furthermore, the "No Free Lunch" theorem by Wolpert and Macready (1997) suggests that an algorithm cannot solve all problems. Thus, in this report, we are opting to experiment with multiple datasets and ML techniques simultaneously to address that gap. Another concern that was pushed towards writing this report is datasets quality and size. When building an IDS, the quality or purity of the dataset plays a role in the quality of the produced IDS. Outliers and extreme values could affect the performance of IDS, and thus filtering such data should not be over-looked. Some of the filtering algorithms will be experimented with such as Decremental Reduction Optimization Procedure 5 (DROP5) and Encoding Length Grow (ELGrow) can also help in reducing dataset's sizes. Many existing Botnet datasets nowadays are huge and cutting portions of them by random sampling may remove important data too.

This report will contribute to the existing knowledge of Botnet by proposing an IDS solution. The followings list the pieces that will be used to achieve that goal:

- Three pre-made datasets, N-BaIoT, IoTID20, and MedBIoT will be used to train the IDS model. That choice was made as an effort to emulate the diversity of IoT devices in real world environments, as much as possible.
- Noise and instance filtering will be attempted on those datasets before they get fed into the trainer. Removing noise and pruning dataset's instances could potentially enhance the performance of the trained IDS model. Some of the algorithms that fit the purpose are Repeated Edited Nearest Neighbor (RENN), Encoding Length (Explore), and Decremental Reduction Optimization Procedure 5 (DROP5).
- To further test the above filtering algorithms, some artificial noises will be injected in the datasets. Afterward, the filtering process will be employed again to filter these injected noises. This technique tests the resilience of such algorithms towards unseen noises.
- Feature selection optimizations (FSO) is another entirely different technique for optimizing datasets. They remove features or columns from the dataset instead of instances. Thus, less time spent by ML techniques on the optimized datasets. For that purpose, the report will experiment with three nature inspired algorithms: Particle Swarm Optimization (PSO), Grey Wolf Optimizer (GWO), Multi-Verse Optimizer (MVO).
- Once the filtering and feature selection is applied to datasets, three supervised ML techniques will be used to train them and produce the IDS classifier. The techniques to be used for that are Neural Network (NN), Decision Tree (DT), and Random Forest (RF).
- Finally, the performance of produced IDS models will be evaluated with metrics derived from confusion matrix: Accuracy, Precision, Recall, Specificity, in addition to harmonic mean (F-Score) and geometric mean (G-Mean).

The remainder of this report will be organized as follows. Section 2 provides backgrounds on IoT, Botnets, IDS and ML. Section 3 will be a literature review of previous works that incorporated ML for Botnet attack detection. A methodology along with the datasets and algorithms is proposed in section 4. Section 5 demonstrates how the work can be reproduced again. Afterward, in section 6, the experiment results are revealed and discussed. Finally, section 7 concludes the report with important findings and possible future work in Botnet and IDS.

2. Internet of Things (IoT)

IoT are devices or networks of devices that have the property of being connected. They are gaining popular and becoming ubiquitous in homes and businesses due to their ability to automate and add the smart element to appliances that typically was not basic, like a washing machine or a security camera, with Internet or network capability adding on them. In addition, that makes them heterogeneous. They can be a big washing machine that sends notification via the Internet when it has done with laundry or they could be a thermostat sensor that can fit into someone's pocket. Figure 1 portrays few IoT devices connected over the Internet.

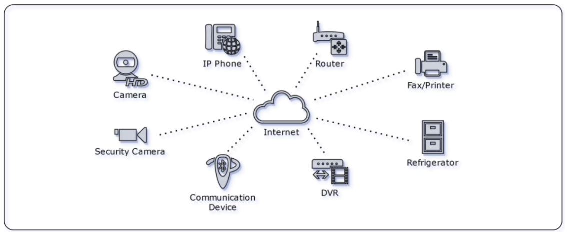
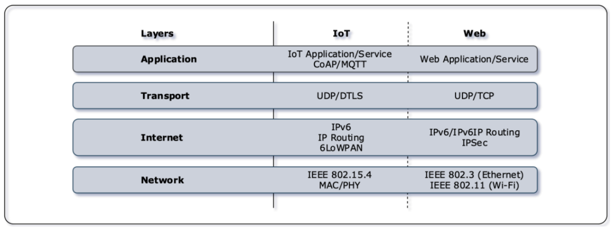


Fig. 1. IoT devices connected over the Internet



Layers	IoT	Web
Application	IoT Application/Service CoAP/MQTT	Web Application/Service
Transport	UDP/DTLS	UDP/TCP
Internet	IPv6 IP Routing 6LoWPAN	IPv6/IPv6IP Routing IPSec
Network	IEEE 802.15.4 MAC/PHY	IEEE 802.3 (Ethernet) IEEE 802.11 (Wi-Fi)

Fig. 2. Network IoT Stack Compared to Web

IoT devices have constrained resources (Mafarja et al., 2020). It is not unusual to find them extremely limited in their processing capabilities or optimized for energy efficiency. These characteristics can be useful in cases where IoT devices are used for gathering or capturing Information in situations where power availability is scarce. Like the case with security cameras connected to power generators. Mobility is also another benefit of constrained resources in IoT devices. A battery-powered heartbeat sensor, wrapped around a patient arm, would be impractical without such optimizations Besides their main functionalities, which can be measuring temperature or sensing movement, IoT devices have communication components (Mafarja et al., 2020). A sensor would, for instance, capture data in real time and then send it via a communication medium, such as Wi-Fi, Bluetooth, or Mobile Networks, to a central place, where data from other devices is collected too. The communication protocols used IoT devices might look like the one used in a typical network environment. i.e., they can use common protocols such as TCP, UDP and IP to deliver data. However, other protocols that are more fitting to IoT exist too. These include 6LoWPAN, CoAP and MQTT protocols (Ullah & Mahmoud, 2020a). For a comparison, Fig. 2 shows the communication stack of IoT and Web servers. The software that powers IoT devices are predominantly Linux based (Al Shorman, Faris, Aljarah, 2020). IoT is gaining some benefits from that situation. Many chipset makers, like Samsung and Broadcom, contribute codes to Linux repositories. That makes it easy for IoT makers because such hardware components would then work out-of-the-box. Being on Linux also gives the IoT vendors access to software that are typically made for larger computing devices. However, there are consequences for running Linux software too. IoT devices may inherit bugs and vulnerabilities from that platform. Bashlite is a Botnet malware that targets IoT devices with Linux OS preinstalled (Kim et al., 2020), for instance.

2.1. Botnet

DDoS are still one of the main attacks that keep disrupting organizations in different industries. The attack takes place when a group of hijacked hosts, or zombies, send a flood of requests, simultaneously, to a victim's server. These requests, which can be one form of ping of death or SYN floods, cause the services or network stacks running in the victim's server to stop responding to legit requests. Despite being effective, DDoS attacks are not easy to set up. That is due to recruiting zombies requiring bypassing an IDS to begin with. And nowadays desktops and mobile devices, such as Microsoft Windows and Apple iOS, are equipped with relatively good security designs. Which is something that cannot be said about IoT devices (Kim et al., 2020; Meidan et al., 2018). Mirai and Bashlite (Kim et al., 2020) are two of the most popular malwares used in Botnets infections. Botnets have a process that involves searching for IoT devices, then infecting them to create a network of robots or zombies. After being taken over, a remote server acting as a command-and-control center (C&C) orders them to launch DDoS attacks on target organizations over the Internet, as illustrated in Fig. 3.

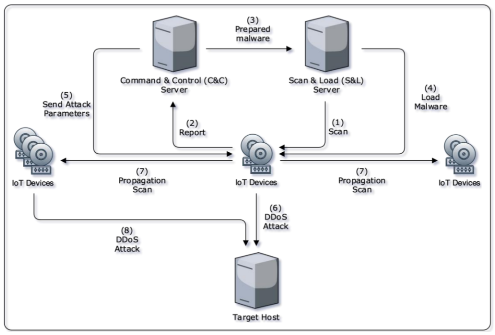


Fig. 3. Miri Attack Steps

The following list the steps that Mirai goes through to launch the DDoS attack on target hosts (McDermott, Majdani, Petrovski, 2018).

- Mirai takes advantage of security holes in IoT devices, such as the weak Telnet default password by instructing the scan and load (S&L) server to scan over the internet for IoT devices with vulnerable Telnet service. When devices are found, the S&L server attempts to brute-force login into the devices with a predefined number of hardcoded passwords. If successful, the working credentials and the IoT information is reported back to the C&C server.

- The C&C server uses the gathered information to prepare for the infection. This instructs the S&L server to load the executable to do the DDoS attack into the IoT device. Transferring such files can be as simple as using the TFTP service in these IoT devices. The executable by now is loaded in the IoT device's memory, and it declares it as a new addition to the Botnet.

- The C&C server sends the attack instruction to the infected IoT devices. The instruction may include the target host, launch time and duration. The attacks can be any of the typical Denial of Service (DoS) attacks, such as UDP, SYN or ACK floods. When the time ticks, the attack is launched.

- Miri botnet has self-propagation properties. Therefore, the infected IoT devices themselves also begin scanning for other vulnerable IoT devices and report them to the C&C server. And the process continues.

2.2. Intrusion Detection System (IDS)

An intrusion is a non-authorized access to a system that may compromise any part of the security triad, confidentiality, integrity or availability (CIA) (Khraisat et al., 2019). To protect against intrusions, IDS solutions are deployed into infrastructures where IoT devices exist. There are various designs and approaches for such solutions. Based on detection methods, these designs can be categorized into two major types: signature-based or an anomaly-based IDS (Khraisat et al., 2019). Fig. 4 compares the components of each type.

- Signature-based IDS: The IDS system looks for patterns that match predefined entry in a database of known malicious activities. For instance, it looks for sequences in a log file that may identify with an entry in the IDS database. This method of detection gave it the other name: misuse IDS. That database is built over time, and it covers malicious activity that has been encountered and accumulated over years. Since this type of IDS already has knowledge about attack signatures, they will produce superior accuracy if the attack already exists in that database. Which also brings attention to a problem this type has: zero-day attacks will not be detected if that database is out-of-date.

- Anomaly-based IDS. This type of IDS builds a pattern or model of normal behavior. Then it looks for a deviation from that model. When such deviation is found, the activity is flagged as a possible attack. Since this type relies on a change of behavior rather than on a database of attack signatures, it is possible for them to catch zero-day attacks. However, that also means that a normal activity can be flagged as an attack even if they were not when their behaviors change for any legitimate reason. False positive rate is one of the important measures for IDS built this way.

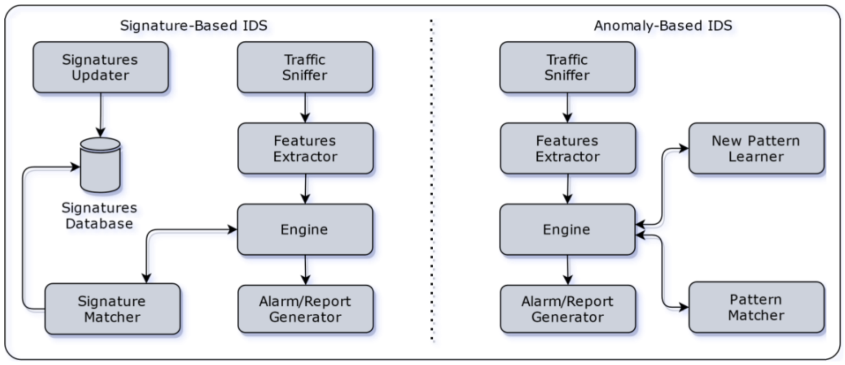


Fig. 4. Components of IDS (Signature-Based Left and Anomaly-Based Right)

The location or deployment scenario can be used, as will, to categorize IDS into two types. Host-based and Network-based IDS (Khraisat et al., 2019):

- Host-based IDS. Typically installed in the host's systems. Because of that placement, this type of IDS gets access to underlying data in the hosts Operating System (OS), such as logs and audits. Being this close the host also allows the IDS to inspect tunneled or encrypted data that has been just decrypted by the host.

- Network-based IDS. Attach to a network environment where devices are exchanging data. An advantage that could be gained from this setup is the possibility of detecting attacks before they reach every other host in the network. Another advantage is that this type can overcome the limited computation capabilities in some hosts, like IoT devices. However, because it is installed outside the host, it has no access to low-level system processes. Therefore, tunneled traffic might not be as inspected in this approach.

Nowadays, there are designs incorporating all the above types into a hybrid one. A signature-based ID can be combined with anomaly-based IDS to take advantage of each at the same time. In addition, an IDS can be deployed both in the network and in each host for further security insight. The designs are a matter of security requirements and available resources. Especially in an IoT environment.

2.3. Machine Learning (ML)

People make every day's decisions based on knowledge but that cannot be applied in complicated scenarios where such knowledge is hard to comprehend by people. Computers solve this problem with the help of ML techniques. These techniques can be categorized into four types. Supervised, unsupervised, Simisupervised (Khraisat et al., 2019) and reinforced ML. Fig. 5 illustrates the difference between the supervised and unsupervised ML.

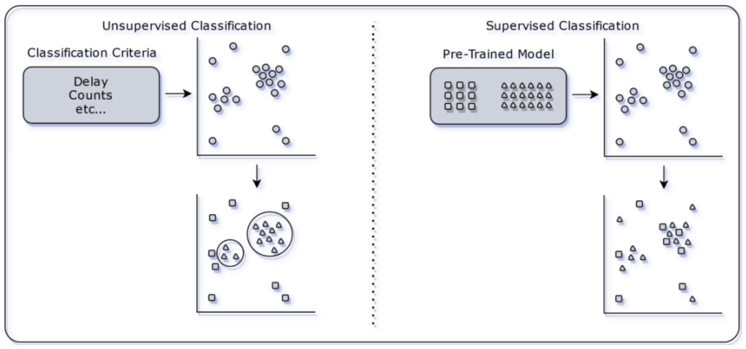


Fig. 5. Unsupervised ML Left vs. Supervised ML Right

- Supervised: in this technique, ML learns from data that has been already labeled or classified. The datasets in supervised ML are split usually into two sets: training and testing sets. The training set is used to build the model that will do the predictions. The testing set on the other hand is used to validate the choices made on the built models. Supervised ML can be applied using techniques such as Support Vector Machine (SVM), Decision Tree (DT), Neural Network (NN), Random Forest (RF), and Naïve Bayes (NB).

- Unsupervised: in this technique, the dataset that is required for building the prediction model does not need to be labeled. Instead, the classifier group points into classes based on criteria or relatedness amongst them. K-Mean is the most prominent classifier for unsupervised ML. In K-Mean, the user supplies the required number of classes or clusters Which the classifier would use to group the points based on their means.
- Semi-supervised: Combine the benefit of supervised and unsupervised ML. A small portion of labeled data, for example, can enhance the performance of an unsupervised classifier and cut the time needed for classification. Expectation Maximization and Semi-Supervised SVM are two examples of this type of ML.
- Reinforcement: Rather than needing labels, like the case with supervised ML, reinforcement takes an approach of rewarding selections in a cumulative way. This is done by the classifier through a continuous process of exploration and exploitation. During the exploration, unknown data is explored for possible knowledge. While in exploitation, the found data is knowledge being exploited.

3. Literature Review

Researchers have studied IDS and Botnets and their malware for years. In general, the goal has been building a performant IDS model that can detect attacks, using varieties of approaches and classification techniques. The following review of some of these works. Khraisat et al. (2019) surveyed techniques, dataset, and challenges IDS. These systems were broadly categorized into signature-based and anomaly-based IDS. They pointed to the major difference between the two as that signature-based can detect well-known attacks, whereas anomaly is able to detect zero-day attacks. Anthi et al. (2018) considered another difference among IDS. In particular, the implementation of such systems can take the host-based or network-based approach. A host-based IDS is installed in the host system that needs protection. If it is network-based, it is independent of the hosts. They also mentioned another hybrid approach that combines host-based and network-based IDS. Drawing conclusions regarding attacks can be done either via deep packet inspection or network flow analysis, according to Koroniotis et al. (2018). The difference between the two is that the deep packet inspection examines each packet's content, which may reveal rich information about each packet. Network flow analysis, on the other hand, inspect the traffic in groups or flows aggregated by related packets that have shared source IP address and port, for instance. The authors believe that the former yields deep and rich information about packets, while the latter has less overhead. The network flow demonstrated excellent accuracy in research done Ullah & Mahmoud (2020a) where only packet statistics, not contents, are inspected for anomaly patterns.

The ML techniques: supervised, unsupervised, semi-supervised and reinforced have all been utilized in building IDS. Xiao et al. (2018) confirmed, in their review, that IoT-based IDS has been using these too. They also went through the usual issues associated with IoT such as computation and communication. The computation issue has been one of the motivations that pushed towards network-based IDS, over host-based. In their work, Meidan et al. (2018) affirmed that notion by building a network-based IDS that can detect subsequent malicious activities of devices that have been infected with a Botnet malware. Traffic was extracted from nine devices and compiled into a new dataset, N-BaIoT. Neural Networks based algorithms, Autoencoders, were applied on the dataset to perform both classification and optimization. The results were then compared with Single Class SVM, Local Outlier Factor (LOF) and Isolation Forest (IF) classifiers. Results showed that autoencoders produced superior false positive 13 rates (FPR). The study also found that the device that showed the best FPR, is the one that produced the largest traffic and suggested more instances is likely to produce better ML models. Reducing the number of features could positively affect the performance of IDS. Habib et al. (2020) attempted to counter the weak processing in IoT devices by suggesting an IDSs based on evolutionary multi-objective Particle Swarm Optimizer (PSO) for IoT networks. By treating the feature selection as a multiobjective problem, the number of features can be minimized along with classification error rate. The results achieved that, but the researchers warned of computational costs associated with such algorithms. Kumar and Lim (2019) did not use features optimizations algorithms in their proposed solution which they called EDIMA. Instead, the important features were selected by hand. Unlike Meidan et al. (2018) proposal, which aimed at detecting attacks after infection, EDIMA went for detecting Botnets in earlier stages. i.e., during scanning and infection phases. k-NN and RF classifiers were applied during the training and testing phases. Supervised ML has been used extensively for intrusion detection systems (Beigi et al., 2014; Doshi et al., 2018; Koroniotis et al., 2018). The reliance on labeled dataset has been the main reason for supervised ML use on intrusion detection. Taking into consideration that most attacks are already known, supervised ML would perform well in detecting them, according to the researchers.

Using multi-phase classification by applying more than one classification on a problem is not unheard of when building IDS. Mohamed et al. (2018) built such a system by first applying a RF classifier on the initial detection phase. Then, the NN categorized the attack type. The attack detection did well. Although the authors found that the second phase lacked accuracy. Rathore and Park (2018) created a distributed solution for attack detection for IoT fog network. The authors attributed IoT devices requirements, such as low latency, scalability and resource constraints, on top to the need for labeled dataset to the failed attempts of supervised-based IDS. So, they instead recommended a semi-supervised ML, which combines both supervised and unsupervised techniques. The attack detection suggested combined techniques of semisupervised Fuzzy C-Means and Extreme Learning Machine for the classification. They found that this system gave better performance when compared against many other ML such as SVM, Logistic Regression (LR), k-NN, DT, RF, and BN.

The supervised ML demand for labeled datasets, and their low performance in detecting unseen or new Botnets pushed for anomaly-based IDS that uses unsupervised ML. Al Shorman et al. (2020) demonstrate that with their IDS that detect Botnets launched from compromised IoT devices. To achieve that, 14 unsupervised One Class SVM classifiers were used along with Grey Wolf Optimization (GWO). In addition to the feature's selection reduction, the optimizer also enabled improved parameters selection for the classifier. Which in turn resulted in better performance in both true and false positive rates. The performance of detection systems is directly affected by the quality and size of datasets. This has been suggested by Meidan et al. (2018) previously when they observed better performance in a dataset with the largest number of instances. It was also hinted at by Doshi et al. (2018) too. Al Shorman et al. (2020) applied cleaning, integration, and normalization on datasets to produce a newer version that they called NN-BaIoT, out of the original N-BaIoT. The cleaning part removed duplicate features from instances. The integration part integrated a couple of devices together, which resulted in lower numbers of generated models. Normalization turns features values with variance into values that are in the range zero to one. Noise injection and removal from datasets is an important technique to address in ML. It helps in creating intrusion detection solutions that are ready for real-world use. In their intrusion detection paper, Al-Gethami et al. (2021) studied the noise influence and traced noise to outliers and extreme values. In one of the experiments, noise filtering algorithms were applied in raw datasets to remove the outliers and extreme values. In general, the filtering gave better accuracy, though, results varied between different ML classifiers and datasets. In another experiment, noise was intentionally injected to the dataset, in levels from 5% to 30%, by modifying the labeling of the instances. They concluded that noise injection also gave varying results. Labonne (2020) experimented with several balancing techniques for their anomaly IDS. ENN and RENN were used to reduce the number of instances in an imbalanced dataset. Both algorithms produced fluctuating results depending on the attack category. Ayad et al., (2019) employed ENN and SMOTE to balance their dataset. They used SMOTE to over sample the minority class. Then they performed down sampling with ENN, to remove noisy instances. This report intends to incorporate algorithms that can remove noise and reduce instances, regardless of dataset balance.

The aim of the report is to be a continuation of the knowledge contribution in the areas of IoT, IDS and ML. It intends to create an IDS that is able to detect Botnet attacks. It will do so by employing multiple ML techniques on multiple datasets. The datasets will also go under extensive manipulation though instance and noise removal and injecting. It will explore the possibility of feature optimization algorithms.

4. The Proposed Methodology

4.1. Datasets

Datasets are a result of a data extraction process that is applied on a real or simulated environment. The captured data is raw, usually, and thus might require further preprocessing. Data integration, cleaning, normalization and reduction (Al Shorman et al., 2020) are techniques that can be applied on raw data so that ML techniques can produce the best possible results. In the domain of IoT devices, the data capturing process is facilitated by network capturing applications such as Wireshark or TShark, the command line companion of Wireshark. These applications tap into the device network interface and then capture the raw traffic passing in that network. The output can be saved into a pcap file after being captured. While the pcap file can be considered a dataset, more dataset creators tend to use feature extractors. Some feature extractors support calculating statistical values and collapsing packets into flows. Then the extracted data is saved to a comma separated values (CSV) file. Datasets consist of rows representing the total number of instances. Each column in a row is one of the instance's features. In a classified dataset, the label or class column provides the correct class of the instance. Table 1 shows a portion of a dataset in csv format.

Table 1
Dataset sample

FlowID	*SrcIP*	*SrcPort*	*Protocol*	**…..**	*IdleMax*	*IdleMin*	*Label*
000512	192.168.0.13	9020	TCP	…..	43394	1394	Anomaly
000513	192.168.0.13	56361	UDP	…..	74	73	Anomaly
000514	192.168.0.13	9020	TCP	…..	120	120	Normal
000515	192.168.0.13	56361	UDP	…..	154	154	Anomaly

For this report, three datasets: N-BaIoT, IoTID20, and MedBIoT have been chosen as data sources. All of the datasets represent captures of IoT traffic in an environment that is being exploited by a form of Botnet malware. What follows is a brief introduction to each dataset.

4.1.1 N-BaIoT (UCI Machine Learning Repository)

Meidan et al. (2018) built the N-BaIoT datasets out of nine real IoT devices. The devices were infected with Bashlite and Mirai malware. It has 7,062,606 instances. The extracted features are 23 multiplied by five time-windows, totaling 17 to 115 features. The dataset provides fine-gained categorization of the attacks, and thus it can be used for multiclassification problems, in addition to binary problems. Table 2 previews the features available in N-BaIoT along with description of each. The features name has the structure Aggregator, Timeframe, and statistic type.

4.1.2 IoTID20 (Ontario Tech University)

In IoT, there are a limited number of flow-based and network-based datasets. This issue motivated Ullah & Mahmoud (2020b) to generate a new dataset out of an existing one, Bot-Net (Kang et al., 2019). The dataset replicates modern IoT network traffic and contains 86 features covering a wide range of modern attacks on IoT devices, including Botnets. It also supports multi-level attack categorization which allows for binary and multiclassification. The dataset has 625,783 instances and 86 features. Table 3 lists all this dataset's features.

Table 2
N-BaIoT Features (UCI Machine Learning Repository)

Aggregated Feature	*Time Frame (LX)*	*Description*
MI_dir_LX_weight	Each of the 23 Aggregators is captured at 5 different times frames.	Number of items, mean, variance in the traffic from the host (IP + MAC)
MI_dir_LX _mean		
MI_dir_LX _variance		
H_LX_weight		Number of items, mean, variance of traffic from the host
H_LX_mean		
H_LX_variance		
HH_LX_weight		Number of items, mean, standard deviation, in traffic going from the host (IP) to the destination host
HH_LX_mean		
HH_LX_std		
HH_LX_magnitude		Root squared sum of the two streams' means and variances of traffic going from the host (IP) to the destination host
HH_LX_radius		
HH_LX_covariance	LX represents different times:	Covariance and correlation coefficient between two streams of traffic going from the host (IP) to the destination host
HH_LX_pcc		
HH_jit_LX_weight	1 min, 10 sec, 1.5 sec, 500 msec, and 100 msec	Number of items, mean, variance in the jitter of the traffic going from this host (IP) to the destination host
HH_jit_LX_mean		
HH_jit_LX_variance		
HpHp_LX_weight	Totaling to 115 features	Number of items, mean, standard deviation in the traffic going from the host + port (IP) to the destination host/port
HpHp_LX_mean		
HpHp_LX_std		
HpHp_LX_magnitude		Root squared sum of the two streams' means and variances of the traffic going from host: port to the destination host: port
HpHp_LX_radius		
HpHp_LX_covariance		Covariance and correlation coefficient between two streams of the traffic going from host: port to the destination host: port
HpHp_LX_pcc		

4.1.3 MedBIoT (Tallinn University of Technology)

The MedBIoT is a dataset that has captures of 83 IoT devices (GuerraManzanares et al., 2021). It covers a combination of both real and emulated devices. Mirai, Bashlite, and Torii attacks were released on real devices with the intention of capturing their traffic in the earlier stages of Botnets. i.e., before launching the DDoS attack. The dataset is relatively large with 37,433,560 instances and 100 features with binary and multi-classification support. Table 4 lists the available features.

Table 3
IoTID20 Features (Ontario Tech University)

Feature	*Description*
Flow_ID	Flow ID
Src_IP Src_Port	Source IP/Port
Dst_IP Dst_Port	Destination IP/Port
Protocol	Protocol type
Timestamp	Timestamp
Flow_Duration	Flow duration
Tot_Fwd_Pkts Tot_Bwd_Pkts	Forwarded/backward packets counts
TotLen_Fwd_Pkts TotLen_Bwd_Pkts	Forwarded/backward packets length
Fwd_Pkt_Len_Max Fwd_Pkt_Len_Min Fwd_Pkt_Len_Mean Fwd_Pkt_Len_Std	Max, min, mean and Std of Forward packet length
Bwd_Pkt_Len_Max Bwd_Pkt_Len_Min Bwd_Pkt_Len_Mean Bwd_Pkt_Len_Std	Max, min, mean and Std of backward packet length
Flow_Byts/s	Flow in bytes/sec
Flow_Pkts/s	Flow in packets/sec
Flow_IAT_Mean Flow_IAT_Std Flow_IAT_Max Flow_IAT_Min	Mean, std, max, min of flow inter-arrival time
Fwd_IAT_Tot Fwd_IAT_Mean Fwd_IAT_Max Fwd_IAT_Min	Total, mean, max, min of forward inter arrival time
Bwd_IAT_Mean Bwd_IAT_Tot Bwd_IAT_Mean.1 Bwd_IAT_Std Bwd_IAT_Max Bwd_IAT_Min	Total, mean, std, max, min of backward inter arrival time
Fwd_PSH_Flags Bwd_PSH_Flags	Forward/backward PSH flags
Fwd_URG_Flags Bwd_URG_Flags	Forward/backward URG flags
Fwd_Header_Len Bwd_Header_Len	Forward/backward header length
Fwd_Pkts/s	Forward packets/sec
Bwd_Pkts/s	Backward packets/sec
Pkt_Len_Min	Min, max, mean, std, variance of packet length
Pkt_Len_Max	
Pkt_Len_Mean	
Pkt_Len_Std	
Pkt_Len_Var	
Down/Up_Ratio	Down/Up Ratio
Pkt_Size_Avg	Average packet size
Fwd_Seg_Size_Avg	Forward/backward average segment size
Bwd_Seg_Size_Avg	
FIN_Flag_Cnt	Counts of FIN, SYN, RST, PSH, ACK, URG, CWE flags
SYN_Flag_Cnt	
RST_Flag_Cnt	
PSH_Flag_Cnt	
ACK_Flag_Cnt	
URG_Flag_Cnt	
CWE_Flag_Count	
ECE_Flag_Cnt	
Fwd_Byts/b_Avg	Forward average bytes, packets, bulk rate
Fwd_Pkts/b_Avg	
Fwd_Blk_Rate_Avg	
Bwd_Byts/b_Avg	Backward average bytes, packets, bulk rate
Bwd_Pkts/b_Avg	
Bwd_Blk_Rate_Avg	
Subflow_Fwd_Pkts	Subflow forward/backward bytes and packets
Subflow_Fwd_Byts	
Subflow_Bwd_Pkts	
Subflow_Bwd_Byts	
Init_Fwd_Win_Byts	Forward/backward Initial window bytes
Init_Bwd_Win_Byts	
Fwd_Act_Data_Pkts	TCP payload packets
Fwd_Seg_Size_Min	Min forward seg size
Active_Mean	Mean, std, max, min of active time
Active_Std	
Active_Max	
Active_Min	
Idle_Mean	Mean, std, max, min of idle time
Idle_Std	
Idle_Max	
Idle_Min	
Label	Benign/Attack
Cat	Attack category
Sub_Cat	Attack subcategory

Table 4
MedBIoT Features (Tallinn University of Technology)

Aggregated Feature	*Time Frame (X)*	*Description*
MI_dir_LX_weight MI_dir_LX _mean MI_dir_LX _variance	Each of the 20 Aggregators is captured at 5 different times frames.	Number of items, mean, standard deviation in the traffic from the host (IP + MAC)
HH_LX_weight HH_LX_mean HH_LX_std		Number of items, mean, standard deviation, in traffic going from the host (IP) to the destination host
HH_LX_magnitude HH_LX_radius	So, each X should be replaced with:	Root squared sum of the two streams' means and variances of traffic going from the host (IP) to the destination host
HH_LX_covariance HH_LX_pcc		Covariance and correlation coefficient between two streams of traffic going from the host (IP) to the destination host
HH_jit_LX_weight HH_jit_LX_mean HH_jit_LX_variance	X represents different times:	Number of items, mean, standard deviation in the jitter of the traffic going from this host (IP) to the destination host
HpHp_LX_weight HpHp_LX_mean HpHp_LX_std	1 min, 10 sec, 1.5 sec, 500 msec, and 100 msec	Number of items, mean, standard deviation in the traffic going from the host + port (IP) to the destination host + port
HpHp_LX_magnitude HpHp_LX_radius		Root squared sum of the two streams' means and variances of the traffic going from host: port to the destination host: port
HpHp_LX_covariance HpHp_LX_pcc	Totaling to 115 features	Covariance and correlation coefficient between two streams of the traffic going from host: port to the destination host:port

A Summary of information related to all three datasets is listed in Table 5. Since all the aforementioned datasets support both binary and multiclassification, the last column was added with each possibly supported category.

Table 5
Datasets Summaries

Name	*Devices*	Format	Instances	Features	*Labels*	*Instances per Label*	
N-BaIoT (Meidan et al., 2018, p.)	Real	CSV	7,062,606	115	2	Benign	555,932
						Malicious	6,506,674
					11	Benign	555,932
						Mirai-ACK	643,821
						Mirai-Scan	537,979
						Mirai-SYN	733,299
						Mirai-UDP	1,229,999
						Mirai-UDP Plain	523,304
						Gafgyt-Combo	515,156
						Gafgyt-Junk	261,789
						Gafgyt-Scan	255,111
						Gafgyt-TCP	859,850
						Gafgyt-UDP	946,366
IoTID20 (Ullah & Mahmoud, 2020b)	Sim	CSV	625,783	86	2	Normal	40,073
						Attack	585,710
					5	Mirai	415,677
						DoS	59,391
						Scan	75,265
						Normal	40,073
						ARP Spoofing	35,377
					9	Mirai-Ack Flooding	55,124
						DoS-Syn Flooding	59,391
						Scan Port OS	53,073
						Mirai-Host Brute force	121,181
						Mirai-UDP Flooding	183,554
						Mirai-HTTP Flooding	55,818
						Normal	40,073
						Scan Host Port	22,192
						ARP Spoofing	35,377
MedBIoT (Guerra Manza-nares et al., 2021)	Sim + Real	CSV	37,433,560	100	2	Benign	31,956,092
						Malicious	5,477,468
					4	Bashlite	3,911,890
						Mirai	1,243,802
						Torii	321,776
						Benign	31,956,092

4.2. Overall Methodology

The steps that are to be taken to build the Botnet IDS is illustrated in Fig. 6.

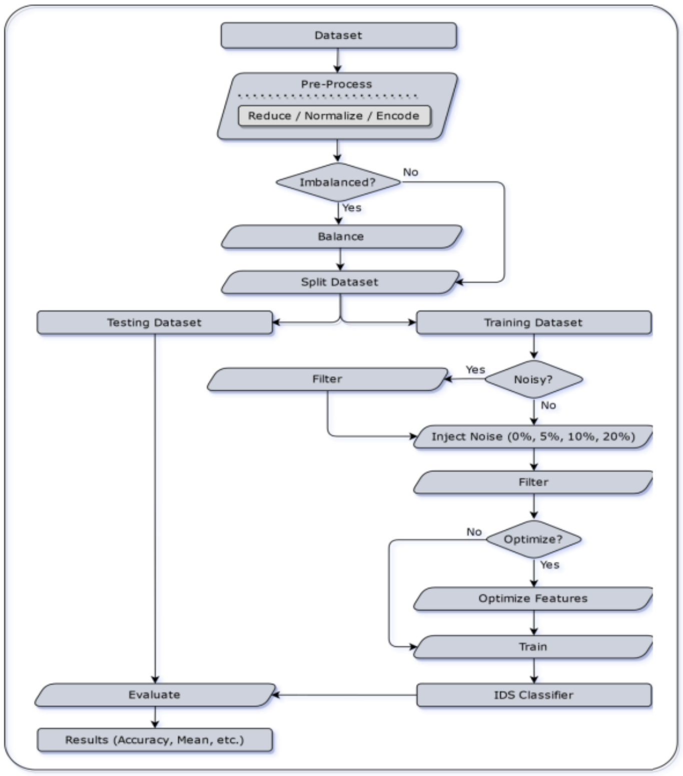


Fig. 6. The Experiment Methodology

4.3 Data Normalization

Normalizing data is a process that unites variation in numbers into a limited scale, typically in the range [0, 1]. It is performed so that machine learning does not become skewed when they are fitted. For instance, SVM expects instance values to be centered around zero. When a feature has values with variance that is an order of magnitude, the learning becomes less effective and as such values becomes the dominant in the produced model. Min-Max Scaling is one of the techniques used to scale datasets (Han et al., 2011). Since all three datasets under study have features with such characteristic, the Min-Max Scaling was applied on their features, as necessary. The operation was performed with help of Eq. (1):

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

4.4 Dataset Balancing

Oftentimes, captured data can be imbalanced. i.e., one or more classes in the dataset is obviously larger than the others. In such circumstances, ML techniques may exhibit bias toward the dominant class over the underrepresented ones. Which could eventually affect the classifier accuracy in a negative way. Imbalanced datasets can be converted into a balanced dataset. Some of the techniques to accomplish that include undersampling and oversampling. In undersampling, the larger samples are trimmed until balance is reached. In oversampling, the lesser class is increased using techniques such as Synthetic Minority Oversampling Technique (SMOTE) as shown in Fig. 7.

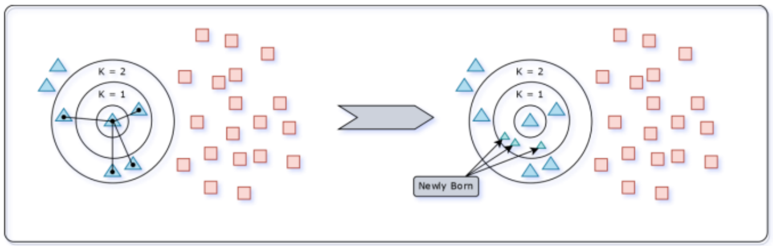


Fig. 7. Balancing a dataset with SMOTE (imbalanced is left, balanced is right)

To achieve the balance, SMOTE select a sample from minority class and then collect samples using the k of nearest neighbor (Chawla et al., 2002). Then the collected samples are synthesized to create new samples, as formulated in Eq. (2). This process is repeated until the minority class reaches the same number as the majority class.

$$X_{new} = X_{origin} + rand \times \ (X_i - X_{origin}) \quad (2)$$

where:

X_i are the randomly selected instances

i is the amount of selected sample

$rand$ is random number uniformly distributed within [0,1]

4.5 Instance and Noise Filtering

In the real world, datasets are not free of issues. Datasets may contain noise in the form of outliers or extreme values. One concern of such data is that they may lead to difficulties such as overfitting in ML. Overfitting results in models with good accuracy but lacking in generalization. Removing such instances could smooth out the dataset for ML. Another issue with the dataset is size. Appling ML on a large dataset can be computationally prohibitive. One of the solutions would be to take a smaller random sample of the dataset. This might be delivered in the size reduction department. But random sampling is blind and may remove instances that have high weight from the dataset. To handle these issues, three noise and instance filtering algorithms will be applied on the datasets. These filters will either filter the noise or reduce the instances or do both operations at the same time. For the noise and instance reduction algorithms to work, filtering algorithms need to calculate the distances between instances. Several distance functions exist such as Euclidean for numeric values and Value Difference Metric (VDM) for nominal (Stanfill & Waltz, 1986). The IoTID20 dataset contains both nominal features such as Src_Port, Dst_Port and Protocol in addition to numeric values. Therefore, Heterogeneous Value Difference Metrics (HVDM) are utilized in Eq. (3) to support both types (Wilson & Martinez, 1997). Then, depending on the value type, if it is numeric, the Euclidean function is applied as in Eq. (4). If it is nominal, it uses VDM as in Eq. (5):

$$HVDM(x,y) = \sqrt{\sum_{a=1}^{f} d_a^2\,(x_a, y_a)} \quad (3)$$

$$d(x_a, y_a) = \frac{|x-y|}{6\sigma_a} \quad (4)$$

$$d_a(x,y) = vdm_a(x,y) = \sqrt{\sum_{c=1}^{c}\left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}}\right)^2} \quad (5)$$

where:

$d_a\,(x, y)$ is the distance of feature a.

σ_a is the standard deviation of a values.

$N_{a,}$ indicate how many times a took the value of x in training set.

$N_{a,,c}$ indicate how many times a took the value of x in class c.

C is the number of classes

4.5.1 Repeated Edited Nearest Neighbor

Repeated Edited Nearest Neighbor (RENN) is a noise filtering algorithm that utilizes k nearest neighbors (Wilson & Martinez, 2000). It is an enhanced version of another noise filtering algorithm, the Edited Nearest Neighbor (ENN). In ENN, an instance is noisy if it is labeled differently than its k nearest neighbors. The major RENN enhancement over ENN is that RENN does that same thing but in several iterations. There are two issues with ENN and its enhanced RENN. One is that, due to the rather cautious algorithms they are using, they do not prune sizable portions of datasets. i.e., sometimes, the pruned datasets end up being almost the same size as the original dataset. The other issue is the ordering in the dataset can have an influence on the reduction performance.

4.5.2 Explore

This algorithm uses Encoding Length heuristic (Cameron-Jones, 1995) to decide on whether a subset of the dataset is good enough to describe the dataset. i.e., it adds instances to a subset based on cost criteria. In a growing phase, it creates a new subset then calculates the cost of adding instances to that new subset. If adding an instance does not increase the subset cost, it gets added. Otherwise, it is omitted. When finished, this subset becomes the pruned dataset. The Encoding Length Grow (ELGrow) algorithm uses this approach in its filtering process. Explore enhances ELGrow by applying the process with 1000 mutations to improve the classifier. Encoding Length algorithm is aggressive with instance reduction. They, sometimes, remove almost the entire instances, which may result in datasets with degraded accuracy. Moreover, since this algorithm works in a growing way, the order of the dataset can greatly influence the pruning process.

4.5.3 Decremental Reduction Optimization Procedure 5

In this algorithm, each instance creates their neighbors. This would create networks of interleaving neighbors. Then, an instance gets removed if its removal does not affect other instances that have it as a neighbor. In other words, instances are removed if the level of generalization does not degrade. Decremental Reduction Optimization Procedure 5 (DROP5) is an improved version of DROP2 (Wilson & Martinez, 2000). The difference is that DROP2 considers neighbors with the same class, while DROP5 relies on neighbors with opposite classes for the decision. The size reduction that can be gained from DROP is huge. However, that comes at the expense of the high computation requirement. Such added complexity also makes them less sensitive to dataset ordering.

4.6 Features Selection Optimizations

Plain data has features that describe the bits that make up that data. And these pieces might or might not be relevant when solving a particular problem. For instance, a person's age could be significant in predicting diabetes but not so for a car accident problem. Feature selection is the process of picking a minimum number of features from a larger set of features such that the omitted features will not affect the problem at hand. While feature selection can be done by humans (Kumar & Lim, 2019), machines are able to do it in more efficient ways. One of the most important companions to ML classifiers is features selection optimization (FSO) algorithms (Faris et al., 2018). When ML classiers looks for hidden patterns, those FSO attempt to find the most relevant features for such models. That results in a boost in prediction performance, while maintaining accuracy as high as possible. Several FSO already exists in field of data science:

- Particle Swarm Optimization (PSO): Information in this algorithm which was authored by Kennedy and Eberhart (1995) is optimized with the help of community interactions. Initially, PSO chooses candidate particles as solutions, randomly. These particles then look for global maximums in a space that has the possible solutions. Optimization is achieved when the best value is reached from all particles collectively.
- Multi-Verse Optimizer (MVO): Based on the theory of multi-versa, Mirjalili et al. (2016) created this optimizer to mimic the diversity of the universe, especially with interactions of white holes, blackholes and wormholes. The optimization global optimum is estimated from the collected solutions.
- Grey Wolf Optimizer (GWO): Mirjalili et al. (2014) established this optimization from observations on grey wolves hierarchical social organization. That social hierarchy ranks wolves into levels, with each level having specific strength and interaction such as preying and leadership.

4.7 Machine Learning Techniques

Three ML techniques will be conducted on the three datasets to develop IDS models. Those techniques are Neural Network (NN), Decision Tree (DT), and Random Forest (RF). The following gives a short explanation along with each classifier's advantages and disadvantages.

4.7.1 Neural Network (NN)

Are a kind of ML that takes inspiration from the biological nervous system. They have interrelated structures, which allow them to represent different processing systems. The main components of NN are the neurons and the weights. The neurons are elements that are interconnected inside the neural network. The weights are fractions that influence the decisions that go through the neurons. Fig. 8 illustrates a simplified version of NN. There are advantages and disadvantages to NN (Al-Gethami et al., 2021). Some of the advantages are that their classifiers are forgiving noise, and that they support a wide range of problems. Also because of their architecture, it is also possible to utilize processing parallelism on them. However, their models suffer from long training times, and they can be hard to deduce.

The multilayer perceptron (MLP) is the most popular implementation in NN. It gives this type of ML the ability of having multiple layers of artificial neurons that can be considered perceptions. In a back propagation network (Hecht-Nielsen, 1992), where a neuron can be described as in Equation 6, information moves in forward direction. Eq. (7) and Eq. (8) are used for hidden and output layers, respectively. Errors, which have Eq. (9) would then propagate back to tweak the inputs.

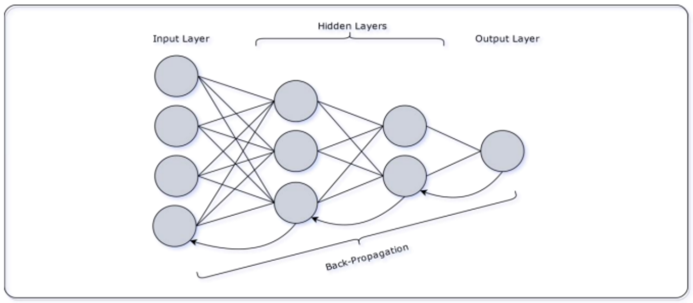


Fig. 8. Neural Network

$$Y_k = f\left(\sum_{i=1}^{m} w_{ik}x_i + b_k\right) \tag{6}$$

$$a1_i = f1\left(\sum_{j=1}^{r} w1_{ji}p_j + b1_i\right) \tag{7}$$

$$a2_k = f2\left(\sum_{i=1}^{s1} w2_{ik}a1_i + b2_k\right) \tag{8}$$

$$\varepsilon = \frac{1}{2}\sum_{k=1}^{s2}(t_k - a2_k)^2\,, e_k = t_k - a2_k \tag{9}$$

where:

p is the input.

r is the neuron.

i and k are the hidden and output neurons, respectively.

$s1$ and $s2$ are number of hidden and output neurons, respectively.

t is the target.

4.7.2 Decision Tree (DT)

One of the things that made DT popular is that they resemble the decision-making process in the human brain. A Human brain can see the output of this type of ML and understand it quite easily, especially if it gets represented graphically. DT has nodes. These nodes can either be internal or leaf. In the case of leaves, the decision process stops. If it is internal, it gets split into two new child nodes if a testing criterion is met. This process continues until the end where these internal organs become another leaf. The desired outcome of DT is getting pure nodes having one class. Figure 9 shows a sample of a decision tree.

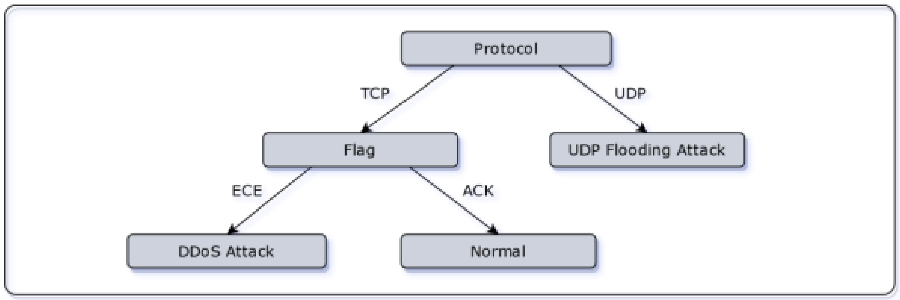


Fig. 9. Decision Tree

Being simple, fast, understandable, and accurate are some of the advantages of DT (Al-Gethami et al., 2021). However, they are not suitable for a wide range of problems since they consume memory, need long training time, suffer from replication and feature arrangement could affect their model's performance.

Reducing uncertainty in DT for a classification problem depends on the tests. The better the test, the lower uncertainty about a problem. Entropy is a function that can be used to calculate that uncertainty (Safavian & Landgrebe, 1991). The function is given in Eq. (10).

$$H(X) = -\sum_{k}^{n} p(x_i) \log_b p(x_i) \tag{10}$$

where:

n holds the number of result or outcomes.

(xi) is the probability of a particular result i

And b being commonly 2, 10 or e.

4.7.3 Random Forest (RF)

Are ensemble techniques that build on DT. It uses bootstrapped samples at each stage of the ensemble. When only a portion of the features are selected, RF may produce non-corelated trees. This could produce accurate scores due to reduction in variance caused by averaging non-correlated trees. Fig. 10 illustrates RF tree.

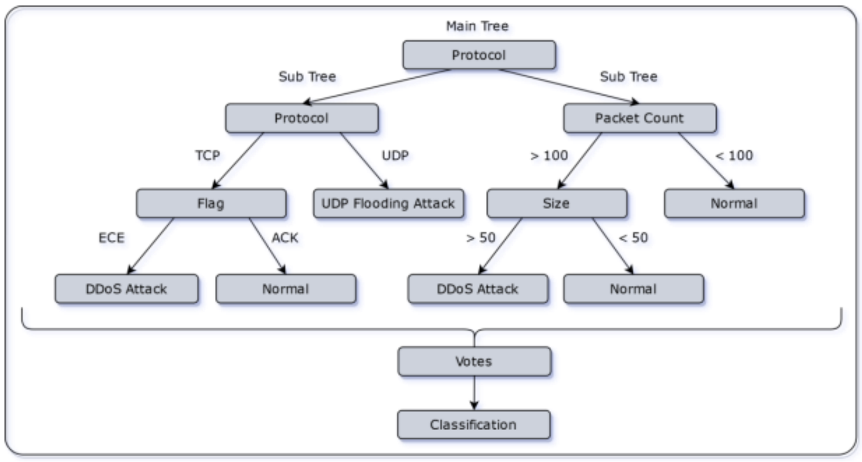


Fig. 10. Random Forest

Due to having many DT, tolerance for features selection (Al-Gethami et al., 2021) is one of the advantages of RF. Though, not every tree sees all the features. Also, bias is unaffected by the number of trees. Finally, in contrast to DT, models normally are not overfitted. However, the price to pay for that is interpretation difficulties and performance degradation when variables are correlated. Samples of DT with different feature combinations is the building block of RF. A prediction is a combination of the majority predictions from those trees. RF work is made possible with the help of Bierman's bagging and random feature selection. Generally, more trees result in better predictions. But also, more computational requirements. RF has Eq. (11). To divide the tree, Gini impurity is employed as in Eq. (12).

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\acute{x}) \tag{11}$$

where:

f is the final prediction.

B is total number of trees

b is one of the trees

x' is data to be fitted

$$I_G(f) = \sum_{i=1}^{m} f_i\,(1 - f_i) \tag{12}$$

where:

fi is probability of correct classification.

4.8 Evaluation Metrics

To evaluate the constructed intrusion detection model, we use the confusion matrix. As seen from Table 6, we derive the four concepts used for classifiers evaluation: True Positive (TP), True Negatives (TN), False Positive (FP) and False Negatives (FN).

Table 6
Confusion Matrix

Prediction\Truth	*Positive*	*Negative*
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

- Accuracy. Also known as Classification Rate. It measures the rates of the correctly classified positive and negative instances. It uses Eq. 13:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

- Precision: Out of all the predicted positive instances, how many are positive? It uses Eq. (14):

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

- Recall: Also known as Sensitivity or True Positive Rate (TPR). Out of all the positive instances, how many are predicted as positive. It uses the Eq. (15):

$$Recall = \frac{TP}{TP+FN} \tag{15}$$

- Specificity. Also known as True Negative Rate (TNR). Out of all the negative instances, how many are predicted as negative. It uses Eq. (16):

$$Specificity = \frac{TN}{FP + FN} \tag{16}$$

- G-Mean is the geometric mean and it combines Sensitivity and Specificity into a single score. This measurement can be useful on imbalanced datasets. It uses the Eq. (17):

$$G - Mean = \sqrt{Sensitivity \cdot\ Specificity} \tag{17}$$

- F-Score is the harmonic mean of precision and recall. It signifies the balance between the values of recall and precision. It has the Eq. (18):

$$F - Score = 2 \cdot \frac{Precision \cdot Recall}{Precisioin + Recall} \tag{18}$$

5. Implementation

This section provides information about the project's implementation. It shows the platform where the code was executed. It also lists the development software and the needed libraries. Finally, it briefly explains some of the codes that were used to complete the project.

5.1 Development Platform

The specifications of the personal computers (PC) that was used for this report is listed in Table 7

Table7
Computer Specifications

OS1 (Host)	Windows 10 64-Bit v20H2. Build: 19042.928
OS2 (Virtual)	Ubuntu 20.04 on Windows WSL
CPU	Intel Core i7-3615QM CPU (8 Cores @ 2.30GHz)
Memory	16 GB
Storage	SSD 254 GB
GPU	Integrated

5.2 Development Software

Different software packages were used to complete parts of this report. Python was the code interpreter. The "Drop" binary is the filtering program.

5.2.1 Python

Almost the entire scripts were written using Python programming language and were executed with their native interpreter version 3.9.4. Along with python's included libraries, Table 8 lists the other 3rd parties' libraries with their purposes. It should be noted that installing these libraries requires running Python's bundled "PIP" command. Overwise, a link to the library is provided.

Table 8
Python 3

Library name	*Version*	*Purpose*
numpy	1.20.2	Required by other libraries
pandas	1.2.4	Dataset loading/manipulation/saving
scikit-learn	0.24.1	Machine Learning
Imbalanced-learn	0.8.0	Dataset Balancing
EvoloPy-FS	Beta	Feature Selection Optimizations (Aljarah, 2019)

5.2.2 Drop

The filtering process in this report was performed with the help of the Drop program. The python scripts above are only automating the process. The code is available at the author website at: https://axon.cs.byu.edu/~randy/pubs/ml The author does not supply binaries for the program. So, it needs to be compiled with a compatible C compiler, such as GCC in a Unix platform. Since Drop has difficulties running on Windows, it is advisable to have it compiled and executed in a Linux system.

5.2.3 Integrated Development Environment (IDE)

The IDE that was utilized to develop the code is Microsoft's open-source Visual Studio Code (VSCode). For easier development, it is recommended to install the extensions: Pylance, Python and IntelliCode.

5.3 Datasets

No datasets were created as part of the report's ML process. All datasets were acquired from their respective authors. Table 9 below list these datasets and a link to download them:

Table 9
Datasets Information

Name	*Authors*	*Institutes*	*Download Link*
N-BaIoT	(Meidan et al., 2018, p.)	Machine Learning Repository (UCI)	https://archive.ics.uci.edu/ml/ datasets/detection_of_IoT_bo tnet_attacks_N_BaIoT
IoTID20	(Ullah & Mahmoud, 2020b)	Ontario Tech University	https://sites.google.com/view /iot-network-intrusion-dataset/home
MedBIoT	(Guerra- Manzanares et al.,2021)	Tallinn University of Technology	https://sites.google.com/view /iot-network-intrusion-dataset/home

6. Experiments, Results and Discussions

The three datasets to be experimented with were sampled from their respective original datasets. To widen the experiment, a choice has been made to collect these datasets differently. Given the available computational resources, filtering an entire dataset was not feasible. Thus, only samples from those rather large datasets were taken. IoTID20 was randomly sampled at 1% from the original imbalanced dataset. That resulted in a new imbalanced dataset with 6,257 instances. Of which, 5,857 were attack instances and 401 were normal. Since that dataset is imbalanced, SMOTE techniques were applied on it before it got split into training and testing datasets. On the other hand, N-BaIoT and MedBIoT were randomly sampled as 5,000 instances, with equally distributed attack and normal instances.

The type of attacks that are covered by each dataset varies. IoTID20 has both Botnet and non-Botnet attacks. N-BaIoT is only concerned with attacks launched from Botnets. MedBIoT covered the infection of IoT devices in addition to Botnet attacks. Table 10 lists the total number of instances, their distribution and attack types covered by the dataset.

Table 10
Datasets Samples, Instances, and their Types

FlowID	*IoTID20 (Before)*	*IoTID20 (After)*	*N-BaIoT*	*MedBIoT*
Attacks instances	5,856	5,856	2,500	2,500
Normal instances	401	5,856	2,500	2,500
Total instances	6,257	11,712	5,000	5,000
Attack Types	Botnet + Other Attack types		Botnet	Full Botnet

In this experiment, the datasets were split into testing and training dataset. With testing taking 30% and training retaining the rest, 70%. In the subsequent experiments, instance filtering, feature optimization and ML techniques will be applied into the training dataset. The testing dataset is only to evaluate the produced IDS models. For each training dataset, instances will be filtered first. Then, artificial noise will be injected and filtered again. Afterwards, the filtered datasets will be fed to the feature optimization process. Finally, the optimized datasets are trained and evaluated against the testing dataset.

6.1 Filtering and Noise Filtering

Three noise and instance filtering algorithms, RENN, Explore and DROP5 were applied to the three datasets: IoTID20, N-BaIoT and MedBIoT. One of the main objectives of those algorithms is to prune instances from datasets. The algorithms deem these instances as noise or having little to no effect on overall dataset accuracy, and as such decide that they are not necessary. The amount of reduction varies depending on the goal of each of these algorithms. Whether it is instance reduction or just noise removal.

6.1.1 Instance and Noise Filtering with No Added Noise

This experiment begins with no noise (0%) inserted into the datasets. The unfiltered dataset kept all instances intact, as can be observed in Fig. 11. This unfiltered dataset provides a reference point for other algorithms. In RENN, a noisy instance is one that has a different label than its near k neighbors. Since this algorithm removes noise only, it was not surprising to see it removing less than 1% of IoTID20 and N-BaIoT. It did slightly better in MedBIoT though, cutting about 4% of the dataset. That might indicate that MedBIoT has more noise than the other datasets. However, a more substantial reduction of instances was noticed with both Explore and DROP5. Explore algorithm filters a dataset by creating an empty dataset first (Cameron-Jones, 1995), and then adds instances to that set if adding them does not increase that set's cost. This means that if the algorithm adds a low cost instance at the beginning, then subsequent instances will likely not get added. This explains the huge reduction in instances seen from the Explore algorithm. The reduction was so aggressive that it removed almost, all instances across the three datasets. DROP5 did well too and filtered about 99% of IoTID20 and N-BaIoT and 94% of MedBIoT.

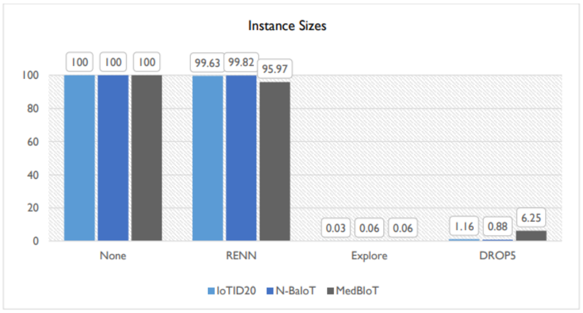


Fig. 11. Amount of reduction in instances

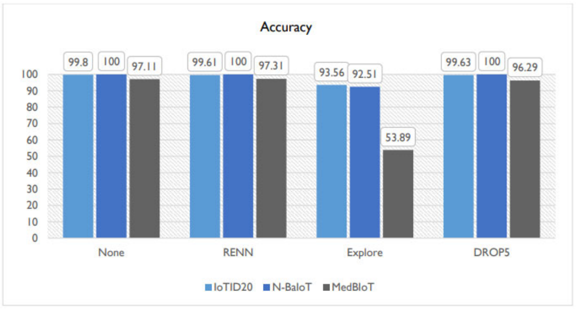


Fig. 12. Accuracy of the filtered datasets

The instance filtering affected the accuracy of some dataset more than others. Figure 12 shows the accuracy after applying the filters on the datasets. Of all filters, explore reported the worst accuracy. Dropping as much as 53% in MedBIoT. As mentioned previously, the cost criteria that Explore uses probably resulted in dropping instances that should never have been dropped (Cameron-Jones, 1995). With RENN, the accuracy was excellent, especially with N-BaIoT scoring 100%. But that was expected considering that RENN pruned just a few instances. The surprising performance was seen in DROP5. Despite removing about 99% of the instances from IoTID20 and N-BaIoT, it still managed to maintain a very high accuracy of 99.63% and 100% respectively. DROP5 performs both noise and instance filtering differently than Explore (Wilson & Martinez, 2000). Each instance in DROP5 creates their own neighbors. This creates networks of interleaving neighbors. The decision to remove an instance is made based on whether the instance to be removed exists in other instances neighbors. That is a computational expensive operation but it results in accuracy maintenance, even with the substantial instance reduction. Table 11 shows the filtering results in detail.

Table 11
Filtering Accuracy and Size Reductions

FlowID	*IoTID20 (Before)*	*IoTID20 (After)*	*N-BaIoT*	*MedBIoT*
Attacks instances	5,856	5,856	2,500	2,500
Normal instances	401	5,856	2,500	2,500
Total instances	6,257	11,712	5,000	5,000
Attack Types	Botnet + Other Attack types		Botnet	Full Botnet

6.1.2 Instance and Noise Filtering with Added Noise

It was obvious from the previous experiment that RENN and DROP5 could be a candidate to pick for the subsequent optimization and classification steps. But the three datasets represent only three datasets out of all other available Botnet datasets. Not to mention that the experiment is conducted in a much smaller sample. Therefore, basing the choice on such a smaller scale sample might not guarantee that either will perform well if larger and noisy datasets are encountered. So, in this experiment, artificial noise is infected in the datasets to simulate the possibility of a noisy environment. Three levels of noise: 5%, 10% and 20% will be injected in each dataset. The filtering application achieved that using a simple trick of changing the class labels in these datasets between “Attack” and “Normal”. After these changes, the filtering process is performed, and accuracy is measured again after these algorithms filter the injected noise. Fig. 13 compares the accuracy at each noise level. Table 12 shows the detailed results. The previous experiment gave us an indication of the expected size reduction. Therefore, the focus in this experiment will be on accuracy. Or specifically, how accuracy will be affected as the noise gets raised from 0% (no noise), to 5%, 10% and 20%. Without applying filtering, the accuracy went down in the three datasets. Nearly 15% and 10% of accuracy were lost in IoTID20 and N-BaIoT datasets, respectively. This shows how unhandled noise could degrade the quality collected data. In the previous experiment, i.e., without noise, DROP5 did a phenomenal job in filtering instances. It performed a tad better than RENN in accuracy and complemented that with superior performance in pruning instances. However, once noise was encountered, DROP5 accuracy kept going down as noise level went up. This can be explained with DROP5's tendency to cut a lot of instances. By doing so, it starved itself from good instances. In contrast, RENN was not as affected with noise level change when compared to DROP5. That is because RENN only filters a small number of instances and. thus retaining more instances. And those retained instances contribute to the overall accuracy. Those findings conclude the search for the best choice among filters. That filter would be RENN.

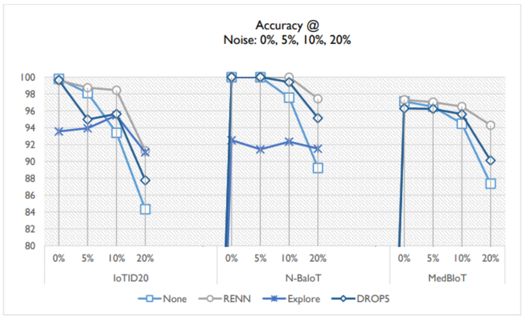


Fig. 13. Accuracy at Different Noise Levels

Table 12
Noise Injection Impact on filters performance

Dataset	*Noise Level*	*None*	*RENN*	*Explore*	*DROP5*
IoTID20	0%	99.8	99.61	93.56	**99.63**
	5%	98.11	**98.74**	93.93	94.97
	10%	93.41	**98.45**	95.46	95.63
	20%	84.34	**91.27**	91.08	87.77
N-BaIoT	0%	100	**100**	92.51	**100**
	5%	100	**100**	91.43	**100**
	10%	97.57	**100**	92.34	99.4
	20%	89.23	**97.43**	91.51	95.14
MedBIoT	0%	97.11	**97.31**	53.89	96.29
	5%	96.49	**97.03**	60.57	96.23
	10%	94.49	**96.51**	69.4	95.6
	20%	87.37	**94.29**	59.71	90.11

6.2 Feature Selection

Now that RENN has been chosen to be the filtering algorithm, the next step would be to apply FSO on the filtered dataset. Four filtered datasets from each of the IoTID20, N-BaIoT and MedBIoT will apply the feature selections: the baseline RENN filtered dataset (RENN-00). And the three noise injected datasets RENN05, RENN-10 and RENN-20. To perform the feature selection, three FSO will be used: Particle Swarm Optimization (PSO), Multi-Verse Optimizer (MVO) and Grey Wolf

Optimizer (GWO). Those nature-inspired algorithms will attempt removing features or attributes from the filtered datasets. By removing features, ML will be able to get better generalization, faster predictions, and lower sizes. The Python library used for FSO, the EvoloPy-FS, exposes some parameters. The general parameters for all optimizers were set to do the optimization just for one time. The number of iterations was 50 iterations, and the population size was fixed at 30. For brevity, in this step, two experiments will be performed on one of the datasets only, the IoTID20. The first experiment discusses the votes outcomes of the three FSO optimizers with no noise (0%), from a security perspective. The second experiment compares the votes at various noise levels (0%, 5%, 10% and 20%) to see how the noise affects the number of selected features.

6.2.1 Security Features selected by the FSO

The sampled IoTID20 dataset has a total of 79 features relating to security. Some features have more influence than others on deciding whether an activity is an attack or not. When applying FSO optimization on that dataset, the expectation is that these optimizers will agree on at least a few features. Fig. 14 shows the votes of the three FSOs after they have analyzed the IoTID20 dataset. Features that have the most relevance should have a unanimous vote from those optimizers (represented as a full bar or 3). On the other hand, features that got zero votes are the ones that have the least relevance from a security point of view. The three optimizers, PSO, MVO and GWO gave a unanimous vote on the following IoTID20 dataset features:

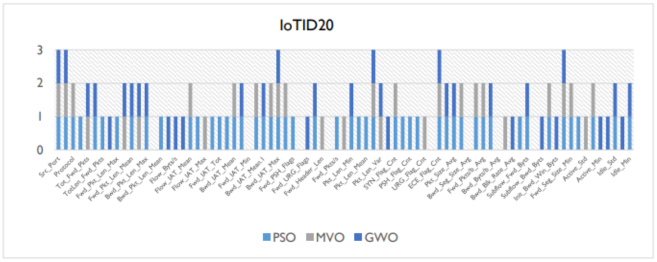


Fig. 14. Features Selected by PSO, MVO, and GWO

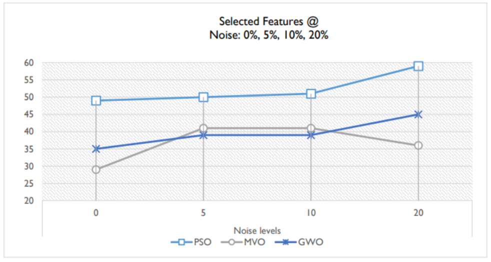


Fig. 15. Effect of Noise Levels on Feature Optimizers

- Src_Port and Dst_Port: The source and destination port of a traffic flow. These could indicate an attack that uses props. For example, Botnet scanning for potential IoT devices may use ICMP, which has no port number (represented in the dataset by 0). So, a port of zero may alert the IDS of Botnet scan.
- Pkt_Len_Std: A sudden increase in the packet length leads to a change in its standard deviation. Attacks cause these changes.
- Bwd_IAT_Max: responding packets inter-arrival times show the difference, in time, between two packets arriving from the destination host. When an IoT device performs a DDoS attack on a server, the response times from that server will change due to being exhausted from the DDoS.
- ECE_Flag_Cnt: This is congestion notification flag that is typically sent from servers that are under DDoS attack. It tells clients that there are congestion right now. Of course, the attacker will not respond to that notification and will keep attacking.
- Act_Data_Pkts: the counts of packet that has data in them. If there are a lot of these, it may indicate a flooding attack on servers.

6.2.2 Comparing Optimizers at Different Noise Levels

The effect of noise on the instance filtering has been analyzed previously. To see how the noise affects the optimizers' performance, the three optimizers, PSO, MVO and GWO optimizer will be applied on the baseline RENN filtered dataset (RENN-00) and the noise injected datasets (RENN-05, RENN-10 and RENN-20). Fig. 15 compares the votes at these noise levels of these optimizers. Table 13 shows the detailed results.

Table 13
Number of Features Selected at Various Noise Levels

Optimizer	**Noise Level**			
	0%	***5%***	***10%***	***20%***
PSO	49	50	51	59
MVO	29	41	41	36
GWO	35	39	39	45

It can be observed that the noise has changed the selected features on the noise-injected datasets (RENN-05, RENN-10 and RENN-20), compared to the baseline (RENN-00). Both PSO and GWO gave the least number of features at 0%, But then the selection increased gradually with the increase of injected noise. The MVO experienced the same thing, except that at noise level 20%, it selected less features than in noise levels 5% and 10%. Overall, FSO performance is clearly affected by the injected noise in the datasets, even though the noise should have been filtered previously.

6.3 Machine Learning

This is the last step in the experiment. The three datasets that have been filtered and feature-optimized will be used now to construct the Botnet IDS. Three ML techniques will train on these datasets: Decision Tree (DT), Random Forest (RF), and Neural Network (NN). Metrics such as accuracy, recall, sensitivity will measure the performance of the models with the help of the testing dataset. Other measurements include prediction speed.

6.3.1 Accuracy

The IDS model produced by IoTID20 managed to achieve excellent accuracy in DT and RF. Nearly all FSOs had an accuracy above 99.5%, with GWO and PSO being slightly ahead than MVO. NN had the worst results of the three ML in all the three FSOs. It only reached around 92%. Fig.16 compares the ML accuracy of IoTID20. Table 14 shows the detailed results.

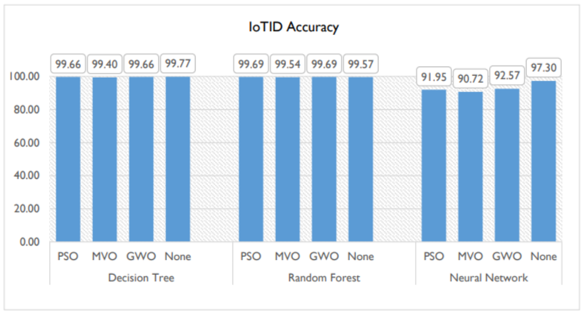


Fig. 16. IDIoT20 Prediction Accuracy from DT, RF and NN classifiers

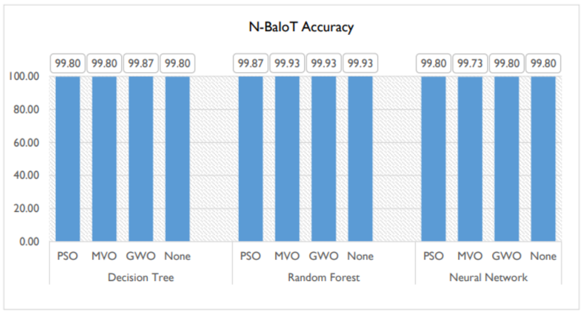


Fig. 17. N-BaIoT Prediction Accuracy from DT, RF and NN classifiers

The best results of the three datasets were produced by N-BaIoT. The accuracy across all ML techniques and FSOs were above 99%. Overall, GWO was a hair better in prediction accuracy than the other two FSOs. Fig. 17 compares the accuracy of N-BaIoT. Table 15 shows the detailed results.

Table 14
IoTID20 Detailed results of DT, RF and NN classifiers

Classifier	*Optimizer*	*Accuracy*	*Recall*	*Precision*	*F1-Score*	*G-Mean*
Decision Tree	***PSO***	99.66	99.72	99.61	99.67	99.66
	MVO	99.40	99.50	99.34	99.42	99.40
	GWO	99.66	99.67	99.67	99.67	99.66
	None	99.77	99.78	99.78	99.78	99.77
Random Forest	***PSO***	99.69	99.83	99.56	99.70	99.68
	MVO	99.54	99.83	99.28	99.56	99.54
	GWO	99.69	99.78	99.61	99.70	99.68
	None	99.57	99.67	99.50	99.58	99.57
Neural Network	***PSO***	91.95	95.29	89.67	92.39	91.79
	MVO	90.72	99.39	85.05	91.66	90.05
	GWO	92.57	99.45	87.72	93.22	92.12
	None	97.30	98.39	96.41	97.39	97.26

Table 15
N-BaIoT Detailed results of DT, RF and NN classifiers

Classifier	*Optimizer*	*Accuracy*	*Recall*	*Precision*	*F1-Score*	*G-Mean*
Decision Tree	***PSO***	99.80	99.73	99.87	99.80	99.80
	MVO	99.80	99.87	99.73	99.80	99.80
	GWO	99.87	99.73	100.00	99.87	99.87
	None	99.80	99.87	99.73	99.80	99.80
Random Forest	***PSO***	99.87	99.73	100.00	99.87	99.87
	MVO	99.93	99.87	100.00	99.93	99.93
	GWO	99.93	99.87	100.00	99.93	99.93
	None	99.93	99.87	100.00	99.93	99.93
Neural Network	***PSO***	99.80	99.73	99.87	99.80	91.79
	MVO	99.73	99.73	99.73	99.73	90.05
	GWO	99.80	99.73	99.87	99.80	92.12
	None	99.80	99.87	99.73	99.80	97.26

Table 16
MedBIoT Detailed results of DT, RF and NN classifiers

Classifier	*Optimizer*	*Accuracy*	*Recall*	*Precision*	*F1-Score*	*G-Mean*
Decision Tree	***PSO***	94.60	92.60	96.36	94.44	94.56
	MVO	94.00	91.12	96.58	93.77	93.93
	GWO	94.40	92.06	96.47	94.21	94.35
	None	94.27	91.92	96.33	94.08	94.22
Random Forest	***PSO***	96.20	93.27	99.00	96.05	96.13
	MVO	95.73	92.73	98.57	95.56	95.66
	GWO	95.93	92.87	98.85	95.77	95.86
	None	96.00	92.87	99.00	95.83	95.92
Neural Network	***PSO***	87.27	77.12	96.46	85.71	91.79
	MVO	86.47	76.18	95.61	84.79	90.05
	GWO	88.93	85.60	91.51	88.46	92.12
	None	89.27	82.64	95.05	88.41	97.26

The worst accuracy of the three datasets was made by MedBIoT. In DT, the accuracy was hanging around 94% across the three FSOs. RF did better than DT in this regard and pulled around 95% in accuracy. The lowest scores were observed in NN with the best accuracy achieved by GWO at 89%. Fig. 18 compares the accuracy of MedBIoT. Table 16 shows the detailed results. From the above ML results of the three datasets, it can be noticed that the N-BaIoT dataset was ahead of IoTID20 and MedBIoT in accuracy, regardless of the ML techniques used. Also, all the FSOs had very close scores, with GWO being marginally ahead. Although N-BaIoT was ahead of the pack, one thing worth mentioning is that N-BaIoT also had the least number of attack coverage. i.e., it only covered attacks launched from compromised IoT devices. While the two other datasets covered scanning, infection, and attacks.

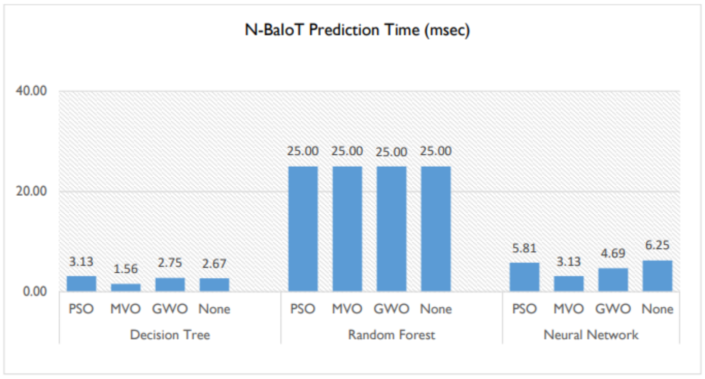


Fig. 18. N-BaIoT Prediction Times

6.3.2 Prediction Performance

The three FSOs did very well with the N-BaIoT dataset. They all achieved an accuracy that is above 99% across all ML techniques. In this part, attention is paid in the prediction time of the FSOs. In DT, MVO predicted the classes of the testing in less than two milliseconds. That was about 150% and 200% faster than GWO and PSO, respectively. The same was achieved in NN too. Though, the margin was less than those from DT. Figure 19 compares the prediction time of GWO, MVO and PSO in the IDS built with N-BaIoT. RF is one of the most resource intensive ML techniques. It is composed of trees of DTs that conclude their decision with votes. In RF, all FSO performed the same. For us, we could not find any explanation for that strange similarity.

7. Conclusions and Future Work

There were two objectives to this report. One is experimenting with different datasets and ML techniques, to address IoT diversity. And two looking into techniques to remove noise and filter dataset instances. The process began by collecting three datasets: IoTID20, N-BaIoT and MedBIoT. These three datasets are IoT focused and have coverage of Botnet attacks.

Under the assumption that these datasets might have noises, three filtering algorithms were incorporated in the report: RENN, Explorer and DROP5. RENN supports noise filtering. While Explorer can reduce instances. DROP5 on the other hand supports both noise and instance reduction. The filters were applied into the training datasets to either remove noise or to reduce the number of instances. To assure the quality of filtering algorithms, artificial noise was also injected and filtered in one stage.

After choosing the best filter, the feature optimization algorithms were used to select the important features in each dataset. Three algorithms were used: Particle Swarm Optimization (PSO), Grey Wolf Optimizer (GWO), Multi-Verse Optimizer (MVO). The optimizations were applied only on the chosen filter, which was decided to be as RENN then. Afterwards, three machine learning techniques were used for classification: Decision Tree, Random Forest, and Neural Network.

The results of the experiment have shown that both RENN and DROP5 delivery sold accuracy on the filtered datasets. RENN only removed less than 1% of the instances representing noise, DROP5 managed to remove more than 98% of the original dataset. However, after injecting noise in the datasets, DROP5 accuracy kept going down as the injected noise was increased. Thus, the choice to pick RENN instead of DROP5.

The Feature optimizations were applied on the datasets that were filtered by RENN. The main goal of these optimizations is to lower the number of features. The experiments revealed that noise also affected the quality of feature selection too. i.e., the increase of noise, increased the number of selected features.

The filtered and optimized dataset were used to create the IDS models. The N-BaIoT dataset produced excellent results across all ML and optimizations with accuracy above 99%. IoTID20 did slightly worse, especially in the Neural Network.

For future experiments, DROP5, or one of its variations could be used to filter full datasets instead of their samples. One of the issues of ML is dataset size. In this experiment, the datasets that were used were sampled. Since sampling does not guarantee that the instances that were removed are irrelevant, we suggest using one of the filtering algorithms for the task. DROP5 did a phenomenal job in both removing the majority of instances while keeping accuracy high. Though, we also suggest that some kind of hardware acceleration as DROP and its variation are extremely resource intensive.

References

Al Shorman, A., Faris, H., & Aljarah, I. (2020). Unsupervised intelligent system based on one class support vector machine and Grey Wolf optimization for IoT botnet detection. *Journal of Ambient Intelligence and Humanized Computing, 11*(7), 2809–2825.

Al-Gethami, K. M., Al-Akhras, M. T., & Alawairdhi, M. (2021). Empirical evaluation of noise influence on supervised machine learning algorithms using intrusion detection datasets. *Security and Communication Networks*, *2021*, 1-28.

Aljarah, I. (2019). Aljarrahcs/EvoloPy-FS [Python]. https://github.com/aljarrahcs/EvoloPy-FS (Original work published 2019)

Anthi, E., Williams, L., & Burnap, P. (2018). Pulse: An adaptive intrusion detection for the internet of things.

Ayad, A., Zamani, A., Schmeink, A., & Dartmann, G. (2019). Design and Implementation of a Hybrid Anomaly Detection System for IoT. https://doi.org/10.1109/IOTSMS48152.2019.8939206

Beigi, E. B., Jazi, H. H., Stakhanova, N., & Ghorbani, A. A. (2014). Towards effective feature selection in machine learning-based botnet detection approaches. *2014 IEEE Conference on Communications and Network Security*, 247–255.

Cameron-Jones, R. M. (1995). Instance selection by encoding length heuristic with random mutation hill climbing. *Eighth Australian Joint Conference on Artificial Intelligence,* 99–106.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357.

Demeter, D., Preuss, M., & Shmelev, Y. (2019, October 15). IoT: A malware story. SecureList By Kaspersky. https://securelist.com/iot-a-malware-story/94451/

Doshi, R., Apthorpe, N., & Feamster, N. (2018). Machine learning ddos detection for consumer internet of things devices. *2018 IEEE Security and Privacy Workshops (SPW)*, 29–35.

Faris, H., Hassonah, M. A., Ala'M, A.-Z., Mirjalili, S., & Aljarah, I. (2018). A multiverse optimizer approach for feature selection and optimizing SVM parameters based on a robust system architecture. *Neural Computing and Applications, 30*(8), 2355–2369. 51

Feingold, J. (2016, October 27). Dyn issues analysis of 'complex and sophisticated' cyberattacks. NH Business Review. https://www.nhbr.com/dyn-issues-analysisof-complex-and-sophisticated-cyberattacks/

Guerra-Manzanares, A., Medina-Galindo, J., Bahsi, H., & Nõmm, S. (2021). MedBIoT: Generation of an IoT Botnet Dataset in a Medium-sized IoT Network. 207–218. https://www.scitepress.org/Link.aspx?doi=10.5220/0009187802070218

Habib, M., Aljarah, I., & Faris, H. (2020). A Modified Multi-objective Particle Swarm Optimizer-Based Lévy Flight: An Approach Toward Intrusion Detection in Internet of Things. *Arabian Journal for Science and Engineering, 45*(8), 6081–6108.

Han, J., Pei, J., & Kamber, M. (2011). *Data Mining: Concepts and Techniques*. Elsevier.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In Neural networks for perception (pp. 65–93). Elsevier.

Kang, H., Ahn, D. H., Lee, G. M., Yoo, J. D., Park, K. H., & Kim, H. K. (2019). IoT network intrusion dataset. IEEE Dataport.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of ICNN'95-International Conference on Neural Networks, 4, 1942–1948.

Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity, 2*(1), 1– 22.

Kim, J., Shim, M., Hong, S., Shin, Y., & Choi, E. (2020). Intelligent Detection of IoT Botnets Using Machine Learning and Deep Learning. *Applied Sciences, 10*(19), 7009.

Koroniotis, N., Moustafa, N., Sitnikova, E., & Turnbull, B. (2018). Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset. ArXiv:1811.00701 [Cs]. http://arxiv.org/abs/1811.00701

Krebs, B. (2017, January 17). Who is Anna-Senpai, the Mirai Worm Author? Krebs on Security. https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-miraiworm-author/

Kumar, A., & Lim, T. J. (2019). EDIMA: Early detection of IoT malware network activity using machine learning techniques. 2019 *IEEE 5th World Forum on Internet of Things (WF-IoT)*, 289–294.

Labonne, M. (2020). Anomaly-based network intrusion detection using machine learning. Institut Polytechnique de Paris.

Mafarja, M., Heidari, A. A., Habib, M., Faris, H., Thaher, T., & Aljarah, I. (2020). Augmented whale feature selection for IoT attacks: Structure, analysis and applications. *Future Generation Computer Systems, 112*, 18–40.

McDermott, C. D., Majdani, F., & Petrovski, A. V. (2018). Botnet detection in the internet of things using deep learning approaches. *2018 International Joint Conference on Neural Networks (IJCNN), 52,* 1–8.

Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Breitenbacher, D., Shabtai, A., & Elovici, Y. (2018). N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders.

Mirjalili, S., Mirjalili, S. M., & Hatamlou, A. (2016). Multi-verse optimizer: A natureinspired algorithm for global optimization. *Neural Computing and Applications, 27(*2), 495–513.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. Advances in Engineering Software, 69, 46–61.

Mohamed, T., Otsuka, T., & Ito, T. (2018). Towards Machine Learning Based IoT Intrusion Detection Service. In M. Mouhoub, S. Sadaoui, O. Ait Mohamed, & M. Ali (Eds.), Recent Trends and Future Technology in Applied Intelligence (pp. 580–585). Springer International Publishing. https://doi.org/10.1007/978-3-319- 92058-0_56

Rathore, S., & Park, J. (2018). Semi-supervised learning based distributed attack detection framework for IoT. *Applied Soft Computing,* 72. https://doi.org/10.1016/j.asoc.2018.05.049

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics, 21*(3), 660– 674.

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM, 29*(12), 1213–1228.

Statista, I. H. S. (2018). Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions).

Ullah, I., & Mahmoud, Q. H. (2020a). A two-level flow-based anomalous activity detection system for IoT networks*. Electronics, 9*(3), 530.

Ullah, I., & Mahmoud, Q. H. (2020b). A Scheme for Generating a Dataset for Anomalous Activity Detection in IoT Networks. In C. Goutte & X. Zhu (Eds.), Advances in Artificial Intelligence (pp. 508–520). Springer International Publishing. https://doi.org/10.1007/978-3-030-47358-7_52

Wilson, D. R., & Martinez, T. R. (1997). Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research, 6*, 1–34.

Wilson, D. R., & Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning, 38*(3), 257–286.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation, 1*(1), 67–82.

Xiao, L., Wan, X., Lu, X., Zhang, Y., & Wu, D. (2018). IoT Security Techniques Based on Machine Learning. ArXiv:1801.06275 [Cs]. http://arxiv.org/abs/1801.06275