



**Birla Institute of Technology and Science, Pilani**

**2021**

**Project Report**

on

**IoT Network Anomaly Detection Using Machine Learning and Deep  
Learning**

Submitted In Fulfilment of the Course

**Study Oriented Project – CSF266**

Submitted by

**SNEHA**

**2019A7PS0042P**

Under the esteemed guidance and supervision of

**Dr. L Rajya Lakshmi**

Faculty, Department of Computer Science & Information Systems

Birla Institute of Technology & Science, PILANI, 333031

Rajasthan, INDIA

## **Acknowledgments**

I am highly indebted to my project guide and supervisor, Dr. L Rajya Lakshmi, for initiating me into such a fruitful endeavor. Without her support, I would have been foraying in the world of Machine Learning and Data Learning. Her constant guidance and encouragement motivated me to go the extra mile throughout this project and sparked a profound interest in the topic. I am immensely grateful to her. Her valuable inputs were vital to the success of this project.

Sincerely,

Sneha

2019A7PS0042P

# Table of Contents

## **1. Introduction**

## **2. Background and Motivation**

- a. Machine Learning
- b. Deep Learning
- c. Internet of Things
- d. Anomaly Detection

## **3. Literature Review**

- a. Implementation
- b. Metrics Used
- c. Comparative Study

## **4. Brief of supervised ML models used (Along with Parameter Tuning)**

- a. Decision Trees
- b. Random Forest
- c. Naive Bayes
- d. Support Vector Machine
- e. AdaBoost
- f. XGBoost
- g. Multi-Layer Perceptron
- h. Convolutional Neural Networks

## **5. Dataset - IoT23**

## **6. Data Preprocessing**

- a. Data Encoding

- b. Handling missing values
- c. Dropping Irrelevant Information
- d. Statistical Correlation
- e. Outlier Handling

## **7. Results**

- a. Graphs Obtained
- b. Analysis of Results
- c. Conclusion

## **8. Appendix I - Code for Data Preprocessing**

## **9. Appendix II - Code for ML Models**

## **10. References**

## Introduction

The Internet of Things (abbreviated as IoT), the "Computer of the Twenty-First Century," is the notion of linking multiple objects to a network. This network is then used to send data between the devices without human interaction automatically. Advances in cloud computing, data analytics, and increased hardware power have given IoT new dimensions, transforming it into what it is today.

Currently, as the scale of IoT grows, the most pressing concern for over half of all potential users of IoT equipment is security. As a result, academics have begun to investigate more complex security solutions to address this problem. One of the new approaches is to use machine learning to detect and classify assaults. The two technologies are a good fit as IoT systems can offer the vast amounts of data that machine learning algorithms require to create their detection models. Furthermore, the vast number of different sorts of attacks and their manifestations makes it nearly impossible for human operators to identify and classify. The primary purpose of this study is to construct Machine Learning and Deep Learning algorithms for network-based anomaly identification in IoT devices and then to evaluate them using the IoT-23 dataset.

IoT-23 is a new dataset that includes malicious and benign network captures from several IoT devices.

## **Background and Motivation**

### **Machine Learning**

Machine learning is a branch of computer science that enables computers to learn without human intervention. There is no explicit programming as Machine Learning performs a task with the least human interaction possible. Usually, a black-box approach is applied where the machine uses large amounts of data to identify and relearn trends. The learning process is automated and enhanced based on the machines' experiences along the way. As its name suggests, it provides the computer with learning, making it more human-like.

### **Deep Learning**

Deep learning is a type of machine learning and artificial intelligence (AI) that mimics how people acquire knowledge. Data science, which includes statistics and predictive modeling, has deep learning as an essential component. Deep learning is beneficial for data scientists responsible for gathering, analyzing, and interpreting massive amounts of data; it speeds up and simplifies the process.

Deep learning is regarded as a means to automate predictive analytics at its most basic level. While typical Machine Learning algorithms are built linearly, Deep learning algorithms are made in a hierarchy of increasing complexity and abstraction.

## **Internet of Things**

The Internet of Things, or IoT, refers to the billions of physical devices connected to the internet that collect and exchange data worldwide. It is based on the concept of linking multiple objects to a network. This network is then used to send data between the devices automatically, without the need for human interaction. The advent of super-cheap computer chips and the widespread availability of wireless networks has opened up new horizons in the world of IoT. Connecting all these various products and attaching sensors to them gives devices that would otherwise be dumb a level of digital intelligence, allowing them to convey real-time data without involving a person. The Internet of Things brings the digital and physical worlds together to make the world around us more intelligent and more responsive.

## **Anomaly Detection**

Anomaly detection is very similar to outlier analysis. It is a step in data mining that identifies data points, events, and observations that deviate from a dataset's normal behavior. Anomalous data can indicate critical incidents, such as technical glitches, etc. The complex networks behind IoT and its increasing applicability in solving complex and everyday challenges make it highly prone to security failure and its enormous ramifications. Hence, anomaly detection is of utmost importance in the Internet of Things.

## Literature Review

The use of Machine Learning for IoT Detection is still relatively new. Although some frameworks have been developed, most research work focuses on implementation and testing. However, the scope of using Machine Learning and Deep Learning is immense in IoT since the devices are less complex than traditional systems, making them more predictable, and data for analysis is readily available.

The portability of these algorithms remains a significant challenge in their widespread use and application. Bypassing security levels by taking advantage of other IoT network flaws poses another problem. In conclusion, machine learning should be viewed as an additional layer of security for IoT networks rather than as a comprehensive solution.

The following conclusions were obtained by fellow researchers who worked on anomaly detection in IoT networks using Machine Learning/ Deep Learning. These points were used as a base while carrying out this project:

### 1. Implementation of Machine Learning

According to Zeadally and Tsikerdekis<sup>[7]</sup>, there are two significant ways of implementing Machine Learning algorithms in IoT networks:

- Network-based: Using metadata from the IoT network



- Host-based: Using the information present on the device

This project will be focused on a network-based implementation of Machine Learning while also performing classification using Convolutional Neural Networks (CNN) and Multi-Layer Perceptrons (MLP).

## **2. Metrics for Testing**

Shafiq, Tian, Sun, et al.<sup>[8]</sup>, 2020 tested 44 features on the Bot-IoT dataset using an implementation of the Naive Bayes algorithm to find a framework model for testing attack detection algorithms. They conclusively stated that the best four metrics for testing the models built are the actual positives rate (TPRate), the precision, the accuracy, and the time taken to build the model. These metrics were used in this project to compare and contrast the results obtained from various Machine Learning algorithms and Deep Learning models for anomaly detection.

## **3. Comparative Study of Various Machine Learning models**

After deciding how to measure the results should be done, it is also imperative to find out which models work best for the work of this project. For this purpose, a comparative analysis was performed against a few similar projects.

- 1) Hasan, Islam, Zerif, et al.<sup>[9]</sup>, 2019 performed a complex and comprehensive analysis using machine learning algorithms to detect whether the system is performing abnormally and then using algorithms to detect the type of attack the device is under. Their research on the DS2OS IoT dataset found that the Random Forest algorithm gave them the best results. They achieved an accuracy of 99.4%. Following this, they also employed an artificial neural network, which produced similar accuracy, but its performance showed lower scores on other metrics. This correlated directly with the results obtained through this project, as well.
- 2) Anthi, Williams, and Burnap, 2018<sup>[10]</sup> proposed a novel model for a network-based real-time malware detection system called Pulse. For them, implementing the Naive Bayes algorithm served as the most performing classifier for the proposed model with precision between 81% and 97.7%.
- 3) Lastly, Revathi and Malathi<sup>[11]</sup> also used the Random Forest algorithm to analyze the NSL-KDD IoT dataset.

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. It should also be noted that even in comparison to significantly more complex algorithms, such as neural networks (Artificial Neural Networks), most of the studies found the best results using algorithms such as Naive Bayes and Random Forest. The findings of this report also corroborated these results.

## **Brief of supervised ML models used (Along with Parameter Tuning):**

As the predictions are made on several categories, all the algorithms used for this project are of multi-class classification type.

### **1. Decision Tree**

A Decision Tree is a classifier that obtains conclusions about a data point value on the basis of information about the data points. While decision trees are not very robust classifiers, their utility lies in acting as a basis for other more complex classifiers like Random Forest (where the number of estimators points to the size of the forest i.e. the number of trees in the forest) or boosters like Adaboost that are based on Decision Trees.

The trees try to simulate a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. It incrementally breaks down a dataset into smaller and smaller subsets while also developing an associated decision tree. The training set determines the order and content of questions. This model is prone to overfitting.

Decision Trees have an inverted structure with the root node at the top and the leaves at the

bottom. A decision tree can be used to simulate any algorithm.

## 2. Random Forest

This supervised classification algorithm uses decision trees as its base and takes the average of the trees in its forest to calculate the result. It is based on the concept of ensemble learning, which combines multiple classifiers to solve a complex problem and improve the performance of the model.

Since it takes the prediction from each tree and predicts the final output on the basis of the majority votes of predictions instead of relying on just one decision tree for its result, its accuracy is more, as is also seen in the results obtained through this report. A greater number of trees in the forest leads to higher accuracy. In SickitLearn, the number of trees is specified by the 'number of estimators' parameter. After a particular value, the increase in accuracy values is only marginal and eventually saturates.

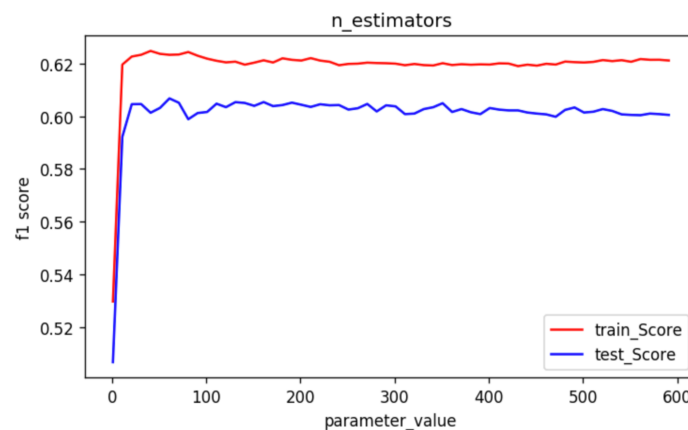


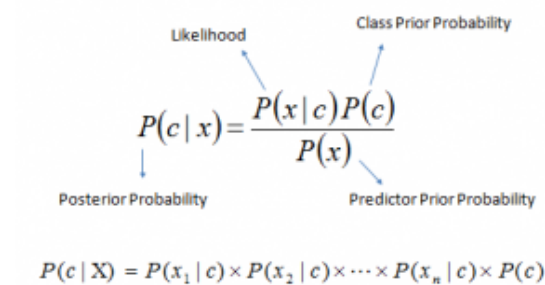
Fig 1: Plot of f1 score against the number of estimators; shows that the

RF algorithm hits saturation value a little before 100

Another advantage of Random Forest is that it prevents the problem of overfitting because of the randomness. It is also computationally efficient, taking significantly less time to train than the rest of its counterparts.

### 3. Naive Bayes

Naive Bayes is a probabilistic classifier that is based on the Bayes Theorem with an assumption of independence among predictors. In simple words, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.



The diagram shows the Bayes Theorem formula with labels pointing to its components:

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

Labels and their corresponding terms in the formula:

- Likelihood** points to  $P(x | c)$
- Class Prior Probability** points to  $P(c)$
- Posterior Probability** points to  $P(c | x)$
- Predictor Prior Probability** points to  $P(x)$

Below the formula, the joint probability formula is given:

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

Fig 2: Bayes Theorem

(Second formula of importance since a class variable is used instead of a numerical variable.)

It is one of the simplest models to use. It is also computationally very light, hence ideal for large datasets.

## 4. Support Vector Machine

Support Vector Machine is a binary classifier that works best with extreme dataset cases. Hence, it is very well suited for anomaly detection, which can be simplified as a version of outlier detection. Maximizing the margin distance provides some degree of reinforcement and separation so that test data points can be classified with more confidence. The algorithm tries to draw the hyperplane while keeping this most significant distance between the two extremes of the classes as constant. SVM essentially separates the classes in the data. Then it classifies the new point depending on whether it lies on the positive or negative side of the hyperplane.

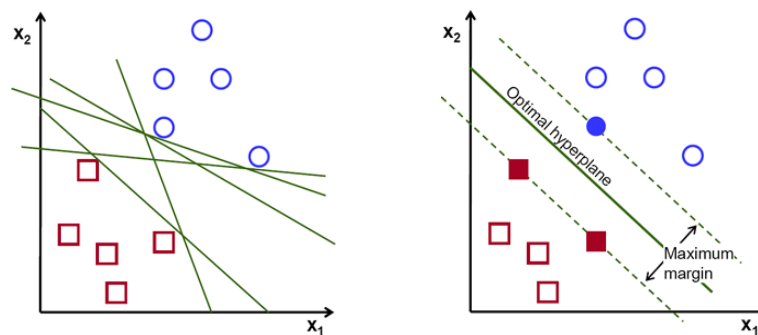


Fig 3: The first figure shows the possible hyperplanes. The second figure depicts the optimal hyperplane; it is decided according to the maximum margin obtained.

The kernel functions translate the data into a higher dimensional feature space to perform the linear separation. It works by creating a separating line called a hyperplane in N-dimensional space (N is the number of features) - the largest distance between the extremes of the two classes

in the dataset.

One of the major advantages of using SVM is working with non-linear data. It does this by using the kernel trick i.e. projecting data with a particular number of dimensions into another value of a number of dimensions. This is usually done for larger to lower dimensions.

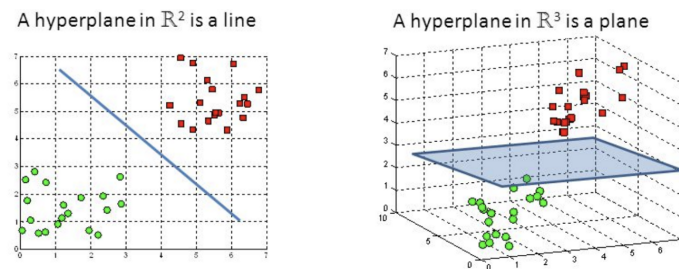


Fig 4: Hyperplanes in different dimensional planes

The major disadvantage of SVM is that it does not provide probability estimates inherently. Hence, we have to use k-fold cross-validation for data validation. Here, k corresponds to the number of classes the dataset is split into for cross-validation.

Since SVM is a binary classifier, in the case of multi-class classification, we need to form pairs of classes in which to run this model. After being split into several binary problems, classification is then carried out in a one-versus-one manner or one-versus all. Here, all classes will be compared individually, or one class will be compared to all other classes, respectively.

The SVM model is SickilLearn by default, only works on CPU. Since the time complexity of SVM is of the order of N cubed, it becomes exceptionally computationally expensive for large

datasets. Hence, for this project, we imported another library, ThunderSVM, to enforce that GPU runtime is used for the model.

## 5. AdaBoost

AdaBoost (Adaptive Boosting) is a classifying algorithm that fits several weak meta-classifiers on a dataset. Boosting is also a class of ensemble machine learning algorithms. The AdaBoost algorithm is an iterative ensemble that involves using very short (one-level) decision trees (called decision stumps) as weak learners added sequentially to the ensemble. Each subsequent model attempts to correct the predictions made by the model before it in the sequence.

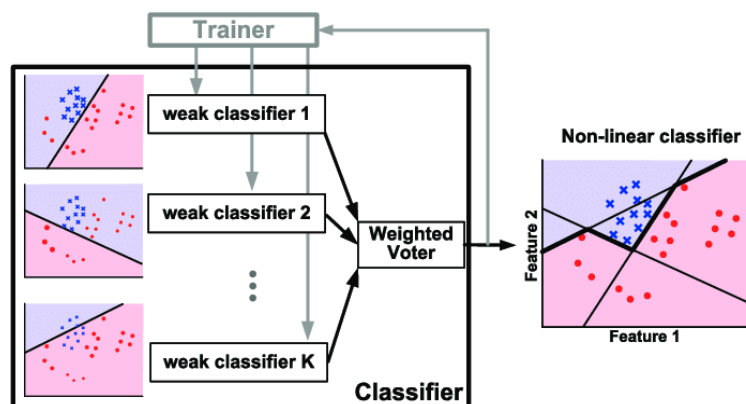


Fig 5: Illustration showing Adaboost iteratively and sequentially creating a stronger classifier by using weighted sums.

Adaboost involves combining the predictions from many weak learners. This improves the chance slightly. Then, it continues fitting additional classifiers on the dataset. However, this



action is performed with adjusted weights to the classifiers previously wrong. Thus, this algorithm is better suited to datasets where the values are harder to classify. This makes for an extremely efficient algorithm as it only iterates over cases that have not yet been classified.

Adaboost in SckitLearn has two parameters that are generally used for finetuning: the number of estimators and the learning rate. The learning rate values around the range of 1 generally give the best values of accuracy. The number of estimators (similar to random forest) increases the accuracy until a certain value, post which the accuracy values are saturated.

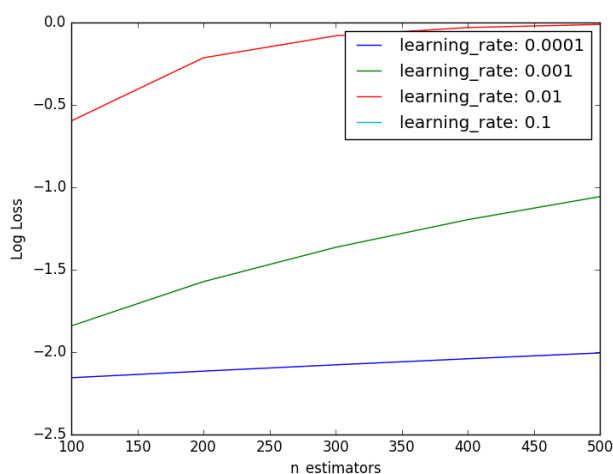


Fig 6: Plot of logarithmic loss against the number of estimators for different values of learning rates

## 6. XGBoost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance that is dominative competitive machine learning. It is generally modeled as an improvement over other boosting algorithms owing to its increased speed as compared to other

algorithms, such as AdaBoost. Its regularization parameter successfully reduces variance too.

However, for this project, the difference in the performance of AdaBoost and XGBoost was not a significant amount.

## 7. Multi-Layer Perceptron

Artificial Neural Networks (ANN) are more complex classifiers that are inspired by the brain. It has many inputs that are termed neurons. ANN is a biologically inspired sub-field of artificial intelligence modeled after the brain. The input is the same as the neuron and then the output is generated on the basis of the neurons and their passing through many hidden layers of networks. These neurons are known as nodes. An ANN is a computational network that tries to mimic the human brain.

The ANN takes input and computes the weighted sum of the inputs and adds a bias to it too.

Each neuron has its internal state and that is controlled by a large number of parameters. To this, the internal state is added, referred to as the bias. The sum of all such values obtained from the neurons is then added. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^n W_i * X_i + b$$

Fig 7:  $W_i$ : Weight of a connection,  $b$ : bias attached

The weighted total is passed as an input to an activation function to produce the final output.

Activation functions choose whether a node should fire or not. Only the fired neurons make it to the output layer. There are distinctive activation functions available that can be applied to the sort of task we are performing. Commonly used activation functions are ReLU, GELU, etc. This can be fed as a parameter in SickitLearn. The activation function that is used in this model is the ReLU activation function (which outputs 1 for all positive values and outputs 0 for all other values.)

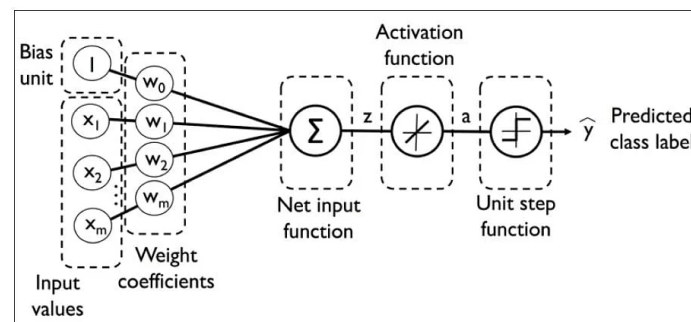


Fig 8: Structure of a Multi-Layer Perceptron (MLP)

Multi-Layer perceptron (MLP) is a type of feed-forward neural network. It consists of three types of layers—the input layer, output layer, and hidden layer, as shown in Fig. 8. The input layer receives the input signal to be processed. The output layer, which receives the activated value of the weighted sum is responsible for performing the classification tasks.

The actual hard computation is performed by the hidden layers that are placed in between the input and output layers. The number of these hidden layers is decided arbitrarily. Since MLP is a feedforward network, the data flow is in the forward direction from the input to the output layer.

The neurons in the MLP are trained with the backpropagation learning algorithm.

MLPs have extremely complex structures and can be used to design a black-box approach that can approximate any continuous function and can even solve a problem that is not linearly separable.

## **8. Convolutional Neural Networks (CNN)**

A Convolutional Neural Network (ConvNet/ CNN) is a Deep Learning algorithm that generally takes in an image as an input. It then assigns importance (learnable weights and biases) to various aspects/objects in the image. It uses this to differentiate one image from the other. For this dataset, we have used the data values that are in the tabular form directly to feed into the neural network.

The pre-processing required in a ConvNet is much lower as compared to other classification algorithms in Deep Learning since they are based on learning. As compared to CNNs, in the past, the primitive method filters used hard-engineered or coded algorithms.

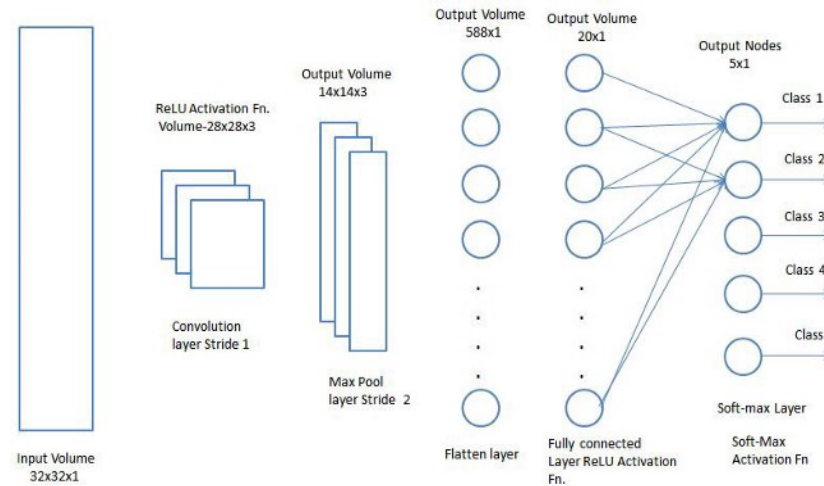


Fig 9: Structure of a Fully-Connected (FC) layer in a Convolutional Neural Network (CNN)

Since CNNs can successfully capture spatial and temporal dependencies, they are much better at classification tasks. Even manipulated data can be fed into this network but would still produce the correct output. This gives CNNs a considerable advantage over feed-forward methods like Multi-Layer perceptrons. The results obtained for MLP and CNN through this project on the IoT dataset also successfully demonstrate the advantage that the CNNs enjoy over traditional feed-forward networks.

## Dataset

The data set used in this project is IoT-23 <sup>[5]</sup>.

The Avast AIC laboratory creates this dataset. The dataset consists of 20 malware captures from

various IoT devices, and 3 captures for benign anomalies. Thus, the total number of labels in this dataset is 23, hence deriving the name of the dataset. The data was collected in partnership with the Czech Technical University in Prague. The timeline for data collection was between 2018 and 2019. The data set contains a total of 325,307,990 captures, of which 294,449,255 are malicious. Two versions of this dataset have been made available on their official site by the creators. They offer two download options for the dataset- the complete version (which contains both the versions of the files presented above), and a lighter version (which only contains the conn.log.labeled files). The latter was chosen for this project.

The dataset in its complete form contains both pcap files and conn.log.labeled files. The pcap files are the original network capture files while the conn.log.labeled files have been created by running a network analyzer on them. The network analyzer is called Zeek. The .pcap files are created by the network capture program Wireshark and can only be opened using it. Since working with them proved unnecessary difficult for this project, they were discarded. It is easier to work exclusively with the conn.log.labeled files and only those were used in this project.

The data has the following types of attacks listed:

Type of attack	Explanation
Attack	the generic label that is attributed to anomalies that cannot be identified
Benign	the 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- File Download	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- Heart Beat	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- Heart Beat -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- Heart Beat - 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 Horizon talPort Scan-Attack	the same as above, but methods that cannot be identified properly are used

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

S. No.	Column	Description	Type
1	ts	the time when the capture was done, expressed in Unix Time	integer
2	uid	the ID of the capture	string



3	id_orig.h	the IP address where the attack happened, either IPv4 or IPv6	string
4	id_orig.p	the port used by the responder	integer
5	id_resp.h	the IP address of the device on which the capture happened	string
6	id_resp.p	the port used for the response from the device where the capture happened	integer
7	proto	the network protocol used for the data package	string
8	service	the application protocol	string
9	duration	the amount of time data was traded between the device and the attacker	float
10	orig_bytes	the amount of data sent to the device	integer
11	resp_bytes	the amount of data sent by the device	integer
12	conn_state	the state of the connection	string
13	local_orig	whether the connection originated locally	boolean
14	local_resp	whether the response originated locally	boolean
15	missed_bytes	number of missed bytes in a message	integer

16	history	the history of the state of the connection	string
17	orig_pkts	number of packets being sent to the device	integer
18	orig_ip_bytes	number of bytes being sent to the device	integer
19	resp_pkts	number of packets being sent from the device	integer
20	resp_ip_bytes	number of bytes being sent from the device	integer
21	tunnel_parents	the id of the connection, if tunneled	string
22	label	the type of capture, benign or malicious	string
23	detailed_label	if the capture is malicious, the type of capture, as described above	string

The column conn-state is a variable specific to the Zeek originated smaller version of the dataset. It represents the state of the connection between two devices. e.g., S0 means a connection is attempted by a device, but the other side is not replying.

In this dataset, missing values have already been handled. All values that were missing from any of the entries were marked with a dash (“-”). In particular, the missing values of the IP address were marked with two colons (“::”).

## Data Preprocessing

### Data Encoding

The data for the labels against which the classification needs to be done is present in two separate columns. Although, classification can only be done on one column at a time. Additionally, there were too many labels that were actually all symptoms of the same attack. These unnecessary and extra classes impose a burden on the computational power of the models that we will use to train on the dataset. The computation is already heavy since there are around 15,00,000 rows in the dataset. Hence, the columns ‘label’ and ‘detailed-label’ were merged into one. They were also numerically encoded according to the following table so that they could 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	PartOfAHorizontalPortScan	8
Malicious	PartOfAHorizontalPortScan-Attack	8
Malicious	C&C-PartOfAHorizontalPortScan	8
Malicious	Attack	9

Fig 10: Numerical Encoding to obtain the values for the labels by  
merging two columns in the dataset

Thus, there are 13 classes/ labels in the dataset now according to which classification needs to be performed. These account for thirteen different types of malicious attacks.

Additionally, one hot encoding was performed for two columns: 'proto' and 'conn\_state' and dummies were obtained for them. This coding helps in raining the data faster as the data becomes compatible with the model that is going to be used to train on it. Also, the correlation matrix that is drawn later provides better results with this form of numerical encoding.

## **Handling missing values**

For different reasons, several real-world datasets can contain missing values. They're sometimes encoded as NaNs, blanks, or other types of placeholders. The IoT-23 dataset has handled these missing values with dashes. Training a machine learning model with a dataset that contains a large number of missing values can have a significant effect on the model's output. Some algorithms, such as ScikitLearn estimators, presume that all values are numerical and that they have and carry significance. Thus, even dashes can't be used as it is in the dataset. It needs to necessarily be a numerical value.

Getting rid of the observations with missing data is one solution to this issue. There is, however, a chance of losing data points containing useful information. A better method would be to impute

the missing values i.e. infer those missing values from the existing part of the data.

It was observed that replacing the missing values with zero gave the best results. The NaN values were replaced by -1 to differentiate them from already present dashes in the dataset. This also helps in separating them from data that has a value of zero by default.

## **Dropping Irrelevant Information**

Three columns: 'uid', 'id.orig\_h', 'id.resp\_h' were also dropped since they only contain information about the ID of the capture, the IP address of the location where the attack happened, and the port that was used by the responder. While this information provides detailed identification of where the attack was detected, these columns are very weakly related to the labels. The information that these columns provide is of no use to us in classifying the attack under various labels.

## **Statistical Correlation**

After carrying out the preliminary data preprocessing above, statistical correlation needs to be applied to the dataset so as to eliminate the data whose values are only weakly related to the 'label' column. Having the values from these columns in the dataset leads to poorer results on training since even values that do not have a strong correlation with the label classifier are being used to identify trends in the complete data.

This correlation matrix along with the confusion matrix was obtained in SickitLearn and then plotted using MatPotLib.

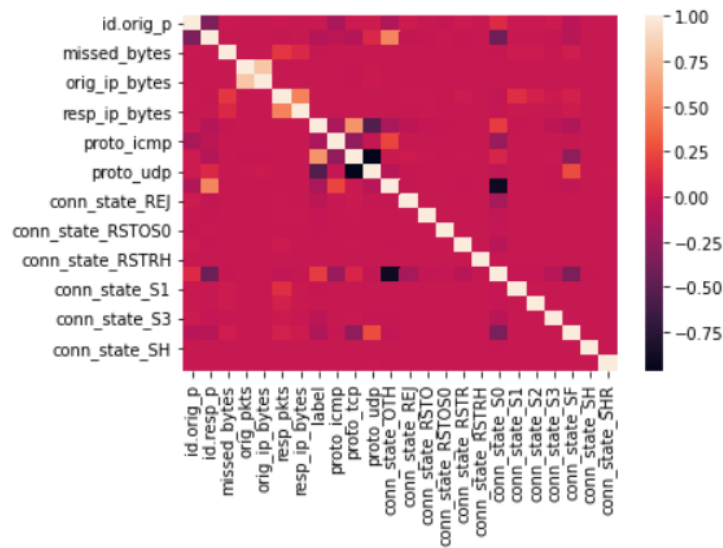


Fig 11: Correlation Matrix for IoT -23

The correlation matrix uses a scale of red, pink, and black to show the correlation for one column and the others. Since we will perform classification using only the ‘label’ column, the matrix cell values of interest to us are the ones corresponding to the ‘label’ column. The color purple-red represents a weak or no correlation, while white represents that there wasn’t enough data to raw any correlation. This mostly corresponds to NaN values in the dataset.

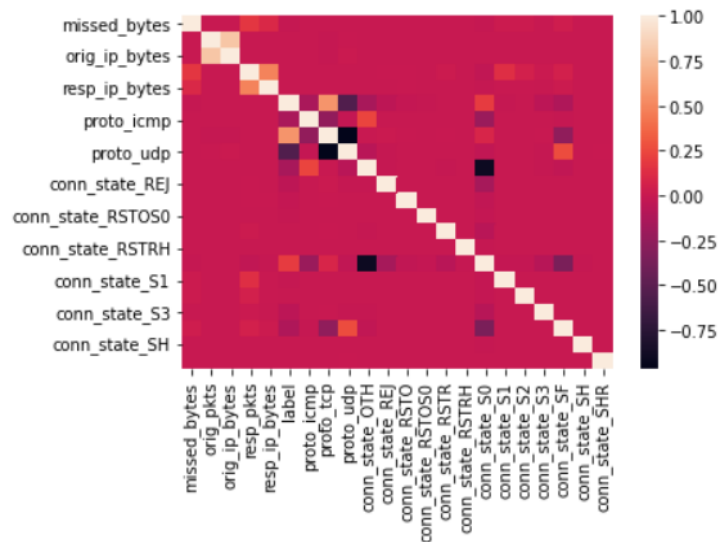


Fig 12: Correlation Matrix for IoT -23 after dropping irrelevant columns.

This now shows a high correlation between the values of the columns.

The following columns were eliminated on the basis of the Correlation Matrix:

'ts', 'local\_orig', 'local\_resp', 'id.orig\_p', 'id.resp\_p', 'service'.

The first three columns were eliminated since, on average, they were very weakly correlated to the classifier column: 'label'. The last three columns were eliminated since they had a lot of missing values. Thus, it was hard to draw any correlation between these columns and the 'label' column.

As is visible from the heat map of the Correlation Matrix above, the dataset now features columns and values that are relatively strongly related to one another. Thus, carrying out classification tasks on this modified dataset will lead to better results for classification algorithms that use correlation techniques and identify patterns.

It is observed that the outliers only exist in one column: 'orif ip bytes df'. Thus these outliers are



removed by taking values close to the standard deviation. We only keep the values within +3 to -3 standard deviations in the column 'orif\_ip\_bytes\_df.'

Finally, all the 33 files (the data frames) are converted into one CSV file that will be fed into the algorithms and models.

## Results:

### Analysis Methods:

To evaluate the algorithms described above, the metrics presented below were used. There are four concepts are pre-requisites for the metrics that are discussed:

- **True Positive (TP):** number of actual positives that were correctly identified.
- **True Negative (TN):** number of actual negatives that were correctly identified
- **False Positive (FP):** number of actual positives that were identified as negatives
- **False Negative (FN):** number of actual negatives that were identified as positives

To evaluate the algorithms described above, the metrics presented below were used:

1. **Precision:** Precision evaluates the model by calculating the fraction of correctly

identified positives.

2. **Recall:** The recall score evaluates the model by calculating the fraction of actual positives that were correctly identified.
3. **Accuracy:** Accuracy evaluates the model by calculating the fraction of correct predictions over the total number of predictions.
4. **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 F1 - micro-average metric.
5. **F-1 Score:** The F-1 Score is a metric that calculates the harmonic mean of the precision and recall score.
6. **Time:** Time is a metric that is used to measure the amount of time taken for a model to run.

## Graphs for Results Obtained and Inference:

The following graphs have been constructed for displaying the comparative analysis of these models along the metrics defined above. The following charts demonstrate the values of the F-1 score and the Time Taken for each algorithm.

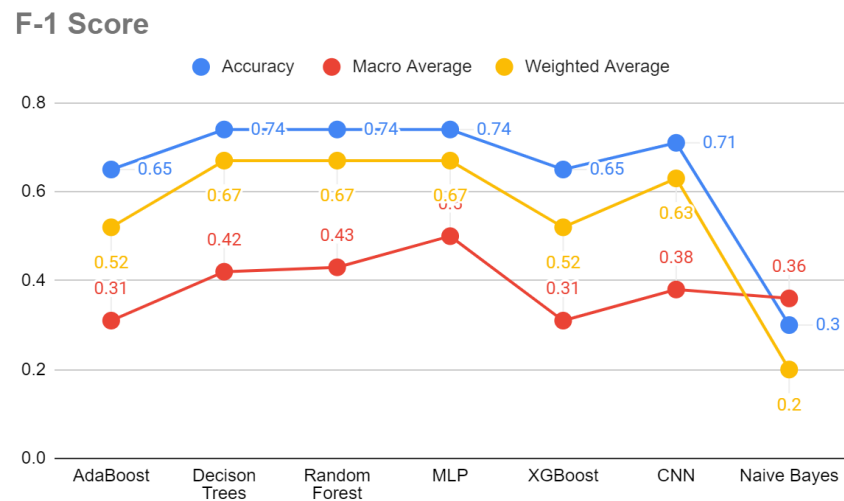


Fig 16: Plot of F-1 Values for all algorithms

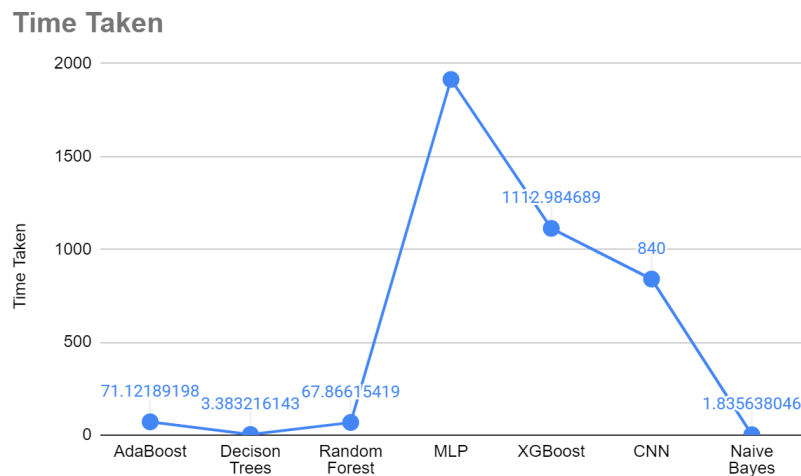


Fig 16: Plot of Time Taken (in seconds) for all algorithms

As can be seen in Figure 16, Random Forest Algorithm provides us with the best results. The Accuracy achieved by it is 74% for the F-1 score. The next best algorithm is Decision Tree since it also takes considerably less amount of time to train and test (a mere 3.38 seconds).

Although MLP and CNN have managed to provide us with results that lie within the ballpark of the best results achieved by the computationally lighter ML models, they aren't considered the optimum models to use because of their complex structure and construction.

Similarly, even though SVM produces decent results, it takes a significant amount to train. Thus, it is not ideal for carrying out analysis on large datasets like the IoT-23 dataset (which has 15,00,000 rows).

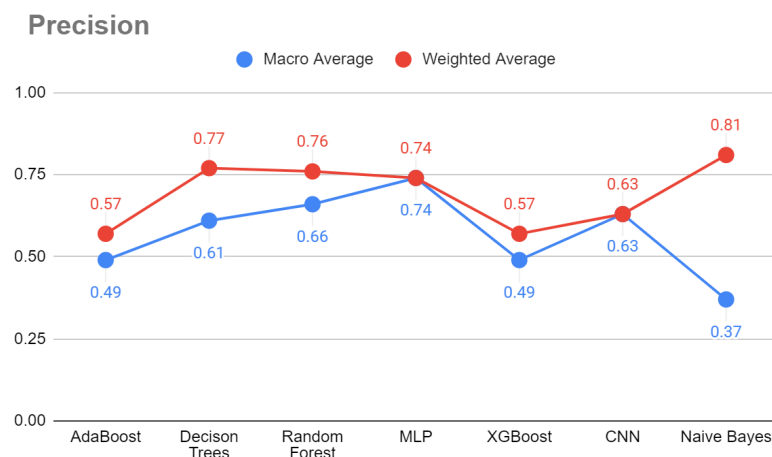


Fig 14: Plot of Precision Values for all algorithms

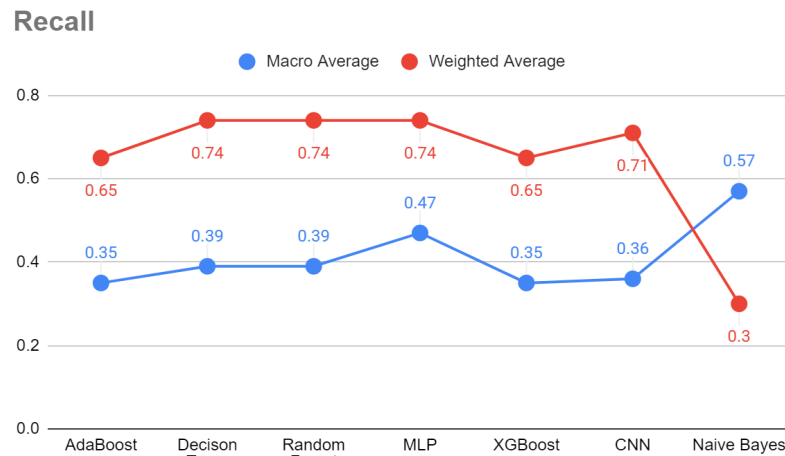


Fig 15: Plot of Recall Values for all algorithms

## Analysis of the Results:

Since Decision Trees and Random Forest algorithms are based on the same base model, their results being in the proximate range isn't surprising. The AdaBoost algorithm also provided decent results (Accuracy: 65%). This is because both AdaBoost and Random Forest algorithms use Decision trees as their meta classifier.

A possible explanation for this heightened performance could be attributed to the fact that some of the columns in the dataset are heavily correlated to the 'label' column. Hence, making simple assumptions is enough for correctly predicting the majority of classes. This is also corroborated by the fact that the Decision Trees algorithm shows excellent results on the IoT-23 dataset. This is just a hypothesis though and would require further research to prove it.

The reason why Naive Bayes performs badly is since not all columns in this dataset are

independent. The poor results obtained by it are most likely a result of the data under consideration not being independent. This is further exaggerated by the data-preprocessing steps that we carried out. Since we removed the data that was weakly correlated to the 'label' column, we increased the dependence of one column on the other. Hence, the naive assumptions made in this case prove to be wrong.

We used two Deep Learning methods: Multi-Layer Perceptron (MLP) and Convolutional Neural Networks (CNN) for this project. Even though both these models produced decent results, they weren't a big improvement on the less computationally expensive ML algorithms used. From looking at the confusion matrix obtained for these, it was clear that the models were showing a certain amount of bias towards the classes that had more occurrences.

A possible explanation for this might be the configuration of the neurons. Perhaps, the bias in their weight function was not large enough to allow the flexibility necessary for correct prediction classes that had scarce occurrences. Again, the cause of this lack of accuracy is not clear and should be further researched upon.

## **Conclusion:**

In conclusion, the Random Forest algorithm works the best for anomaly detection and classification in the IoT-23 dataset. More complex algorithms like CNN and MLP: Artificial Neural Networks also don't provide much improvement over these computationally lighter ML

models. The prediction tends to have a certain bias towards classes that have a greater number of occurrences. The problem in the detection of smaller classes lies in the fact that their proportion in the huge amount of data being used is minuscule.

## Appendix I - Code for Data Preprocessing

```
# -*- coding: utf-8 -*-
"""SOP_IoT-23 - Data Preprocessing.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/12YBljurUK3J8-6WWHHcLD86B93Z-KiFU
"""

import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/gdrive')

capture_34 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-1-1/bro/conn.log.labeled"
capture_43 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-43-1/bro/conn.log.labeled"
capture_44 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-44-1/bro/conn.log.labeled"
capture_49 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-49-1/bro/conn.log.labeled"
capture_52 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-52-1/bro/conn.log.labeled"
capture_20 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-20-1/bro/conn.log.labeled"
capture_21 =
```



```
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-21-1/bro/conn.log.labeled"
capture_42 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-42-1/bro/conn.log.labeled"
capture_60 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-60-1/bro/conn.log.labeled"
capture_17 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-17-1/bro/conn.log.labeled"
capture_36 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-36-1/bro/conn.log.labeled"
capture_33 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-33-1/bro/conn.log.labeled"
capture_8 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-8-1/bro/conn.log.labeled"
capture_35 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-35-1/bro/conn.log.labeled"
capture_48 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-48-1/bro/conn.log.labeled"
capture_39 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-39-1/bro/conn.log.labeled"
capture_7 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-7-1/bro/conn.log.labeled"
capture_9 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-9-1/bro/conn.log.labeled"
capture_3 =
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-3-1/bro/conn.log.labeled"
capture_1 =
```

```
"/content/gdrive/MyDrive/iot_23_data/opt/Malware-Project/BigDataset/IoTSce
narios/CTU-IoT-Malware-Capture-1-1/bro/conn.log.labeled"
```

```
# captures = [capture_1, capture_17, capture_20, capture_21, capture_3,
capture_33, capture_34, capture_35, capture_36, capture_39, capture_42,
capture_43, capture_44, capture_48, capture_49, capture_52, capture_60,
capture_7, capture_8, capture_9]
```

```
# def parse_dataframe(file_path, skiprows=10, nrows=100000):
#     df = pd.read_table(filepath_or_buffer=file_path, skiprows=skiprows,
nrows=nrows)
```

```
#     df.columns=['ts',
#                 'uid',
#                 'id.orig_h',
#                 'id.orig_p',
#                 'id.resp_h',
#                 'id.resp_p',
#                 'proto',
#                 'service',
#                 'duration',
#                 'orig_bytes',
#                 'resp_bytes',
#                 'conn_state',
#                 'local_orig',
#                 'local_resp',
#                 'missed_bytes',
#                 'history',
#                 'orig_pkts',
#                 'orig_ip_bytes',
#                 'resp_pkts',
#                 'resp_ip_bytes'
#                 'label']
#     df.drop(df.tail(1).index,inplace=True)
#     return df
```

```
# df_c = pd.DataFrame()
# for capture in captures:
#     df_c = pd.concat([df_c, parse_dataframe(capture)])
```

```

df34 = pd.read_table(filepath_or_buffer=capture_34, skiprows=10,
nrows=100000)
df34.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']
df34.drop(df34.tail(1).index,inplace=True)

```

```

df43 = pd.read_table(filepath_or_buffer=capture_43, skiprows=10,
nrows=100000)
df43.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',

```

```

        'local_orig',
        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']
df43.drop(df43.tail(1).index,inplace=True)

df44 = pd.read_table(filepath_or_buffer=capture_44, skiprows=10,
nrows=100000)
df44.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']
df44.drop(df44.tail(1).index,inplace=True)

df49 = pd.read_table(filepath_or_buffer=capture_49, skiprows=10,
nrows=100000)
df49.columns=['ts',

```

```

        'uid',
        'id.orig_h',
        'id.orig_p',
        'id.resp_h',
        'id.resp_p',
        'proto',
        'service',
        'duration',
        'orig_bytes',
        'resp_bytes',
        'conn_state',
        'local_orig',
        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']

df49.drop(df49.tail(1).index,inplace=True)

df52 = pd.read_table(filepath_or_buffer=capture_52, skiprows=10,
nrows=100000)
df52.columns=['ts',
        'uid',
        'id.orig_h',
        'id.orig_p',
        'id.resp_h',
        'id.resp_p',
        'proto',
        'service',
        'duration',
        'orig_bytes',
        'resp_bytes',
        'conn_state',
        'local_orig',
        'local_resp',
        'missed_bytes',

```

```

        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']
df52.drop(df52.tail(1).index,inplace=True)

df20 = pd.read_table(filepath_or_buffer=capture_20, skiprows=10,
nrows=100000)
df20.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']
df20.drop(df20.tail(1).index,inplace=True)

df21 = pd.read_table(filepath_or_buffer=capture_21, skiprows=10,
nrows=100000)
df21.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',

```

```

        'id.resp_h',
        'id.resp_p',
        'proto',
        'service',
        'duration',
        'orig_bytes',
        'resp_bytes',
        'conn_state',
        'local_orig',
        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']

df21.drop(df21.tail(1).index,inplace=True)

df42 = pd.read_table(filepath_or_buffer=capture_42, skiprows=10,
nrows=100000)
df42.columns=['ts',
               'uid',
               'id.orig_h',
               'id.orig_p',
               'id.resp_h',
               'id.resp_p',
               'proto',
               'service',
               'duration',
               'orig_bytes',
               'resp_bytes',
               'conn_state',
               'local_orig',
               'local_resp',
               'missed_bytes',
               'history',
               'orig_pkts',
               'orig_ip_bytes',

```

```

        'resp_pkts',
        'resp_ip_bytes',
        'label']
df42.drop(df42.tail(1).index,inplace=True)

df60 = pd.read_table(filepath_or_buffer=capture_60, skiprows=10,
nrows=100000)
df60.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']
df60.drop(df60.tail(1).index,inplace=True)

df17 = pd.read_table(filepath_or_buffer=capture_17, skiprows=10,
nrows=100000)
df17.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',

```



```

        'service',
        'duration',
        'orig_bytes',
        'resp_bytes',
        'conn_state',
        'local_orig',
        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']

df17.drop(df17.tail(1).index,inplace=True)

df36 = pd.read_table(filepath_or_buffer=capture_36, skiprows=10,
nrows=100000)
df36.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']

```

```
df36.drop(df36.tail(1).index,inplace=True)
```

```
df33 = pd.read_table(filepath_or_buffer=capture_33, skiprows=10,
nrows=100000)
```

```
df33.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']
```

```
df33.drop(df33.tail(1).index,inplace=True)
```

```
df8 = pd.read_table(filepath_or_buffer=capture_8, skiprows=10,
nrows=100000)
```

```
df8.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
```

```

        'resp_bytes',
        'conn_state',
        'local_orig',
        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']

df8.drop(df8.tail(1).index,inplace=True)

df35 = pd.read_table(filepath_or_buffer=capture_35, skiprows=10,
nrows=100000)
df35.columns=['ts',
              'uid',
              'id.orig_h',
              'id.orig_p',
              'id.resp_h',
              'id.resp_p',
              'proto',
              'service',
              'duration',
              'orig_bytes',
              'resp_bytes',
              'conn_state',
              'local_orig',
              'local_resp',
              'missed_bytes',
              'history',
              'orig_pkts',
              'orig_ip_bytes',
              'resp_pkts',
              'resp_ip_bytes',
              'label']

df35.drop(df35.tail(1).index,inplace=True)

df48 = pd.read_table(filepath_or_buffer=capture_48, skiprows=10,

```

```

nrows=100000)
df48.columns=['ts',
               'uid',
               'id.orig_h',
               'id.orig_p',
               'id.resp_h',
               'id.resp_p',
               'proto',
               'service',
               'duration',
               'orig_bytes',
               'resp_bytes',
               'conn_state',
               'local_orig',
               'local_resp',
               'missed_bytes',
               'history',
               'orig_pkts',
               'orig_ip_bytes',
               'resp_pkts',
               'resp_ip_bytes',
               'label']

df48.drop(df48.tail(1).index,inplace=True)

df39 = pd.read_table(filepath_or_buffer=capture_39, skiprows=10,
nrows=100000)
df39.columns=['ts',
               'uid',
               'id.orig_h',
               'id.orig_p',
               'id.resp_h',
               'id.resp_p',
               'proto',
               'service',
               'duration',
               'orig_bytes',
               'resp_bytes',
               'conn_state',
               'local_orig',

```

```

        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']
df39.drop(df39.tail(1).index,inplace=True)

df7 = pd.read_table(filepath_or_buffer=capture_7, skiprows=10,
nrows=100000)
df7.columns=['ts',
            'uid',
            'id.orig_h',
            'id.orig_p',
            'id.resp_h',
            'id.resp_p',
            'proto',
            'service',
            'duration',
            'orig_bytes',
            'resp_bytes',
            'conn_state',
            'local_orig',
            'local_resp',
            'missed_bytes',
            'history',
            'orig_pkts',
            'orig_ip_bytes',
            'resp_pkts',
            'resp_ip_bytes',
            'label']
df7.drop(df7.tail(1).index,inplace=True)

df9 = pd.read_table(filepath_or_buffer=capture_9, skiprows=10,
nrows=100000)
df9.columns=['ts',
            'uid',

```

```

        'id.orig_h',
        'id.orig_p',
        'id.resp_h',
        'id.resp_p',
        'proto',
        'service',
        'duration',
        'orig_bytes',
        'resp_bytes',
        'conn_state',
        'local_orig',
        'local_resp',
        'missed_bytes',
        'history',
        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']

df9.drop(df9.tail(1).index,inplace=True)

df3 = pd.read_table(filepath_or_buffer=capture_3, skiprows=10,
nrows=100000)
df3.columns=['ts',
            'uid',
            'id.orig_h',
            'id.orig_p',
            'id.resp_h',
            'id.resp_p',
            'proto',
            'service',
            'duration',
            'orig_bytes',
            'resp_bytes',
            'conn_state',
            'local_orig',
            'local_resp',
            'missed_bytes',
            'history',

```

```

        'orig_pkts',
        'orig_ip_bytes',
        'resp_pkts',
        'resp_ip_bytes',
        'label']
df3.drop(df3.tail(1).index,inplace=True)

df1 = pd.read_table(filepath_or_buffer=capture_1, skiprows=10,
nrows=100000)
df1.columns=['ts',
            'uid',
            'id.orig_h',
            'id.orig_p',
            'id.resp_h',
            'id.resp_p',
            'proto',
            'service',
            'duration',
            'orig_bytes',
            'resp_bytes',
            'conn_state',
            'local_orig',
            'local_resp',
            'missed_bytes',
            'history',
            'orig_pkts',
            'orig_ip_bytes',
            'resp_pkts',
            'resp_ip_bytes',
            'label']
df1.drop(df1.tail(1).index,inplace=True)

frames=[df1, df17, df20, df21, df3, df33, df34, df35, df36, df39, df42,
df43, df44, df48, df49, df52, df60, df7, df8, df9]

df_c=pd.concat(frames)

df_c

```

```
df_c.shape
```

```
df_c['label'].value_counts()
```

```
df_c.loc[(df_c.label == '- Malicious PartOfAHorizontalPortScan'),
'label'] = 'PartOfAHorizontalPortScan'
df_c.loc[(df_c.label == '(empty) Malicious
PartOfAHorizontalPortScan'), 'label'] = 'PartOfAHorizontalPortScan'
df_c.loc[(df_c.label == '- Malicious Okiru'), 'label'] = 'Okiru'
df_c.loc[(df_c.label == '(empty) Malicious Okiru'), 'label'] = 'Okiru'
df_c.loc[(df_c.label == '- Benign -'), 'label'] = 'Benign'
df_c.loc[(df_c.label == '(empty) Benign -'), 'label'] = 'Benign'
df_c.loc[(df_c.label == '- Malicious DDoS'), 'label'] = 'DDoS'
df_c.loc[(df_c.label == '- Malicious C&C'), 'label'] = 'C&C'
df_c.loc[(df_c.label == '(empty) Malicious C&C'), 'label'] = 'C&C'
df_c.loc[(df_c.label == '- Malicious Attack'), 'label'] = 'Attack'
df_c.loc[(df_c.label == '(empty) Malicious Attack'), 'label'] =
'Attack'
df_c.loc[(df_c.label == '- Malicious C&C-HeartBeat'), 'label'] =
'C&C-HeartBeat'
df_c.loc[(df_c.label == '(empty) Malicious C&C-HeartBeat'), 'label'] =
'C&C-HeartBeat'
df_c.loc[(df_c.label == '- Malicious C&C-FileDownload'), 'label'] =
'C&C-FileDownload'
df_c.loc[(df_c.label == '- Malicious C&C-Torii'), 'label'] =
'C&C-Torii'
df_c.loc[(df_c.label == '- Malicious C&C-HeartBeat-FileDownload'),
'label'] = 'C&C-HeartBeat-FileDownload'
df_c.loc[(df_c.label == '- Malicious FileDownload'), 'label'] =
'FileDownload'
df_c.loc[(df_c.label == '- Malicious C&C-Mirai'), 'label'] =
'C&C-Mirai'
df_c.loc[(df_c.label == '- Malicious Okiru-Attack'), 'label'] =
'Okiru-Attack'
```

```
"""13 types of malicious attack"""
```

```
df_c['label'].value_counts()
```



```

pd.options.display.max_rows = 300
pd.options.display.max_columns = 300

# df_c = df_c.replace('-', '0')
# #read about missing value handling
# # df_c['service'] = df_c['service'].str.replace('-', '0')
df_c['duration'] = df_c['duration'].str.replace('-', '0')
df_c['orig_bytes'] = df_c['orig_bytes'].str.replace('-', '0')
df_c['resp_bytes'] = df_c['resp_bytes'].str.replace('-', '0')

df_c.fillna(-1, inplace=True)

df_c

df_c.info()

# df_c =
df_c.astype({'ts': 'float64', 'duration': 'float64', 'orig_bytes': 'float64', 'r
esp_bytes': 'float64', 'local_orig': 'float64', 'local_resp': 'float64'})

# df_c.info()

df_c.uid.unique()

df_c = df_c.drop(columns=['uid', 'id.orig_h', 'id.resp_h'])

df_c.proto.unique()

df_c = pd.get_dummies(df_c, columns=['proto'])

# df_c = df_c.replace({"proto": {'tcp': 1, 'udp': 2, 'icmp': 3}})

# df_c.proto.unique()

df_c.service.unique()

df_c = df_c.replace({"service": {'-2': -2, 'http': 1, 'dns': 2, 'irc': 3,
'ssh': 4, 'dhcp': 5, 'ssl': 7}})

```

```

df_c.service.unique()

df_c.conn_state.unique()

df_c = pd.get_dummies(df_c, columns=['conn_state'])

# df_c = df_c.replace({"conn_state":{"S0":1, 'REJ':2, 'SF':3, 'OTH':4,
'RTOS0':5, 'RSTR':6, 'S2':7, 'RSTRH':8, 'RSTO':9, 'S1':10, 'SHR':11,
'SH':12, 'S3':13}}})

# df_c.conn_state.unique()

df_c.history.unique()

df_c = df_c.drop(columns=['history'])

df_c.label.unique()

df_c =
df_c.replace({"label":{"Benign":0, "C&C":1, "C&C-FileDownload":1, "C&C-HeartB
eat":2, 'C&C-HeartBeat-FileDownload':2, "PartOfAHorizontalPortScan":8, 'Okiru
':7, 'DDoS':5, 'C&C-Torii':4, 'Attack':9, 'FileDownload':6,
'C&C-Mirai':3}}})

df_c.label.unique()

df_c.info()

#df_c =
df_c.astype({'proto':'float64', 'service':'float64', 'conn_state':'float64',
'label':'float64'})

# df_c.info()

corrMatrix = df_c.corr()
sn.heatmap(corrMatrix)
plt.show()

corrMatrix

```

```

df_c = df_c.drop(columns=['ts', 'local_orig', 'local_resp'])

df_c = df_c.drop(columns=['id.orig_p', 'id.resp_p', 'service'])

corrMatrix = df_c.corr()
sn.heatmap(corrMatrix)
plt.show()

df_c

df_c.plot(kind='box');

# red_circle = dict(markerfacecolor='red', marker='o',
#                    markeredgecolor='white')

# fig, axs = plt.subplots(1, len(df_c.columns), figsize=(20,10))

# for i, ax in enumerate(axs.flat):
#     ax.boxplot(df_c.iloc[:,i], flierprops=red_circle)
#     ax.set_title(df_c.columns[i], fontsize=20, fontweight='bold')
#     ax.tick_params(axis='y', labelsize=14)

# plt.show()

# df_c = df_c[np.abs(df_c.orig_bytes-df_c.orig_bytes.mean()) <=
# (3*df_c.orig_bytes.std())]
# # keep only the ones that are within +3 to -3 standard deviations in the
# column 'Data'._

# keep only the ones that are within +3 to -3 standard deviations in the
# column 'Data'._
df_c = df_c[np.abs(df_c.orig_ip_bytes-df_c.orig_ip_bytes.mean()) <=
(3*df_c.orig_ip_bytes.std())]

# from scipy import stats
# df_c = df_c[(np.abs(stats.zscore(df_c)) < 3).all(axis=1)]

df_c

```

```
# df_c.orig_bytes.unique()

df_c.isna().sum()

print(df_c.columns.tolist())

df_c.to_csv('iot23_combined.csv')
```

## Appendix II - Code for ML Models

### Decision Trees

```
# -*- coding: utf-8 -*-
"""Decision_Trees-IoT23.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1IRAW9PA_gRxgLoBxWAHL1-pCw6BNdxtW
"""

import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import time
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
import warnings

from google.colab import drive
drive.mount('/content/gdrive')

filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"

df = pd.read_csv(filepath)

del df['Unnamed: 0']

df
```

```

df['label'].value_counts()

DT = DecisionTreeClassifier()

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10,
test_size=0.2)

start = time.time()
print('program start...')
print()

DT.fit(X_train, Y_train)
print()

print('prediction:')
y_pred = DT.predict(X_test)
print(y_pred)
print()

print('Score:')
score = DT.score(X_test, Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

warnings.filterwarnings('ignore')

print("Classification Report :")

```

```
print(classification_report(Y_test, y_pred))
```

## Random Forest

```
# -*- coding: utf-8 -*-
```

```
"""Random_Forest-IoT23.ipynb
```

```
Automatically generated by Colaboratory.
```

```
Original file is located at
```

```
https://colab.research.google.com/drive/1oI74klVIK5EpPt3Uc4b08PUZ-dMOQugY
```

```
"""
```

```
import pandas as pd
```

```
from matplotlib import pyplot as plt
```

```
from sklearn.svm import SVC
```

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.model_selection import train_test_split
```

```
import time
```

```
import numpy as np
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
import warnings
```

```
from google.colab import drive
```

```
drive.mount('/content/gdrive')
```

```
filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"
```

```
df = pd.read_csv(filepath)
```

```
del df['Unnamed: 0']
```

```

df

df['label'].value_counts()

warnings.filterwarnings('ignore')

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10,
test_size=0.2)

clf=RandomForestClassifier(n_estimators =100, random_state=10)

start = time.time()
print('program start...')
print()

clf.fit(X_train,Y_train)
print()

print('prediction:')
y_pred=clf.predict(X_test)
print(y_pred)
print()

print('Score:')
score = clf.score(X_test,Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

```



```

print("Classification Report :")
print(classification_report(Y_test, y_pred))

clf=RandomForestClassifier(n_estimators =200, random_state=10)

start = time.time()
print('program start...')
print()

clf.fit(X_train,Y_train)
print()

print('prediction:')
y_pred=clf.predict(X_test)
print(y_pred)
print()

print('Score:')
score = clf.score(X_test,Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

clf=RandomForestClassifier(n_estimators =100, random_state=10)

param_grid = {
    'max_features': ['auto', 'sqrt', 'log2',0.2],
    'max_depth' : [5,6,7,8],
    'criterion' :['gini', 'entropy']
}

```

```

CV_rfc = GridSearchCV(estimator=clf, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train, Y_train)

CV_rfc.best_params_

clf=RandomForestClassifier(n_estimators =100, random_state=10, criterion =
'entropy', max_depth = 8, max_features = 'auto')

start = time.time()
print('program start...')
print()

clf.fit(X_train,Y_train)
print()

print('prediction:')
y_pred=clf.predict(X_test)
print(y_pred)
print()

print('Score:')
score = clf.score(X_test,Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

```

## Naive Bayes

```
# -*- coding: utf-8 -*-
```

```
"""Naive_Bayes-IoT23.ipynb
```

```
Automatically generated by Colaboratory.
```

```
Original file is located at
```

```
https://colab.research.google.com/drive/13\_2C4RXcmIsRozPAiOisBImHN34x6N2z
"""
```

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import time
import numpy as np
import warnings

from google.colab import drive
drive.mount('/content/gdrive')
filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"

df = pd.read_csv(filepath)

del df['Unnamed: 0']
df

df.shape

df['label'].value_counts()

for col in df.columns:
    print(col)

print(df.columns.tolist())

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
```

```

'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
normalized_x = scaler.transform(X)

from sklearn.naive_bayes import GaussianNB

X_train, X_test, Y_train, Y_test = train_test_split(normalized_x, Y,
random_state=100, test_size=0.2)

clf = GaussianNB()
clf.fit(X_train, Y_train)

start = time.time()
print('program start...')
print()

clf = GaussianNB().fit(X_train, Y_train)
print()
print(clf.score(X_test, Y_test))
print()

y_pred = clf.fit(X_train, Y_train).predict(X_test)
print(y_pred)
print()

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

warnings.filterwarnings('ignore')

```

```
print("Classification Report :")
print(classification_report(Y_test, y_pred))
```

## Support Vector Machine

```
# -*- coding: utf-8 -*-
"""SVM-IoT23.ipynb
```

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/drive/17vXQhxXKdyTPsFuWja2GW13CIT43ulk6>

```
!pip install thundersvm
```

```
!wget
https://developer.nvidia.com/compute/cuda/9.0/Prod/local_installers/cuda-r
epo-ubuntu1704-9-0-local_9.0.176-1_amd64-deb
!ls # Check if required cuda 9.0 amd64-deb file is downloaded
!dpkg -i cuda-repo-ubuntu1704-9-0-local_9.0.176-1_amd64-deb
!ls /var/cuda-repo-9-0-local | grep .pub
!apt-key add /var/cuda-repo-9-0-local/7fa2af80.pub
!apt-get update
!sudo apt-get install cuda-9.0

!nvcc --version
```

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import time
import numpy as np
```

```

import thundersvm
import warnings

from google.colab import drive
drive.mount('/content/gdrive')

filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"

df = pd.read_csv(filepath, nrows=800000)

df

del df['Unnamed: 0']

df.shape

df['label'].value_counts()

df.columns = ['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'label',
'proto_icmp', 'proto_tcp', 'proto_udp', 'conn_state_OTH',
'conn_state_REJ', 'conn_state_RSTO', 'conn_state_RSTOS0',
'conn_state_RSTR', 'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1',
'conn_state_S2', 'conn_state_S3', 'conn_state_SF', 'conn_state_SH',
'conn_state_SHR']

for col in df.columns:
    print(col)

print(df.columns.tolist())

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']

```

```

# SVM_classifier = SVC()
# X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
random_state=10, test_size=0.2)

start = time.time()
print('program start...')
print()

svc_gpu = thundersvm.SVC(C=10, kernel='rbf')
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10,
test_size=0.2)
svc_gpu.fit(X_train, Y_train)
print()

print('Prediction:')
y_pred = svc_gpu.predict(X_test)
print(y_pred)
print()

print('Score:')
score = svc_gpu.score(X_test, Y_test)
print(score)
print()
# accuracy_svc = svc_gpu.score(X_test, Y_test)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

warnings.filterwarnings('ignore')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

# accuracy_svc

# start = time.time()

```

```

# print('program start...')
# print()

# SVM_classifier = SVC(C=1.0, cache_size=1500, verbose=True).fit(X_train,
Y_train)
# print()
# print(SVM_classifier.score(X_test, Y_test))
# print()

# y_pred = SVM_classifier.predict(X_test)
# print(y_pred)
# print()

# end = time.time()
# print('program end...')
# print()
# print('time cost: ')
# print(end - start, 'seconds')

```

## AdaBoost

```

# -*- coding: utf-8 -*-
"""AdaBoost-IoT23.ipynb

```

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/drive/1AUJ7Es2EZUK5aTcDTXiPKGKB4Z6u7PxJ>

```

import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

```



```

import time
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics
import warnings

# from sklearn import datasets
# # Import train_test_split function
# from sklearn.model_selection import train_test_split
# #Import scikit-learn metrics module for accuracy calculation

from google.colab import drive
drive.mount('/content/gdrive')

filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"

df = pd.read_csv(filepath)

del df['Unnamed: 0']

df

df['label'].value_counts()

abc = AdaBoostClassifier()
# Train Adaboost Classifier

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10,
test_size=0.2)

```

```

start = time.time()
print('program start...')
print()

# DT.fit(X_train, Y_train)
model = abc.fit(X_train, Y_train)
print()

print('prediction:')
y_pred = model.predict(X_test)
# y_pred = DT.predict(X_test)
print(y_pred)
print()

print('Score:')
# score = metrics.accuracy_score(y_test, y_pred)
score = model.score(X_test, Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

warnings.filterwarnings('ignore')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

param_grid = {
    "n_estimators": [50, 100, 200],
    "learning_rate" : [0.9, 1.0, 1.1],
}

grid_search_abc = GridSearchCV(abc, param_grid=param_grid)
grid_search_abc.fit(X_train, Y_train)

grid_search_abc.best_params_

```

```

abc = AdaBoostClassifier(n_estimators=50,learning_rate=1.1)

start = time.time()
print('program start...')
print()

# DT.fit(X_train, Y_train)
model = abc.fit(X_train, Y_train)
print()

print('prediction:')
y_pred = model.predict(X_test)
# y_pred = DT.predict(X_test)
print(y_pred)
print()

print('Score:')
# score = metrics.accuracy_score(y_test, y_pred)
score = model.score(X_test,Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

```

## XGBoost

```

"""XGBoost-IoT23.ipynb

```

Automatically generated by Colaboratory.

Original file is located at

[https://colab.research.google.com/drive/1gq7ZtiHdOhWdYmtDP18\\_BpVOq6GVL6gr](https://colab.research.google.com/drive/1gq7ZtiHdOhWdYmtDP18_BpVOq6GVL6gr)

"""

```
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import time
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn import metrics
import warnings

# from sklearn import datasets
# # Import train_test_split function
# from sklearn.model_selection import train_test_split
# #Import scikit-learn metrics module for accuracy calculation

from google.colab import drive
drive.mount('/content/gdrive')

filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"

df = pd.read_csv(filepath)

del df['Unnamed: 0']

df

df['label'].value_counts()

xgb = XGBClassifier()
# Train Adaboost Classifier
```

```

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10,
test_size=0.2)

start = time.time()
print('program start...')
print()

# DT.fit(X_train, Y_train)
model = xgb.fit(X_train, Y_train)
print()

print('prediction:')
y_pred = model.predict(X_test)
# y_pred = DT.predict(X_test)
print(y_pred)
print()

print('Score:')
# score = metrics.accuracy_score(y_test, y_pred)
score = model.score(X_test, Y_test)
print(score)

end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

warnings.filterwarnings('ignore')

print("Classification Report :")

```

```
print(classification_report(Y_test, y_pred))
```

## Multi-Layer Perceptron

```
# -*- coding: utf-8 -*-
```

```
"""MLP-IoT23.ipynb
```

```
Automatically generated by Colaboratory.
```

```
Original file is located at
```

```
https://colab.research.google.com/drive/1xP2KTgApPKNXL2qI1JP8wK8KQN1qtTUs
```

```
"""
```

```
import pandas as pd
```

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.model_selection import train_test_split
```

```
import time
```

```
import numpy as np
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.preprocessing import StandardScaler
```

```
import warnings
```

```
from google.colab import drive
```

```
drive.mount('/content/gdrive')
```

```
filepath = "/content/gdrive/MyDrive/iot_23_data/iot23_combined (2).csv"
```

```
df = pd.read_csv(filepath)
```

```
del df['Unnamed: 0']
```

```
df
```

```

df['label'].value_counts()

mlp = MLPClassifier(random_state=1)

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
Y = df['label']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=10,
test_size=0.2)

start = time.time()
print('program start...')
print()

mlp.fit(X_train, Y_train)
# SC.fit(X_train, Y_train)
# print()

print('prediction:')
y_pred=mlp.predict(X_test)
# y_pred = SC.predict(X_test)
print(y_pred)
print()

print('Score:')
score = mlp.score(X_test, Y_test)
print(score)
print()
# score = SC.score(X_test,Y_test)
# print(score)

end = time.time()
print('program end...')
print()

```

```

print('time cost: ')
print(end - start, 'seconds')

warnings.filterwarnings('ignore')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

```

## Convolutional Neural Networks

```

# -*- coding: utf-8 -*-
"""CNN-IoT23.ipynb

```

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/drive/1qRWOMr6GWtH9CctKhPRENy4sBX78h43f>

```

import numpy as np
import pandas as pd
import tensorflow as tf
from matplotlib import pyplot as plt
from tensorflow.keras import Model
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras import Sequential
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, Input, Dropout, Activation,
Dense, MaxPooling2D, Flatten, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adadelta
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from google.protobuf import text_format
import sklearn

```



```

import warnings
from sklearn.preprocessing import MinMaxScaler

from google.colab import drive
drive.mount('/content/drive')

filepath = "/content/drive/MyDrive/iot_23_data/iot23_combined (2).csv"

df = pd.read_csv(filepath)

df

del df['Unnamed: 0']

df['label'].value_counts()

from sklearn.model_selection import train_test_split

X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH',
'conn_state_SHR']].values

X.shape

Y = pd.get_dummies(df['label']).values

Y.shape

#X = df[['duration', 'orig_bytes', 'resp_bytes', 'missed_bytes',
'orig_pkts', 'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'proto_icmp',
'proto_tcp', 'proto_udp', 'conn_state_OTH', 'conn_state_REJ',
'conn_state_RSTO', 'conn_state_RSTOS0', 'conn_state_RSTR',
'conn_state_RSTRH', 'conn_state_S0', 'conn_state_S1', 'conn_state_S2',
'conn_state_S3', 'conn_state_SF', 'conn_state_SH', 'conn_state_SHR']]
#Y = df[['label']]

```

```
X
```

```
df
```

```
scaler = MinMaxScaler()
```

```
scaler.fit(X)
```

```
normalized_x = scaler.transform(X)
```

```
normalized_x
```

```
normalized_x.shape
```

```
scaler.fit(Y)
```

```
normalized_y = scaler.transform(Y)
```

```
normalized_y
```

```
X_train, X_test, Y_train, Y_test = train_test_split(normalized_x,
normalized_y, random_state=10, test_size=0.2)
```

```
X_train.shape
```

```
model = Sequential()
```

```
model.add(Dense(2000, activation='relu', input_dim=24))
```

```
model.add(Dense(1500, activation='relu'))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(800, activation='relu'))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(400, activation='relu'))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(150, activation='relu'))
```

```
model.add(Dropout(0.2))
```

```
model.add(Dense(10, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
```

```

metrics=['accuracy'])

model.summary()

import time

start = time.time()
print('program start...')
print()

history = model.fit(X_train, Y_train, epochs = 10, batch_size=256,
validation_data=(X_test,Y_test),verbose=1)

print('prediction:')
y_pred=model.predict(X_test)
print(y_pred)
print()

print()
end = time.time()
print('program end...')
print()
print('time cost: ')
print(end - start, 'seconds')

y_pred

model.evaluate(X_test,Y_test)

# y_pred=np.argmax(y_pred, axis=1)
# Y_test=np.argmax(Y_test, axis=1)

warnings.filterwarnings('ignore')

print("Classification Report :")
print(classification_report(Y_test, y_pred))

```

## References

[1] What is Machine Learning?

<https://www.mygreatlearning.com/blog/what-is-machine-learning/>

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

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

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

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

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

[7] 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

[8] 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

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

[10] T. M. Mitchell, “Machine learning,” in Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, Pasadena, CA, USA, July 2009.

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

[12]What are outliers in the data?

<https://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm>