Network Attacks Classification : A Machine Learning Approach

ximena.moure@estudiantant.upc.edu
ange.xu@estudiantant.upc.edu
fatima.zohra.chriki@estudiantat.upc.edu

Advanced Machine Learning, Master in Data Science

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

November 19, 2023

Contents

1 Introduction

This document serves as the comprehensive project report for the Advanced Machine Learning course at Facultat d'Informàtica de Barcelona (UPC). The core objective of this project is to address a real-world problem through the lens of advanced machine-learning techniques. As we navigate through this project, our goal is to synthesize the diverse machine-learning techniques into a cohesive strategy, aiming to offer a solution that is not only academically sound but also practically viable. The culmination of this project is a final deliverable designed to meet the needs of the end-user, showcasing our ability to apply our academic learning to real-world challenges.

1.1 Problem description

Thanks to the advent of low-cost computer chips and high-bandwidth telecommunications, we now have billions of devices connected to the Internet. This means that everyday devices, such as toothbrushes, vacuum cleaners, cars, and machines, can use sensors to collect data and respond intelligently to users. This comes with a high risk, especially in the area of privacy and security. Since cyber threats are continuously evolving, by identifying and understanding different types of cyberattacks, we can develop and implement countermeasures to protect IOT devices. Additionally, the model can be adapted to identify suspicious patterns or activities in the network packets received that could be a potential cyberattack. The goal of this project is to develop a machine learning model with the capability to accurately detect and classify the type of cyberattack received based on the input data.

1.2 Dataset description

The chosen dataset is from the University of New Brunswick, Centre for Cybersecurity and it can be found published at Kaggle[1]. The data was collected by executing 33 attacks in an IoT topology composed of 105 devices. These attacks are classified into seven categories, namely DDoS, DoS, Recon, Web-based, Brute Force, Spoong, and Mirai. Finally, all attacks are executed by malicious IoT devices targeting other IoT devices. Therefore, the dataset also includes benign samples of network data which along with the malign attack samples, can be used to perform network packet classification. Each observation in the dataset corresponds to a set of 47 features that specify the properties of the network packet. The size of the dataset is above 13 GB.
An explanation of each of the seven malign attacks can be found below:

- **DoS (denial-of-service):** The attacker overwhelms a server with a huge amount of internet traffic, aiming to stop people from accessing websites and online services connected to it.
- **DDoS (Distributed Denial-of-Service):** Similar to a Denial of Service (DoS) attack, it's a type of attack where multiple computers or machines are used to flood a specific resource, causing a disruption in its normal functioning.
- **Recon (Reconnaissance):** a process of gathering information about a system to identify vulnerabilities.
- **Web-based:** Criminals take advantage of coding weaknesses to access servers or databases illegally.

- **Brute Force:** The attacker employs trial and error to break passwords, login credentials, and encryption keys. This straightforward method remains dependable for illicitly accessing both individual accounts and the systems and networks of organizations.
- **Spoofing:** When the attacker assumes a false identity to earn our trust, aiming to access our systems, pilfer data, extract money, or distribute malware.
- **Mirai:** This malware variant focuses on consumer devices such as smart cameras and home routers, transforming them into a network of remote-controlled bots known as a zombie network.

2 Related work

Prior research has been done in the dataset. They basically adopt three distinct analytical perspectives: firstly, a multi-class classification approach, aimed at identifying 33 unique attack types; secondly, a grouped classification strategy, which consolidates these into 7 broader attack categories; and thirdly, a binary classification framework that discerns between malicious and benign activities [2].

Among these methodologies, binary classification has demonstrated the highest levels of accuracy and efficiency. In contrast, the multi-class classification of individual attacks, encompassing 34 distinct classes, posed the most significant challenge, underscoring the complexity inherent in distinguishing among a wide range of attack types. The array of techniques deployed in these studies encompasses a diverse set of algorithms, including Logistic Regression, Perceptron, AdaBoost, Random Forest, and Deep Neural Networks.

3 Data Exploration

This section delves into the comprehensive examination of our dataset, which is instrumental in uncovering insights and guiding the subsequent application of machine learning techniques.

3.1 Data quality assessment

Before diving into data preprocessing, performing a data quality assessment is crucial to familiarize ourselves with the chosen dataset and to uncover meaningful insights from it. For this step, we mainly checked the following aspects of the dataset:

- **Missing Data:** No missing values found in the dataset.
- **Duplicates:** No duplicated samples were found.
- **Outliers existence:** as all the variables are continuous, we performed univariate outlier detection using the statistical test Z-score, where data points with a Z-score beyond threshold 3 are considered outliers. The test results show that almost all the variables have data points that are significantly far away from the mean, treatment of these outliers will be conducted in the next section.
- **Data consistency:** An initial analysis was performed to check the variable types, the variable's range correctness, and their distributions.See fig. 7. After this process, the variable "Protocol type" was detected to have incorrect data type format, according to the

standards published by the organization Internet Assigned Numbers Authority(IANA), the protocol identifier is an integer value instead of decimal as we found in the dataset.

Apart from the aforementioned data quality problems detected during this initial analysis, no other systematic errors that could be produced during the data collection were found.

3.2 Data preprocessing

In this section, we conducted the correction of the data quality issues identified in the previous section.

3.2.1 Outliers treatment

Following the pieces of advice described in [3] related to the right steps to follow when working with the data exploration, prior to the removal of the outliers from the dataset, we performed a deep analysis on the outliers in order to see whether they represent true errors(measurement errors, data entry, processing errors) or whether they just represent natural variations in the dataset and contain important information that should not be removed.

- **Univariate Outliers**: after a deep analysis we observed that those outliers are not the result of human errors and they just represent rare or extreme values in the population. For example, more than 50% of the outliers of the variable "Header Length" belong to the attack "Spoofing", which makes sense because this kind of attack consists of falsifying DNS responses, redirecting users to malicious destinations by modifying the response's content such as IP addresses that are in the packet header. Removing these outliers may cause a loss of information that later on could be useful for model training.

- **Multivariate Outliers**: as for the detection of multivariate outliers, the method Local Outlier Factor(LOF)[4] was chosen due to its ability to adapt to varying data densities, which is the case of our dataset. LOF can also identify global outliers (far from all data) and local outliers (far from local neighborhood). It is a density-based technique that uses the nearest-neighbor search to identify anomalous points. As a result, $\simeq 1000$ observations were detected as multivariate outliers, we found out that some of them were detected as univariate in the previous section, which makes sense, because multivariate outliers are extreme values in multiple variables simultaneously, and due to the correlation between variables in a dataset, an extreme value in one variable might lead to having also extreme value in its correlated variable.

  After removing the detected outliers from the dataset we performed the Kolmogorov-Smirnov Test[5] to assess whether the distribution of the variables has changed or remained the same as when outliers were present. As a result, the removal of outliers altered almost 20% of the variables's distribution, taking this into account, we can say that their impact on the overall characteristics of the dataset is not significant.

3.2.2 Data type correction

In this step, we correct the variable "Protocol Type" values to convert them from decimal to integer by rounding. Sequentially, we checked if the converted values exist as protocol types according to the IANA's standards.

3.3 Feature preprocessing

In this section, we conducted the Feature processing steps such as feature extraction and selection. As for feature transformation, we only standardized the variables. The application

of mathematical transformations such as logarithmic, squared, or nth-root operations was not considered to be applied in this step, because we considered that this application depends on the algorithms we are working on, thus specific transformations will be performed when it's necessary for some algorithms.

3.3.1 New Features Derivation

In this section, we created additional features from existing ones in our dataset with the goal of increasing the amount of information fed to the models that might improve their predictive power. Details can be found in the table 2.

3.3.2 Features Selection

Our dataset includes 47 features, including the label. In order to reduce noise and irrelevant features, we followed three strategies to select the most important features for the classification of the attack categories in the data.

1. **Removing variables with unique values:** we found six features with unique values: *ece_flag_number*, *ece_flag_number*, *cwr_flag_number*, *Telnet, SMTP, IRC* and *DHCP*. Variables that contain a unique value for every record (also known as constant features) typically do not contribute to the predictive power of a model. In fact, they can increase computational complexity without adding any informative value. Thus, they were dropped.

2. **Removing correlated variables:** based on the correlation matrix results Figure 5, we found several highly correlated features. We selected all pair of features which a correlation coefficient higher than 0.9 and dropped one variable of each pair. In total, 13 features were removed, namely *Srate, LLC, Std, Number, Magnitue, Radius, Covariance, Weight, ratio_ack, ratio_fin, ratio_urg, ratio_rst,* and *ratioTotalSizeToTotalNumber*

3. **Feature importance:** to eliminate irrelevant features, we employed Random Forest to assess the importance of each variable, as shown in Figure 6. Following the analysis of feature importance, we removed non-relevant features, retaining only 15 significant ones. Subsequently, we reassessed the correlation, and the final features exhibited low correlation among themselves.

3.3.3 Feature transformation

We standardized all the features of our dataset around a zero mean with unit variance. This technique is especially important for algorithms that are sensitive to the scale of the input data, such as SVM and KNN.
For a given feature, we subtract the mean of the feature and then divide it by the standard deviation. The transformed value for the feature 'x' is calculated as $z = \frac{x-\mu}{\sigma}$, where mu is the mean of the feature values, and $sigma$ is the standard deviation.

3.4 Exploratory data analysis

Visualizing the data plays a critical role in comprehending its structure and the interrelationships among various variables. Through graphical representation, complex data sets become more accessible and interpretable, allowing for a clearer grasp of underlying patterns, trends, and anomalies.

We started by visualizing histograms of every feature in the dataset. See Figure **??**. From these plots, we can observe the following:

1. The histograms for *ratioPacketSizeToTotal* and *ratioTotalSizeToDuration* suggest that most of the ratios are low, with a few higher values.

2. Several histograms, such as those for *flow_ duration*, *Header_ Length*, *Rate*, *urg_ count*, *rst_ count*, *Tot sum*, *Min*, *Max*, *AVG*, and *Tot size*, exhibit a right-skewed distribution. This suggests that the majority of observations have low values, with a long tail extending to higher values.

3.4.1 Packet size statistics

Plotting *Min*,*Max* and *AVG* together could reveal interesting relationships between these variables and how packet sizes vary within flows. From Figure 2a we can see the following:

1. Min vs. Max: The scatter plot between *Min* and *Max* shows a trend where as the "Min" value increases, the "Max" value also tends to increase. The distribution of points suggests a positive correlation between these two variables.

2. Min vs. AVG: The scatter plot also indicates a positive correlation. As the "Min" values increase, the "AVG" values tend to increase as well. However, the spread of points is more dispersed compared to the "Min" vs. "Max" plot, suggesting that the relationship might be less strong or more variable.

3. Most data points are concentrated in the lower range for all three variables, which suggests that for most observations, the values of "Min", "Max", and "AVG" are low. There are fewer observations with high values.

3.4.2 Rate by Family of Attack

Figure 2b depicts a bar plot of the feature *Rate* for each type of attack.

There is a clear variation in the 'Rate' among different 'family' categories. This suggests that 'Rate' could be a significant feature to distinguish between these categories. The 'DDoS' category has the highest mean 'Rate' and also shows a large error bar (standard deviation), indicating substantial variability within this category. The large error bar could be due to a wide range of 'Rate' values or outliers in the 'DDoS' data. The 'Benign' category has the lowest mean 'Rate', which is expected as normal traffic typically does not have as high a rate as malicious traffic like DDoS attacks. The close clustering of the bars for most categories except 'DDoS' suggests that 'Rate' has a relatively homogenous distribution across these categories, with 'DDoS' being an exception.

3.4.3 Header length by Family of attack

Different types of attacks may exhibit distinct patterns or signatures in their header lengths. For example, certain types of attacks might generate packets with consistently longer or shorter header lengths. Identifying such patterns can be crucial for understanding attack behaviors and developing detection strategies.
From Figure 3a we can observe that the distribution of *Header_ Length* across categories is not uniform, indicating that this feature could be useful in differentiating between different types of traffic or threat categories

3.4.4 Total Size by Family

Different types of attacks may have characteristic data packet sizes. Outliers or unusual patterns in Total Size can signal anomalous behavior. From Figure 3b we can see the following:

1. The "Mirai", "DDoS", and "Web" categories have wider distributions, indicating a larger variability in "Total Size" compared to other categories. This could suggest a higher diversity in the total sizes associated with these types of network traffic or attacks.

2. It appears that the "Benign" category has a lower median "Total Size" compared to "Mirai", "Recon", and "Web". In contrast, "BruteForce", "DDoS", "DoS", and "Spoofing" have very narrow violins with less variance and lower medians, indicating that "Total Size" tends to be smaller and more consistent within these categories.

3. The narrowness of the violins for "BruteForce", "DDoS", "DoS", and "Spoofing" suggests that there are fewer variations in "Total Size" within these categories, and most of the data points are concentrated around similar values.

3.5 Clustering

After preprocessing our data, we performed clustering in order to discover visually hidden structures within data and to detect if there are outliers or rare data points in our data. Thus K-means was chosen to carry out the process. As we are working with multivariate data, it is not feasible to plot the groups without performing a dimensionality reduction. Therefore, Kernel PCA was applied to capture nonlinear relationships within data and to help K-means in finding a better cluster separation. From Figure 4 we can observe a clear decision boundary between clusters, but most of them are closer to each other. This closeness can imply that the data points in the boundaries share similarities or have overlapping characteristics of two clusters.

4 Experimentation methodology

4.1 Methods considered

In the following subsections a short introduction of the model training methods chosen for the project can be found.

4.2 SVM

Support Vector Machines(SVM) represent a robust supervised learning methodology utilized for both classification and regression tasks. These algorithms operate by computing the optimal separating hyperplane that not only delineates different classes within the dataset but also maximizes the distance, or margin, from the nearest data points of any class. These data points, which anchor the margin, are known as support vectors.
When applying SVM, our goal is to find a Separating Hyperplane(SH) that helps us to: a) Maximizing the margin;b)Accurately classifying the training points.
The equation of that hyperplane is $g(x) : w^i x + b$, where the sign of $g(x)$ indicates if the observation's label is +1 or -1(in a binary classification case). The goal of an SVM classifier is to maximize the margin of the SH with respect to the classes, and this margin defines its confidence, thus wider margin is preferable. There are two types of margins:

- **Functional margin**: The decision boundary of the classifier that defines the equation of the margin. $y_i(w^i x + b) = y_i g(x_i) = \gamma$. γ is the margin value and it is positive when $y_i = 1$, otherwise, negative. From the equation, we can see that the functional margin depends on the values of w and b, which means that it scales up when w and b values increase and vice versa.

- **Geometric margin**: The set of distances from the SH to each observation.It is a scaled version of the functional margin $d(x, \gamma = \frac{y_i g(x)}{||w||})$. The normalizer $||w||$ avoids the margin getting wider when w and b increase.

Knowing the difference between the margin types, we know that the margin of the hyperplane can be defined as the minimum distance of an observation from the hyperplane, and all the observations that achieve minimum distance receive the name of Support Vectors: $y_i g(x) \pm 1$. Therefore, the minimum distance between two classes can be defined as $\frac{2}{||w||}$.
To formulate the task of finding the best Optimal Separating Hyperplane(OSH) as an optimization problem, it is essential to take into account the geometric margin that represents the confidence of the classification, thus the optimization problem will consist of maximizing the geometrical margin:

$$\underset{\mathbf{w,b}}{max}\{\frac{2}{||w||}\} \quad \text{s.t.} \quad y_i((w, x_i) + b) \geq 1, \quad \text{for i=1,...,N} \tag{1}$$

4.2.1 Non-kernelized method: Linear SVC

The Linear SVC was chosen as the non-kernelized method for model training. According to the following soft margins primal formulation:

$$\underset{\mathbf{w,b}}{miin} \frac{1}{2}||w||^2 + C\sum_{\mathbf{i=1}} \zeta_i \quad \text{s.t.} \quad y_i((w, x_i) + b) > 1 - \zeta_i, \quad \zeta_i > 0, \quad \text{for i=1,...,N} \tag{2}$$

where,

- $\zeta_i = (1 - y_i g(x_i))$, if an observation is misclassified then $y_i g(x_i) < 0$, thus $\zeta_i > 0$ and the term $\sum_{\mathbf{i=1}} \zeta_i$ acts as an upper bound on the number of training errors.

- C is the cost hyperparameter that acts as the regularization term to the primal formulation to penalize misclassified observations.

To achieve the best accuracy for Linear SVC, C is the only hyperparameter required to tune. Based on the above equation, the effect of the C on the model performance can be summarized as follow[6]:

- When the value of C is low, a lower penalty will be applied for the misclassifications, this behavior results in a wider margin hyperplane. Lower C values lead to Underfitting.

- When the value of C is high, a higher penalty will be applied. The model will be focused on enhancing the accuracy thus it will tend to create the hyperplane with a small margin to avoid misclassifications. Higher values of C lead to Overfitting.

- When C approaches $+\infty$, as the penalty for misclassifications would be huge, the model will be heavily overfitted for being too fitted to the training data.

- When C approaches 0, with a high probability the model underfits and might be unable to find the optimal separating hyperplane, as it will opt for a larger-margin hyperplane, allowing more misclassified data points.

4.2.2 Kernelized method

A distinctive attribute of SVMs lies in their use of kernel functions. These mathematical constructs enable SVMs to transcend the limitations of the data‘s original dimensionality. When the data is not linearly separable in its initial form, kernel functions project into a higher-dimensional feature space where a separating hyperplane can be more readily identified. Kernel functions serve as a gauge of similarity among data points, effectively capturing the nuances of their relationships within this transformed feature space.
By applying kernel functions, these algorithms can implicitly map input data into high-dimensional spaces without the computational overhead of explicit transformation. This allows for the identification of non-linear decision boundaries in the original input space, which might not be discernible with traditional linear models. The choice of the kernel function is pivotal. Common kernels include the linear, polynomial, radial basis function (RBF), and sigmoid. Each kernel comes with its own parameters, such as degree in polynomial kernels or gamma in RBF, which can be fine-tuned.

Multiclass SVMs

Originally conceived as a binary classification tool, SVMs have undergone significant evolution since their inception [7]. By the 1990s, adaptations of SVMs for multiclass classification challenges had been developed, most notably the One-versus-One (Ovo) and the One-versus-Rest(OvR) strategies. These methodologies expanded the applicability of SVMs, enabling them to discern among multiple classes by decomposing complex classification problems into a series of binary tasks.

One-versus-Rest strategy, applied in the context of SVMs, involves constructing 'c' binary classifiers, where 'c' represents the total number of distinct classes. In this approach, each classifier is tasked with distinguishing a single class from all other classes, essentially creating 'c' distinct hyperplanes in an $\mathbb{R}^d$ dimensional space. During the training phase, 'c' decision functions are derived, each corresponding to one of these classifiers [8]. When a new evaluation point (x', y') is introduced, it is evaluated against all 'c' decision functions, formulated as

$$y'_k = w_k^T x' + b_k \quad \text{for } k = 1 \text{ to } c$$

Given that a point x' could potentially fall into multiple regions demarcated by these hyperplanes, the class is ultimately determined by selecting the maximum value from these decision functions, i.e,

$$y'_k = \arg\max_j \left(w_k^T x' + b_k \right)$$

In contrast, the One-versus-One (OvO) method constructs a binary SVM classifier for each possible pair of classes. With 'c' classes, this amounts to training

$$\frac{c(c-1)}{2}$$

classifiers. For a validation data point x', the class assignment is determined through a process of majority voting, where the class that most frequently classifies x' as positive $(+1)$ is selected as the label.
This distinction between OvR and Ovo lies primarily in the number and type of binary classifiers constructed and the mechanism through which the validation points are classified. While OvR focuses on separating each class from the rest, OvO concentrates on discerning between every possible pair of classes, which can be particularly advantageous in certain datasets where

inter-class distinctions are subtle.

SVMs in Python

For the development of this project we chose Python as our primary programming language. Specifically, we utilized the Scikit-learn library. Within this library, we employed the Support Vector Classifier (SVC), which is underpinned by the libsvm library. This implementation adopts the One-versus-One (OvO) approach [9].
In order to find the best hyperparameters, we conducted a Grid Search. This process involved experimenting with a range of values for the following key hyperparamaters:

- **C**: The regularization parameter, balances the trade-off between minimizing the training error and ensuring a well-generalized model. The following values were tested: [0.001, 0.01, 0.1, 1, 10, 100].
- **kernel**: The kernel type to be used in the algorithm. The following values were tested: ['linear', 'poly', 'rbf', 'sigmoid'].
- **degree**: Degree of the polynomial kernel function. The following values were tested: [1, 2, 3].
- **gamma**: In the RBF kernel, it defines how far the influence of a single training example reaches. In the polynomial kernel, it is used as a scale factor in the kernel function. A small gamma value leads to a more generalized model where the decision boundary is smoother. A large gamma value, conversely, leads to a model that reacts more to the specific characteristics of the training data. This can be beneficial for capturing complex patterns in the data but can also lead to overfitting if the decision boundary becomes too tailored to the training data. The following values were tested: [0.001, 0.01, 0.1, 1, 10, 'scale', 'auto'].
- **max_iter**: it specifies the maximum number of iterations the algorithm is allowed to run before it converges to a solution or terminates. We started by using the default value in this parameter, but it took a very long time to run, so we decided to try with some values. By trial and error, we found that 20000 was well suited. With values less than 20000, we got a warning stating that the algorithm did not converge.
- **class_weight**: This parameter is a vital feature for handling imbalanced datasets. Since our dataset is balanced we did not tune this hyperparameter.

4.3 KNN

The K-Nearest Neighbors (KNN) algorithm is a machine learning technique that can be used in both classification and regression tasks. KNN is considered to be effective in scenarios where decision boundaries are complex and non-linear. It adapts well to the local structure of the data. During the training phase, KNN represents the data points in a feature space and memorizes them. The represented data points will be used as a reference for performing predictions. Each time we want to perform a prediction for a new sample S, KNN follows the following steps:

1. Represent the sample S in the feature space where the training data points are represented.
2. Calculates the distance between the new data point S and all data points in the training set. For distance measure, the Euclidean and Manhattan distances are most commonly used:

(a) Euclidean distance: measures a straight line between two data points.

(b) Manhattan distance: measures the sum of the absolute differences between the coordinates of two data points.

3. Identify the K nearest neighbors based on the calculated distances.

4. It assigns the majority class label among the K neighbors as the predicted label.

The key hyperparameters of the K-Nearest Neighbors (KNN) algorithm include the number of neighbors considered in each prediction (k) and the choice of distance metric. The selection of these parameters is crucial as they significantly influence the overall model performance.
KNN has the advantage of being used for both regression and classification tasks and its capacity to be trained with different types of datasets. However, it is computationally expensive when working with large datasets. Moreover, it is very sensitive to the presence of outliers and the value of K. Therefore, it is crucial to assess and treat the outliers in the data and choose an optimal k value.

4.4 Decision Tree

The Decision Tree is an ML model that emulates a tree-like structure composed of nodes, branches, and leaves. The initial node represents a decision based on the value of a specific feature, each branch represents an outcome of that decision, and each leaf node represents the final predicted class. During the training phase, the tree is created using the main features in the data. When we want to predict the class label of a new sample, the model makes decisions by traversing the tree from the root node to a leaf node. The final prediction is determined by the majority class of the samples within that leaf node. This approach enables the Decision Tree to efficiently categorize new instances based on the learned decision rules, making it a versatile and interpretable model in various machine-learning applications. The main hyperparameters tuned for model training are:

1. **Criterion:** The algorithm chooses the most discriminant feature in each node to split the data at each node, aiming to maximize the homogeneity of sample classes in the resulting subsets. In the context of classification task, the most commonly used impurity criteria are Gini index, Entropy, and Misclassification.

2. **Max depth:** maximum depth of the tree. If the max depth is "none", then de tree can grow as much as needed to achieve the maximum purity in nodes. However, this tends to overfitting, therefore, limiting the tree's depth allows pruning the tree and avoids overfitting.

3. **Min samples split:** minimum number of samples needed to split an internal node in the tree during training, this allows to avoid splitting nodes with few samples.

4. **Min samples leaf:** helps avoid overfitting by setting a maximum number of leafs in the tree.

Decision Trees possess the advantage of working with data in its original range. They automatically select the most important features and maintain robustness in the presence of outliers. Consequently, they require less complex preprocessing compared to other models.

4.5 Random Forest

Random Forest is an ensemble learning algorithm that uses the Bootstrap Aggregation (or bagging) technique to create multiple decision trees to make predictions. Once all the trees are built, predictions are made by each tree for new input and the final prediction is usually the average of the predictions from all the trees.

4.6 XGBoost

XGBoost, short for eXtreme Gradient Boosting, stands as an improved and scalable version of the gradient boosting algorithm, which means that it builds multiple weak models sequentially to create a strong model. Hence, subsequent models in the sequence correct the errors of weaker models to achieve an optimized solution. XGBoost has the flexibility to customize the objective function and evaluation metrics. The following hyperparameters were considered for the XGBoost model training:
a)Maximum Depth(max_depth): Specifies the maximum depth of each tree in the boosting process;
b)Learning Rate (eta): Controls the step size during the boosting process.
c)Number of Trees (n_estimators): Determines the number of boosting rounds or trees to build.
d)Colsample Bytree: Determines the fraction of features used for training each tree.
e)Gamma: Specifies the minimum loss reduction required to make a further partition on a leaf node.
f)Regularization Parameters (lambda, alpha): Control L1 and L2 regularization to prevent overfitting.

5 Evaluation

5.1 Resampling techniques

To ensure an unbiased evaluation of our final selected model's performance, we divided our dataset into 2 distinct independent sets: *train.csv* which accounted for 80% of the data, and *test.csv* includes the remaining 20%. Using the test set allowed us to assess the model's generalization capacity on unseen data.
Given that we are working with multiple models, the incorporation of a validation set is essential for model selection. Therefore, during model training, we split the *train.csv* into training and validation sets. The models were evaluated on both training and validation sets to check the overfitting. Moreover, to have a better robust evaluation of the generalization power of each model, we used 5-fold cross-validation. This approach ensures a robust evaluation throughout the model development and selection process. 5-fold cross-validation was also used during the hyperparameter tuning phase with Grid-search in order to have an unbiased evaluation and robust performance assessment for our models.

5.2 Metrics considered

For the model evaluation, we considered multiple metrics widely used in a classification task problem. We used some specific metrics to evaluate and compare the multiple models we worked with and used another additional visualization tools to evaluate the final selected model. The metrics used for all models evaluation and comparison are the following:

1. **Accuracy:** Accuracy represents the number of samples classified correctly over the number of misclassified samples. Where TP is True Positive, TN is True Negative, FP is False Positive and FN False Negative. $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

2. **Precision:** *"Out of all the positive predictions we made, how many were true?"*. It measures the proportion of true positives to the amount of total positives predicted. $Precision = \frac{TP}{TP+FP}$

3. **Recall:** *"Out of all the data points that should be predicted as true, how many did we correctly predict as true?"* It measures the proportion of true positives to the amount of total positive real samples. $Recall = \frac{TP}{TP+FN}$

4. **F1-score:** harmonic mean of precision and recall. $F1 = \frac{2*Precision*Recall}{Precision+Recall}$

   Since we are working with a multi-class classification task, we need to compute the precision, recall, and f1-score for each class of our target. To obtain the average of each metric for all classes, we used the following methods:

5. **Macro average:** computes the unweighted metric mean per class, regardless of their respective support (number of actual occurrences of the class in the dataset).

6. **Weighted average:** computes the mean of all per-class metric scores while considering each class's support. Through weighted averaging, the resultant average considers the contribution of each class, weighted by the number of examples in that specific class.

6 Modeling

6.1 Hyperparameters tuning process

After preprocessing the data, we started with the modeling stage. In our experiments, we worked with six different machine-learning algorithms widely used in ML-based Attack classification systems: Linear SVM, Kernelized SVM, KNN, Decision Tree, Random Forest, and XGBoost. For each model, we applied the GridSearch method, exploring a range of hyperparameter values to identify the optimal combination that yields the highest performance. In order to have a more robust result, we configured GridSearch to work with a 5-fold CV. In table 1 we observe the best parameter values found for each ML model.

6.2 Model Selection

After determining the optimal configuration for each model, we assessed their performance using three sets of data: training data, validation data, and 5-fold cross-validation (5-CV). We employed the accuracy metric to evaluate both the training and validation data, aiming to detect potential overfitting. Additionally, we utilized the F1-weighted score for the validation set and 5-CV to evaluate the model's generalization capacity in accurately identifying the classes within our target.
In terms of accuracy, from the table 1 we can see that the algorithms Decision Tree, Random Forest, and XGBoost are the ones that perform better, the difference of the model's generalization ability to unseen data is insignificant. However, it is noticeable that both Random Forest and XGBoost models overfitted to training data. Taking this behavior into account, the selection of the best model now becomes a discussion of whether choosing an overfitted model that gives the same performance as the non-overfitted one. After conducting a deep analysis of the aforementioned models, we decided to choose the Decision Tree as the final model due to the following considerations:

- Generalization power: the overfitted models have dropped from 0.95-1 of accuracy in training to ≃86 in validation data, which is a significant drop in performance when working with unseen data. The reason for this decline in accuracy could be that the model has likely learned specific details or noise unique to the training data, resulting in a near-perfect performance.

- Risk of Misleading Results: Although a high training accuracy is desirable, the primary objective is to choose a model capable of adapting to future unseen data or changes in the underlying patterns in the data.

In conclusion, we decided to choose as final model the Decision Tree, although its performance on training data is lower than the other two models. However, it has been demonstrated that it generalizes better to unseen data and is less prone to overfitting.

Table 1: Models performance comparison

		Train	**Valid**		**5-fold CV**
Model	**Best Hyperparameters**	**Accuracy**	**Accuracy**	**F1-weighted**	**F1-weighted**
LinearSVC	C=1	0.57	0.55	0.52	0.53
Kernel SVM	C= 100, degree = 1, gamma='scale', kernel = 'rbf'	0.72	0.63	0.63	0.63
KNeighbors	n_neighbors = 31, weights = 'distance'	1.00	0.64	0.64	0.64
DecisionTree	criterion = 'entropy', max_depth = 10, min_samples_leaf = 1, min_samples_split = 10, splitter = 'best'	0.86	0.84	0.84	0.83
Random Forest	max_depth = None, max_features = 'auto', min_samples_leaf = 1, min_samples_split = 5, n_estimators = 1000	1.00	0.85	0.85	0.84
XGBoost	colsample_bytree = 0.8, reg_alpha = 5, reg_lambda= 5	0.95	0.84	0.84	0.86

7 Final Model Evaluation

In the section, we evaluate in depth using the *test.csv* the selected final model, the Decision Tree. The model was evaluated with four methods: the classification report, confusion matrix, and ROC curve.

7.1 Classification report

The classification report Figure 1a provides an evaluation of the model's performance in classifying cyberattacks using multiple metrics.
The overall accuracy of the model is 83%, which is a good measure of the model's overall class classification. In terms of precision, it is relatively high for most classes, indicating that when the model most of the time predicts correctly each class. Note that Brute Force, Web, DDoS have perfect precision. The model struggles in classifying Spoofing observations, almost 40% of them were misclassified. In terms of recall, the model demonstrates robust performance in

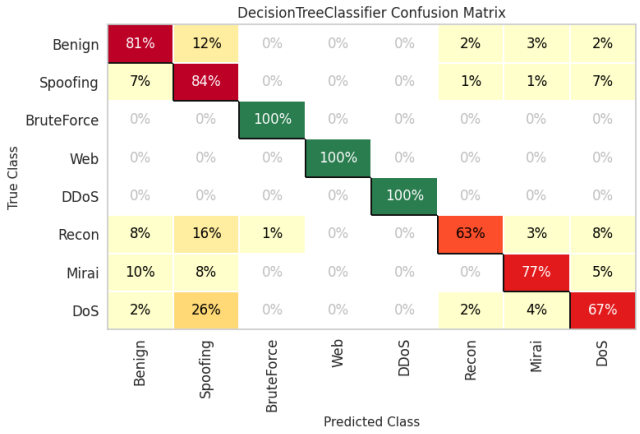
identifying instances across different classes, reflected in a weighted average of 0.83. However, the classes DoS and Mirai show lower recall values, indicating that the model misses some instances associated with these specific attack types.
The F1-weighted score provides the average of F1 scores across all classes. The F1-weighted score is pretty high (0.84), which means that the model performs well classifying on average across the classes.

7.2 Confusion Matrix

Figure 1b represents the confusion matrix of the model. The confusion matrix provides a detailed breakdown of the performance of the model by presenting the counts of TP, TN, FP, and FN. Each cell in the matrix represents the instances of the predicted and actual classes.

Class	Precision	Recall	F1-Score	Support
Benign	0.72	0.83	0.77	180
Spoofing	0.61	0.83	0.70	186
BruteForce	1.00	1.00	1.00	151
Web	1.00	1.00	1.00	152
DDoS	0.99	1.00	0.99	156
DoS	0.95	0.65	0.77	182
Recon	0.89	0.75	0.82	186
Mirai	0.73	0.69	0.71	176
Accuracy			0.83	1369
Macro Avg	0.86	0.84	0.85	1369
Weighted Avg	0.85	0.83	0.84	1369

(a) Decision Tree's Classification Report for test data



(b) Confusion Matrix of the class attacks

Figure 1: Model evaluation

The model demonstrates accurate predictions for observations belong to Brute Force, Web attacks, and DDoS. For all other attack classes, the accuracy exceeds 65%. However, misclassifications occur across various attack categories:

1. **Benign:** several samples (12%) got misclassified as Spoofing attack.
2. **Spoofing:** Same as in the previous case, some Spoofing samples were misclassified as benign.
3. **Recon:** Several Recon observations were classified as benign, Spoofing or DoS.
4. **Mirai:** Same as Spoofing, several observations of this attack are misclassified as benign, Spoofing, or DoS.
5. **DoS:** Almost 30% of the DoS samples were considered by the model as Spoofing.

From the confusion matrix, we can conclude that there exists some similarity between some attacks. Several samples from different attacks get misclassified mainly with the Spoofing class. The statement is generally correct but could benefit from a slight clarification.

7.3 ROC Curve

We also wanted to assess the ROC Curve Figure 9 of each class, which represents the True Positive Rate (Sensitivity) against the False Positive Rate (1 Specificity) for different threshold values. In summary, the AUC for all attack classes exceeds 0.9, indicating that the model effectively discriminates each attack class from the others. The high AUC values suggest strong predictive performance and a robust ability to differentiate between different types of attacks. In conclusion, the model has a highly effective performance in classfying the cyberattack categories across the testing data.

8 Conclusions

Throughout the development of the project, we were able to apply different ML techniques which allowed us to gain practical knowledge and hands-on experience.
Although our best model predicts the right type of attack for some categories more often than not, it also exhibits constraints in reliably classifying certain other categories.

SVM failed to produce particularly strong results. This outcome could stem from multiple factors, one of which might be the method of feature transformation applied. Given that it relies heavily on the distance between data points, inadequate scaling of features can result in those with larger scales disproportionately influencing the decision boundary. thereby compromising the model‘s performance. Additionally, the hyperparameters need careful tuning. We did try different values but due to a lack of computational resources and time, we could not do an exhaustive tune.

9 Limitations

In future research, it would be beneficial to extend the scope of exploration to encompass a broader range of machine-learning techniques that were not examined during the development of this project. Other ensemble methods like Stacking or Voting Classifiers could also be tried, including Deep Neural Networks. Additionally, more kernel functions (custom ones) could be explored in order to improve the results of the SVMs methods used in the project.
Altering our approach from a multi-class to a binary classification(malign vs benign attacks) problem could potentially enhance the performance of our model. This shift in strategy, by simplifying the complexity of the classification task, might lead to more accurate and robust results.

With respect to the dataset, undertaking a more thorough process of feature selection and transformation could prove beneficial. Exploring additional data transformations, such as Box-Cox transformation among others, might provide valuable insights into enhancing the performance of various models. Experimenting with these transformations could reveal ways to optimize the dataset for improved model results.

A significant constraint encountered in our study was the limitation in computational resources. Enhanced computational capabilities would have been ideal for hyperparameter-tuning, as it would have allowed for a more extensive exploration of hyperparameters and a more extensive exploration of different models. With greater computational power, we could have experimented with a wider combination of hyperparameters and assessed their performance across various datasets, each subjected to different transformations and featuring diverse sets of attributes.

References

[1] Neto, E. C. P., Dadkhah, S., Ferreira, R., Zohourian, A., Lu, R., & Ghorbani, A. A. (2023). *CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment*

[2] *https://www.mdpi.com/1424-8220/23/13/5941*

[3] Zuur, A. F., Ieno, E. N., & Elphick, C. S. (2010). *A protocol for data exploration to avoid common statistical problems. Methods in ecology and evolution*, 1(1), 3-14.

[4] Alghushairy, O., Alsini, R., Soule, T., & Ma, X. (2020). *A review of local outlier factor algorithms for outlier detection in big data streams. Big Data and Cognitive Computing*, 5(1), 1.

[5] Berger, V. W., & Zhou, Y. (2014). *Kolmogorov–smirnov test: Overview. Wiley statsref: Statistics reference online.*

[6] *The conundrum of 'C': SVM hyperparameters*, https://medium.com/swlh/the-conundrum-of-c-svm-hyperparameters-3327dfc7354a

[7] Chauhan, Vinod Kumar & Dahiya, Kalpana & Sharma, Anuj , Y. (2019). *Problem formulations and solvers in linear SVM: a review*

[8] Amey Band *Multi-class Classification — One-vs-All One-vs-One* https://towardsdatascience.com/multi-class-classification-one-vs-all-one-vs-one-94daed32a87b

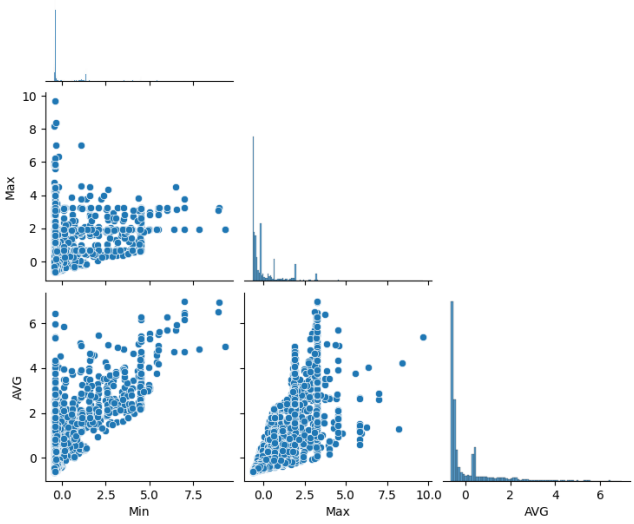
[9] *https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*

10 Appendix

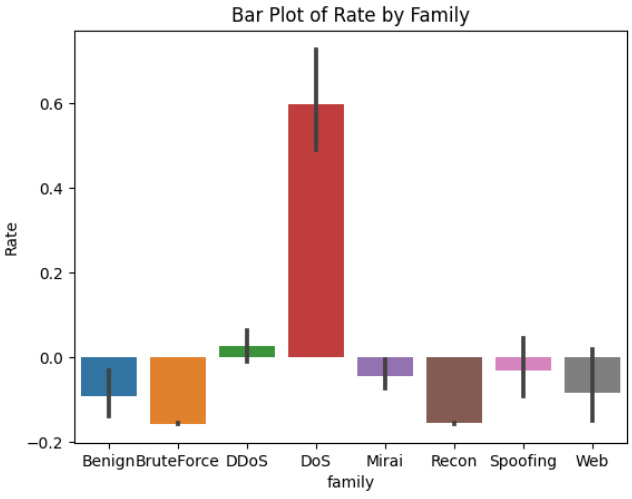
10.1 Figures and Tables

Table 2: New generated features

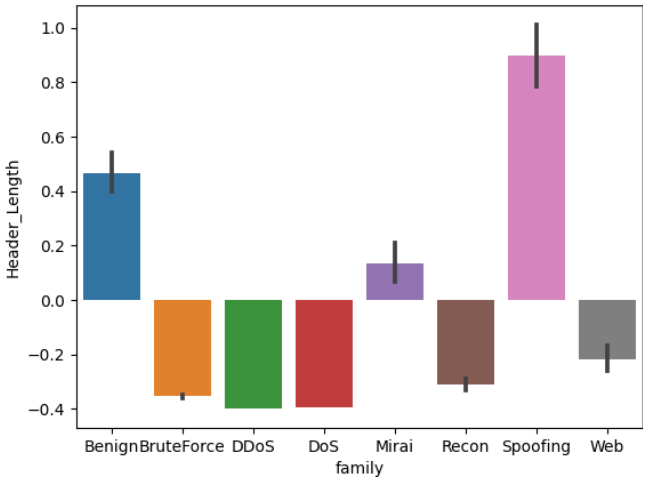
New Feature	Definition
ratio_ack	Ratio of ACK packets to total packets
ratio_syn	Ratio of SYN packets to total packets
ratio_fin	Ratio of fin packets to total packets
ratio_urg	Ratio of urg packets to total packets
ratio_rst	Ratio of rst packets to total packets
ifRskAck	Boolean that indicates if RST and ACK flags are activated.
ifBiggerPacketSize	If the current packet size is bigger than the average packet size in the flow.
ratioPacketSizeToTotal	Ratio of current packet size to the sum of packets size in the flow.
ratioTotalSizeToTotalNumber	Ratio of total packet size to the total number of packets in the flow
ratioTotalSizeToDuration	ratio of total packet size to the average flow duration



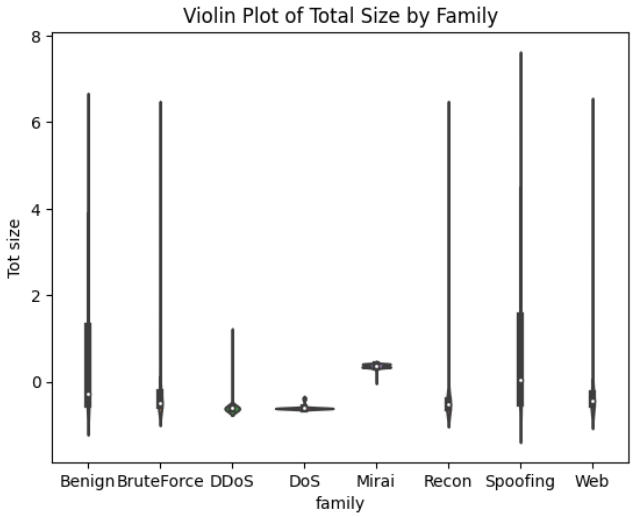
(a) Packet Size Statistics



(b) Rate by Family of Attack



(a) Header length by Family of attack



(b) Total packet size by family of attack

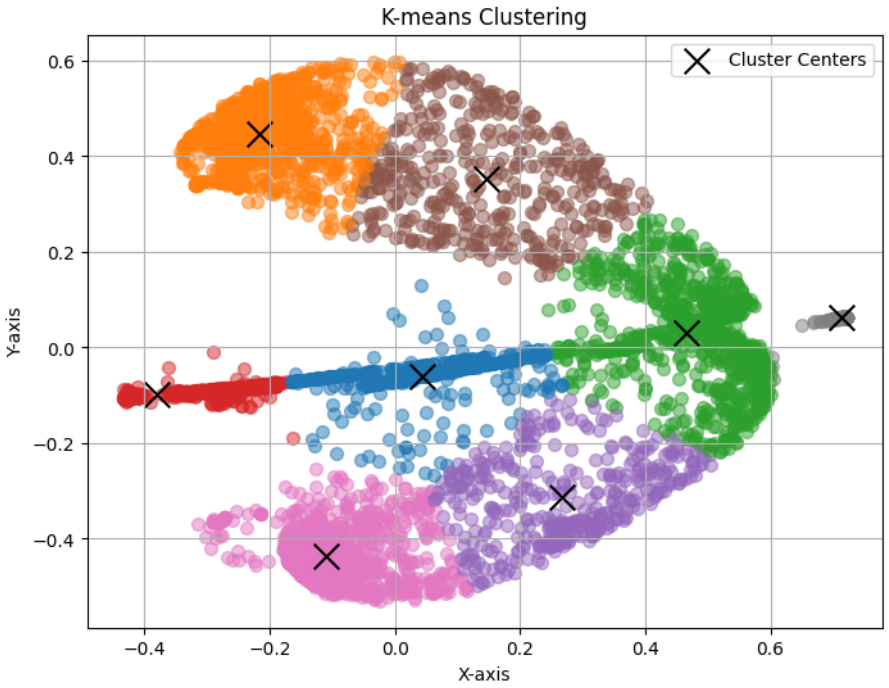


Figure 4: Clusters distribution

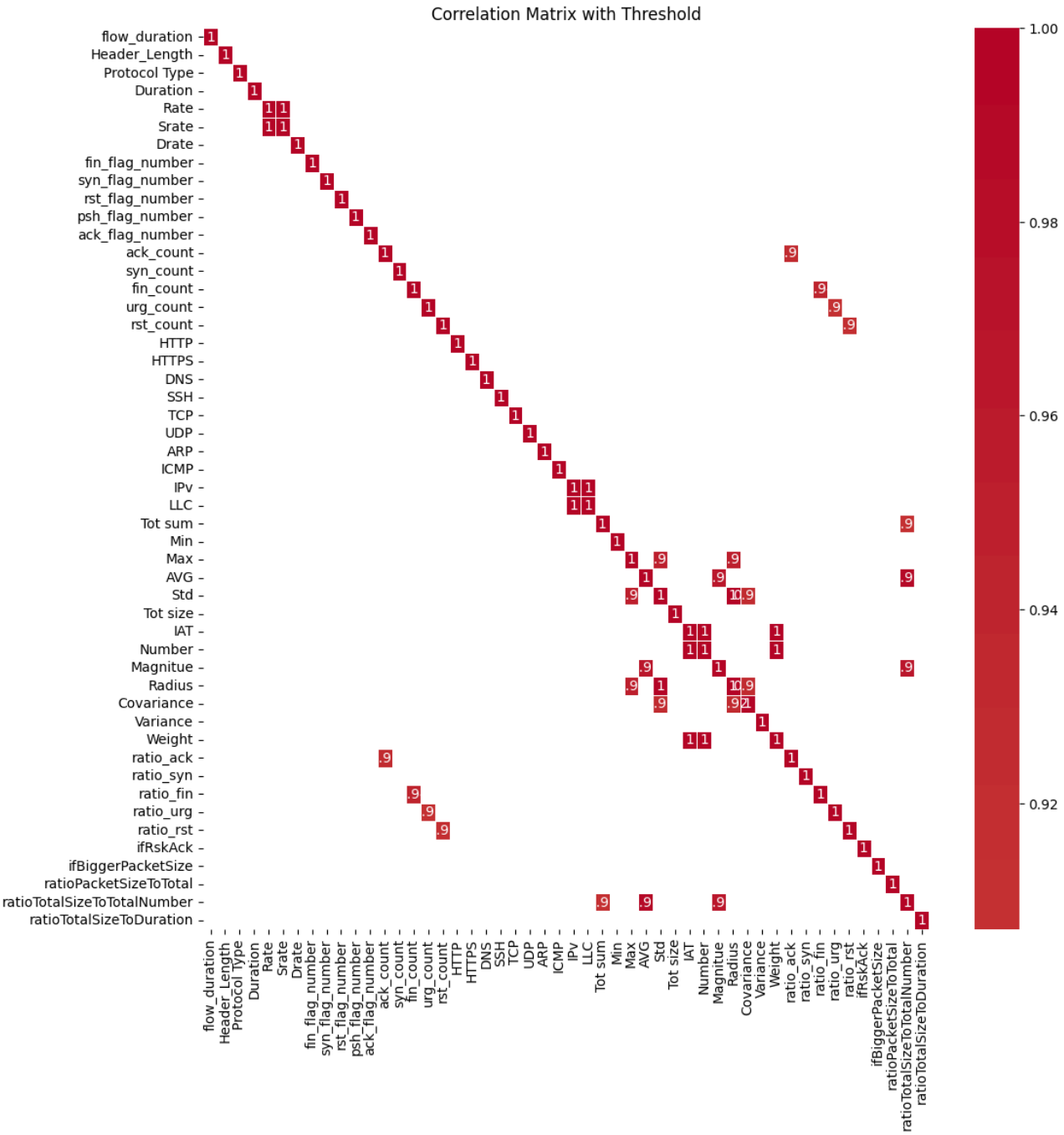


Figure 5: Correlation Matrix

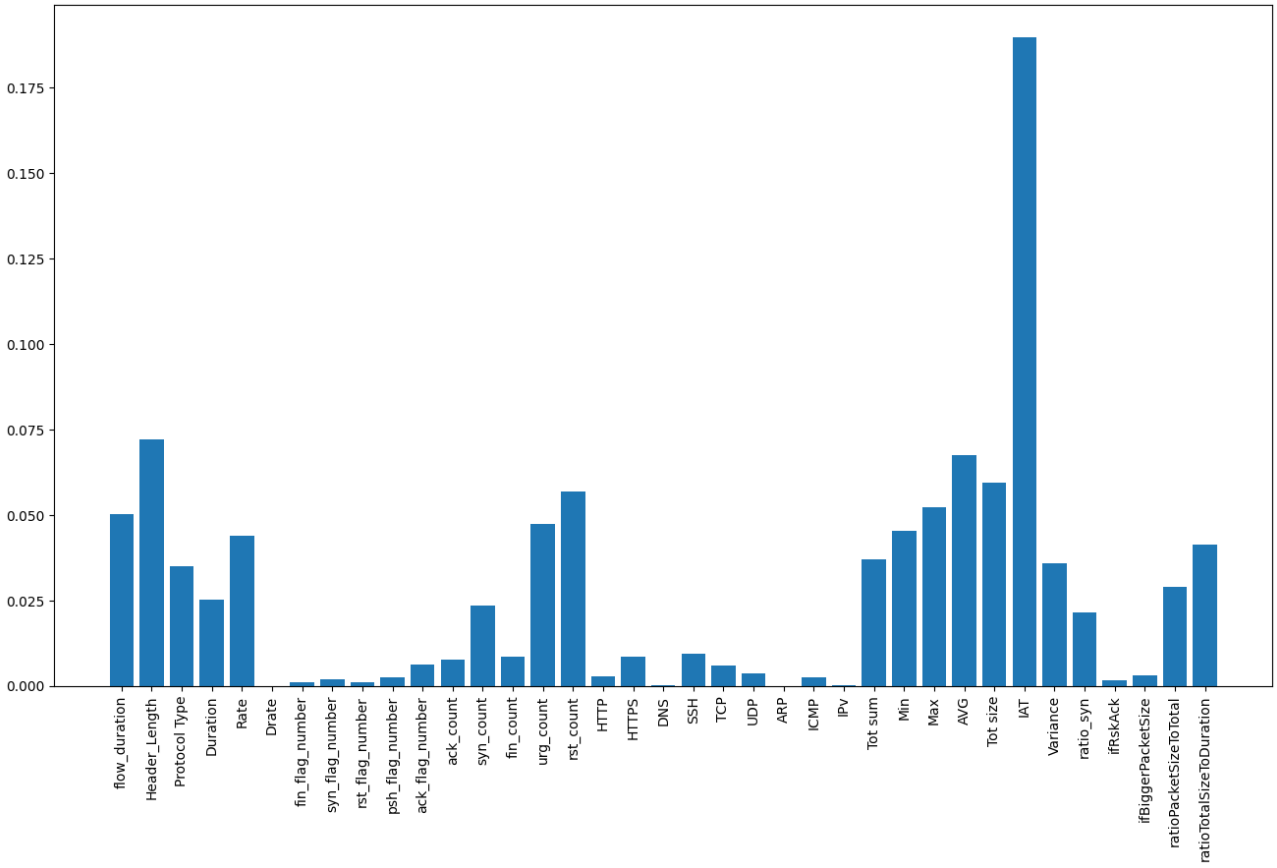
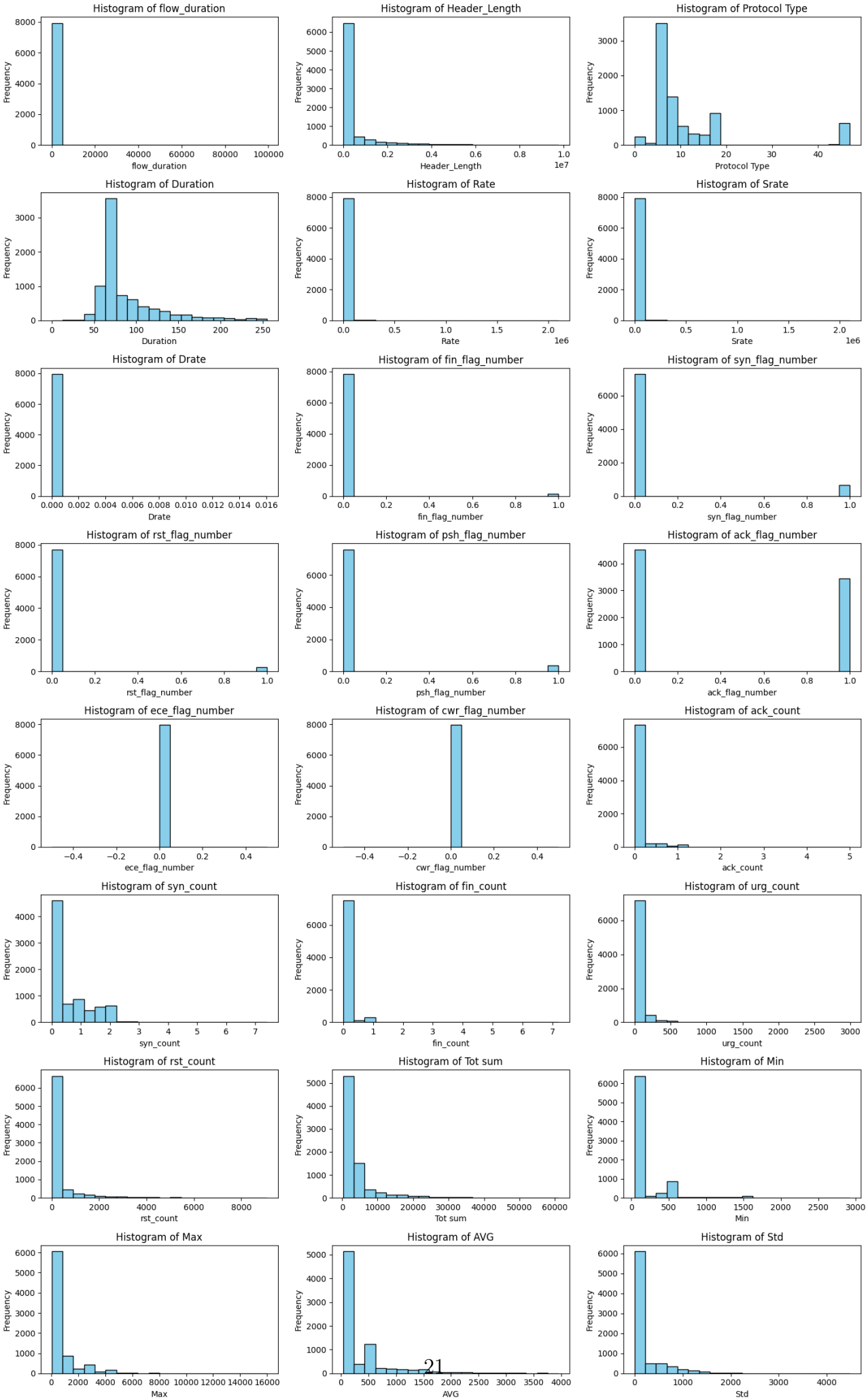
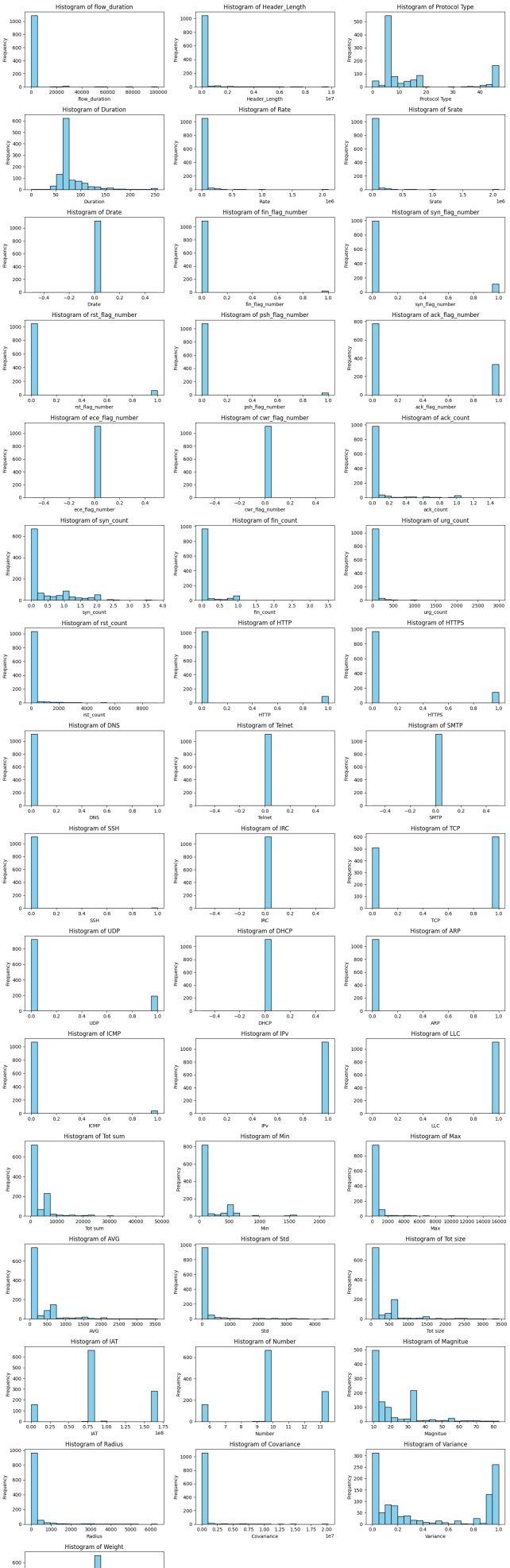


Figure 6: Feature importance computed by Random Forest


Histogram of flow_duration
Frequency
flow_duration
Histogram of Header_Length
Header_Length
1e7
Histogram of Protocol Type
Protocol Type
Histogram of Duration
Duration
Histogram of Rate
Rate
1e6
Histogram of Srate
Srate
1e6
Histogram of Drate
Drate
Histogram of fin_flag_number
fin_flag_number
Histogram of syn_flag_number
syn_flag_number
Histogram of rst_flag_number
rst_flag_number
Histogram of psh_flag_number
psh_flag_number
Histogram of ack_flag_number
ack_flag_number
Histogram of ece_flag_number
ece_flag_number
Histogram of cwr_flag_number
cwr_flag_number
Histogram of ack_count
ack_count
Histogram of syn_count
syn_count
Histogram of fin_count
fin_count
Histogram of urg_count
urg_count
Histogram of rst_count
rst_count
Histogram of Tot sum
Tot sum
Histogram of Min
Min
Histogram of Max
Max
Histogram of AVG
AVG
Histogram of Std
Std
Histogram of Tot size
Histogram of IAT
Histogram of Number


Histogram of flow_duration
Histogram of Header_Length
Histogram of Protocol Type
Histogram of Duration
Histogram of Rate
Histogram of Srate
Histogram of Drate
Histogram of fin_flag_number
Histogram of syn_flag_number
Histogram of rst_flag_number
Histogram of psh_flag_number
Histogram of ack_flag_number
Histogram of ece_flag_number
Histogram of cwr_flag_number
Histogram of ack_count
Histogram of syn_count
Histogram of fin_count
Histogram of urg_count
Histogram of rst_count
Histogram of HTTP
Histogram of HTTPS
Histogram of DNS
Histogram of Telnet
Histogram of SMTP
Histogram of SSH
Histogram of IRC
Histogram of TCP
Histogram of UDP
Histogram of DHCP
Histogram of ARP
Histogram of ICMP
Histogram of IPv
Histogram of LLC
Histogram of Tot sum
Histogram of Min
Histogram of Max
Histogram of AVG
Histogram of Std
Histogram of Tot size
Histogram of IAT
Histogram of Number
Histogram of Magnitue
Histogram of Radius
Histogram of Covariance
Histogram of Variance
Histogram of Weight
Frequency

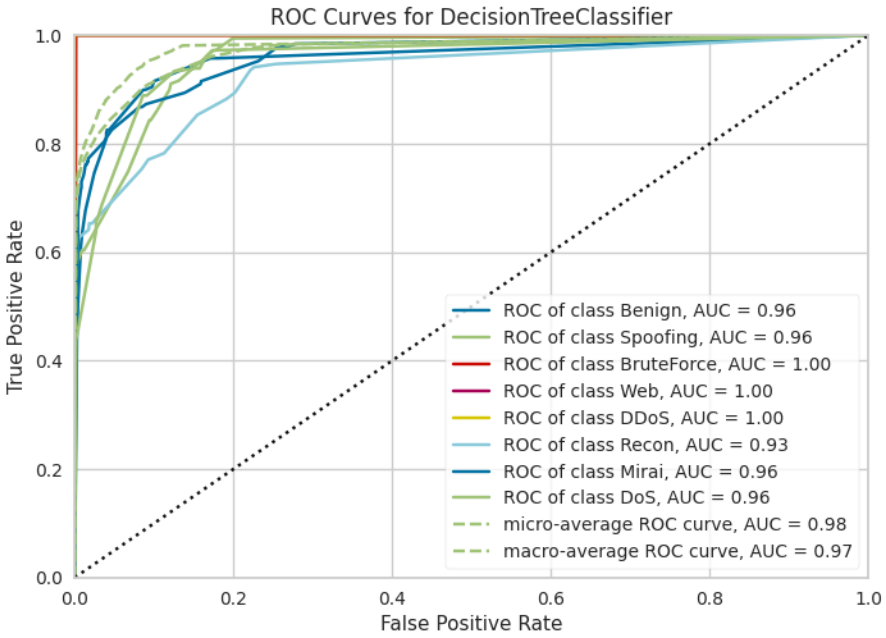


Figure 9: ROC Curve of each attack category