Machine Learning for Anomaly Detection in IoT networks: Malware analysis on the IoT-23 Data set

Nicolas-Alin Stoian
University of Twente
PO Box 217, 7500 AE Enschede
the Netherlands
n.stoian@student.utwente.nl

Abstract

The Internet of Things is one of the newer developments in the domain of the Internet. It is defined as a network of connected devices and sensors, both physical and digital, that generate and exchange large amounts of data without the need for human intervention. As a result of eliminating the need for human operators, the IoT (Internet of Things) can process more data than ever before faster and more efficient.
This paper focuses on the security aspect of IoT networks by investigating the usability of machine learning algorithms in the detection of anomalies found within the data of such networks. It examines ML algorithms that are successfully utilized in relatively similar situations and compares using a number of parameters and methods.
This paper implements the following algorithms: Random Forest (RF), Naïve Bayes (NB), Multi Layer Perceptron (MLP), a variant of the Artificial Neural Network class of algorithms, Support Vector Machine (SVM) and AdaBoost (ADA). The best results were achieved by the Random Forest algorithm, with a accuracy or 99.5%.

Keywords
Internet of Things, anomaly detection, machine learning, IoT-23, malware analysis

Introduction

First described in 1991 as the 'Computer of the 21st century' [19], the Internet of Things (abbreviated as IoT) is the concept of connecting numerous devices to a network which is used to transfer data between them, all happening automatically, without the need for human intervention [16]. While the idea is already 30 years old at this point, its true development only started around 10 years ago, when the number of IoT devices in the world became larger than the number of people [6]. Since then, advancements in fields such as cloud computing or data analytics, along with the increase in hardware power have added new dimensions to the concept, turning it into what it is today [16]. One of the technologies that also benefits from these same advancements is machine learning, the use of artificial intelligence in order to create system that can learn by themselves, without the need for explicit programming [7].
At the moment, the biggest concern for more almost a half of the potential users of IoT systems is security [4]. Thus, in the last few years, researchers have started looking into more advanced security measures to tackle this issue. There are two main categories of security measures: passive (e.g. passwords, encryption) and active. One of those new measures is the use of machine learning to detect attacks and classify them, [20] as in theory, the two technologies seem well fit for each other. Machine learning algorithms require large quantities of data in order to build its detection model, and IoT systems can provide them. Additionally, the sheer number of types of attacks and their manifestations makes identifying and categorising them near impossible for human operators [20].
The main goal of this paper is to develop Machine Learning algorithms to be used in network-based anomaly detection in Internet of Things devices, and then test them using the IoT-23 dataset [1], a new dataset consisting of both malicious and benign network captures from a number of IoT devices.
The research question that this paper will attempt to answer is:

What are the best machine learning algorithms for detecting anomalies produced by IoT devices?

Additionally, the results of this research will be compared to other similar research papers, so a secondary research question is:

How do the algorithms tested on the IoT-23 dataset fare in comparison with algorithms tested on similar data sets?

The following approach will be taken to for this paper:

1. The dataset will be visualized, analysed and fitted to the purposes of this paper
2. The algorithms will be implemented and tested on the IoT-23 dataset
3. The final results will be discussed and compared to similar studies

The structure of this study is as follows:

- Section 3 will review literature relevant to this project.
- Section 4 will discuss the methods used to gather results. The dataset section (4.1) will present the data set and its features. Next, sections 4.2, 4.3 and 4.4 will discuss the way pre-processing of the dataset was done. Section 4.5 describes the theory behind the algorithms, while section 4.6 describes the theory behind the metrics that will be used to compare the results of the algorithms.
- Section 5 presents the results and discusses them.
- Section 6 will provide the conclusions of the research, along with its limitations and will propose future research ideas.

Literature Review

At the moment, the use of machine learning is still in its incipient phases. So far, some frameworks have been developed for this idea [20], while other research focused on implementing and testing the idea [9]. According to Zeadally and Tsikerdekis, 2019 [20], the idea of using machine learning algorithms is relatively new and has clear potential due to a multitude of factors: the devices are less complex than traditional systems, which in turn makes them more predictable and data is easy to come by. There are, however, a few difficulties at the moment, such as the portability of the algorithms. There is also the general problem of simply bypassing layers of security by exploiting other weaknesses in IoT networks. In summary, machine learning should be seen as another layer of security for IoT networks, not as a general solution.
Also according to them, there are two major ways of implementing ML algorithms in IoT networks, network based, by using metadata from the IoT network, or host based, by using the information present on the device.
This project will be focused on a network based implementation.
Shafiq, Tian, Sun et al., 2020 [17] tested 44 features trying to find a framework model for testing attack detection algorithms. For that, they used the Bot-IoT dataset [10]. Their final results were that the best four metrics are the true positives rate (TPRate), the precision, the accuracy and the time taken to build the model. By using those metrics and the Bot-IoT dataset, an implementation of Naïve Bayes was the best algorithm according to their framework.
After discussing how the use of machine learning for anomaly detection is faring so far, and how measuring the results of its use should be done, next up a few projects similar to this research will be discussed for comparative analysis. Hasan, Islam, Zerif et al., 2019 [9] have performed somewhat similar work to this project, but went one step further, by using machine learning algorithms first to detect whether the system is performing abnormally, and if it is, they are using algorithms to detect the type of attack the device is under. For their research, they used the open-source DS2OS dataset [13]. In their case, the Random Forest algorithm was the best choice, with an accuracy of 99.4%, followed by an artificial neural network with the same percentage, but lower scores on other metrics.
Anthi, Williams and Burnap, 2018 [2] proposed a novel model for a network-based real-time malware detection system called Pulse. In their research, an implementation of Naïve Bayes served as the most performing classifier for the proposed model with a precision between 81% and 97.7%, depending on the type of attack.
Lastly, Revathi and Malathi [14] discuss the results they obtained using the NSL-KDD dataset [18] in 2013. In their paper, the Random Forest algorithm obtained by far the most consistent results.
By doing some meta-analysis of this literature review, it can be seen that the use of machine learning on IoT networks is a very recent development, with all the papers being less than 3-years old. Also, it should be noted that even when more complex algorithms, such as neural networks, are used, most of the studies found the best results are coming from algorithms such as Naïve Bayes and Random Forest.

Methods

This part of the paper concerns the data set, the way it was pre-processed, and theoretical discussions about the algorithms and the measurements used in this project.
The first big step is data preprocessing, which consists of data selection, data visualization, data formatting, statistical correlation and data splitting. These steps processed the data so it could be fed into the algorithms.
The data was split randomly in a 80-20 ratio, with the 20 percent becoming the training data and the 80 percent becoming the testing data. All the algorithms are

of the type multi-class. Lastly, the algorithms were compared on accuracy, the f1-score, the recall score and the support score.

4.1 Dataset

The data set used in this project is IoT-23 [1], a dataset created by the Avast AIC laboratory. The dataset contains 20 malware captures from various IoT devices, and 3 captures for benign anomalies. The data was collected in partnership with the Czech Technical University in Prague, with the data being captured between 2018 and 2019 [1]. The dataset in its complete form contains:
.pcap files, which are the original network capture files,
conn.log.labeled files, which are created by running the network analyser called Zeek,
various details and information about each of the captures
Due to the fact that it is easier to work exclusively with the conn.log.labeled files, only those were used in this project. The .pcap files are created by the network capture program Wireshark and can only be opened using it, working with them proved unnecessary difficult for this project, so they were discarded. This approach also seems to be embraced by the creators of the dataset, which offer two download options for it, the complete version, which contains all the file presented above, and a lighter version, which only contains the conn.log.labeled files and the information. The latter was chosen for this project.
The data set contains a total of 325,307,990 captures, of which 294,449,255 are malicious. The data set registered the following types of attacks:

Type of attack	Explanation
Attack	the generic label that is attributed to anomalies that cannot be identified
Benign	generic label for a capture that is not suspicious
C&C	control and command, a type of attack which takes control of the device in order to order it to perform various attacks in the future
C&C- FileDownload	the server that controls the infected device is sending it a file
C&C- Mirai	the attack is performed by the Mirai bot network
C&C- Torii	the attack is performed by the Torii bot network, a more sophisticated version of the Mirai network
DDoS	the infected device is performing a distributed denial of service
C&C- HeartBeat	the server that controls the infected device sends periodic messages the check the status of the infected device, this is captured by looking for small packages being sent periodically from a suspicious source
C&C- HeartBeat -Attack	the same as above, but the method is not clear, only the fact that the attack is coming periodically from a suspicious source
C&C- HeartBeat - FileDownload	the check-up is done via a small file being sent instead of a data packet
C&C- PartOfA-Horizontal-PortScan	the network is sending data packages in order to gather information for a future attack
Okiru	the attack is performed by the Okiru bot network, a more sophisticated version of the Mirai network
Okiru-Attack	the attacker is recognized as the Okiru bot network, but the method of attack is harder to identify
PartOfA Horizontal-PortScan	information is gathered from a device for a future attack
PartOfA HorizontalPort Scan-Attack	the same as above, but methods that cannot be identified properly are used

Table 1: The types of attacks present in the data set

Each of the conn.log.labelled files contain 23 columns of data, whose types are presented in table 1. These columns are:

Table 2: The types of information in the data set

Column	Description	Type
ts	the time when the capture was done, expressed in Unix Time	int
uid	the ID of the capture	str
id_orig.h	the IP address where the attack happened, either IPv4 or IPv6	str
id_orig.p	the port used by the responder	int
id_resp.h	the IP address of the device on which the capture happened	str
id_resp.p	the port used for the response from the device where the capture happened	int
proto	the network protocol used for the data package	str
service	the application protocol	str
duration	the amount of time data was traded between the device and the attacker	float
orig_bytes	the amount of data sent to the device	int
resp_bytes	the amount of data sent by the device	int
conn_state	the state of the connection	str
local_orig	whether the connection originated locally	bool
local_resp	whether the response originated locally	bool
missed_bytes	number of missed bytes in a message	int
history	the history of the state of the connection	str
orig_pkts	number of packets being sent to the device	int
orig_ip_bytes	number of bytes being sent to the device	int
resp_pkts	number of packets being sent from the device	int
resp_ip_bytes	number of bytes being sent from the device	int
tunnel_parents	the id of the connection, if tunnelled	str
label	the type of capture, benign or malicious	str
detailed_label	if the capture is malicious, the type of capture, as described above	str

The column conn-state is a variable specific to Zeek and represents the state of the connection between two devices. As an example, S0 means a connection is attempted by a device, but the other side is not replying. In this dataset, all values that were missing from any of the entries were marked with a dash ("-"), except for the IP address, which were marked with two-colons ("::").

4.2 Data visualization

Before being able to visualize the data, the dataset was converted into text files in order to make it readable for Python. The fig. 3 shows the distribution of each anomaly in the files Malware-17,34,60 and Honeypot-4 of the dataset.

As it can be seen, the files titled 'Honeypot-x' all contain only benign captures, as they are meant to show how normal traffic should look like for a IoT device.

4.3 Data formatting

The files were then converted from .txt to .csv (a type of text file where values are delimited by commas) in order to solve compatibility issues with some of Python's libraries.
Next, the 'label' and the 'detailed-label' columns where merged into just one, and then numerically encoded according to the following table:

Table 3: How the data was encoded in order to be fed into the models

label	detailed-label	encoding
Benign	-	0
Malicious	C&C	1
Malicious	C&C-FileDownload	1
Malicious	C&C-HeartBeat	2
Malicious	C&C-HeartBeat-Attack	2
Malicious	C&C-HeartBeat-FileDownload	2
Malicious	C&C-Mirai	3
Malicious	C&C-Torii	4
Malicious	DDoS	5
Malicious	FileDownload	6
Malicious	Okiru	7
Malicious	Okiru-Attack	7
Malicious	PartOfAHorizontal PortScan	8
Malicious	PartOfAHorizontal PortScan-Attack	8
Malicious	C&C-PartOfAHorizontal PortScan	8
Malicious	Attack	9

There were three major reasons behind this step. First, the data that was important for the algorithms was in two separate columns, which created problems as classification is done on only one column at a time. Second, there were too many labels that were different symptoms of the same attack. Lastly, more classes increase the necessary computational power required for working with the model.

4.3.1 Statistical correlation

After the steps discussed above, statistical correlation was applied to the data set in order to eliminate the data which was not related to the 'label' column. In order to obtain this data, a correlation matrix for each of the 33 files in the data set was drawn and then the results were averaged.
The matrices are all size 23×23, one row and one column for each of the columns in the data set. The correlation matrices use a scale from blue to orange to represent the strength of the correlations, with blue representing negative correlation and orange positive correlation. The colour grey represents a weak or no correlation, while white represents the fact that the column did not contain enough data to draw any correlations.

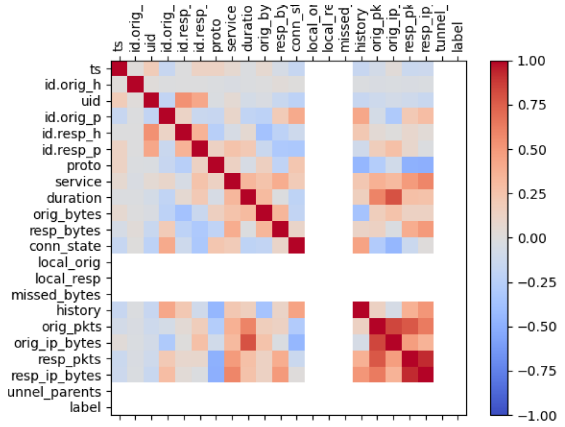

Figure 1: Correlation in Honeypot-5

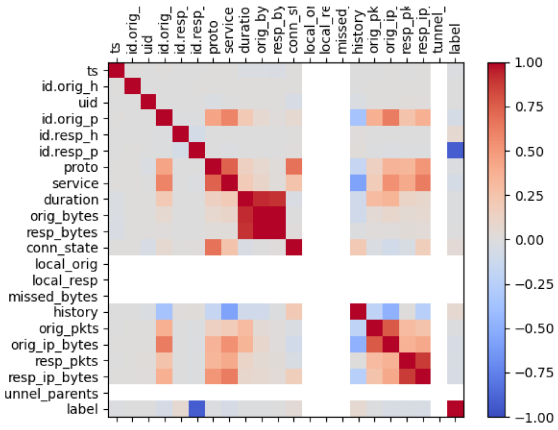

Figure 2: Correlation in Malware-17

The following columns were eliminated:

`ts, uid, id.orig_h, local_orig, local_resp, missed_bytes, tunnel_parents`

The first three were eliminated as on average they were only weakly related to the 'label' column. The last three were eliminated as they were missing so much data that none of the correlation matrices could establish correlations between them and other columns.

4.4 Data splitting

For the last step, all the 33 files were combined into one .csv file that was to be fed into the algorithms. However, due to technical limitations, the dataset had to be split into four smaller files, with the data being randomly distributed between the files.

For each of the algorithms, the data from each file was split 80-20 between the testing and the training set.

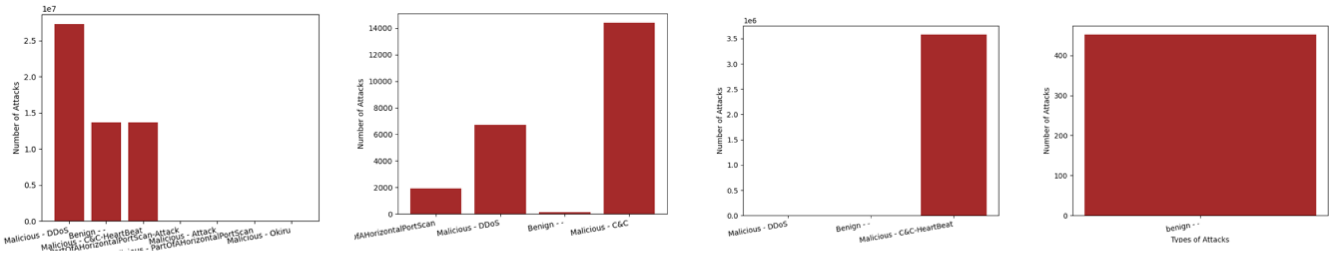

Figure 3: Distribution of anomalies in the presented files

4.5 Data analysis

As the predictions are done on a number of categories, all the algorithms presented in this section are of multiclass classification type.

4.5.1 Decision Tree

Decision Trees are a classifier that tries to go from information about a data point to conclusions about its value within the given system by making a number of successive simple decisions. While decision trees are not very robust classifiers, they are widely used as a basis for other, more complex clasifiers, such as forests and boosters due to the fact that they are very computationally light [12].

4.5.2 Random Forest

Random forest (RF) is a supervised classification algorithm that averages the results of a number of decision trees. In general, the number of decision trees increases the accuracy of the algorithm, although at some point each new tree only increases the accuracy only marginally. The advantage of the random forest algorithm over similar algorithms is that the randomness corrects for the habit of the decision trees to over-fit to the given training set [3]. Also, just like other similar forest algorithms, it is computationally-light compared to other more sophisticated algorithms.

4.5.3 Naïve Bayes

Naïve Bayes (NB) is a probabilistic classifier that is based on Bayes' theorem

$$P(A|B) = [P(B|A) * P(A)]/P(B)$$

where the strong independence between the features of the data is assumed [11]. Since a class variable is used instead of a numerical variable, the theorem becomes:

$$P(C|x_1, ..., x_n) = \frac{P(C) * P(x_1, ..., x_n)}{P(x_1, ..., x_n)}$$

It is one of the simplest models to use, but it is computationally-light, which makes it fast to use.

4.5.4 Support Vector Machine

Support Vector Machine (SVM) is a binary classification algorithm that works best with the extreme cases of a dataset. It works by creating a separation line called hyperplane, which is the biggest distance between the extremes of both classes of the data set. Thus, the algorithm tries to draw the hyperplane while keeping the maximum distance between the extremes constant [5].
One of the main advantages of the algorithm is that it can work with non-linear data by using what is called the kernel trick, which project data with a high number of dimensions onto lower dimensions. The main disadvantage is that the SVM function does not provide probability estimates by itself. Instead it has to use k-fold cross-validation, where k is the number of parts the dataset is split into for cross-validation purposes.
In the case of multi-class classification, the multi-class problem is split into binary problems and then done one-versus-one, where pairs of the classes are compared between each other individually, or one-versus-all, where all the other are considered one class and are then compared with the other remaining class.

4.5.5 Artificial Neural Network

Artificial neural networks (ANN) are more complex classifiers that are inspired by the brain, in that they use units called neurons, units which receive inputs, have an internal state which combined with the input produce the output. These networks contain three 'layers', the input layer, the hidden layer and the output layer, although a network can use multiple hidden layers. Each of the neurons in one layer is connected to the next layer all the way up to the output layer, either randomly, or with each neuron connecting to all the others in the next layer [15].
The main idea of the algorithm is that each neuron has its own internal state, which is controlled by a large number of parameters, and then adds together the value

of the internal state, also called bias, and the sum of all the values received from all the other neurons it is connected to in the previous layer. The formula in this case is:

$$\sum_{i=1}^{n}(x_i * w_i) + bias$$

Where:

- n is the number of neurons that the target neuron is connected to,
- x_i is the information from the neuron i,
- w_i is the weight given to the connection between the target neuron and the neuron i,
- bias is the internal state of the target neuron, given by its internal state parameters

The result of this calculation is then sent to an activator:

$$f(x) = \begin{cases} 1 & \text{if} \sum_{i=1}^{n}(x_i * w_i) + bias > 0 \\ 0 & \text{otherwise} \end{cases}$$

Depending on the value of the activator, the neuron will then send its output $\hat{y}$ to the next layer of neurons. This project uses a type of neural network called Multi Layer Perceptron (MLP), which is one of the most simple types of networks, being just a network comprised of multiple layers of neurons.

4.5.6 AdaBoost

AdaBoost is a classifier which fits a number of weak meta-classifiers (slightly better than chance) on a dataset, and then continues fitting additional classifiers on the dataset, but with adjusted weights to the cases where the previous classifiers were wrong [8]. Thus, the algorithm is better suited datasets where the values are harder to classify.
This algorithm is efficient due to the fact that the algorithm only iterates over the cases which have not yet been classified.

4.6 Analysis Methods

In order to evaluate the algorithms described above, the metrics presented below were used. The discussion of these metrics in the context of this project is done in the ‘Results‘ section.

There are four concepts which have to be presented before the metrics are discussed:

- TP is the number of actual positives that were correctly identified
- TN is the number of actual negatives that were correctly identified
- FP is the number of actual positives that were identified as negatives
- FN is the number of actual negatives that were identified as positives

4.6.1 Confusion Matrix

A confusion matrix is a table that allows for the visualization of the performance of a model by showing which values the model thought belong to which classes. It has a $N \times N$ size, where n is the number of classes, with the columns representing the actual classes and the rows the predicted classes.

4.6.2 Precision

The precision is a metric that evaluates the model by calculating the fraction of correctly identified positives. Its formula is:

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

4.6.3 Accuracy

The accuracy is a metric that evaluates the model by calculating the fraction of correct predictions over the total number of predictions. Its formula is:

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

4.6.4 Recall Score

The recall score is a metric that evaluates the model by calculating the fraction of actual positives that were correctly identified. Its formula is:

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

4.6.5 Support score

The support score is the number of occurrences of a class in the total number of predictions. It is used as part of the F_1 - micro-average metric.

Metrics		Classifiers				
		RF	NB	ANN	SVM	ADA
Precision	Weighted	1.00	0.76	0.71	0.60	0.86
	Macro	0.88	0.27	0.33	0.23	0.55
Recall	Weighted	1.00	0.23	0.66	0.67	0.87
	Macro	0.85	0.38	0.14	0.14	0.35
F-1 Score	Weighted	1.00	0.25	0.52	0.59	0.83
	Macro	0.84	0.10	0.10	0.13	0.37
	Accuracy	1.00	0.23	0.66	0.67	0.87

Figure 4: The results of the classifiers

4.6.6 F-1 Score

The F-1 score is a metric that calculates the harmonic mean of the precision and the recall score. It is considered an extension on the metric of accuracy, as it takes into account both false positives and false negatives. Its formula is:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

However, the formula in this form only works on binary classifiers. For multi-class classifiers, there are two options: micro-averaging (taking into account the frequency of each class) and macro-averaging (all the class are taken into account equally). Macro-averaging is better suited for data sets where the data is relatively uniform size-wise, while micro-averaging is better for data sets where there are major imbalances between the sizes of the classes. The formulas for them are:

$$F_1 - micro = \sum_{i=1}^{n} [F_1(i) * \frac{supportscore_i}{setsize}]$$

$$F_1 - macro = \sum_{i=1}^{n} [F_1(i) * \frac{1}{n}]$$

Where n is the number of classes and $setsize$ is the total number of predictions made.

For the results of this paper, the formula with the micro-average is used, as there are clear imbalances in the sizes of the classes from the data set.

Results

5.1 Experimental Setup

This experiment was conducted on a machine operating on Windows 10 Pro 64-bit, Build 18636, the processor was an Intel(R) Core(TM) i7-9700K, CPU @ 3.60GHz (8 CPUs), 3.6GHz. The memory of the machine was 32768 MB RAM. The graphics card was NVIDIA GeForce GTX 1060 6GB. The algorithms used for this research were implemented using Python 3.8.3 64-bit, loading the data into the models was done using the 'Pandas' library. The models were implemented using the library 'sklearn', and the metrics were also provided by this library.

5.2 Result Analysis

As the data set was split into four parts of similar size, with the data being distributed randomly between them, the metrics obtained from each of the parts can be averaged in order to get results for the entire dataset. As it can be seen from fig. 4, the best algorithm by far is the random forest, with a weighted average precision of 1.00. To be kept in mind, the real results are somewhere between 0.995 and 1.000, as the results are only given in two decimals. The same is true for the recall, and thus for the F-1 score as well.
The next best algorithm is AdaBoost, with a 0.86 precision, a 0.87 recall score and a 0.83 F-1 score. As discussed above, both AdaBoost and Random Forest use Decision Tree as their meta-classifier. A possible explanations for this is that since some of the columns are heavily correlated with the label column, making simple assumptions, such as those made by a decision tree, are all that is necessary for correctly predicting the large majority of the cases. This is just an assumption however, and as it not completely clear what the correlation between the decision tree classifier and the high results is, it is recommended that this be the subject of future research.
The somewhat weak metrics of the Artificial Neural

Network are an interesting result. From also looking at the confusion matrix, it seems that the model had a bias for the categories with the largest number of occurrences and tended to choose the others rarely. A possible explanation for this might be the configuration of the neurons, with the bias in their weight function favouring not being large enough to allow the flexibility necessary for correct predictions on rare occurrences to be made. Again, this cause of this lack of accuracy is not clear and should be further investigated.
The worst results were generated by the Support Vector Machine algorithm, with an accuracy of only 60%. Meanwhile, the recall score and the F-1 score were actually higher than NB and ANN, which indicate a relatively low number of true positives, but a higher number of false negatives. Again, this indicates that the algorithm was better at predicting the classes with more occurrences. This can also be seen in the confusion matrix, where only three classes had correct predictions, with those three being the biggest malware categories. Interestingly, the algorithm was never able to predict benign comportment, which means that benign captures cannot have a hyperplane drawn such that they will be separated from the other anomalies. A possible explanation might be that benign captures simply do not have features that distinguish them clearly from different types of attacks, so they are simply categorized as the attack that resembles them the most.
Lastly, the poor results obtained by Naïve Bayes likely stem form the fact that the data is not independent, so the naive assumption is in this case wrong
Regarding the classes that the algorithms had to classify, one of the clear problems was that none of the algorithms could identify Mirai attacks. This can be attributed to the fact that there are only 16 occurrences of it in the entire data set, so around 4 per file on average. The same problem was present in the case of the Torii botnet, although in this case, some algorithms obtained at least some results. For the largest categories, all the algorithms were able to predict them with at least 69% accuracy, so the problem with the detection of smaller classes of attack are in their proportion within the data set.

5.3 Comparison with other studies

Table 4: The results of other similar studies

Study	Data set	Best Algorithm	Metric
Shafiq, Tian, Sun et al. [17]	Bot-IoT	Naïve Bayes	Precision = 0.99
Hasan, Islam, Zerif et al. [9]	DS2OS	Random Forest	Accuracy = 0.994
Anthi, Williams and Burnap [2]	Custom data set	Naïve Bayes	Accuracy = 0.977
Revathi and Malathi [14]	NSL-KDD	Random Forest	Accuracy = 0.998
This project	IoT-23	Random Forest	Precision = 0.995

The five studies above were chosen such that each one presents the implementation of machine learning algorithms on another data set, in order to have some diversity when comparing this project with other similar approaches. All of them use multi-class classification.
For this part, the lower bound of 0.995 was chosen. As it can be seen, the results of this research are in line with other similar projects.
It is also interesting that even when more complex algorithms were tested, such as artificial neural networks, simpler algorithms have always had some kind of edge over the more complex ones. The same is also true for this project.
Again, as discussed above, the reason why Naïve Bayes performs badly in this experiment is because not all the columns are independent.

Conclusions

6.1 General conclusions

In conclusion, the Random Forest algorithm is the best choice for anomaly detection and classification in the context of the IoT-23 dataset. While the reasons behind this conclusion are not fully understood, this algorithm scored the highest in all metrics and it presented itself as the best choice overall.
To answer the secondary research question, the results of this study are in line with what other similar works found as well.

6.2 Limitations and future research

Most of the limitations of this project were of technical nature. The data set had to be split into smaller parts and the data had to be encoded in such a way as to have less categories, as using all the initial one would create computational problems. Thus, it is recommended that at a future date this experiment be redone by using all the data at once, with the labels as they were in their original format.
Statistical correlation had to be run on each of the

files in the data set, which may have skewed the results. Additionally, only the data that had no statistical correlation with the column to be predicted was eliminated. Of course, even more data can be eliminated from the data set. Thus, another possibility for future research is to find out what the minimum amount of data from the IoT-23 data set is such that the implemented models are still accurate. Also, as mentioned in the Result Analysis section, the reason behind the high accuracy of the Decision Tree classifier should be investigated. Lastly, this project used a Multi Layer Perceptron with mixed results. Possible future research would be to use more advanced types of Artificial Neural Networks and see whether they get different results.

References

[1] Agustin Parmisano, Sebastian Garcia, M. J. E. IoT-23 Dataset: A labeled dataset of Malware and Benign IoT Traffic. — Stratosphere IPS, 2018.

[2] Anthi, E., Williams, L., and Burnap, P. Pulse: An adaptive intrusion detection for the internet of things. *IET Conference Publications* (2018).

[3] Breiman, L. Random forests. *Machine Learning* (2001).

[4] Cale Gutherie Weissman. We Asked Executives About The Internet Of Things And Their Answers Reveal That Security Remains A Huge Concern — Business Insider India.

[5] Cortes, C., and Vapnik, V. Support-Vector Networks. Tech. rep., 1995.

[6] Evans, D. The Internet of Things How the Next Evolution of the Internet Is Changing Everything. Tech. rep., 2011.

[7] Expert System Team. What is Machine Learning? A definition - Expert System.

[8] Freund, Y., and Schapire, R. E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences 55*, 1 (aug 1997), 119–139.

[9] Hasan, M., Islam, M. M., Zarif, M. I. I., and Hashem, M. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things 7* (sep 2019), 100059.

[10] Koroniotis, N., Moustafa, N., Sitnikova, E., and Turnbull, B. Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset.

[11] Lewis, D. D. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. Tech. rep.

[12] Loh, W.-Y. Fifty Years of Classification and Regression Trees 1.

[13] Pahl, M.-O., and Aubet, F.-X. DS2OS traffic traces — Kaggle.

[14] Revathi, S., and Malathi, A. A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection. Tech. rep.

[15] Rosenblatt, F. Principles of Neurodynamics: Perceptrons and the theory of brain mechanisms. Tech. rep., 1961.

[16] Rouse, M. What is IoT (Internet of Things) and How Does it Work?

[17] Shafiq, M., Tian, Z., Sun, Y., Du, X., and Guizani, M. Selection of effective machine learning algorithm and Bot-IoT attacks traffic identification for internet of things in smart city. *Future Generation Computer Systems 107* (jun 2020), 433–442.

[18] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. A Detailed Analysis of the KDD CUP 99 Data Set.

[19] Weiser, M. The Computer for the 21st Century. Tech. rep.

[20] Zeadally, S., and Tsikerdekis, M. Securing Internet of Things (IoT) with machine learning.