IoT Network Traffic Data Malware Detection

Eryk Lorenz
Colorado School of Mines
Golden, Colorado, USA

Abstract

Internet of Things (IoT) devices have been utilized to carry out numerous botnet attacks. With the number of IoT devices predicted to grow at an exponential rate so will the scale and frequency of botnet attacks with IoT devices [4]. This paper focuses on utilizing machine learning to detect malware in IoT devices by analyzing the network traffic data from the devices. The following algorithms are evaluated using the IoT-23 dataset: Naive Bayes, Support Vector Machine, Decision Tree, Random Forest, Multi-layer Perceptron, and Convolutional Neural Networks. The Decision Tree and Random Forest models demonstrate the best performance with accuracies of 99.997% and 99.999%, respectively. However, the two models display very different training times with Random Forest taking approximately 50 times longer to train than the Decision Tree model.

1 Introduction

Internet of Things (IoT) devices are susceptible for use in botnet attacks due to their weaker security and the shear number of IoT devices that are present in the world. One specific malware that has been used in over a dozen attacks is Mirai. One of the first, if not the first, attacks with Mirai was against the French web host OHV which was estimated to use over 145,000 devices in the attack [5]. Mirai was also used to attack Dyn which caused outages for GitHub, Twitter, Reddit, Netflix, and Airbnb [14]. Mirai works by using existing infected devices to replicate the malware and infect other IoT devices. Then, a server will send commands to all infected devices and can use them to conduct a Distributed Denial of Service (DDoS) attack. With an estimation stating that there will be close to 30 billion IoT devices by 2030, the scale of botnets that could be used in attacks quickly becomes very alarming [4]. While securing IoT devices against the vulnerabilities exploited by malware is the best solution, vulnerabilities will always exist. The next best solution for combating the creation of massive botnets is the ability to detect devices that are infected. The work in this paper focuses on detecting the presence of malware in IoT devices through network traffic data. This approach poses no additional computational or memory loads on the IoT devices, which is key as computational power and memory are limited resources. This paper will focus on comparing the performance of Naive Bayes, Support Vector Machine, Decision Tree, Random Forest, Multi-layer Perceptron, and Convolutional Neural Network algorithms on the detection of malware in IoT device network traffic data.

2 Related Work

The idea of utilizing machine learning with network traffic data has been investigated in many other works for IoT devices. Woźniak et al. [15] proposed a Recurrent Neural Network and were able to achieve an accuracy of 99.9996% when testing in a real world scenario. Classical neural networks were also evaluated but did not achieve an accuracy higher than 71.11%. Du et al. [3] took a different approach by capturing network traffic packets and transforming them to images. After the packets were transformed to images, an object detection model was then used to identify malware in the packets. The model was created with the YOLO algorithm which utilizes a Convolutional Neural Network. Through this approach the authors were able to achieve a mean average precision of 0.9945. Shafiq et al. [12] developed a new feature selection algorithm to filter the data and select features that carry enough information for the model. After developing that algorithm it was applied to four different machine learning algorithms: Decision Tree, Support Vector Machine, Naive Bayes, and Random Forest and were able to achieve accuracies as high as 99.9%.

Previous work has already also been done applying machine learning models specifically to the IoT-23 dataset to detect malware in network traffic data. Stoain [13] applied Decision Tree, Random Forest, Naive Bayes, Support Vector Machine, Artificial Neural Network, and AdaBoost algorithms to the IoT-23 dataset and was able to achieve F1 scores as high as 0.998. However, technical challenges resulted in a subset of the data being utilized with many of the fields also being dropped. The work in this paper will focus on overcoming those technical challenges to determine if the dropped fields and reduction in data had any significant impact on the performance of the models.

3 Methodology

3.1 Dataset

The IoT-23 dataset [6] was utilized for training and validation. It consists of network traffic data captures for 20 malicious scenarios and 3 benign scenarios. All the captures are from Stratosphere Laboratory, AIC group, FEL, CTU University, Czech Republic and were captured between 2018 and 2019.

The full dataset from all the malicious scenarios consists of 325,307,990 labeled packet captures. A breakdown of the number of captures for each type of attack can be seen in Table 1 and descriptions for each attack type can be found in Section 3.1.1.

Label	Count
Attack	9,398
Benign	30,858,735
C&C	21,995
C&C-FileDownload	53
C&C-HeartBeat	33,673
C&C-HeartBeat-Attack	834
C&C-HeartBeat-FileDownload	11
C&C-Mirai	2
C&C-PartOfAHorizontalPortScan	888
C&C-Torii	30
DDoS	19,538,713
FileDownload	18
Okiru	47,381,241
Okiru-Attack	13,609,470
PartOfAHorizontalPortScan	213,852,924
PartOfAHorizontalPortScan-Attack	5
Total	325,307,990

Table 1. Count of Attacks in IoT-23 Dataset

For each packet captured there were 23 fields of data. A brief description for each of the fields can be found in Table 2.

Field	Description
ts	Timestamp of capture
uid	ID of capture
id.orig_h	IP Address for origin device
id.orig_p	Port for origin device
id.resp_h	IP address for response device
id.resp_p	Port for response device
proto	Network Protocol (TCP or UDP)
service	Application Protocol
duration	Length of Connection
orig_bytes	Bytes sent by origin device
resp_bytes	Bytes sent by response device
conn_state	State of Connection
local_orig	Boolean of if origin is local
local_resp	Boolean of if response is local
missed_bytes	Missed bytes in message
history	History of state of connection
orig_pkts	Packets sent by origin device
orig_ip_bytes	Bytes sent by origin device
resp_pkts	Packets sent by response device
resp_ip_bytes	Bytes sent by response device
tunnel_parents	ID of tunneled connections
label	Benign or Malicious
detailed_label	Detailed label of attack type

Table 2. Descriptions of Fields in Dataset

3.1.1 Description of Attack Types.

Each of the attacks are defined by the IoT-23 dataset as:

Attack: "This label indicates that there was some type of attack from the infected device to another host. Here we are labeling as attack to any flow that, by analyzing its payload and behavior, tries to take advantage of some vulnerable service. For example, a brute force to some telnet login, a command injection in the header of a GET request, etc." [6].

Benign: "This label indicates that no suspicious or malicious activities where found in the connections" [6].

C&C: "This label indicates that the infected device was connected to a CC server. This activity was detected in the analysis of the network malware capture because the connections to the suspicious server are periodic or our infected device is downloading some binaries from it or some IRC like or decoded orders are coming and going from it" [6].

DDoS: "This label indicates that a Distributed Denial of Service attack is being executed by the infected device. These traffic flows are detected as part of a DDoS attack because of the amount of flows directed to the same IP address" [6].

FileDownload: "This label indicates that a file is being downloaded to our infected device. This is detected by filtering connections with response bytes more than 3KB or 5KB, normally this is combined with some known suspicious destination port or destination IP known to be a C&C server" [6].

HeartBeat: "This label indicates that packets sent on this connection are used to keep a track on the infected host by the C&C server. This was detected by filtering connections with response bytes lower than 1B and with periodic similar connections, normally this is combined with some known suspicious destination port or destination IP known to be a C&C server" [6].

Mirai: "This label indicates that the connections have characteristics of a Mirai botnet. This label is added when the flows has similar patterns as the most common known Mirai attacks" [6].

Okiru: "This label indicates that the connections have characteristics of a Okiru botnet. This labeling decision was made with the same parameters as with Mirai but with the difference that this botnet family is less common" [6].

PartOfAHorizontalScan: "This label indicates that the connections are used to do a horizontal port scan to gather information to perform further attacks. To put these labels we rely on the pattern in which the connections shared the same port, a similar number of transmitted bytes and multiple different destination IPs" [6].

Torii: "This label indicates that the connections have characteristics of a Torii botnet. This labeling decision was made with the same parameters as with Mirai but with the difference that this botnet family is less common" [6].

3.2 Data Preprocessing

The original IoT-23 dataset [6] was separated into 23 separate files. The first step in preprocessing the data was combining the 23 files into a single file. This was done by loading the first 3,000,000 records from each file into a dataframe and then concatenating those dataframes into a single dataframe. It was chosen to take the first 3,000,000 records as this was determined to be the largest amount of data that could be loaded into memory and still have enough memory remaining to train the models. This resulted in a dataset with 37,211,524 packets compared to the 325,307,990 packets in the original dataset. Thus, the dataset utilized was roughly 10% of the original data.

Once the data was combined into a single dataframe two fields were dropped. *tunnel_parents* was dropped as almost every single packet did not have any data for this field. *label* was also dropped as the models were focused on classifying the *detailed_label* field and thus the *label* field should not be included in the training data as it is not part of the original packet capture data.

The next step in preprocessing the data was to fill in any remaining missing values. For the *detailed_label* field any missing values were filled in with the Benign label. For all other fields missing values were filled in with 0.

Looking at the counts of each label in the full dataset, as shown in Table 1, shows that many of the labels are only present a few times in the dataset. To alleviate that issue and improve performance of the multi classification models it was then chosen to combine many of the labels that are similar to each other. The labels were combined based on the primary, or first, listed type of attack. The mapping for combining labels is shown in Table 3. The dataset with the updated labels was saved as a CSV file and then utilized for training all the models.

Original Label	Combined Label
Attack	Attack
Benign	Benign
C&C	C&C
C&C-FileDownload	C&C
C&C-HeartBeat	C&C
C&C-HeartBeat-Attack	C&C
C&C-HeartBeat-FileDownload	C&C
C&C-Mirai	C&C
C&C-PartOfAHorizontalPortScan	C&C
C&C-Torii	C&C
DDoS	DDoS
FileDownload	FileDownload
Okiru	Okiru
Okiru-Attack	Okiru
PartOfAHorizontalPortScan	PartOfAHorizontalPortScan
PartOfAHorizontalPortScan-Attack	PartOfAHorizontalPortScan

Table 3. Mapping of Combined Labels

The final dataset resulted in a slightly different distribution of labels when compared to the original full dataset. The comparison of label counts and distributions are shown in Table 4. It is important to note that the original dataset was very imbalanced. The processed dataset, while still not balanced, is closer to balanced than the original dataset was.

Label	Full Data	Processed Data
Attack	9,398	8,446
Benign	30,858,735	4,515,748
C&C	56,598	24,216
DDoS	19,538,713	3,761,566
FileDownload	18	18
Okiru	60,990,711	7,882,149
PartOfAHorizontalScan	213,853,817	21,019,381
Total	325,307,990	37,211,524

(a) Counts of Labels in Full Dataset and Processed Dataset

Label	Full Data	Processed Data
Attack	0.00289%	0.02270%
Benign	9.48601%	12.13535%
C&C	0.01740%	0.06508%
DDoS	6.00622%	10.10861%
FileDownload	0.00001%	0.00005%
Okiru	18.74861%	21.18201%
PartOfAHorizontalScan	65.73888%	56.48621%

(b) Distribution of Labels in Full Dataset and Processed Dataset

Table 4. Comparison of Label Frequency in Full Dataset and Processed Dataset

3.3 Hardware

The experiments were ran on a desktop computer running Linux Mint 21 with kernel 5.15.0-53-generic. The CPU was an AMD Ryzen 5600X, the GPU was an AMD Radeon 6700XT, and 32 GB of RAM at 3,600MHz.

3.4 Performance Metrics

The models were evaluated on the following performance metrics: Precision, Recall, F1 score, and Matthews Correlation Coefficient (MCC). For each of the metrics except MCC, both a macro average and a weighted average are presented. However, due to the imbalance of the dataset, the weighted average was focused on more than the macro average.

3.4.1 Precision.

Precision is the measure of how well a model does not label or classify a sample as positive when the sample is actually negative. The equation for precision is shown in Equation 1 where TP is the number of true positives and FP is the number of false positives.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

3.4.2 Recall.

Recall is a measure of how well a model is able to identify all the positive samples. The equation for recall is shown in Equation 2 where TP is the number of true positives and FN is the number of false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

3.4.3 F1 Score.

The F1 score is the harmonic mean of Precision and Recall, with equal contributions from both. Thus, it is a way to combine both the precision and recall measurements into a single metric. The equation for the F1 score is shown in Equation 3 where P is precision and R is recall.

$$F1 = 2 * \frac{P * R}{P + R} \tag{3}$$

3.4.4 Matthews Correlation Coefficient.

The Matthews Correlation Coefficient is a measure that takes into account both true and false positives and negatives and works well even with unbalanced classes, as there are with this dataset. It shows the correlation between the observed and predicted classifications where score of 1 is perfect, 0 is no better than random, and -1 is complete disagreement. The equation is shown in Equation 4 where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4}$$

3.5 Hyperparameter Tuning

The hyperparameters of a model, or the parameters that are passed into the model instead of learned, are key to the performance of the model. A grid search was done for each of the models to optimize and find the best values for each parameter. A grid search is performed by specifying a list of values for each parameter and then performing an exhaustive search through all possible combinations of defined values to find the best performing model. Due to the large number of possible candidates for each model, multiple candidate models were evaluated in parallel to reduce the time required to find the optimal set of parameters. However, it was not possible to train multiple models in parallel with the full scale preprocessed dataset due to memory limitations. Instead, a dataset $\frac{1}{5}$ the size was used and then the three best candidates for each model were evaluated on the full scale dataset to ensure the optimal parameters scaled with the dataset. The optimal parameters for each model can be found in their respective subsection of Section 3.6.

3.6 Models

The algorithms evaluated in this paper are Naive Bayes (NB), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN). All algorithms, except CNN, were implemented using the Scikit-learn version 1.3.1 library. The options for each parameter can be found in the API Reference of the library [10]. The CNN model was implemented using TensorFlow version 2.15.0.

3.6.1 Naive Bayes.

Naive Bayes algorithms are based on the naive assumption of independence between all pairs of features given the value of the label. The classification rule for a Naive Bayes classification model is defined by:

$$\hat{y} = max(P(y) \prod_{i=1}^{n} P(x_i|y)) \tag{5}$$

For this paper Gaussian Naive Bayes was utilized which assumes a Gaussian distribution for the likelihood of features. With a Gaussian Naive Bayes algorithm $P(x_i|y)$ is defined as [8]:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(-\frac{(x_i - mu_y)^2}{2\sigma_y^2}) \tag{6}$$

The parameters used in the final model are shown in Table 5.

Metric	Value	Description
$var_{smoothing}$	$1 * 10^{-15}$	Portion of the largest variance of all features that is added to variances for calculation stability

Table 5. Hyperparameters for Naive Bayes Model

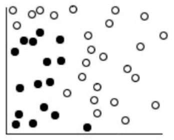
3.6.2 Support Vector Machine.

Support Vector Machines map all input data passed in to a highly dimensional space and then create separators between the different classifications. The data is then transformed to create a hyperplane for boundaries between each classification. This process, referenced from IBM [2], is shown in Figure 1. The transformation of the data is controlled by

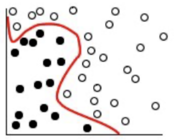
Metric	Value	Description
dual	"auto"	Selects the algorithm to either solve the dual or primal optimization problem

Table 6. Hyperparameters for Support Vector Machine Model

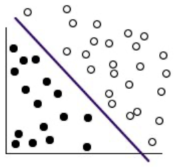
the kernel. A linear kernel was chosen as the default rbf kernel in the Scikit-learn implementation scales at minimum quadratically with the number of samples while the linear kernel scales substantially better [11]. The parameters used in the final model are shown in Table 6.



(a) Original Dataset



(b) Data with Separator Added



(c) Transformed Data

Figure 1. Support Vector Machine Operations [2]

3.6.3 Decision Tree.

Decision Trees create decision rules in the form of a tree that lead to a classification of the data. One of the major benefits of a decision tree is that the cost of predicting with the tree is logarithmic with the size of the training data [7]. The parameters used in the final model are shown in Table 7.

Metric	Value	Description
criterion	"entropy"	The function to measure the quality of a split
$max_{features}$	"sqrt"	The number of features to consider when looking for the best split

Table 7. Hyperparameters for Decision Tree Model

3.6.4 Random Forest.

Random Forest models are a collection of Decision Trees where each tree is comprised of a random data sample from the training data. One of major benefits of Random Forest over Decision Tree is that Random Forest better accounts for variability in the data which reduces the risk of overfitting and bias [1]. However, due to the model being a collection of Decision Trees, the costs are higher than Decision Trees. The parameters used in the final model are shown in Table 8.

Metric	Value	Description
criterion	"gini"	The function to measure the quality of a split
$max_{features}$	"log2"	The number of features to consider when looking for the best split
$n_{estimators}$	100	The number of trees in the forest

Table 8. Hyperparameters for Random Forest Model

3.6.5 Multi-Layer Perceptron.

Multi-Layer Perceptron models have an input and an output layer with at least one hidden layer in the middle where the layers are composed of neurons. Each neuron has a weight associated with it. The summation of neurons are then passed to an activation function and then to the next layer in the model. During training the weights of neurons are adjusted through backpropogration and the models "learns" the optimal weight for each neuron from the training data [9]. An example of a single hidden layer MLP model is shown in Figure 2. The parameters used in the final model are shown in Table 8.

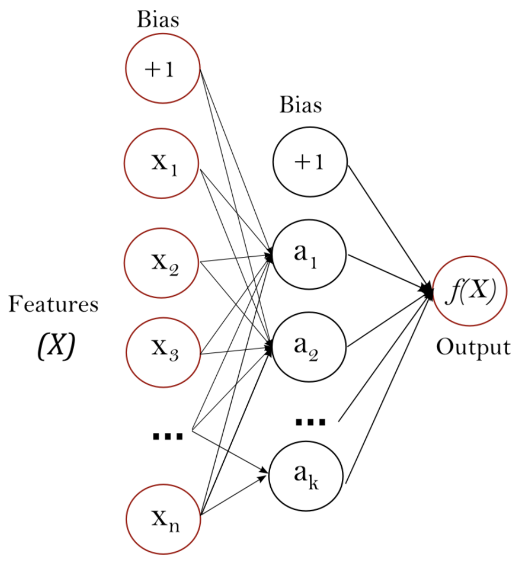


Figure 2. Example of a Single Hidden Layer Multi-Layer Perceptron Model [9]

Metric	Value	Description
activation	"relu"	Activation function for hidden layer
alpha	$1 * 10^{-12}$	Strength of the L2 regularization term
hidden layer sizes	200	Number of neurons in hidden layers
solver	"adam"	Solver for weight optimization

Table 9. Hyperparameters for Multi-Layer Perceptron Model

3.6.6 Convolutional Neural Network.

Convolutional Neural Networks are extremely similar to MLP models. They both consist of neurons and utilize backpropogation to learn the optimal weights of the neurons. However, CNNs consist of a variety of layer types such as convolutional layers, pooling layers, fully connected or dense layers, and dropout layers. The model utilized in this paper consists of dense. The dense layers are fully connected to the next layer and the dropout layers. The dropout layer randomly sets a number of the input units to 0. This is done to reduce the chance of overfitting to the data. The layers and their activation functions of the model utilized in this paper are shown in Figure 3 and the hyperparameters are shown in Table 10.

Metric	Value	Description
Epochs	10	Number of Epochs to Train
Batch Size	256	Number of Samples per Gradient Update
Loss	"categorical crossentropy"	Name of Loss function
Optimizer	"Adam"	Name of Optimizer

Table 10. Hyperparameters for Convolutional Neural Network Model

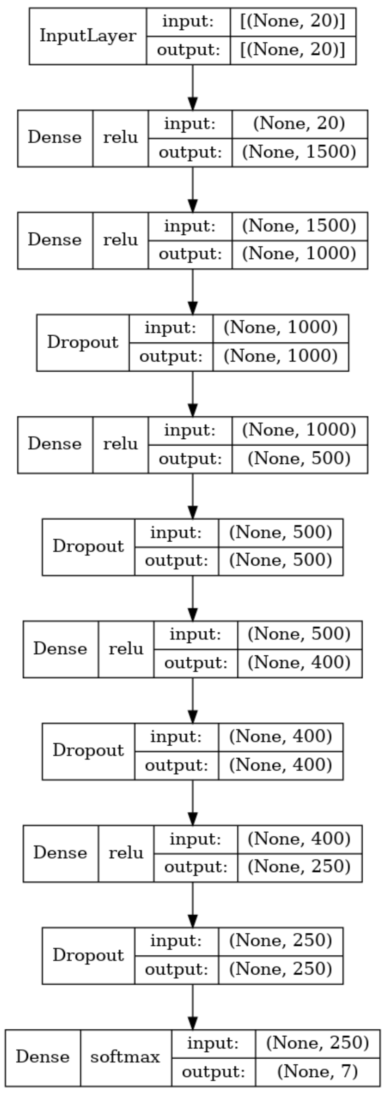


Figure 3. Layers of CNN Model

4 Results

4.1 Naive Bayes

The results for the Naive Bayes model are shown in Table 11. The two labels that had the worst performance are C&C and FileDownload. These two labels also had small percentage of labels in the dataset. In contrast, the heavier weighted labels in the dataset, like Okiru and PartOfAHorizontalPortScan, displayed the best performance. The one outlier was the Benign label, which is one of the most present labels, with a high precision but an extremely low recall. However, overall the model performed well. This can be seen by the Matthews Correlation Coefficient (MCC) score of 0.729. Based on the strong performance of the heavier weighted labels it can be expected that a better balanced dataset would result in better overall performance. This is supported by the large difference between the macro and weighted averaged F1 scores.

4.2 Support Vector Machine

The results for the Support Vector Machine model are shown in Table 12. Similarly to the Naive Bayes model, the SVM model showed strong performance for the labels that were heavily weighted in the dataset and poor performance for the labels that were lacking in the dataset. However, this issue was even worse for the SVM model with 3 labels having F1

Label	Precision	Recall	F1
Attack	0.441	0.856	0.582
Benign	1.000	0.053	0.101
C&C	0.075	0.182	0.106
DDoS	0.659	0.930	0.771
FileDownload	0.153	0.800	0.258
Okiru	1.000	1.000	1.000
PartOfAHorizontalScan	0.819	0.924	0.868
Macro Average	0.593	0.678	0.527
Weighted Average	0.862	0.834	0.793
Accuracy			83.432%
Matthews Correlation Coefficient			0.729
Time (s)			37.637

Table 11. Naive Bayes Performance Metrics

scores of 0. Also, similar to the Naive Bayes model, Benign once again had high precision and low recall.

Label	Precision	Recall	F1
Attack	0.000	0.000	0.000
Benign	0.675	0.186	0.291
C&C	0.000	0.000	0.000
DDoS	0.833	0.797	0.814
FileDownload	0.000	0.000	0.000
Okiru	0.730	0.820	0.772
PartOfAHorizontalScan	0.797	0.891	0.841
Macro Average	0.433	0.385	0.388
Weighted Average	0.770	0.780	0.757
Accuracy			78.023%
Matthews Correlation Coefficient			0.629
Time (s)			1,282.9

Table 12. Support Vector Machine Performance Metrics

4.3 Decision Tree

The results for the Decision Tree model are shown in Table 13. The Decision Tree model displayed very strong performance for all labels, even the labels that were sparsely represented throughout the dataset. Similarly to both of the previously presented models, FileDownload was one of the lowest F1 scores. However, unlike the other models, FileDownload still showed strong performance for the Decision Tree model with an F1 score of 0.889. Overall, Decision Tree had extremely strong performance achieving an F1 score and MCC both of 1 and an accuracy of 99.997%.

Label	Precision	Recall	F1
Attack	0.996	0.997	0.997
Benign	1.000	1.000	1.000
C&C	0.999	0.997	0.998
DDoS	1.000	1.000	1.000
FileDownload	0.571	0.800	0.667
Okiru	1.000	1.000	1.000
PartOfAHorizontalScan	1.000	1.000	1.000
Macro Average	0.938	0.971	0.952
Weighted Average	1.000	1.000	1.000
Accuracy			99.997%
Matthews Correlation Coefficient			1.000
Time (s)			79.781

Table 13. Decision Tree Performance Metrics

4.4 Random Forest

The results for the Random Forest model are shown in Table 14. Like all the other models, the lowest performing label was FileDownload with an F1 score of 0.889. However, Attack and C&C both performed well with F1 scores of 0.999 and 1, respectively. Similar to the Decision Tree model, the Random Forest model displayed exceptional performance with an MCC score and F1 score of 1 and an accuracy of 99.999%. However, the training time for the Random Forest model took almost 50 times as long for the Random Forest model compared to the Decision Tree model.

Label	Precision	Recall	F1
Attack	1.000	0.999	0.999
Benign	1.000	1.000	1.000
C&C	1.000	1.000	1.000
DDoS	1.000	1.000	1.000
FileDownload	1.000	0.800	0.889
Okiru	1.000	1.000	1.000
PartOfAHorizontalScan	1.000	1.000	1.000
Macro Average	1.000	0.971	0.984
Weighted Average	1.000	1.000	1.000
Accuracy			99.999%
Matthews Correlation Coefficient			1.000
Time (s)			3,930.923

Table 14. Random Forest Performance Metrics

4.5 Multi-Layer Perceptron

The results for the Multi-Layer Perceptron model are shown in Table 15. It can be seen that the MLP model struggled for all labels except for Benign, PartOfAHorizontalPortScan, and Okiru. Benign and Okiru both had exceptional precision but extremely poor recall. The imbalance in the training data is causing the model predict PartOfAHorizontalPortScan nearly every time which is evident by the recall of 1 for that label and recall of 0 for all other labels.

Label	Precision	Recall	F1
Attack	0	0	0
Benign	0.995	0	0
C&C	0	0	0
DDoS	0	0	0
FileDownload	0	0	0
Okiru	1.000	0	0
PartOfAHorizontalScan	0.565	1.000	0.722
Macro Average	0.366	0.143	0.103
Weighted Average	0.652	0.565	0.408
Accuracy			56.514%
Matthews Correlation Coefficient			0.007
Time (s)			1,016.884

Table 15. Multi-Layer Perceptron Performance Metrics

4.6 Convolutional Neural Network

The results of the Convolutional Neural Network model are shown in Table 16. TensorFlow does not provide support for viewing the precision, recall, and F1 metrics at a per class level. As a result only the metrics for the entire model are presented. However, based on the low macro averaged F1 score and higher averaged micro F1 score similar results to the MLP model can be assumed. The model is able to successfully predict PartOfAHorizontalPortScan as it is the most present label in the dataset but is not able to predict the other labels especially the labels that are only present in a small percentage of the dataset. However, the CNN model was able to recognize many of the labels better than the MLP model. This is evident by the substantially higher MCC score.

Metric	
Loss	1.145
Macro Averaged F1	0.103
Weighted Averaged F1	0.408
Matthews Correlation Coefficient	0.493
Accuracy	56.5%
Training Time (s)	12,867.4

Table 16. Convolutional Neural Network Performance Metrics

4.7 Comparison of Models

A high level comparison of the results for the models are shown in Table 17. When comparing the results it is clear that two models outperformed the rest. Both the Decision Tree and Random Forest models performed exceptionally well with the Random Forest model slightly outperforming the Decision Tree model. However, Random Forest took roughly 50 times as long to train. Depending on if the slight increase in accuracy or the faster training time is preferred either model could be argued as being the best model. In contrast, both the Multi-layer Perceptron and Convolutional Neural Network models had the lowest performance with the highest training times. The poor performance for both of those models can be attributed to the poorly balanced dataset. Additional neurons were added in an attempt to improve performance. However, while this did result in small performance increases, it also rapidly caused the training times of both models to increase.

	Classifier					
Metric	NB	SVM	DT	RF	MLP	CNN
F1	0.793	0.757	1.000	1.000	0.408	0.408
Accuracy (%)	83.432	78.023	99.997	99.999	56.514	56.5
MCC	0.729	0.629	1.000	1.000	0.007	0.493
Time (s)	37.7	1,282.9	79.8	3,930.9	1,016.9	12,867.4

Table 17. Performance Comparison of Models

4.8 Analysis of Performance Improvement with Additional Fields

As mentioned previously, the work done by Stoain [13] encountered technical issues and had to reduce the scope of the dataset. A comparison of the results from this work with the work done by Stoain are presented in Table 18. The Random Forest model did not have any noticeable improvements in performance with the inclusion of additional data and fields. However, all results are presented in the table for the Random Model have values of 1, so improvement was not possible. The Support Vector Machine did show some improvement with the inclusion of additional data as the F1 score increased from 0.59 to 0.76. The Naive Bayes model showed the largest improvement from the inclusion of additional data with the F1 score increasing substantially from 0.25 to 0.79. This highlights that the additional data did have a considerable impact on improving performance. However, that increased performance also comes at an increase in training costs. Stoain [13] did not present any metrics for training time, so the actual increases in training times cannot be evaluated.

4.9 Correlation of F1 Score and Label Frequency in the Dataset

Consistently across the models evaluated it was observed that the higher performing labels were the ones that were

Metric	NB		SVM		RF	
	This Work	Stoain	This Work	Stoain	This Work	Stoain
Precision	0.86	0.76	0.77	0.60	1.00	1.00
Recall	0.83	0.23	0.78	0.67	1.00	1.00
F1	0.79	0.25	0.76	0.59	1.00	1.00

Table 18. Comparison of Results with Work by Stoain [13]

more frequently present in the dataset and the poorer performing labels were the ones that were less frequently present in the dataset. Table 19 shows the correlation between individual label F1 scores and the frequency of the attacks for each model. A hypothesis test was conducted, with $\alpha = 0.1$, to determine if there was a statistically significant correlation. The two worst performing models, Support Vector Machine and Multi-Layer Perceptron, both showed a statistically significant correlation between the F1 score and the frequency of labels in the dataset. The Naive Bayes model, while not statistically significant, did show a fairly strong correlation between F1 scores and frequency of each label. In contrast, the two best performing models, Decision Tree and Random Forest, displayed a substantially weaker correlation between F1 scores and frequency of each label.

Model	F1	Correlation	P-value	Significant
NB	0.793	0.587	0.166	No
SVM	0.757	0.744	0.055	Yes
DT	1.000	0.317	0.489	No
RF	1.000	0.315	0.492	No
MLP	0.408	0.919	0.003	Yes

Table 19. Correlation between Label Weighted F1 Scores and Label Frequency for each Model

5 Future Work

As previously stated, one issue that was encountered with all models during training was poor performance for labels that were weakly represented in the dataset, especially C&C and FileDownload. This issue was most apparent in the MLP and CNN models. One possible direction for future research is to attempt to mitigate this issue and reevaluate the models. One possible approach to this is to attempt to subset the data differently and better balance the labels. Another approach to mitigating this issue would be to weight the data such that the weakly represented labels have each packet be more heavily weighted.

Another area where future work could be conducted is with real world testing of the models. The models were not evaluated outside of the verification done during the training process. Each model could be deployed to a lab environment, with both the devices present in the dataset and other IoT devices, containing malware and test if the models are able to detect malware within the traffic data. One proposed question for this future work would be how well do the models perform on traffic data from devices that were not included within the training data. Do the models perform similar when evaluating traffic data from devices they were not trained on or is it necessary to train the models with traffic from all devices that would be encountered in a real world scenario?

6 Conclusion

One of the keys to addressing IoT botnets is the ability to detect malware in IoT devices. The work in this paper built upon previous work by utilizing machine learning to detect different malwares through network traffic data of IoT devices utilizing the IoT-23 dataset. The Decision Tree and Random Forest models were able to detect the different malwares the best with Random Forest slightly outperforming Decision Tree. However, the Random Forest model took substantially longer to train than the Decision Tree model did. Also, the Naive Bayes and Support Vector Machine models demonstrated large increases in performance compared to previous work. This increase in performance is attributed to overcoming previous technical limitations that severely limited the amount of the IoT-23 dataset that could be used. One of the main issues that was consistent across the models was poor performance on the malwares that were sparsely represented in the dataset. It is recommended that in the future the work of this paper is continued with efforts to mitigate the impacts of the large imbalance of the dataset. Another recommendation in the future is the evaluation of the models in a real world scenario.

References

[1] IBM Corporation. What is random forest?

[2] IBM Corporation. How svm works, Aug 2017.

[3] Chunlai Du, Shenghui Liu, Lei Si, Yanhui Guo, and Tong Jin. Using object detection network for malware detection and identification in network traffic packets. *Computers, Materials & Continua*, 64(3):1785–1796, 2020.

[4] Fabio Duarte. Number of iot devices, 2023.

[5] Center for Internet Security. The mirai botnet: Threats and mitigations, Jul 2021.

[6] Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. Iot-23: A labeled dataset with malicious and benign iot network traffic (version 1.0.0), 2020.

[7] scikit-learn developers. Decision trees, 2023.

[8] scikit-learn developers. Naive bayes, 2023.

[9] scikit-learn developers. Neural network models, 2023.

[10] scikit-learn developers. scikit-learn 1.3.2 api reference, 2023.

[11] scikit-learn developers. Support vector classification, 2023.

[12] Muhammad Shafiq, Zhihong Tian, Ali Kashif Bashir, Xiaojiang Du, and Mohsen Guizani. Corrauc: A malicious bot-iot traffic detection method in iot network using machine-learning techniques. *IEEE internet of things journal*, 8(5):3242–3254, 2021.

[13] Nicolas-Alin Stoain. Machine learning for anomaly detection in iot networks: Malware analysis on the iot-23 data set, 2020.

[14] Chris Williams. Today the web was broken by countless hacked devices, Oct 2016.

[15] Marcin Wozniak, Jakub Silka, Michal Wieczorek, and Mubarak Alrashoud. Recurrent neural network model for iot and networking malware threat detection. *IEEE transactions on industrial informatics*, 17(8):5583–5594, 2021.