



Machine Learning for Malware Detection in IoT Networks

Huu Phuc CHU - Ngoc Khanh DAM

Contents

1	Dataset description	2
2	Data preprocessing	2
2.1	Handling noise	4
2.2	Handling missing data	5
2.3	Data transformation	5
2.4	Data normalization	5
2.5	Feature selection	5
3	Data visualization	6
4	Implementation result	10
4.1	Supervised learning methods	10
4.1.1	XGBoost	11
4.1.2	Random Forest	11
4.1.3	Naive Bayes	12
4.1.4	Support Vector Machine	13
4.2	Unsupervised learning methods	14

1 Dataset description

The IoT-23 dataset consists of network traffic data collected from IoT devices under both normal and malicious scenarios. It contains 23 labeled scenarios, with 20 scenarios involving malicious activity (e.g., Mirai botnet attacks, network scans, and exploits) and 3 scenarios with purely benign traffic. The data is provided in PCAP format, allowing for detailed packet-level analysis.

IOT23 is a very large dataset, including a total of 325,307,990 captures. Due to the scope of this project and computational constraints, we will extract a subset of the dataset, including 1,044,258 samples, 21 features, 2 labels with malicious and benign values, 3 detailed labels for attack types (C&C, DDos, PartOfAHorizontalPortScan). The dataset that we use is a combination of multiple files:

- Malware files containing both malicious and benign traffic:
 - CTU-IoT-Malware-Capture-8-1: generated by the Hakai malware.
 - CTU-IoT-Malware-Capture-1-1: generated by the Hide and Seek malware.
 - CTU-IoT-Malware-Capture-34-1: generated by the Mirai malware.
- Files containing only benign traffic:
 - CTU-Honeypot-Capture-4-1
 - CTU-Honeypot-Capture-5-1
 - CTU-Honeypot-Capture-7-1

Table 1 describes the features of the dataset:

2 Data preprocessing

After reading data into dataframes, we get the following statistics:

Column name	Description	Type	# missing value rows
ts	The time when the capture was done, expressed in Unix Time	float	0
uid	The ID of the capture	str	0
id.orig_h	The IP address where the attack happened, either IPv4 or IPv6	str	0
id.orig_p	The port used by the responder	float	0
id.resp_h	The IP address of the device on which the capture happened	str	0
id.resp_p	The port used for the response from the device where the capture happened	float	0
proto	The network protocol used for the data package	str	0
service	The application protocol	str	1.037.933
duration	The amount of time data was traded between the device and the attacker	float	820.614
orig_bytes	The amount of data sent to the device	float	820.614
resp_bytes	The amount of data sent by the device	float	820.614
conn_state	The state of the connection	str	0
local_orig	Whether the connection originated locally	float	1.044.258
local_resp	Whether the response originated locally	float	1.044.258
missed_bytes	Number of missed bytes in a message	float	0
history	The history of the state of the connection	str	17.562
orig_pkts	Number of packets being sent to the device	float	0
orig_ip_bytes	Number of bytes being sent to the device	float	0
resp_pkts	Number of packets being sent from the device	float	0
resp_ip_bytes	Number of bytes being sent from the device	float	0
tunnel_parents	The ID of the connection, if tunnelled	str	25.107
label	The type of capture, benign or malicious	str	0
detailed_label	If the capture is malicious, the type of attack	str	475.341

Table 1: Feature descriptions of the dataset

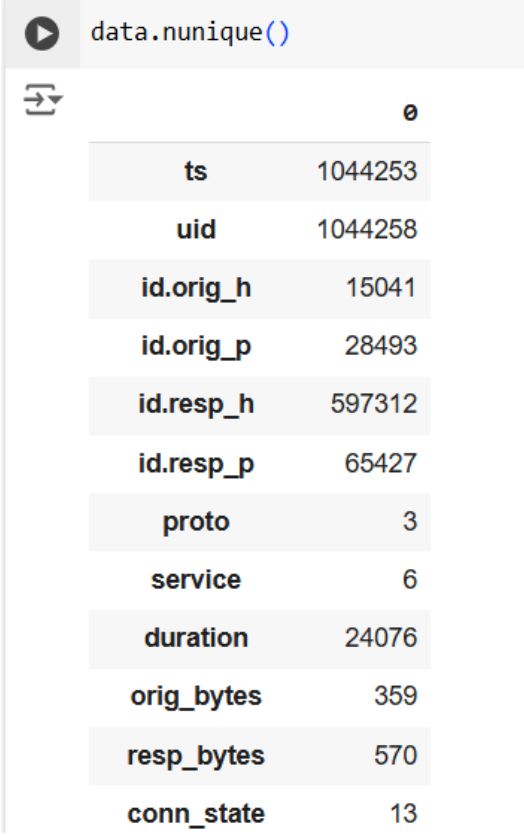
```

<class 'pandas.core.frame.DataFrame'>
Index: 1044258 entries, 0 to 130
Data columns (total 23 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ts                    1044258 non-null  object
1   uid                   1044258 non-null  object
2   id.orig_h            1044255 non-null  object
3   id.orig_p            1044252 non-null  float64
4   id.resp_h            1044252 non-null  object
5   id.resp_p            1044252 non-null  float64
6   proto                1044252 non-null  object
7   service              6325 non-null     object
8   duration              223644 non-null   float64
9   orig_bytes           223644 non-null   float64
10  resp_bytes           223644 non-null   float64
11  conn_state           1044252 non-null   object
12  local_orig           0 non-null        float64
13  local_resp           0 non-null        float64
14  missed_bytes         1044252 non-null   float64

```

2.1 Handling noise

The number of unique (no duplicate) values for each feature show in Fig 2.



	0
ts	1044253
uid	1044258
id.orig_h	15041
id.orig_p	28493
id.resp_h	597312
id.resp_p	65427
proto	3
service	6
duration	24076
orig_bytes	359
resp_bytes	570
conn_state	13
local_orig	0
local_resp	0
missed_bytes	7
history	177
orig_pkts	110
orig_ip_bytes	1497
resp_pkts	88
resp_ip_bytes	1292
tunnel_parents	1
label	3
detailed-label	3

Figure 2: The number of unique values for each feature

Features that are not important in training the model or have constant values:

- **ts**: Timestamp of the event or connection start time.
- **uid**: Unique identifier for the connection/session.
- **id.orig_h**: IP address of the originator (source) of the connection.
- **id.orig_p**: Port number of the originator (source) of the connection.
- **id.resp_h**: IP address of the responder (destination) in the connection.
- **id.resp_p**: Port number of the responder (destination) in the connection.
- **local_orig**: Indicates whether the originator is part of the local network (T for true, F for false). But all value of this feature is null
- **local_resp**: Indicates whether the responder is part of the local network (T for true, F for false). But all value of this feature is null
- **tunnel_parents**: Identifies parent tunnel(s) (if any) associated with this connection, such as VPN or encapsulated traffic.

2.2 Handling missing data

In the IoT-23 dataset, `#close` typically indicates the end of a connection or session. It is a marker used in logs (e.g., generated by Zeek) to signify that no further data will be logged for the corresponding network flow or session. This helps signal the closure or completion of the recorded interaction. So, we will remove flow have `timestamp = #close..`

Information fields with null and Nan values such as `duration`, `orig_bytes`, `resp_bytes` will be filled with the average value of all values.

2.3 Data transformation

With features of type objects such as `proto`, `service`, `conn_state`, `history`, we use `LabelEncoder` to encode them into labels that can be used in model training. For labels, `benign` is encoded as 0 and `malicious` is encoded as 1. As for using multi-class classification, detailed `malicious` labels will be encoded from the `detailed_label` feature.

2.4 Data normalization

Data normalization is a necessary step to ensure that all input features have similar scales and units of measurement. This helps the machine learning model learn more efficiently and improves the accuracy of predictions.

In this IOT23 dataset, I choose the `MaxAbsScaler` data normalization method, dividing all the values of each feature by the absolute value of the maximum value of that feature according to the formula 1, thus obtaining a dataset with values in the range $[-1, 1]$

$$v_i(new) = \frac{v_i}{|v_{max}|} \quad (1)$$

There are two factors that influence my decision. First, applying Max Absolute Scaling ensures that the original scale of the network traffic features is preserved. This is beneficial when the magnitude of the features is important and there is an uneven distribution among the features. Furthermore, Max Absolute Scaling preserves the sparseness of the network traffic data without converting it to a dense format.

2.5 Feature selection

We use Mutual Information to select feature. Mutual information between two random variables X and Y is defined above entropy. This method calculates the dependence index between each characteristic variable and the label, called MI score, has a joint probability distribution $P(f, l)$ between each feature F and L labels will have an MI score between them, with the formula 2:

$$M(F, L) = \sum_{m,l} P(f, l) \log\left(\frac{P(f, l)}{P(f)P(l)}\right) \quad (2)$$

The higher the MI score index, the higher the correlation with the label, if the value is closer to 0 the greater the independence of the two variables. From there, features with indexes can be eliminated MI score is lower than the self-set threshold so that model performance improves.

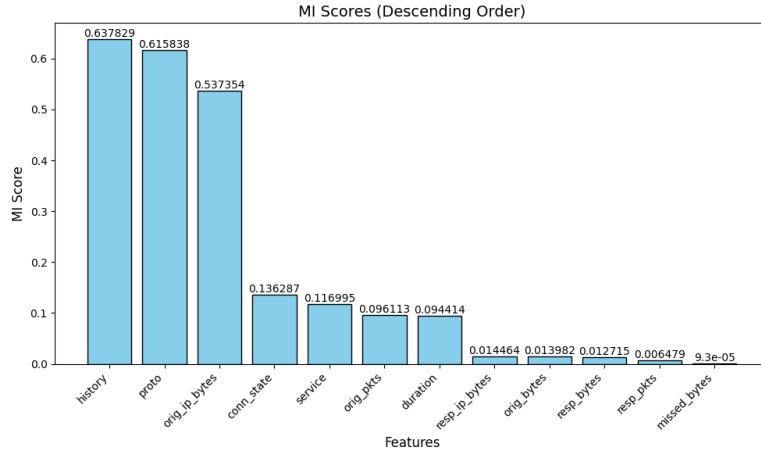


Figure 3: Mi Score of Mutual Information

We removed the first 2 features with MI score around 0 (< 0.001): resp_pkts, missed_bytes. Now we have 10 features left that will be used to train the model.

3 Data visualization

The dataset contains two primary labels: Malicious and Benign. As shown in the bar plot (figure 4), the dataset is relatively balanced between these two categories, with slightly more malicious samples than benign ones.

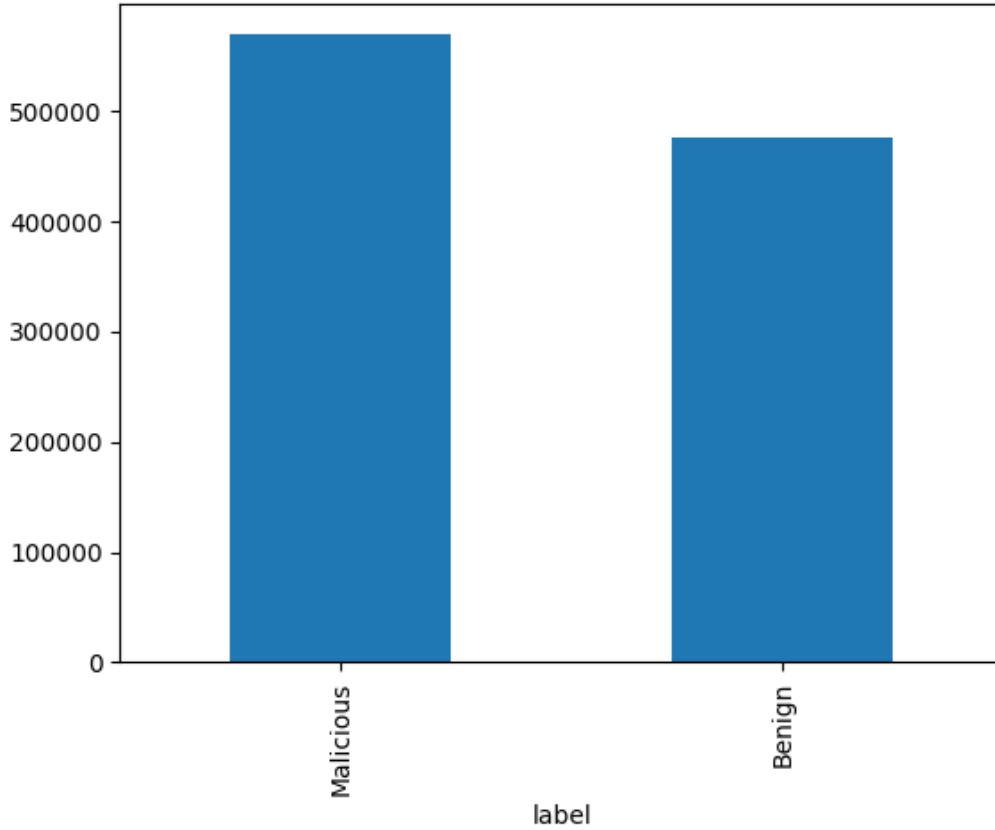


Figure 4: Label distribution of dataset

For malicious traffic, the dataset provides additional granularity through the **detailed-label**

feature. The distribution of detailed malicious activities is as follows and figure 5:

- **PartOfAHorizontalPortScan**: the majority of malicious activity (539,585 samples) is associated with horizontal port scanning, where attackers scan multiple ports across various IPs.
- **C&C (Command and Control)**: a smaller portion (14,931 samples) corresponds to command-and-control traffic, typically used by attackers to communicate with compromised devices.
- **DDoS (Distributed Denial of Service)**: another significant subset (14,393 samples) involves DDoS traffic, characterized by high-volume attacks aimed at overwhelming a target system.

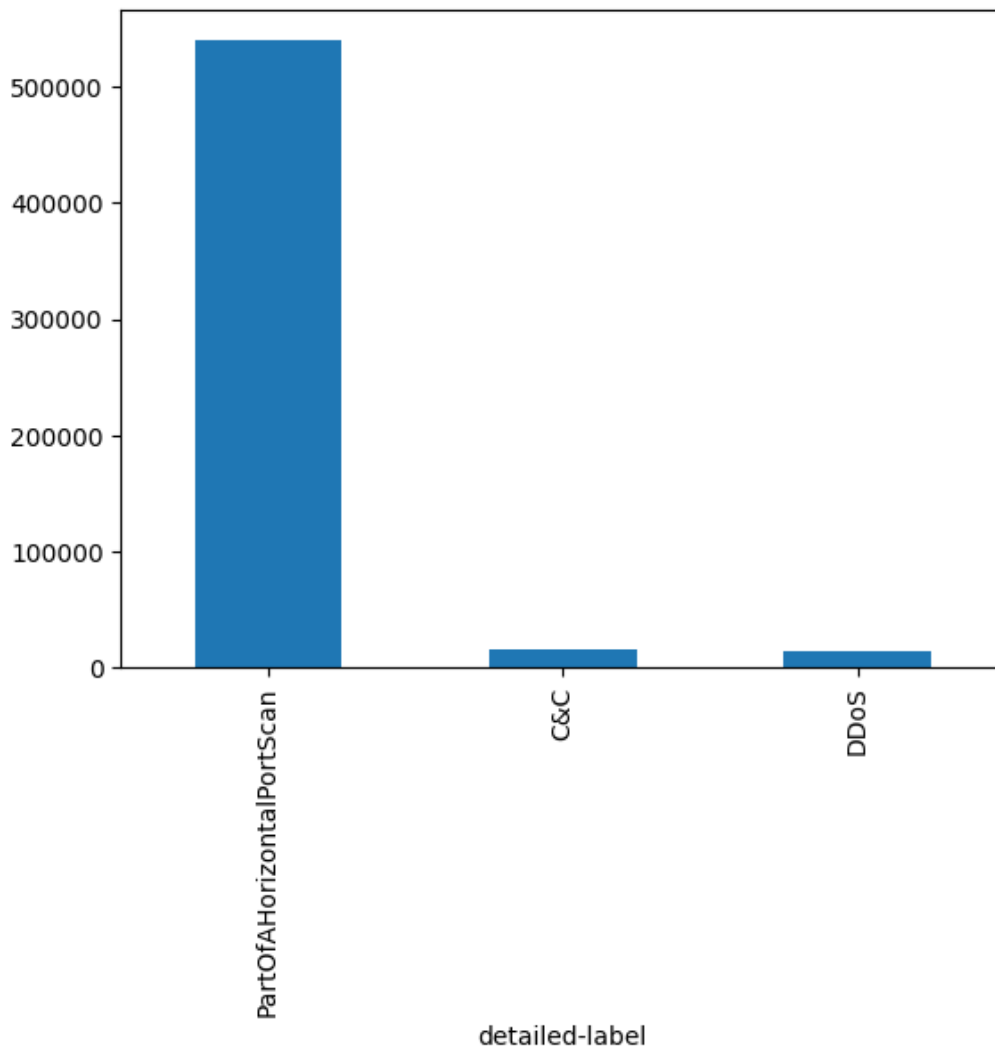


Figure 5: Detailed label distribution

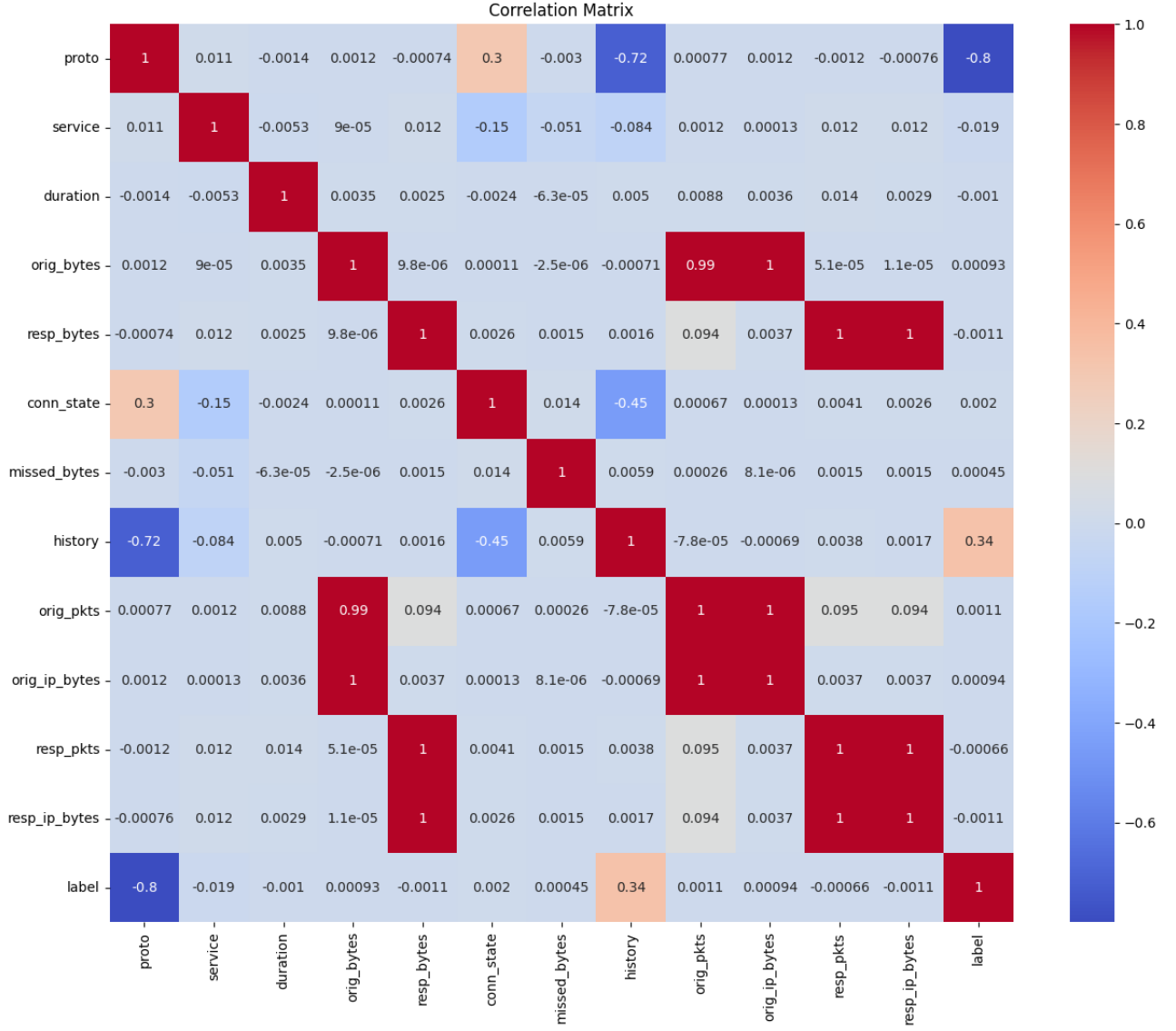


Figure 6: Correlation matrix

The correlation matrix (figure 6) visualizes the relationships between numerical features in the dataset. A strong positive correlation is observed between *orig_bytes* and *orig_ip_bytes*, as well as between *resp_bytes* and *resp_ip_bytes*. This indicates that the volume of bytes transferred at the IP layer corresponds closely to the volume transferred at the application layer. Conversely, notable negative correlations include the relationship between *proto* and *history*, with a value of -0.72 , suggesting that specific protocols are associated with distinct patterns of connection history. Similarly, the *proto* feature exhibits a strong negative correlation with the *label* feature at -0.8 , implying that certain protocols are more prevalent in either malicious or benign traffic.

Most other features, such as *duration*, *missed_bytes*, and *conn_state*, display weak or negligible correlations with each other. This suggests that these features capture independent aspects of the network traffic, making them potentially valuable for machine learning models.

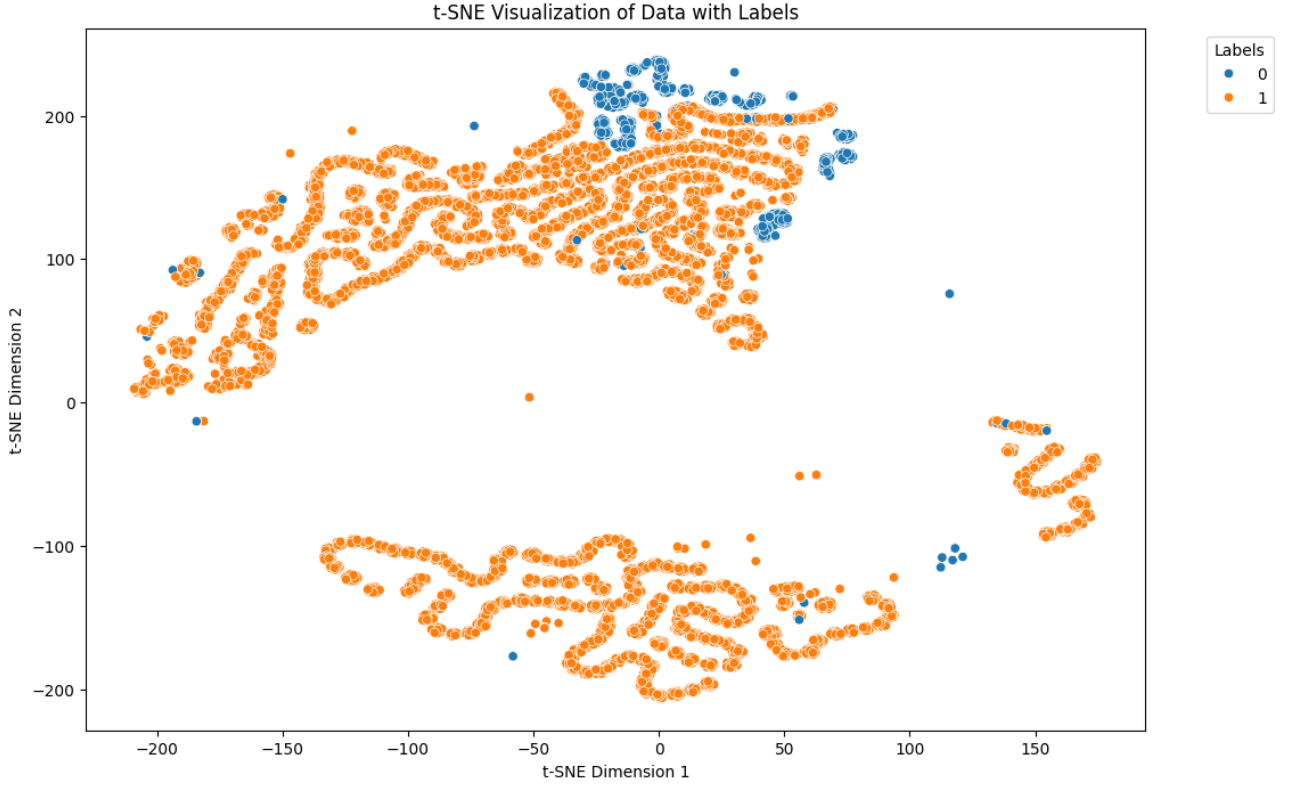


Figure 7: t-SNE visualization of data with labels

The t-SNE visualization (figure 7) demonstrates the separability of the dataset in a two-dimensional space, with each point representing a sample and colors indicating the labels: blue for benign (label 0) and orange for malicious (label 1) traffic. The visualization reveals distinct clustering patterns, where regions dominated by either malicious or benign traffic can be observed. This structure suggests that the dataset is well-suited not only for classification tasks, as machine learning models can exploit the separability, but also for clustering algorithms, which can identify natural groupings in the data. However, some areas of overlap between clusters indicate the presence of ambiguous or borderline samples, which may challenge both classification and clustering methods.

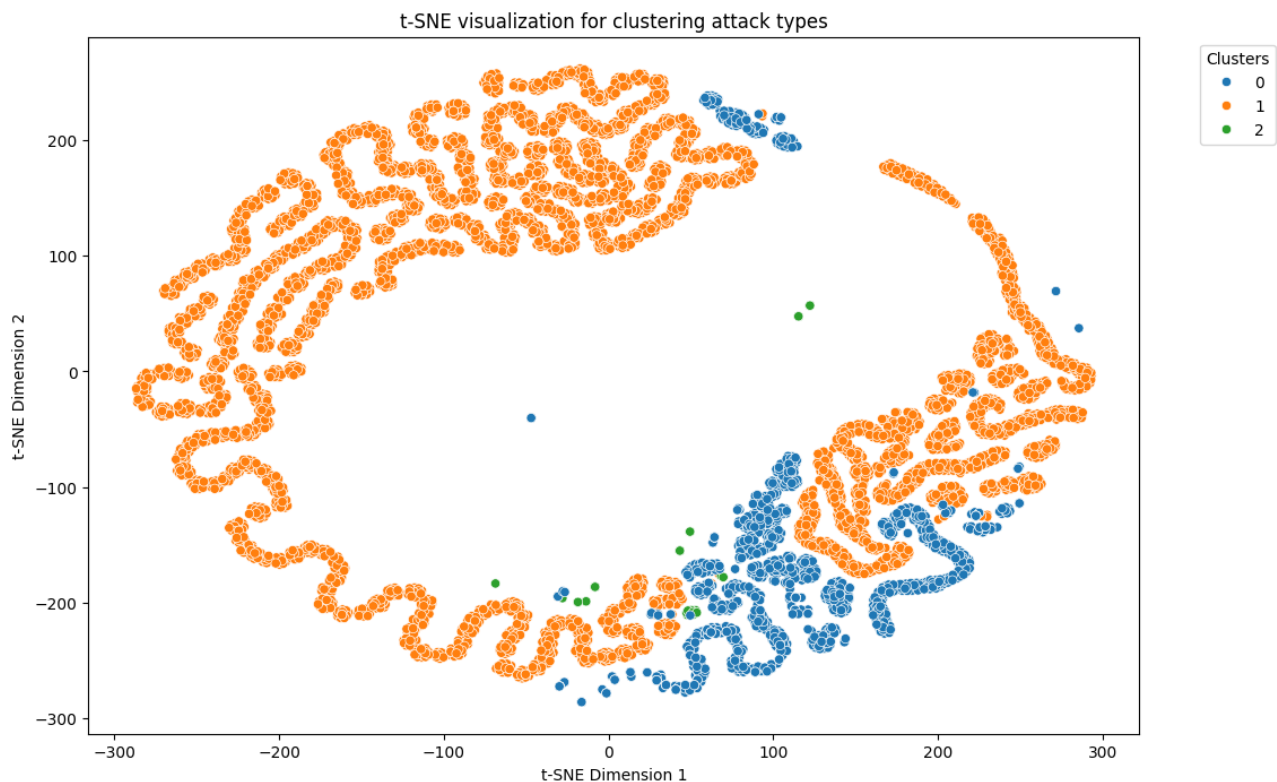


Figure 8: t-SNE visualization for attack types clustering

Similarly, figure 8 represents a t-SNE visualization that focuses on clustering different attack types in a malicious dataset. The attack types have been encoded into numeric labels for better interpretation, where 'PartOfAHorizontalPortScan' is replaced with 0, 'C&C' with 1, and 'DDoS' with 2. The visualization reveals distinct groupings of the attack categories, which are plotted in different colors to highlight their separation.

4 Implementation result

Link to the source code: <https://colab.research.google.com/drive/1KW2GUVYvvtUoQTqJov5bVmbEpHQ49N?usp=sharing>

Scikit-learn version 1.6 modified the API around its "tags", and that's the cause of this error. XGBoost has made the necessary changes in PR11021, but at present that hasn't made it into a released version. You can either keep your sklearn version <1.6, or build XGBoost directly from github (or upgrade XGBoost, after a new version is released).

4.1 Supervised learning methods

In this project, We split the dataset into 2 parts with 70% for training and 30% for testing. We apply binary clasification with 2 label benign(0) and malicious(1).

We use machine learning algorithms:

- XGBoost
- Naive bayes
- Random Forest

4.1.1 XGBoost

XGBoost is a machine learning algorithm based on the boosting method, using decision trees to improve the model. It works by building decision trees sequentially, where each new tree learns from the errors of the previous one. The model is updated by adding a new tree to minimize the loss.

The loss function used in XGBoost is:

$$\text{Loss Function} = \sum_{i=1}^N L(y_i, \hat{y}_i) + \Omega(f)$$

where:

- $L(y_i, \hat{y}_i)$ is the loss function that measures the difference between the true value y_i and the predicted value \hat{y}_i .
- $\Omega(f)$ is a regularization term that controls the complexity of the model.

The model is updated after each round by adding a new decision tree:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta \cdot h_t(x_i)$$

where:

- $\hat{y}_i^{(t)}$ is the prediction after round t .
- η is the learning rate.
- $h_t(x_i)$ is the prediction of the t -th decision tree.

We use GridSearchCV to find the best parameter among the parameters we set.

XGBoost	Result
Accuracy	95.84%
Recall	92.90%
Precision	99.99%
F1-Score	96.31%

Table 2: XGBoost model evaluation results

The accuracy index of 95.84% and Precision index of 92.9% shows that the model's prediction rate still has a number of errors, but the Recall index is very high at 99.99%, meaning that the model is detecting malicious samples well but there are also false predictions that are malicious but in reality are benign. This is not too bad in the actual detection of malware, but with a large number of connections, the number of false predictions and the experts have to check will be quite large.

4.1.2 Random Forest

Random Forest is an ensemble learning algorithm that uses multiple decision trees to improve prediction accuracy. It works by training multiple decision trees on random subsets of the data and then combining their results to make the final prediction. Each tree is trained using a random sample of the data, and the final prediction is determined by averaging (for regression) or majority voting (for classification) the predictions of all trees.

The model works as follows:

- A random subset of data points and features is selected to build each decision tree.
- Each tree is built using a standard decision tree algorithm (e.g., CART).
- The final prediction is made by averaging the outputs of all trees in the case of regression or using a majority vote for classification.

The final prediction for an individual sample is:

$$\hat{y}_i = \frac{1}{T} \sum_{t=1}^T h_t(x_i)$$

where:

- \hat{y}_i is the final prediction for the i -th sample.
- T is the number of trees in the forest.
- $h_t(x_i)$ is the prediction of the t -th decision tree.

The results of the Random Forest algorithm are quite similar to the results of the XGBoost algorithm. These two algorithms have similarities in that they detect Malicious well but still have errors (mistakenly calling it malicious while in fact it is benign).

Random Forest	Result
Accuracy	95.80%
Recall	92.91%
Precision	99.90%
F1-Score	96.28%

Table 3: Random Forest model evaluation results

4.1.3 Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem, assuming independence between features. It is used to classify data by calculating the posterior probability of each class and assigning the class with the highest probability. The "naive" assumption is that all features are conditionally independent given the class.

The posterior probability of a class C_k given the input features $x = (x_1, x_2, \dots, x_n)$ is calculated using Bayes' theorem:

$$P(C_k|x) = \frac{P(C_k) \prod_{i=1}^n P(x_i|C_k)}{P(x)}$$

where:

- $P(C_k|x)$ is the posterior probability of class C_k given the input features x .
- $P(C_k)$ is the prior probability of class C_k .
- $P(x_i|C_k)$ is the likelihood of feature x_i given class C_k .
- $P(x)$ is the evidence or marginal probability of the features, which can be ignored for classification as it is constant across classes.

To classify a new instance, we compute the posterior probability for each class and choose the class with the highest probability:

$$\hat{C} = \arg \max_{C_k} P(C_k) \prod_{i=1}^n P(x_i|C_k)$$

With the Naive Bayes algorithm, the model evaluation results are lower than the two algorithms XGBoost and Random Forest but the detection of malicious samples is still quite good. The results are shown in the table 5.

Random Forest	Result
Accuracy	94.74%
Recall	92.71%
Precision	98.03%
F1-Score	95.30%

Table 4: Random Forest model evaluation results

4.1.4 Support Vector Machine

Support Vector Machine (SVM) is a supervised learning algorithm used for classification. It finds the hyperplane that best separates the data into different classes by maximizing the margin between the support vectors.

The decision function for SVM is:

$$f(x) = w^T x + b$$

where w is the weight vector, and b is the bias term. SVM aims to maximize the margin by solving the following optimization problem:

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1, \quad \forall i$$

For non-linearly separable data, SVM uses a kernel function $K(x_i, x)$ to map data into a higher-dimensional space.

The final prediction is made by:

$$\hat{y} = \text{sign}(f(x))$$

SVM	Result
Accuracy	95.78%
Recall	92.81%
Precision	99.97%
F1-Score	96.26%

Table 5: Support Vector Machine evaluation results

Thus, although the results of the above 4 machine learning algorithms are quite good, in reality, with a large number of connections in a network, analyzing false alarms is costly. Moreover, the prediction results are only independent of each connection, which also loses the connectivity of the connections in a network. Therefore, in the future, we can apply graph neural network algorithms in a network with each ip as a node and the connections are the characteristics of edge attributes. This will significantly reduce the analysis of false predictions while taking advantage of the relationship between nodes in a network.

4.2 Unsupervised learning methods

In this project, we apply two distinct clustering strategies to evaluate the structure and patterns within the dataset:

- **Binary clustering:** involves clustering the dataset into two categories: *benign* and *malicious*.
- **Clustering by attack types:** focuses on clustering malicious activity further based on the *detailed label* present in the dataset. This allows us to evaluate how well clustering algorithms can differentiate between specific types of attacks, including *PartOfAHorizontalPortScan*, *C&C*, and *DDoS*.

Since the original dataset contains over 1,044,258 samples, we extract a representative subset of 100,000 rows to ensure computational efficiency, preserve key patterns, and enable effective visualization without compromising the integrity of the analysis.

The dataset used for clustering consisted of numerical features after dropping the ‘label’ and ‘detailed-label’ columns.

To evaluate the clustering performance, the following metrics are used:

- **Adjusted Rand Index (ARI):**

$$\text{ARI} = \frac{\text{Index} - \text{Expected Index}}{\text{Max Index} - \text{Expected Index}}$$

where Index represents the number of agreements between clustering and ground truth, Expected Index is the expected value of Index under a random assignment, and Max Index is the maximum possible value of Index. ARI ranges from -1 to 1.

- **Normalized Mutual Information (NMI):**

$$\text{NMI} = \frac{2 \cdot I(X; Y)}{H(X) + H(Y)}$$

where $I(X; Y)$ is the mutual information between the predicted clusters X and the true labels Y , and $H(X)$ and $H(Y)$ are the entropies of X and Y , respectively. NMI ranges from 0 to 1.

- **Accuracy:**

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Samples}}$$

It measures the proportion of samples that are correctly clustered when true labels are available.

- **Davies-Bouldin Index (DBI):**

$$\text{DBI} = \frac{1}{N} \sum_{i=1}^N \max_{j \neq i} \left(\frac{S_i + S_j}{d_{i,j}} \right)$$

where S_i is the average distance of points in cluster i to the cluster center, $d_{i,j}$ is the distance between cluster centers i and j , and N is the number of clusters. Lower values of DBI indicate better clustering.

- **Silhouette Score:**

$$S = \frac{b - a}{\max(a, b)}$$

where a is the average distance of the sample to other points in the same cluster (cohesion), and b is the average distance to points in the nearest cluster (separation). The overall Silhouette Score is the mean of S across all samples. The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters.

For each strategy, we apply two clustering methods: K-means and Hierarchical Clustering with single linkage. For binary clustering, the number of clusters is set to 2 (benign and malicious). For attack-type clustering, the number of clusters is equal to the number of unique attack labels in the dataset, which is 3 ('PartOfAHorizontalPortScan', 'C&C', and 'DDoS').

The clustering results for binary clustering and attack-type clustering are summarized in Table 6.

Table 6: Clustering evaluation results for binary and attack-type clustering

Strategy	Method	ARI	NMI	ACC	DB	Silhouette
Binary clustering	K-means	0.0009	0.0000	0.6590	0.3139	0.8864
	Hierarchical	-0.0004	0.0000	0.6576	0.3343	0.8800
Attack-type clustering	K-means	0.5289	0.6283	0.7889	0.1149	0.9887
	Hierarchical	0.5258	0.6243	0.7870	0.1225	0.9898

The binary clustering task aim to separate benign and malicious activities, but both algorithms struggle to achieve high alignment with ground truth labels, as reflected by the low ARI and NMI scores. This suggests that the features used in the dataset may not fully capture the nuanced distinctions between benign and malicious behavior in a binary context. However, the high Silhouette Scores indicate that the clusters themselves were well-formed and distinct in the feature space, which may point to the presence of underlying structures that do not align perfectly with the binary labels.

In contrast to binary clustering, the task of distinguishing between specific attack types achieve much better results. Both K-means and Hierarchical Clustering achieved significantly higher ARI, NMI, and Accuracy scores. This demonstrates that the clustering algorithms were better suited for capturing the differences between specific malicious behaviors rather than the more general benign versus malicious distinction. The improved performance could also indicate that the attack types are inherently more separable in the feature space, reflecting clear patterns or characteristics specific to each type of attack.

Across all metrics and clustering strategies, K-means showed slightly better performance than Hierarchical Clustering. K-means' ability to optimize intra-cluster variance likely contributed to its superior results, particularly in datasets with well-separated clusters. Hierarchical Clustering, on the other hand, may have been influenced by the choice of linkage method, which can sometimes lead to chaining effects and less compact clusters.