DDoS attacks detection and characterization

Simone Sampognaro s322918 , Manuel Firrera s329226
Luca Nobili s331461 , Alessandro Mulassano s330263

Prof. Luca Vassio
01DSMBG - Machine learning for networking

January 29, 2024

Politecnico
di Torino

Contents

1 Abstract

DDoS attacks are among, if not the most, common type of cyber-attacks. Even in modern era internet, these kinds of attacks never cease to exist due to a number of vulnerabilities of the protocols on which the World Wide Web is based. In this short paper we will conduct a feasibility study on the possibility of preventing such attacks by training an AI with real-world data and using it in an IDPS defence system.

2 Introduction

A DDoS (Distributed Denial of Service) attack is a type of cyber attack that aims to disrupt the normal functioning of a targeted system, network, or service by overwhelming it with a flood of traffic from multiple sources. In a DDoS attack, the attacker typically uses a network of compromised computers (referred to as "bots" or "zombies") to send an excessive amount of traffic to the target, thereby causing it to become slow, unresponsive, or completely inaccessible to legitimate users.
Among the numerous cyber attacks that we can find in the wild, DDoS comes always as the most common because of it's simplicity and highly disruptive effects. Looking at the numbers, Tera-bits/s attacks are now the standard. For the http class it's in the order of millions of requests every second. This outrageous quantity of traffic is able to take down even the biggest and most powerful servers like the ones owned by Google, CloudFlare or AWS, Amazon web services' company, which have reported numerous large scale attacks in the last five years. For this reason companies, even the smaller ones, should invest into defence systems to block any kind of attack, in our case, DDoS ones. With the advent of AI and Machine Learning it is becoming increasingly cheaper to adopt said solutions, thanks to the automatic nature of an AI based Intrusion Detection and Prevention System (IDPS). Our studies are based on a dataset provided by our professors for this project. We will dig deeply into it's features, with numerous plots, then we will train and test numerous machine learning models with various approaches to demonstrate the effectiveness of Machine Learning for defence purposes and visualize their capabilities.
From now on we will refer to the attacks as label numbers in this way:

Labels	Ground truth
0	Ddos_dns
1	Benign
2	Ddos_ldap
3	Ddos_mssql
4	Ddos_netbios
5	Ddos_ntp
6	Ddos_snmp
7	Ddos_ssdp
8	Ddos_udp
9	Ddos_syn
10	Ddos_tftp
11	Ddos_udp_lag

Table 1

3 Data exploration and pre-processing

Exploring various aspects, including flow, IP, and ports within a dataset, is essential for gaining insights into the dataset's structure and temporal evolution. In the context of machine learning, understanding the intricacies of the data is paramount for effective analysis and modeling. For instance, in the domain of network data, a detailed examination of information flow can yield valuable insights into communication patterns among different components.

The provided dataset contains numerous flow-level metrics, as well as a 'Ground Truth' column (GT). An identificator, composed by the combination of: Source IP, Source port, Destination IP, Destination port, Protocol, is used to distinguish between different flows.
The presence of both categorical and numerical features has prompted us to dig deeper into the dataset and eventually take care of them.
Then we can find data regarding the traffic such as features about the length of individual packets and within the context of the flow, time-related features such as sending rate, flow duration and inter-arrival times of the packets, various flags, inbound/outbound and so on.
Given the quantity of different features and data points, the initial step in this project involves showcasing the dataset using diverse data visualization methods and conducting analyses.

3.1 Dataset investigation

In this section, we will analyze the dataset based on the associated flows and IP addresses.
The first chart in the report is dedicated to showcasing the top five source and destination IP addresses most frequently used in the dataset. This graphical representation provides an immediate overview of the primary players in network communication, highlighting the IP addresses that are most active as both senders and receivers.

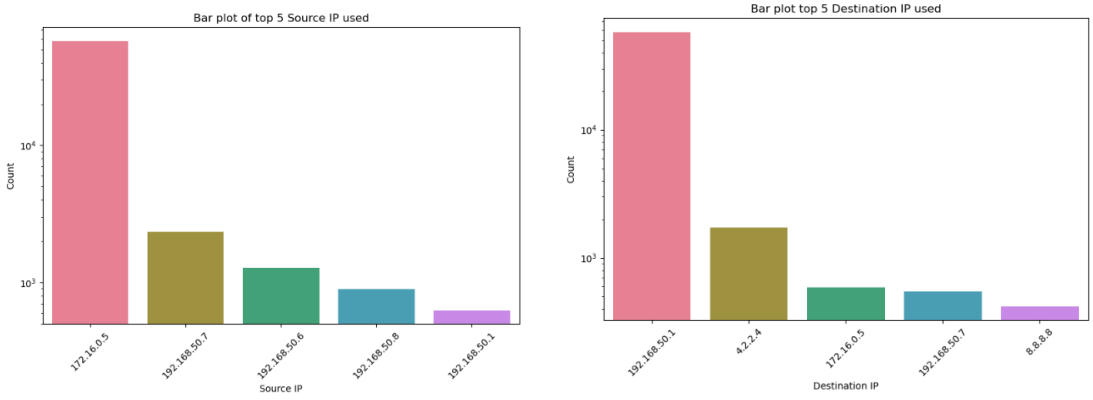


Figure 1

An intriguing observation common to both plots is the pronounced dominance of a singular IP address, suggenting that two 'main actors' will be involved in the majority of the traffic exchanged.
In this upcoming visualization, we'll continue our exploration of the dataset by examining the top 5 source IP addresses. Each IP will be presented through a box plot, offering insights into the flow packets per second (packets/s) within the flows it is engaged in.
Moreover, this investigation extends to the top 5 destination IP addresses. Similar box plots will depict the corresponding flow packets per second (packets/s) for each of these destination IPs.

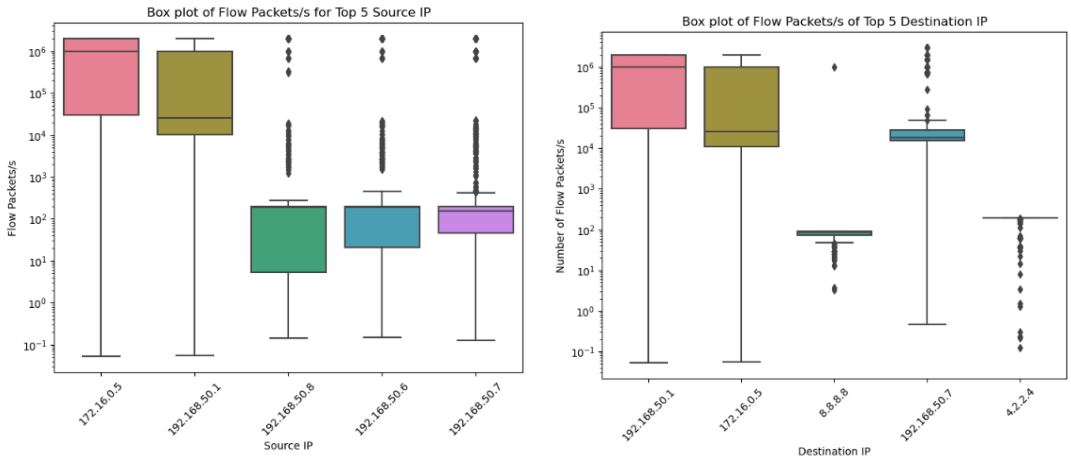


Figure 2

In both charts, a striking similarity is observed in the box plots associated with the top two IP addresses. This suggests a resemblance in the flow characteristics, particularly concerning the packets per second (packets/s).
However, a notable difference arises in the chart related to source IPs, where the third, fourth, and fifth IPs exhibit remarkably similar box plots. This discrepancy in the source IP chart suggests shared characteristics in flow duration among these specific addresses, prompting further exploration to unveil potential commonalities or distinct behaviors.

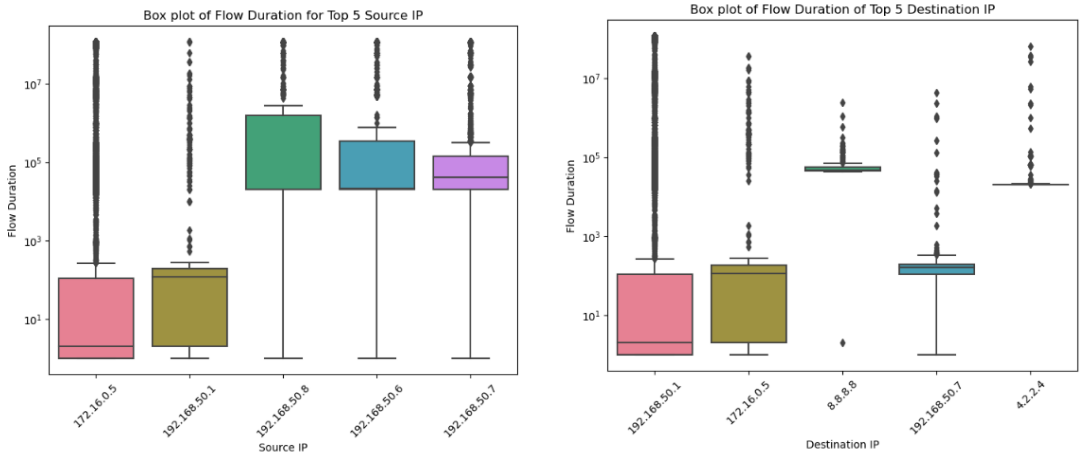


Figure 3

The same exploration for the top 5 source and destination IPs extends to features related to Flow IAT (Inter-Arrival Time) and packet length. In this instance, bar plots are employed to depict the averages of the maximum and minimum values for these features across the flows of each IP. This visual representation provides insight into the central tendencies of Flow IAT and packet length for the identified IPs.

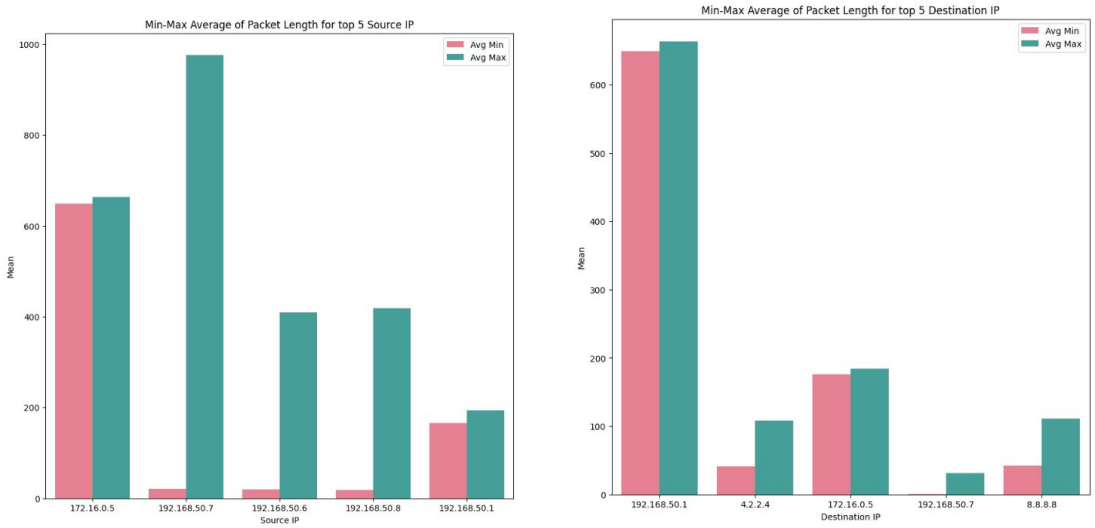


Figure 4

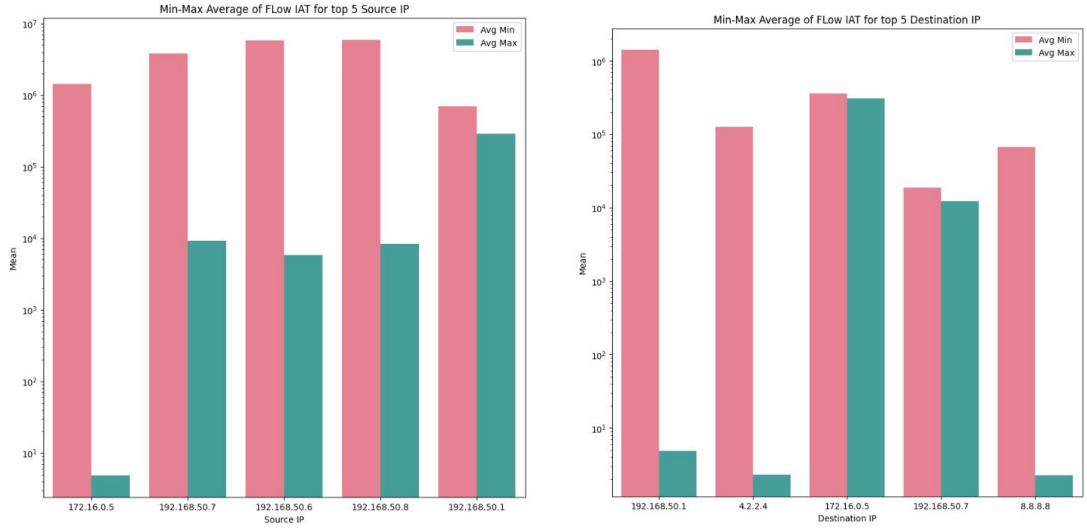


Figure 5

It's important to highlight that the average of maximum and minimum values is nearly identical for the source IP 172.16.0.5 concerning the packet length feature. Similarly, for the destination IP 192.168.50.1.
Another noteworthy observation is that the IP address 192.168.50.1, which appears in the most frequently used destination IP addresses, and the source IP address 172.16.0.5 exhibit very similar values in both analyzed features. From this particularity, one can infer that these two IPs predominantly communicate with each other.
Similar to our approach for identifying the top 5 source and destination IP addresses, we will extend our analysis to include the top 10 ports commonly utilized for both source and destination. This step is crucial for a comprehensive understanding of the dataset, as depicted in the subsequent figures.

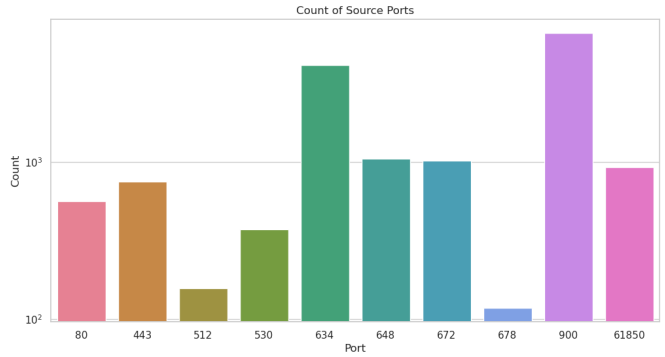


Figure 6

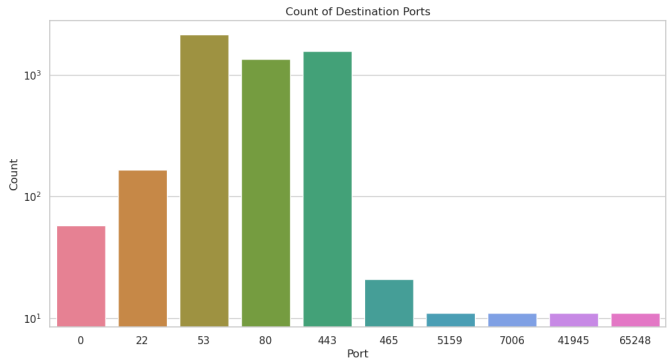


Figure 7

As evident from the two frequency charts depicting the top 10 ports for both source and destination, it is noteworthy that in the latter only 5 of them exhibit a significantly higher frequency compared to the remaining 5. This pattern suggests a crucial decision point for the subsequent steps. As done for the top 5 IPs, we will now analyze two features for the top 10 source ports. First, we will examine 'Total Forward Packets' using box plots, where a significant number of outliers is observed in port 634, and a similar trend for the remaining 5 ports. The second feature under analysis is 'Flow Duration' Here, the port trends closely resemble those observed in the previous feature, with the additional observation of a high presence of outliers across all ports. Furthermore, port 634 exhibits a distinctly different pattern in both graphs compared to other ports.

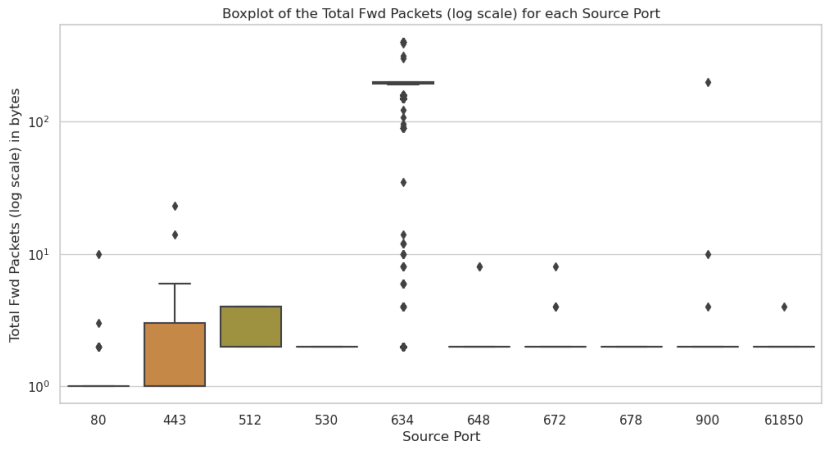


Figure 8

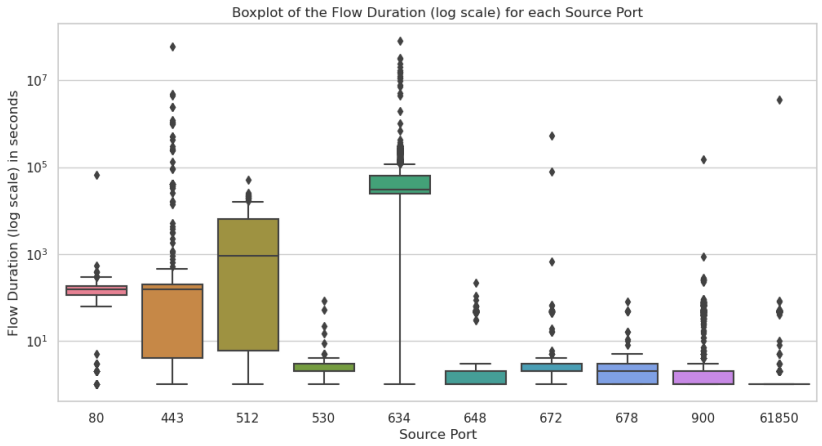


Figure 9

The same analysis is now being presented for the top 10 destination ports. In these two graphs, it is immediately noticeable that the box plots for the last 4 destination ports exhibit strikingly similar patterns.

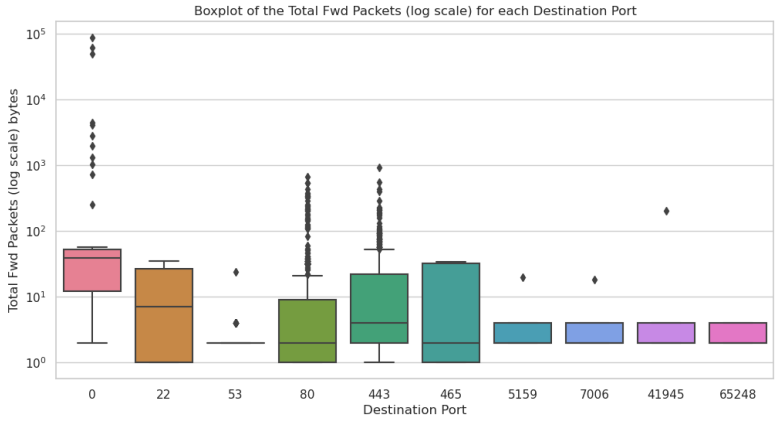


Figure 10

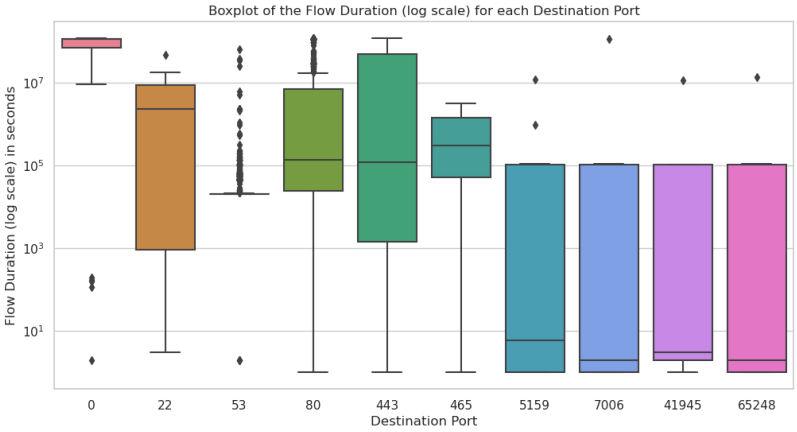


Figure 11

This similarity may introduce complexities for machine learning algorithms attempting to classify or cluster data based on these features.

3.2 Ground truth level visualization

Our exploration spans both the generic traffic (GT) level, involving the distribution of features and characterizing GT classes.
To better comprehend the essence of our flows, we start by examining the distribution of labels within our dataset. The use of a pie chart enables a visual representation of label proportions, providing a quick snapshot of the composition of our data. Also, a bar plot is provided.

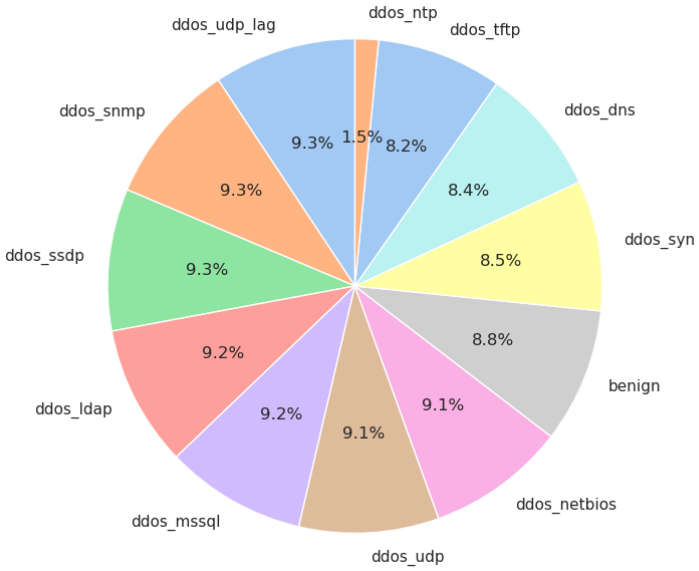


Figure 12

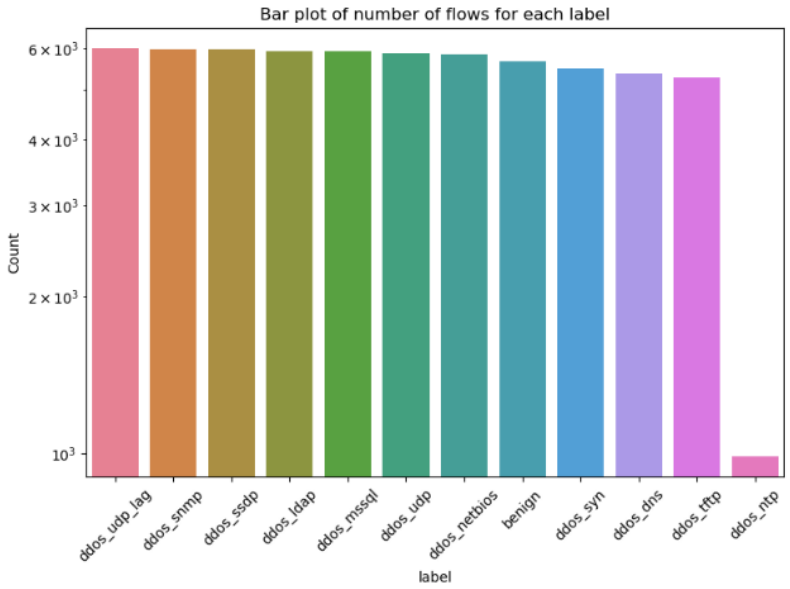


Figure 13

It is evident upon examination that the dataset is organized in a consistent manner, with the exception of the ddos ntp attack, which is notably underrepresented in terms of quantity.
As we embark on an exploration of our dataset, our attention turns towards unraveling the distribution dynamics of UDP, TCP, and HOPOPT protocols. To initiate this investigation, our

approach involves creating a pie chart that dissects the frequencies associated with each protocol. The visual representation promptly draws attention to the omnipresence of UDP. Conversely, HOPOPT appears elusive, contributing a mere 0.1 percent to the dataset.

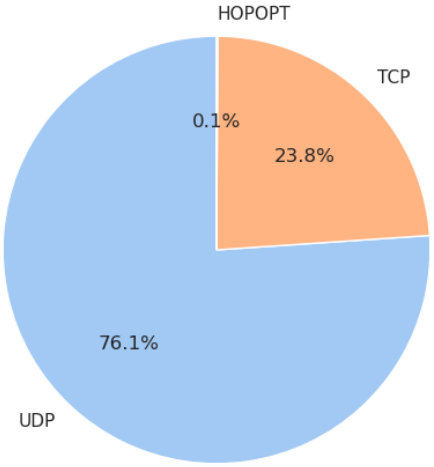


Figure 14

As we transition towards a more detailed examination, our focus shifts to understanding how these protocols are partitioned across various labels. This phase aims to uncover the intricate relationships between the UDP, TCP, and HOPOPT protocols and the associated labels within our dataset.

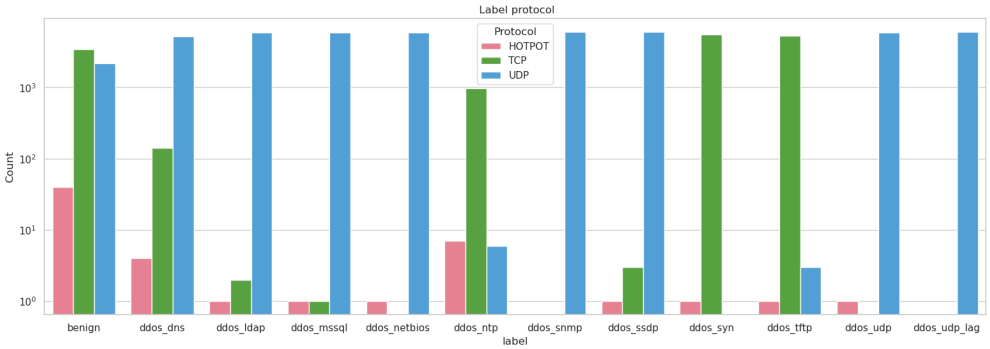


Figure 15

As anticipated, the visual representation of the partitioning of protocols among various labels reaffirms the robust prevalence of UDP within our dataset. Conversely, the presence of the HOPOPT protocol remains notably sparse, aligning with our earlier findings of its limited representation.
To enhance our understanding of the data, we've included bar plots depicting the mean, minimum, and maximum values for the total forwarded packets, the total flow duration and about the total length of packets for each label in our dataset.
In the accompanying visualizations, the bars represent the mean values, while the error bars provide insights into the variability by indicating the minimum and maximum values.

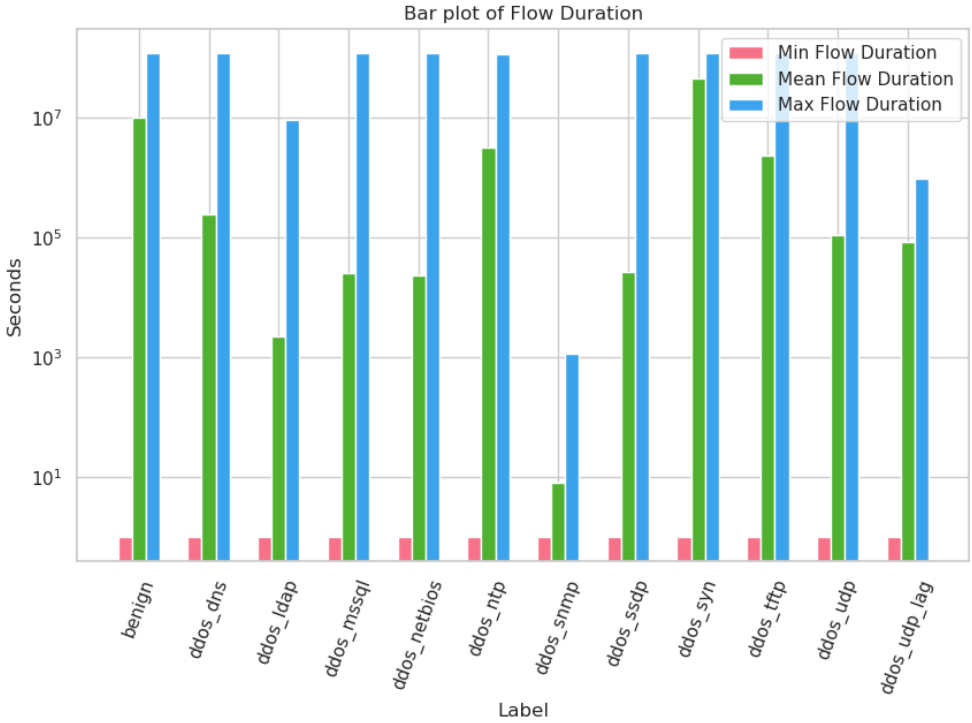


Figure 16

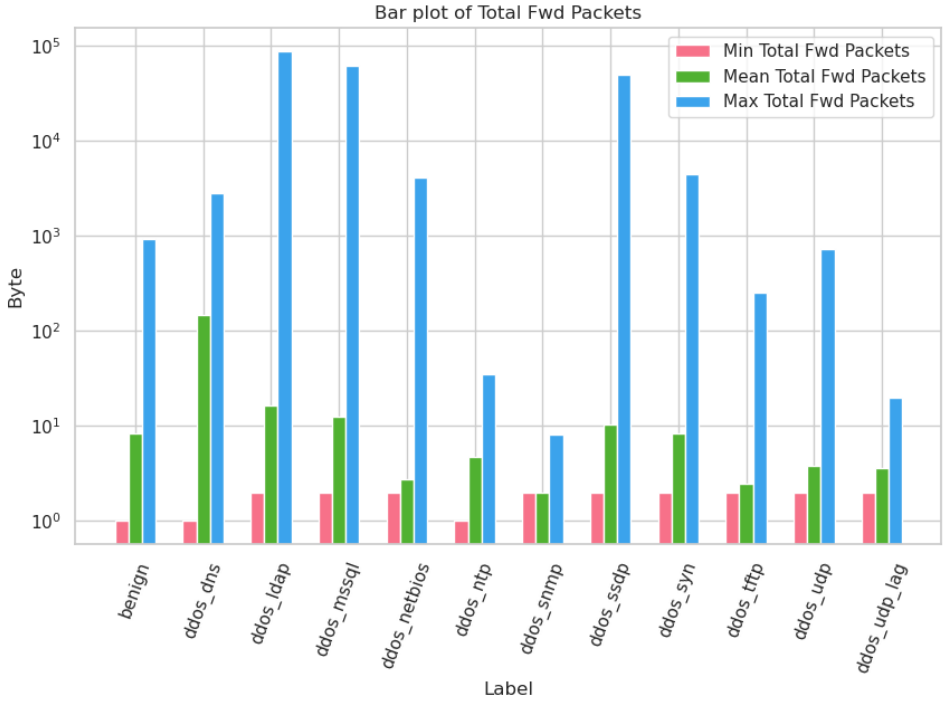


Figure 17

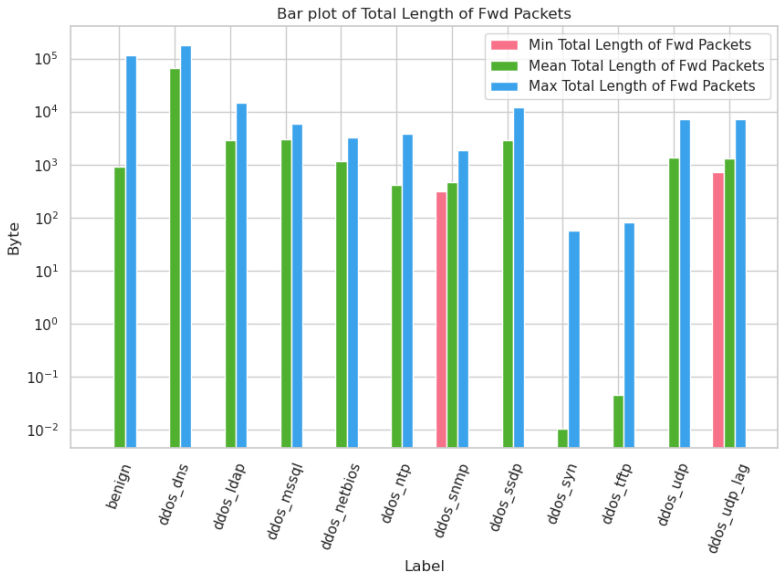


Figure 18

As evident from the graphical representation of total length of forwarding packets (Figure 18), it is apparent that, for numerous labels, the minimum packet length is in close proximity to zero. This observation implies that instances exist within the dataset where packets with minimal length, closest to 0, are prevalent across various labels. However, for the flow duration (Figure 16), all the bar plots related to the minimum value are identical across all labels.
The "Min Packet Length" is another crucial feature to visualize. In this regard, the figure below illustrates its distribution for each label through a boxplot. It is essential to observe that each attack exhibits a distinct distribution of values for this feature compared to others. This observation suggests that this feature may play a significant role in discriminating between different attacks in the models utilized in subsequent sections of the analysis.

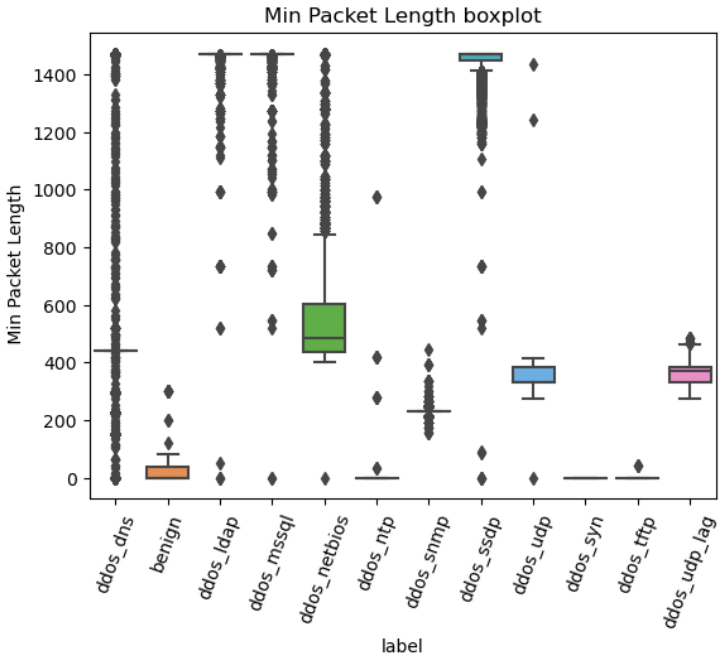


Figure 19

Another way to visualize the distribution of features across different labels is through the Empirical Cumulative Distribution Function (ECDF). The ECDF represents the cumulative probability distribution of a dataset, providing insights into how the data is spread out.
By utilizing the ECDF, we can explore how source and destination ports are distributed for each label in our dataset (Figure 20). This visualization allows us to understand the cumulative distribution of these specific features, offering a comprehensive perspective on their prevalence and spread within each label.
In terms of source ports, depending on the type of attack, we observe a prevalence of a specific port for each label. For destination ports, this occurs only for two labels, while the others evenly utilize a large number of ports, resulting in our plot having an almost linear trend.

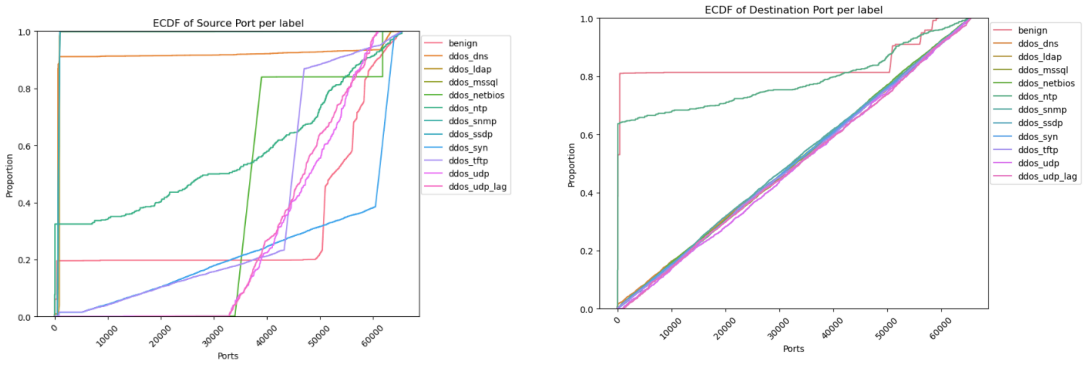


Figure 20

Furthermore, it is essential to investigate the segmentation of various flows not only based on source and destination IP addresses but also considering the most frequent communications. Specifically, we aim to analyze the quadruplet of source IP, destination IP, source port, and destination port that occurs most frequently.

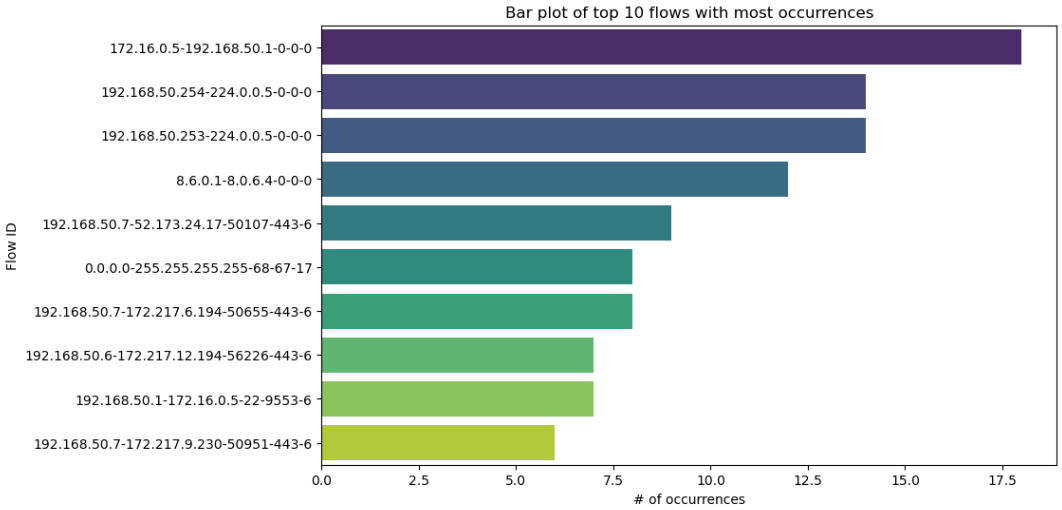


Figure 21

In the last analysis, we examine the distribution of number of flows over time. Specifically, we consider a fraction of a second, namely 0.001s, to observe how many flows we have for each attack. From the figure we can observe that initially there is a prevalence of two attacks for an extended period, followed by sporadic attacks that rapidly send a large quantity of flows. In

fact, there are some attacks that send more than 20/25 flows in 0.001s, as ddos_udp,ddos_syn and ddos_udp.

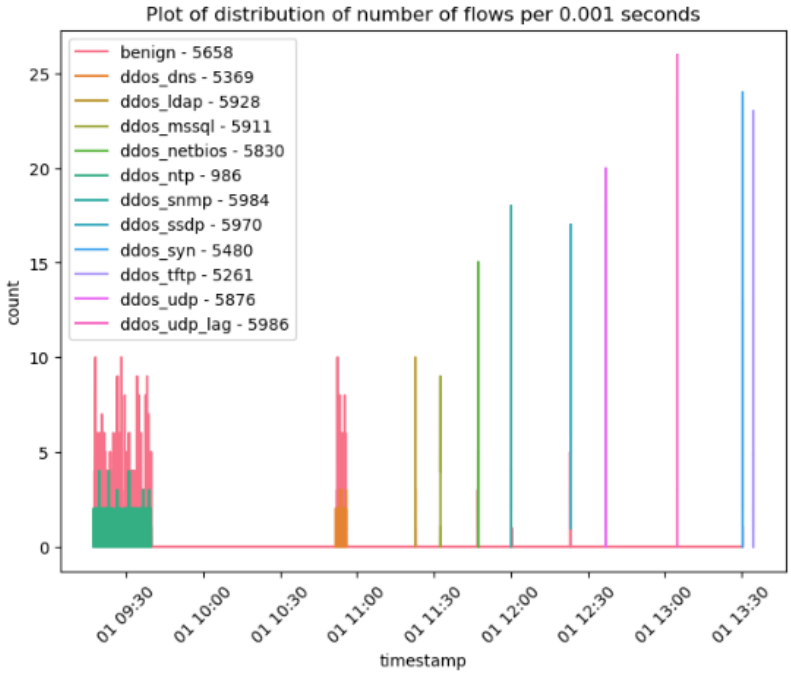


Figure 22

3.3 Pre-processing

In this segment of our machine learning pipeline, we are engaged in the crucial task of preprocessing the dataset. Let's break down the steps we've taken to prepare our data for effective model training.

In addressing the complexity of our machine learning model for network data analysis, a crucial consideration emerged concerning the multitude of source and destination ports. Due to their categorical nature, since the port numbers do not possess an inherent order or ranking, a decision was made to leverage one-hot encoding as a preprocessing technique.

Upon delving into the dataset, a noteworthy insight surfaced a significant number of source and destination ports exhibited a wide range of frequencies. To effectively capture this variability in our model, we opted to create specific features for the top 10 frequently occurring source ports and the top 5 destination ports. These features indicate the presence or absence of each port in a given data instance.

Less frequent source and destination ports are efficiently grouped into a collective 'others' category, namely '-1'. This streamlined approach not only enhanced the model's interpretability but also addressed the challenges posed by the extensive range of port possibilities.

In summary, the strategic use of one-hot encoding, driven by the observed frequency distribution of source and destination ports, ensures a focused representation of the most relevant information. This tailored approach allows our machine learning model to effectively discern patterns in the predominant ports, while efficiently handling the variability inherent in the less frequent ones (Figure 6 and Figure 7).

Then, we utilize the One-Hot Encoding again for the 'Protocol' column. This technique expands the single 'Protocol' column into three separate columns ('Protocol 0', 'Protocol 6', 'Protocol 17' : those that we had previously defined as UDP, TCP, and HOPOPT), each denoting the presence or absence of a specific protocol.

To streamline our dataset further, we remove 'all 0s' columns such as source and destination IP addresses, timestamp, flow ID, 'SimilarHTTP' and 'Protocol.' These exclusions contribute to a more focused and efficient feature set for training our machine learning model.

Afterwards, we divide our preprocessed dataset into training and testing sets using the train_test_split function. The selected features for DDoS classification, denoted by 'features_ddos,' are separated into X_train and X_test, while the corresponding labels are split into y_train and y_test.

The 'stratify' parameter ensures a balanced distribution of class labels in both sets, crucial for robust model evaluation. With 80% of the data allocated for training, this division sets the stage for the subsequent stages of model development and assessment.

Following the dataset split, the next vital step in the pipeline involves standardization. Standardization is a crucial preprocessing technique that ensures all our features are on the same scale, preventing certain features from dominating the learning process due to differences in their magnitudes. In this case, after having observed empirical gaussian PMFs across various features, the normalization approach was taken.

To achieve this, we employ standardization on both our training (X_train) and testing (X_test) sets. This process involves centering the data by subtracting the mean and scaling it by dividing by the standard deviation.

Standardization not only facilitates a more stable and efficient training process but also ensures that our machine learning model is better equipped to generalize well to unseen data. This standardized data is then ready for input into our chosen machine learning algorithm, contributing to a more robust and accurate model for DDoS attack classification.

Afterwards we perform a correlation analysis. The purpose is to understand the relationship between different variables in our dataset.

In particular we want to identify if there are patterns, dependencies and potential insights that can help us to highlight redundant or highly correlated features. Removing such features can lead us to a more efficient and accurate model.

To do this, first, we calculate the correlation matrix, which provides the correlation of each feature with the others (a number between 0 and 1, the higher means more correlation between features). Then, we extract individual correlations between two features (feature1 - feature2) that are greater than 80%, and we order them based on two criteria: the first in terms of the

correlation between the two features, and for the second, we calculate the sum of the squares of the correlations between the first feature (feature1) and all the other features; applying the square to each correlation gives more weight to correlation values approaching 1. This ensures that among two features with the same correlation, the one with a higher correlation with all others is ranked higher than the other.
Finally, we iterate over all pairs of features and remove the less important one, saving the other to prevent its removal in subsequent steps due to a feature that is undoubtedly less important. This is because, as mentioned before, they are ordered. The remaining features are uncorrelated, as shown in the heatmap (Figure 23), therefore highly descriptive for our dataset.

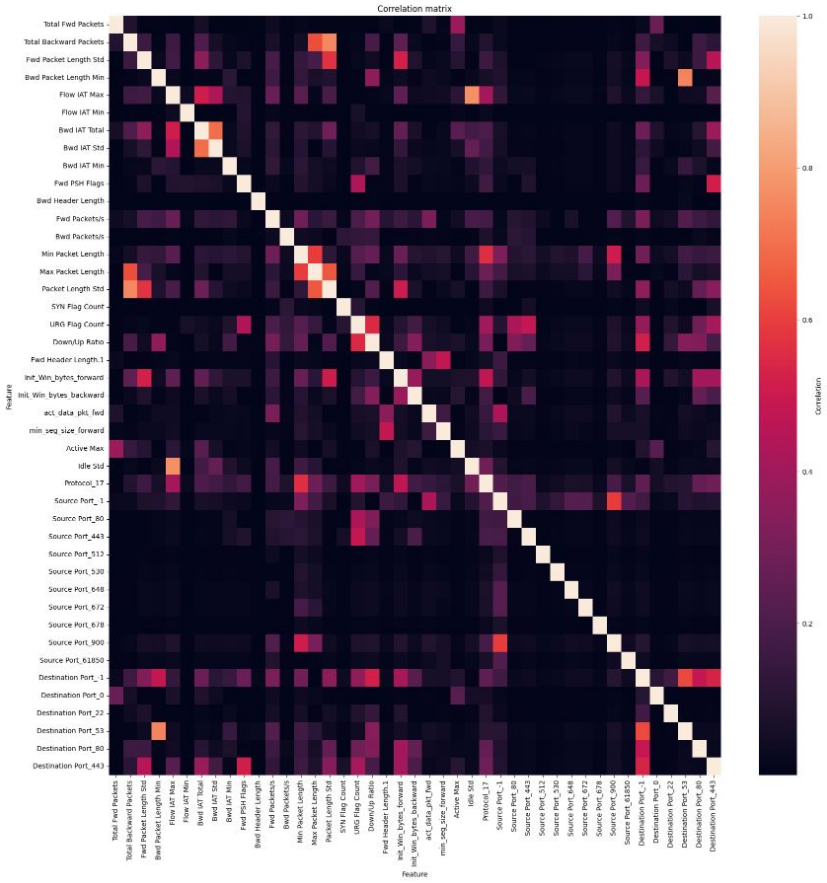


Figure 23

This is confirmed by the high loading scores (Figure 25) of said features in the Principal Component (PC) Analysis. The chosen number of principal components is 18. This decision was based on the absence of an elbow point within the figure 24. Instead, we followed the rule of positioning the number of PCs immediately after 67% of the total variance.

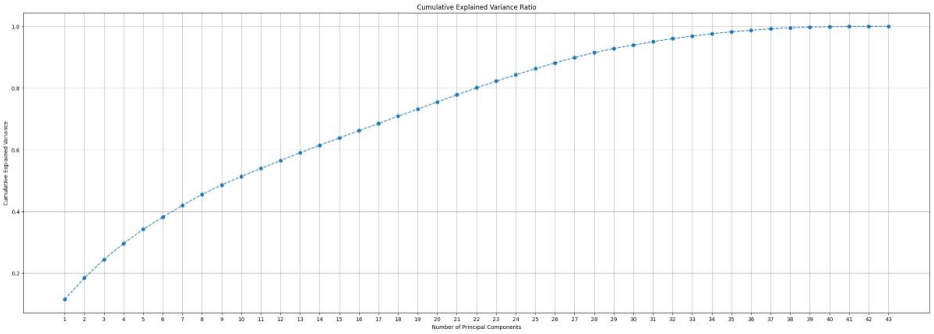


Figure 24

To get a visual representation of the dataset we can refer to yet another technique called tSNE. By plotting the projections of the data points onto a 2D plane, the differences between the attacks and benign traffic are made evident. It is also possible to spot numerous overlappings between labels 10 and 11. This problem will be addressed in the next sections.

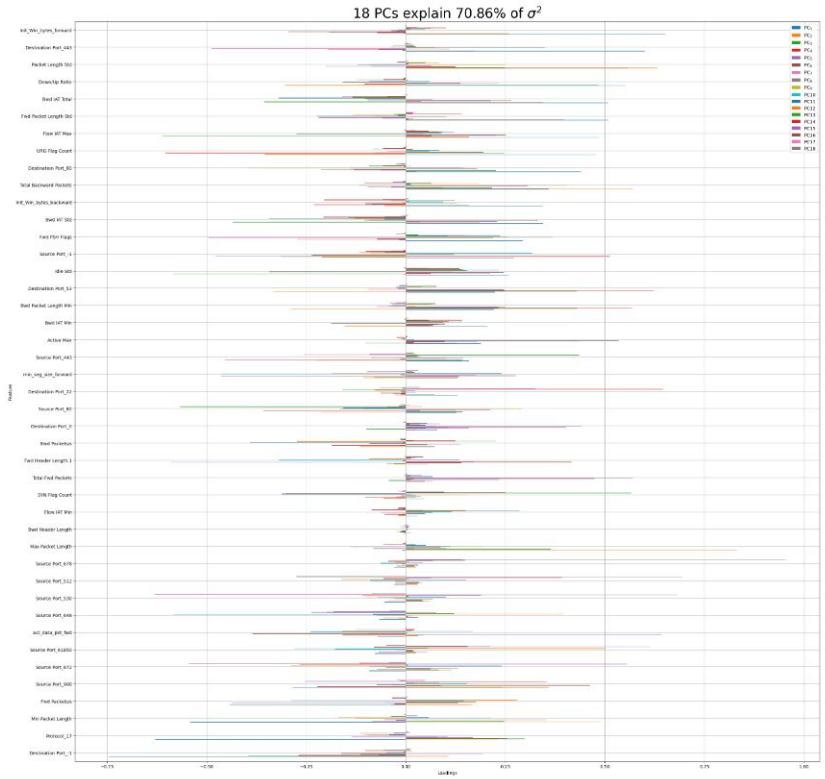


Figure 25

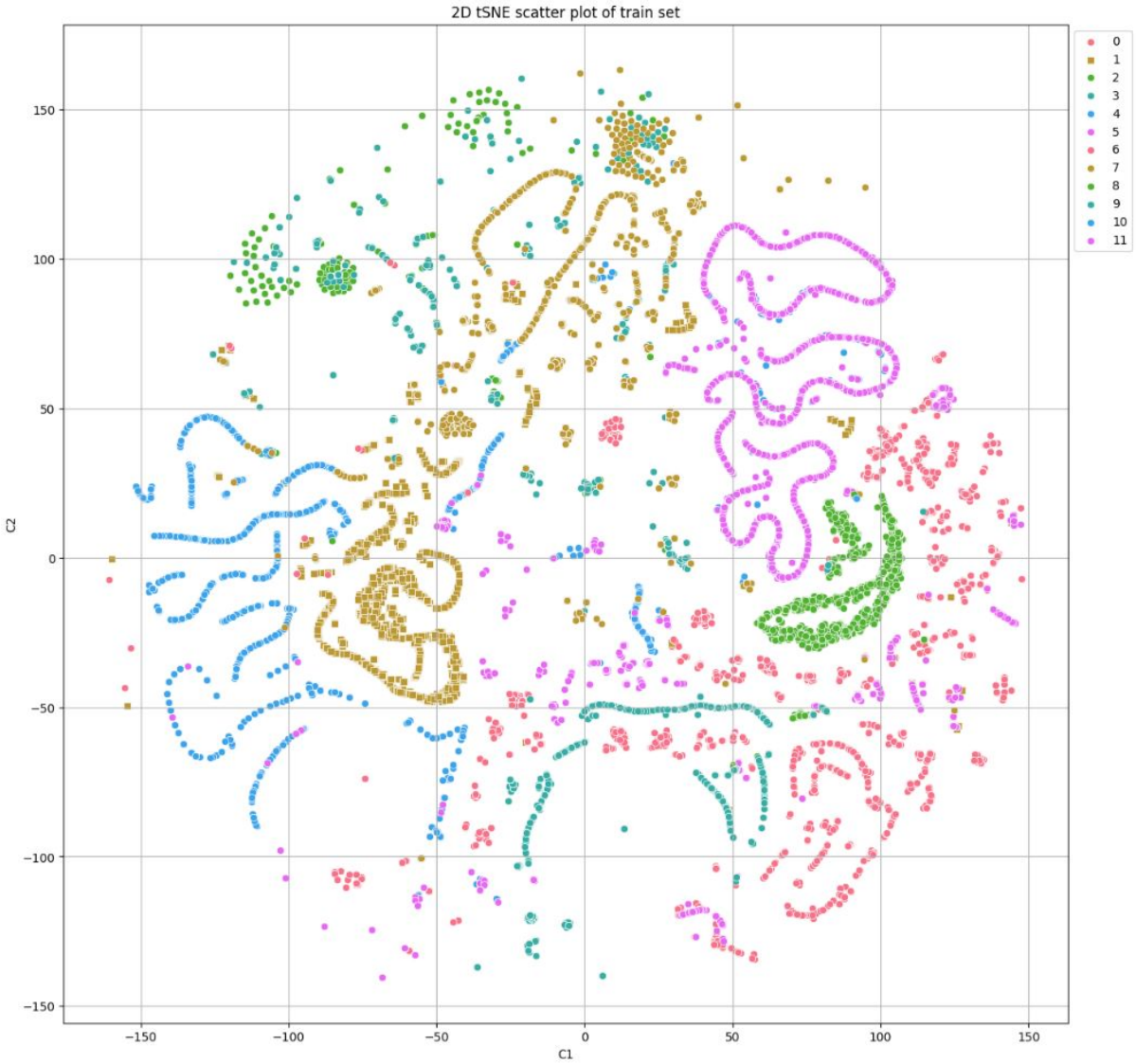


Figure 26: tSNE of train set, squares are benign flows

4 Supervised learning – classification

The primary aim of supervised learning is to teach a model to map input data to corresponding output labels by providing it with a labeled training dataset. In this learning paradigm, the algorithm learns from examples in the training data where the correct output is known. The model's goal is to generalize its learning to make accurate predictions or classifications on new, unseen data.
The second section involves categorizing the data flows based on identified attacks through a supervised classification approach. The ground truth dataset contains names corresponding to different types of attacks.
Divided the dataset into training and testing sets, the training set is used to train the model, while the testing set is reserved for evaluating the model's performance on unseen data.
We select 4 suitable machine learning model that differ in the type of algorithm, nature of output, handling non-linearity, etc. and evaluate the performance on both training and test set.
Performance are high and similar, it suggests a well-generalized model that does not incur in overfitting either underfitting.
In the process of developing and fine-tuning our machine learning model, we employed a comprehensive approach known as hyperparameter tuning. This involved systematically searching for the optimal combination of hyperparameter values, which are external configurations influencing the behavior of the model.
To efficiently explore the hyperparameter space, we utilized a technique called grid search. This approach involves defining a grid of hyperparameter values, systematically testing each combination, and evaluating the model's performance for each configuration.
Usually, we only have a vague idea of the best hyperparameters and thus we want to narrow our search through:

- **Validation Curve**: Graphical representation of the model's performance on both training and validation datasets across a range of key hyperparameter values.
- **Random Search**: Using Scikit-Learn's **RandomizedSearchCV** method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing cross-validation with each combination of values.

Random search and validation curve allowed us to narrow down the range for each hyperparameter. Now that we know where to concentrate our search, we can explicitly specify every combination of settings to try. We do this with **GridSearchCV** making another grid based on the best values provided by random search.
To obtain robust and reliable performance estimates for each hyperparameter combination, we integrated the stratified k-fold cross-validation, a variation of the traditional k-fold cross-validation. This method that ensures to maintain the same class distribution in each fold, into our random and grid search process.
This is particularly important when dealing with imbalanced datasets, where certain classes may be underrepresented.
Dividing our dataset into k subsets (folds), training the model on k-1 folds, and evaluating it on the remaining fold. This process was repeated k times, with each fold serving as the validation set exactly once.
By using cross-validation, we aimed to mitigate the risk of overfitting to a particular subset of the data, ensuring that our model's performance estimates were more representative of its generalization ability.
Throughout the random and grid search and cross-validation process, the benchmark for comparing different hyperparameter configurations is the F1 score macro-averaged, it is computed by averaging the F1 scores of individual classes, giving equal weight to each class regardless of its size.

$$F1_{\text{macro-avg}} = \frac{1}{N} \sum_{i=1}^{N} \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

where:

- N is the total number of classes.
- $F1_i$ is the F1 score for class i.
- Precision_i is the precision for class i.
- Recall_i is the recall for class i.

The precision and recall for each class are defined as:

$$\text{Precision}_i = \frac{\text{True Positives}_i}{\text{True Positives}_i + \text{False Positives}_i}$$

$$\text{Recall}_i = \frac{\text{True Positives}_i}{\text{True Positives}_i + \text{False Negatives}_i}$$

The F1 score macro-averaged is a metric used in multiclass classification to assess the overall performance of a model by considering the harmonic mean of precision and recall across all classes, there are several reasons for its importance:

- **Equal Weighting of Classes:** The macro-averaged F1 score assigns equal importance to each class. This is particularly important when all classes are considered equally important, and you want to avoid giving more weight to the performance on larger classes.
- **Global Assessment:** It provides a global assessment of the model's ability to balance precision and recall across all classes. It gives you a comprehensive understanding of how well the model is performing on average, irrespective of class sizes.
- **Diagnostic Tool:** If the macro-averaged F1 score is low, it indicates that the model is struggling with one or more classes, prompting further investigation into potential issues.
- **Comparability Across Models:** When comparing multiple models or tuning hyperparameters, it provides a consistent metric that allows fair comparisons across different experiments and configurations.

Upon completing the grid search and stratified cross-validation iterations, we identified the set of hyperparameters that yielded the best average performance across all folds. This set represents the configuration that optimized our model's predictive capabilities.
In our effort to enhance the effectiveness of our model, we propose incorporating ensemble learning to enhance our results. This strategic approach involves combining the predictions of multiple fine-tuned models to create a more robust and accurate final prediction. Below are key points explaining the rationale behind this decision:

- We have fine-tuned multiple models with varied hyperparameters, algorithms, or features, each exploring different facets of the data and learning patterns. This diversity in models can be harnessed through ensemble methods.
- The ensemble's collective decision-making can often outperform individual models, especially when each model contributes unique insights. This collaborative approach tends to yield more accurate predictions.
- In the event that one of the fine-tuned models underperforms on specific instances or scenarios, the ensemble approach provides a safety net, mitigating the impact of individual model weaknesses.

The final prediction is determined through a majority vote among the four models. In the event of a tie, we consider a weighted voting scheme, where each model's prediction has a certain influence on the final decision based on its assigned weight. The weight in the model's vote is derived from its performance during tuning, specifically the F1 score macro-averaged obtained from the best hyperparameters combination during grid search.

4.1 Random Forest

Default Parameter Configuration

```
Classification Report (Training):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4526
           1       0.99      0.94      0.97      4295
           2       0.84      0.94      0.89      4742
           3       0.85      0.62      0.72      4729
           4       0.99      0.97      0.98      4664
           5       0.99      1.00      0.99       789
           6       1.00      1.00      1.00      4787
           7       0.79      0.95      0.86      4776
           8       0.97      0.48      0.64      4384
           9       0.65      0.98      0.78      4209
          10       0.63      0.74      0.68      4701
          11       0.69      0.56      0.62      4789

    accuracy                           0.84     51391
   macro avg       0.86      0.85      0.84     51391
weighted avg       0.86      0.84      0.83     51391

Classification Report (Test):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1132
           1       0.98      0.90      0.94      1074
           2       0.85      0.94      0.89      1186
           3       0.83      0.61      0.70      1182
           4       0.98      0.97      0.97      1166
           5       0.88      0.97      0.92       197
           6       1.00      1.00      1.00      1197
           7       0.76      0.94      0.84      1194
           8       0.91      0.45      0.60      1096
           9       0.62      0.95      0.75      1052
          10       0.54      0.66      0.60      1175
          11       0.58      0.46      0.51      1197

    accuracy                           0.81     12848
   macro avg       0.83      0.82      0.81     12848
weighted avg       0.82      0.81      0.80     12848
```

Figure 27: Random Forest Default Parameter Configuration

```
Confusion Matrix (Test):
[[1131    0    0    0    0    0    1    0    0    0    0    0]
 [   1  966    0   11   10   25    1   58    0    2    0    0]
 [   0    0 1109   63    1    0    0   13    0    0    0    0]
 [   0    6  193  720    4    0    0  259    0    0    0    0]
 [   0    0    0   12 1131    0    0   18    0    0    5    0]
 [   2    4    0    0    0  191    0    0    0    0    0    0]
 [   0    0    0    0    0    0 1197    0    0    0    0    0]
 [   1    5    4   57    4    0    0 1123    0    0    0    0]
 [   0    0    0    0    0    0    0    0  488  608    0    0]
 [   0    0    0    0    0    0    0    0   48 1004    0    0]
 [   0    0    0    0    1    0    0    1    0    0  776  397]
 [   0    0    0    0    3    0    0    0    0    0  648  546]]
```

Figure 28: Random Forest Default Parameter Configuration Confusion Matrix

Validation Curve

n_estimators: In a Random Forest algorithm, the n_estimators hyperparameter specifies the number of trees in the forest. Each tree in the forest is constructed using a different random subset of the training data, and the final prediction is an aggregate of the predictions made by individual trees.

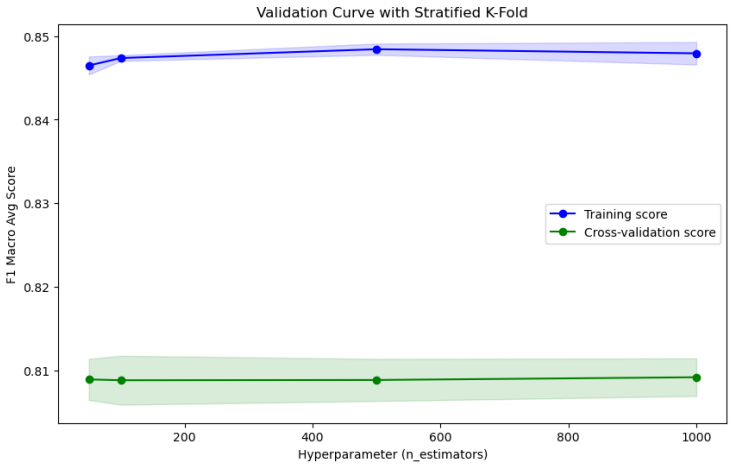


Figure 29

max_depth: This hyperparameter controls the maximum depth of each individual decision tree in the forest. The depth of a tree refers to the length of the longest path from the root node to a leaf node.

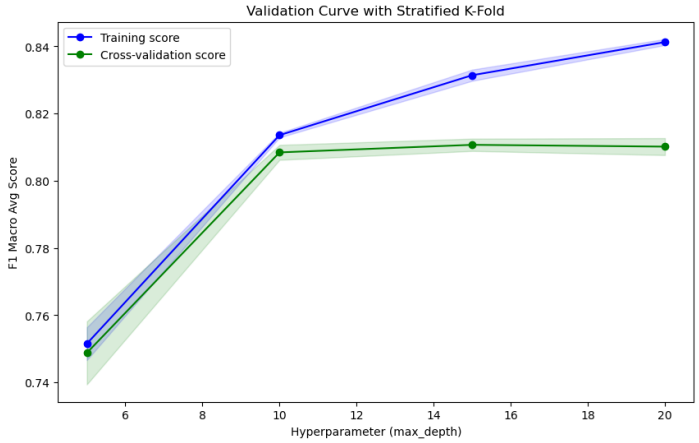


Figure 30

Randomized Search

```
RandomizedSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
                   estimator=RandomForestClassifier(), n_iter=100, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [15, 20, 25, 30, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 500, 800, 1000]},
                   random_state=42,
                   scoring=make_scorer(f1_score, average=macro))
```

Figure 31: Random Forest Randomized Search

Best Parameter Randomized Search

```
Best parameters values for Randomized Search:
 {'n_estimators': 800, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 20, 'bootstrap': True}
```

Figure 32: Best Parameter Randomized Search

Grid Search

```
GridSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
             estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_depth': [18, 20, 22],
                         'max_features': ['auto'], 'min_samples_leaf': [1, 2],
                         'min_samples_split': [5, 6, 7],
                         'n_estimators': [750, 800, 850, 900]},
             scoring=make_scorer(f1_score, average=macro))
```

Figure 33: Random Forest Grid Search

Best Parameter Configuration

```
Best parameters values for Grid Search:
 {'bootstrap': True, 'max_depth': 18, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 7, 'n_estimators': 800}
Weight of this model:
 0.811445179110929
```

Figure 34: Random Forest Best Parameter Configuration

4.2 Support Vector Machines

Default Parameter Configuration

Classification Report (Training):

	precision	recall	f1-score	support
0	0.96	0.99	0.97	4526
1	0.98	0.91	0.94	4295
2	0.83	0.94	0.88	4742
3	0.73	0.51	0.60	4729
4	0.98	0.78	0.87	4664
5	0.84	0.72	0.78	789
6	0.99	0.99	0.99	4787
7	0.71	0.89	0.79	4776
8	1.00	0.42	0.60	4384
9	0.62	1.00	0.77	4209
10	0.48	0.91	0.63	4701
11	0.56	0.15	0.24	4789
accuracy			0.77	51391
macro avg	0.81	0.77	0.75	51391
weighted avg	0.80	0.77	0.75	51391

Classification Report (Test):

	precision	recall	f1-score	support
0	0.96	0.98	0.97	1132
1	0.98	0.89	0.93	1074
2	0.85	0.94	0.89	1186
3	0.75	0.53	0.62	1182
4	0.97	0.77	0.86	1166
5	0.81	0.74	0.77	197
6	0.99	0.99	0.99	1197
7	0.70	0.88	0.78	1194
8	1.00	0.42	0.59	1096
9	0.62	1.00	0.76	1052
10	0.47	0.92	0.62	1175
11	0.52	0.13	0.21	1197
accuracy			0.77	12848
macro avg	0.80	0.77	0.75	12848
weighted avg	0.80	0.77	0.75	12848

Figure 35: SVM Default Parameter Configuration

Confusion Matrix (Test):

```
[[1115    0    0    0    0    9    7    0    0    1    0    0]
 [   6  955    0    8   13   26    4   61    0    0    0    1]
 [   0    0 1118   49    1    0    0   18    0    0    0    0]
 [   0    4  192  627    7    0    0  352    0    0    0    0]
 [   0    0    0   23  898    0    0   20    0    0  180   45]
 [  40    0    0    0    0  146    0    0    0   11    0    0]
 [   0    6    0    0    0    0 1189    0    0    0    2    0]
 [   0    7    3  127    1    0    1 1055    0    0    0    0]
 [   1    0    0    0    0    0    0    0  458  637    0    0]
 [   0    0    0    0    0    0    1    0    0 1051    0    0]
 [   0    0    0    0    0    0    0    1    0    0 1078   96]
 [   0    0    0    0    2    0    0    0    0    0 1042  153]]
```

Figure 36: SVM Default Parameter Configuration Confusion Matrix

Validation Curve

kernel: Defines the type of decision boundary used by the SVM model. SVMs are capable of performing both linear and non-linear classification by transforming the input features into a higher-dimensional space.

- **rbf**: Also known as the Gaussian Kernel, allows SVMs to model complex decision boundaries in high-dimensional spaces, making it especially effective when the data is not linearly separable.
- **linear**: Corresponds to a linear decision boundary.
- **poly**: Introduces polynomial features to capture non-linear relationships.

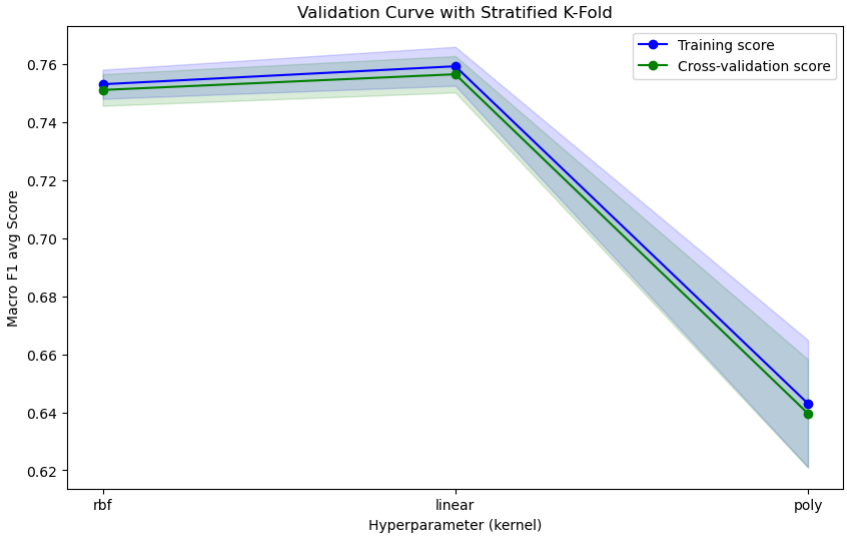


Figure 37

In selecting the kernel for our SVM model, we opt for the Radial Basis Function (RBF) kernel instead of the linear or polynomial kernel based on computational efficiency. The linear kernel involves computing the dot product between each pair of input features, resulting in a time complexity that scales linearly with the number of data points. Given our large dataset, the computational burden of the linear kernel was significant and resulted in impractical training times.
On the other hand, the RBF kernel, being a non-linear kernel, allows us to capture more complex relationships in the data without the same computational overhead. Its time complexity is more favorable for large datasets, making it a suitable choice for our specific circumstances.

C: Regularization parameter, often referred to as the "cost" parameter. It controls the balance between achieving a smooth decision boundary and accurately classifying the training data.

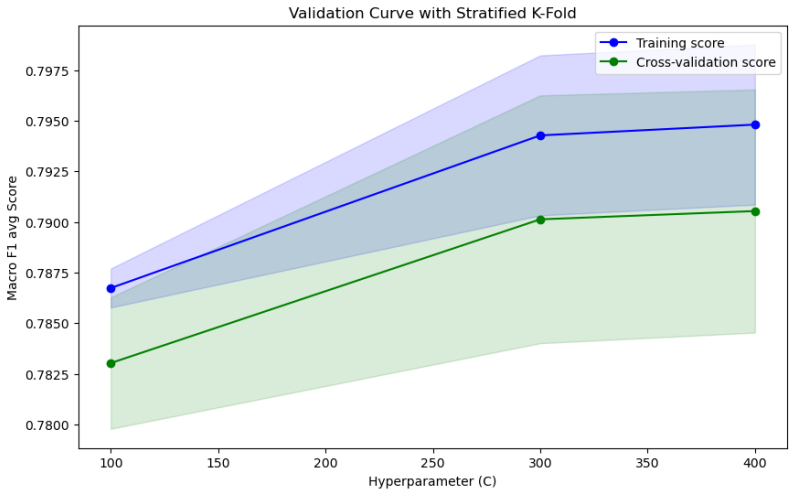


Figure 38

gamma: it has a significant impact on the shape of the decision boundary, especially when using non-linear kernels. It determines the influence of a single training example, and the choice of gamma can affect the model's performance and generalization.

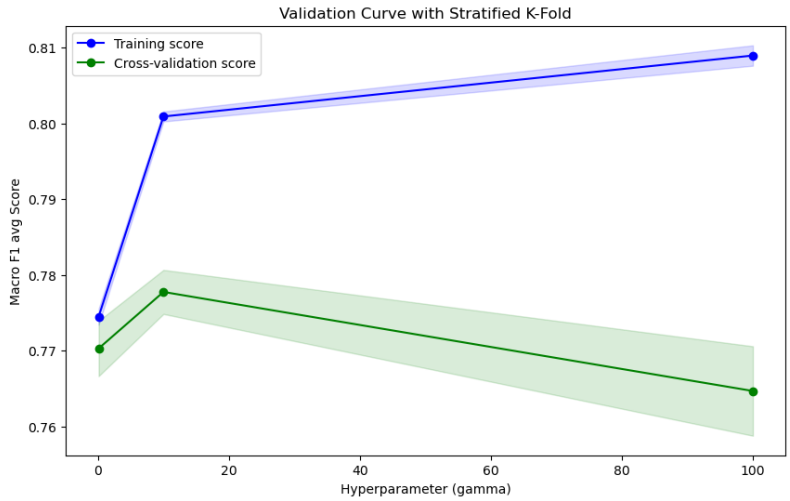


Figure 39

Randomized Search

```
RandomizedSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
                   estimator=SVC(), n_iter=4, n_jobs=-1,
                   param_distributions={'C': [300, 375, 450],
                                        'gamma': [5, 10, 15]},
                   random_state=42,
                   scoring=make_scorer(f1_score, average=macro))
```

Figure 40: SVM Randomized Search

Best Parameter Randomized Search

```
Best parameters values for Randomized Search:
 {'gamma': 5, 'C': 300}
```

Figure 41: Best Parameter Randomized Search

Grid Search

```
GridSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
             estimator=SVC(), n_jobs=-1,
             param_grid={'C': [300, 325, 350], 'gamma': [3, 5, 7]},
             scoring=make_scorer(f1_score, average=macro))
```

Figure 42: SVM Grid Search

Best Parameter Configuration

```
Best parameters values for Grid Search:
 {'C': 350, 'gamma': 3}
Weight of this model:
 0.7920232972101876
```

Figure 43: SVM Best Parameter Configuration

4.3 k-Nearest Neighbor

Default Parameter Configuration

Classification Report (Training):

	precision	recall	f1-score	support
0	0.99	1.00	0.99	4526
1	0.97	0.92	0.95	4295
2	0.84	0.91	0.87	4742
3	0.81	0.62	0.70	4729
4	0.98	0.97	0.98	4664
5	0.89	0.96	0.92	789
6	1.00	1.00	1.00	4787
7	0.78	0.94	0.86	4776
8	0.69	0.67	0.68	4384
9	0.67	0.69	0.68	4209
10	0.63	0.60	0.61	4701
11	0.62	0.65	0.63	4789
accuracy			0.82	51391
macro avg	0.82	0.83	0.82	51391
weighted avg	0.82	0.82	0.82	51391

Classification Report (Test):

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1132
1	0.97	0.90	0.93	1074
2	0.85	0.92	0.88	1186
3	0.82	0.62	0.70	1182
4	0.97	0.96	0.97	1166
5	0.85	0.94	0.89	197
6	0.99	1.00	1.00	1197
7	0.76	0.94	0.84	1194
8	0.68	0.66	0.67	1096
9	0.65	0.67	0.66	1052
10	0.52	0.52	0.52	1175
11	0.53	0.53	0.53	1197
accuracy			0.79	12848
macro avg	0.80	0.80	0.80	12848
weighted avg	0.80	0.79	0.79	12848

Figure 44: k-NN Default Parameter Configuration

Confusion Matrix (Test):

```
[[1119    2    0    0    0    7    2    0    0    2    0    0]
 [   3  962    1   10    8   27    4   57    0    1    0    1]
 [   0    0 1086   86    1    0    0   13    0    0    0    0]
 [   0    4  182  728    6    0    0  262    0    0    0    0]
 [   0    2    0   17 1123    0    0   16    0    0    2    6]
 [   7    4    0    0    0  186    0    0    0    0    0    0]
 [   0    3    0    0    0    0 1193    0    0    0    1    0]
 [   0   12    3   51   11    0    0 1117    0    0    0    0]
 [   0    0    0    0    0    0    0    0  722  374    0    0]
 [   0    0    0    0    0    0    1    0  346  705    0    0]
 [   0    0    0    0    1    0    0    1    0    0  612  561]
 [   0    0    0    0    2    0    0    0    0    0  562  633]]
```

Figure 45: k-NN Default Parameter Configuration Confusion Matrix

Validation Curve

n_neighbors: Represents the number of neighbors used to make predictions. It is a crucial parameter in k-NN because it determines the size of the neighborhood around a data point, and thus, it affects the smoothness or granularity of the decision boundary.

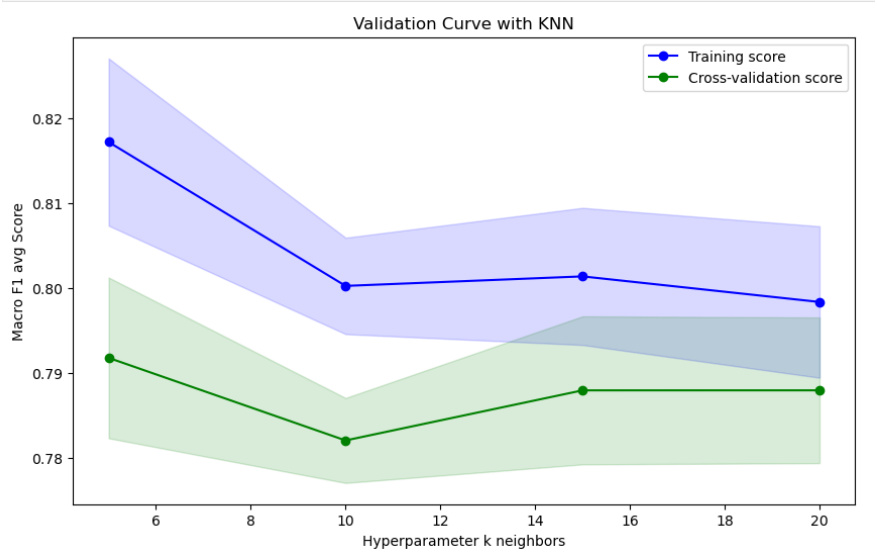


Figure 46

Randomized Search

```
RandomizedSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
                   estimator=KNeighborsClassifier(), n_iter=20, n_jobs=-1,
                   param_distributions={'metric': ['euclidean', 'manhattan',
                                                   'chebyshev'],
                                        'n_neighbors': array([ 3,  7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47])},
                   random_state=42,
                   scoring=make_scorer(f1_score, average=macro))
```

Figure 47: k-NN Randomized Search

Best Parameter Randomized Search

```
Best parameters values for Randomized Search:
 {'n_neighbors': 3, 'metric': 'manhattan'}
```

Figure 48: Best Parameter Randomized Search

Grid Search

```
GridSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
             estimator=KNeighborsClassifier(metric='manhattan'), n_jobs=-1,
             param_grid={'n_neighbors': [3, 5, 7, 9]},
             scoring=make_scorer(f1_score, average=macro))
```

Figure 49: k-NN Grid Search

Best Parameter Configuration

```
Best parameters values for Grid Search:
 {'n_neighbors': 3}
Weight of this model:
 0.8039361338011974
```

Figure 50: k-NN Best Parameter Configuration

4.4 Logistic Regression

Default Parameter Configuration

Classification Report (Training):

	precision	recall	f1-score	support
0	0.94	0.95	0.94	4526
1	0.98	0.90	0.94	4295
2	0.83	0.94	0.88	4742
3	0.76	0.50	0.61	4729
4	0.97	0.89	0.93	4664
5	0.64	0.65	0.64	789
6	0.99	1.00	0.99	4787
7	0.70	0.90	0.79	4776
8	1.00	0.42	0.60	4384
9	0.62	1.00	0.77	4209
10	0.50	0.76	0.60	4701
11	0.54	0.30	0.39	4789
accuracy			0.77	51391
macro avg	0.79	0.77	0.76	51391
weighted avg	0.80	0.77	0.76	51391

Classification Report (Test):

	precision	recall	f1-score	support
0	0.95	0.93	0.94	1132
1	0.98	0.89	0.93	1074
2	0.85	0.94	0.89	1186
3	0.77	0.53	0.63	1182
4	0.97	0.88	0.92	1166
5	0.62	0.71	0.66	197
6	0.99	0.99	0.99	1197
7	0.70	0.89	0.79	1194
8	1.00	0.42	0.59	1096
9	0.62	1.00	0.76	1052
10	0.49	0.75	0.59	1175
11	0.52	0.29	0.38	1197
accuracy			0.77	12848
macro avg	0.79	0.77	0.76	12848
weighted avg	0.80	0.77	0.76	12848

Figure 51: Logistic Regression Default Parameter Configuration

Confusion Matrix (Test):

```
[[1058    4    1    0    0   60    7    0    1    1    0    0]
 [   5  954    0    9   13   25    6   60    0    1    0    1]
 [   0    0 1118   49    1    0    0   18    0    0    0    0]
 [   0    4  192  627    8    0    0  351    0    0    0    0]
 [   0    0    0   16 1021    0    0   19    0    0   76   34]
 [  49    0    0    0    0  139    0    0    0    9    0    0]
 [   0    4    0    0    0    0 1191    0    0    0    2    0]
 [   0    6    4  110    9    0    0 1064    0    0    1    0]
 [   0    0    0    0    0    0    0    0  459  637    0    0]
 [   0    0    0    0    0    0    1    0    0 1051    0    0]
 [   0    0    0    0    1    0    0    1    0    0  886  287]
 [   0    0    0    0    3    0    0    0    0    0  843  351]]
```

Figure 52: Logistic Regression Default Parameter Configuration Confusion Matrix

Validation Curve

multi_class: Specifies how the model should handle multi-class classification problems. It defines the strategy used to extend binary classification to the multi-class case.

- **One-vs-Rest** For each class, it fits a binary classification model considering that class as one of the positive classes and the rest as the negative class
- **Multinomial** It uses the cross-entropy loss for multi-class classification and fits a single model for all classes simultaneously, considering them jointly.

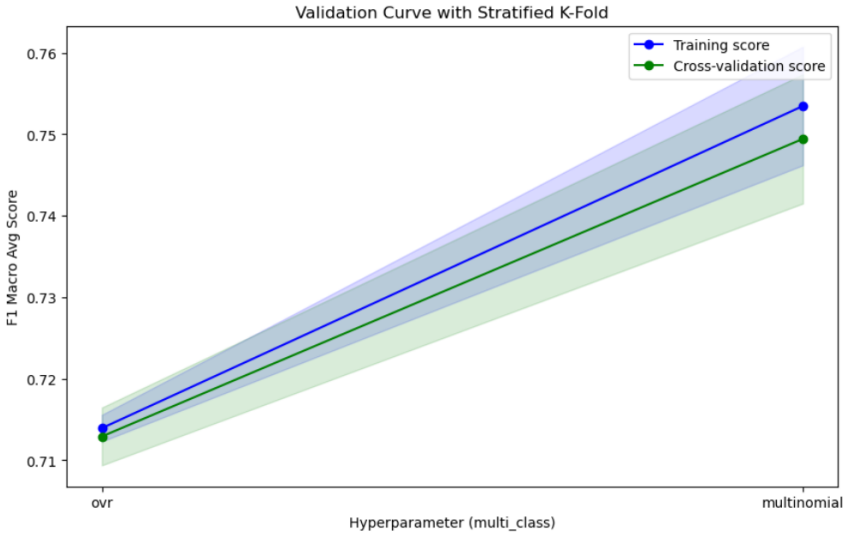


Figure 53

We encountered convergence issues while training the logistic regression model. Despite increasing the maximum number of iterations, the optimization process failed to converge using solvers like lbfgs and penalty L1, so the model did not learn the optimal set of parameters during training. Models with too many features or a high degree of complexity relative to the size of the dataset may struggle to converge, we experiment alternative solvers (e.g., 'newton-cg', 'sag', 'saga') to see if a different solver resolves the convergence issue. However, it's important to note that not all combinations of solvers, multi-class and penalty settings are possible or supported due to algorithmic constraints.

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Figure 54

penalty: Regularization term added to the cost function during the model training process. Regularization is a technique used to prevent overfitting and improve the generalization performance of a model by discouraging overly complex models with large coefficients.

- **L1 Penalty**: The L1 penalty adds the absolute values of the coefficients to the cost function.
- **L2 Penalty**: The L2 penalty adds the squared values of the coefficients to the cost function.

solver: Specifies the optimization algorithm to be used during the training process to find the coefficients that minimize the logistic loss.

- **newton-cg**: It is based on Newton's method with conjugate gradient.
- **sag**: It performs a stochastic gradient descent with an averaging procedure.

- **saga**: It is an extension of the SAG solver with support for both L1 and L2 regularization

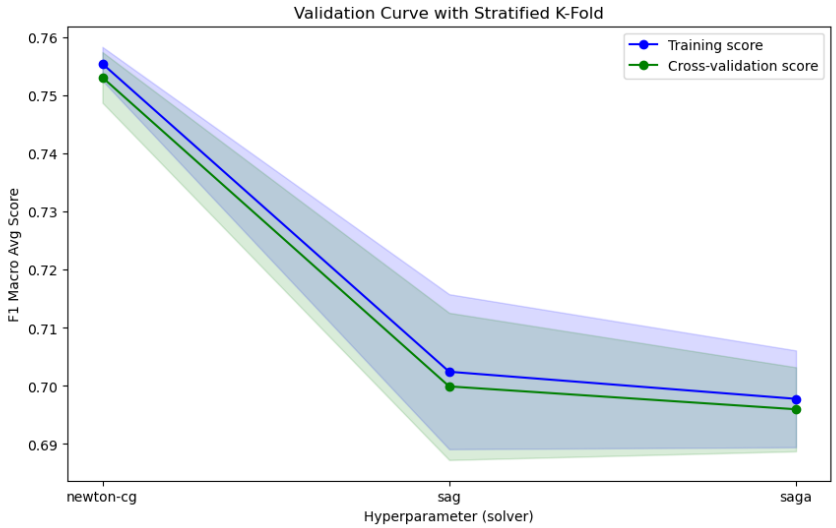


Figure 55: Multinomial + L2

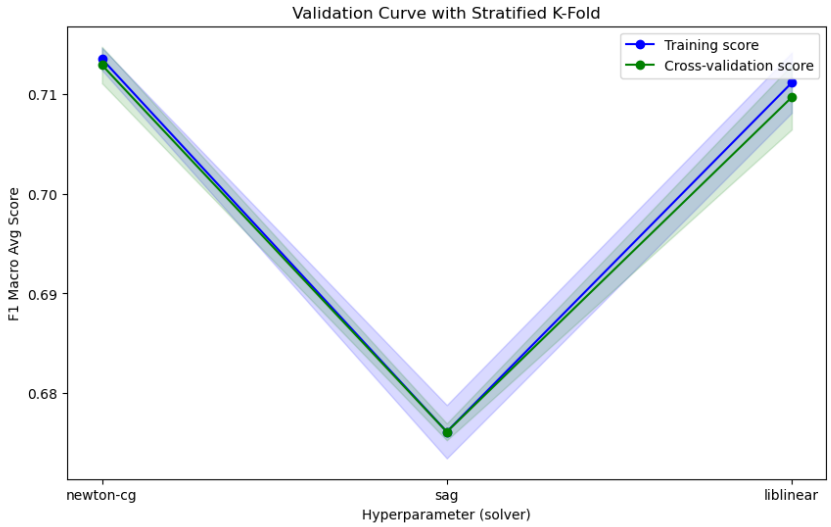


Figure 56: OvR + L2

In this combination, we encountered convergence issues, as evidenced by the figure below; the score is significantly lower compared to other parameter configurations.

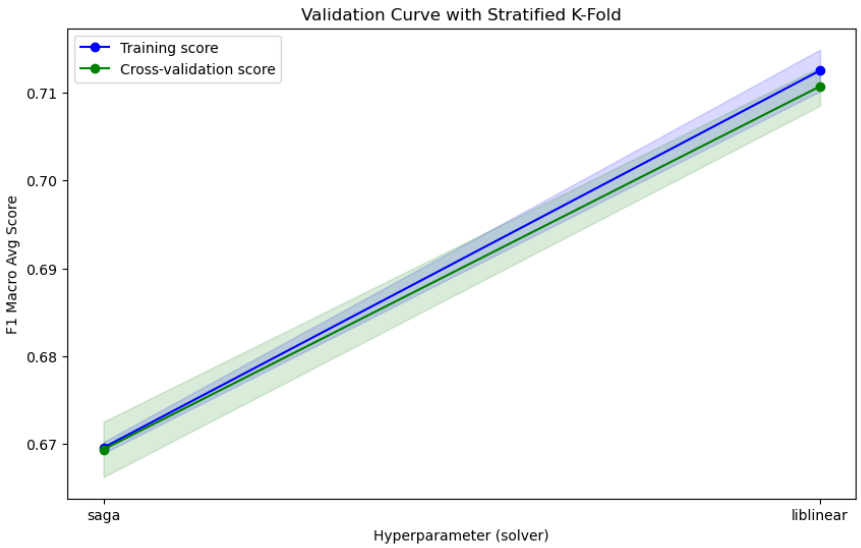


Figure 57: OvR + L1

C: It is the inverse of regularization strength. Regularization is a technique used to prevent overfitting by penalizing large coefficients. The default values for multi_class and penalty parameters are used.

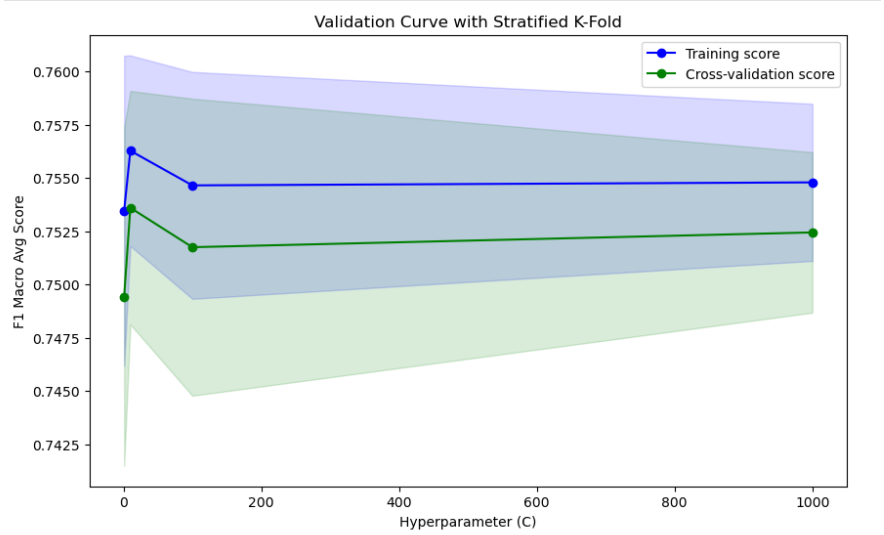


Figure 58

Random Search

This model demands a considerable computational effort, given the previous results, therefore, we decide to use the 'newton-cg' solver and the 'multinomial' approach to narrow down the hyperparameter space and to increase the number of iterations.
The aim now is to find the optimal value for the parameter 'C' with a grid search tecnique.

```
RandomizedSearchCV(cv=StratifiedKFold(n_splits=4, random_state=42, shuffle=True),
        estimator=LogisticRegression(multi_class='multinomial',max_iter=1000),
        n_iter=4, n_jobs=-1,
        param_distributions={'C': [100, 10, 1.0, 0.1],
                'solver': ['newton-cg', 'sag']},
        random_state=42,
        scoring=make_scorer(f1_score, average=macro))
```

Figure 59: Logistic Regression Random Search

Best Parameter Configuration

Despite the challenges encountered with execution times, particularly during the grid search, we managed to obtain an algorithm that converges during the training process with one thousand iterations and using the 'multinomial' approach.

```
Best parameters values for Random Search:
 {'solver': 'newton-cg', 'C': 100}
Weight of this model:
 0.757771587384698
```

Figure 60: Logistic Regression Best Parameter Configuration

4.5 Ensemble Classifier

Let's now combine the results of our configured models. The final prediction will be determined by a majority vote. In case of a tie, we will consider the label for which there is the highest individual model weight or the sum of weights for models predicting that label.

Model	Weight
Random Forest	0.811
Support Vector Machine	0.792
k-Nearest Neighbor	0.804
Logistic Regression	0.758

Table 2: Models and their weights

Classification Report (Training):

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4526
1	0.99	0.92	0.96	4295
2	0.84	0.94	0.89	4742
3	0.86	0.59	0.70	4729
4	0.98	0.97	0.98	4664
5	0.93	1.00	0.96	789
6	1.00	1.00	1.00	4787
7	0.77	0.97	0.86	4776
8	0.97	0.47	0.63	4384
9	0.64	0.99	0.78	4209
10	0.60	0.67	0.63	4701
11	0.63	0.56	0.59	4789
accuracy			0.83	51391
macro avg	0.85	0.84	0.83	51391
weighted avg	0.84	0.83	0.82	51391

Classification Report (Test):

	precision	recall	f1-score	support
0	0.99	1.00	1.00	1132
1	0.99	0.90	0.94	1074
2	0.85	0.94	0.89	1186
3	0.86	0.58	0.69	1182
4	0.98	0.97	0.97	1166
5	0.87	0.98	0.92	197
6	1.00	1.00	1.00	1197
7	0.75	0.96	0.84	1194
8	0.94	0.44	0.59	1096
9	0.62	0.97	0.76	1052
10	0.54	0.62	0.57	1175
11	0.56	0.48	0.52	1197
accuracy			0.81	12848
macro avg	0.83	0.82	0.81	12848
weighted avg	0.82	0.81	0.80	12848

Figure 61: Ensemble Classification Report

Model	Performance
Random Forest	0.81
Support Vector Machine	0.79
k-Nearest Neighbor	0.80
Logistic Regression	0.76
Ensemble	0.81

Table 3: Models and their Average F1 Macro on Test Set

We proceeded to analyze where the ensemble goes wrong.
Specifically, we focused on cases where there was a tie pair-wise (i.e., among the 4 models, two by two, voting for different labels), and selected the samples where the prediction was subsequently incorrect.
A lot of imprecision in our model can be observed for samples belonging to labels 10 and 11 (Figure 61), both for individual models and the ensemble. They are not able to significantly improve the accuracy of predictions for these classes.
For many samples in these classes, there is indeed a tie in votes, as in the case of the sample with index 62920, which will be resolved by considering the votes of the most performing models during training.
In the next sections, the similarity between these classes will be thoroughly examined.
As shown in Figure 62, for example, in the case of the sample at index 54750, the predicted label is 8, whereas the ground truth is 9.

```
Sample index 62920, Label 11, Prediction 10
Sample index 54750, Label 9, Prediction 8
Sample index 41152, Label 7, Prediction 3
```

Figure 62: Notable Samples

Given the complexity of directly obtaining feature importance from the ensemble, we addressed this challenge by exploring which features do not contribute significantly to the model's ability to distinguish between labels.
Focusing on a specific case (Sample 54750) involving labels 9 and 8, we exclusively analyzed these two attacks in the dataset. To gain insights into feature importance, we conducted a random forest analysis with default parameters, selecting this model for its highest score on training among others.
The results, as illustrated in figure 63, highlight that certain features, such as the one-hot encoding features of the ports, are zero, likely because these ports are not utilized by these two attacks.
Additionally, we observe many low values for feature importance it suggests that the model relies less on those features for making predictions.
It might indicate that these feature are not strong discriminators between the classes. In other words, the information provided by these features does not significantly help the model distinguish between the two labels.
To confirm this hypothesis, let's plot boxplots for some features with low scores and compare them with one high score feature. (Figure 64)

```
Feature Importance in ascending order:
 Destination Port_443: 0.0
 Fwd Header Length.1: 0.0
 Source Port_80: 0.0
 URG Flag Count: 0.0
 SYN Flag Count: 0.0
 Packet Length Std: 0.0
 Source Port_443: 0.0
 Source Port_512: 0.0
 Source Port_530: 0.0
 min_seg_size_forward: 0.0
 Source Port_648: 0.0
 Source Port_672: 0.0
 Source Port_678: 0.0
 Source Port_900: 0.0
 Destination Port_22: 0.0
 Bwd Packet Length Min: 0.0
 Fwd Packet Length Std: 0.0
 Destination Port_53: 0.0
Fwd PSH Flags: 0.0
 Destination Port_80: 0.0
 Destination Port_-1: 4.6740956322509534e-05
 Max Packet Length: 6.510152154217189e-05
Init_Win_bytes_forward: 7.046631884191949e-05
Total Length of Fwd Packets: 8.045673221316954e-05
 Destination Port_0: 8.486697367361077e-05
 Min Packet Length: 8.587310289417919e-05
Protocol_17: 0.00011023994630338426
 Source Port_-1: 0.00017404300306749485
```

Figure 63: Feature Importance for Random Forest with Label 8 and 9

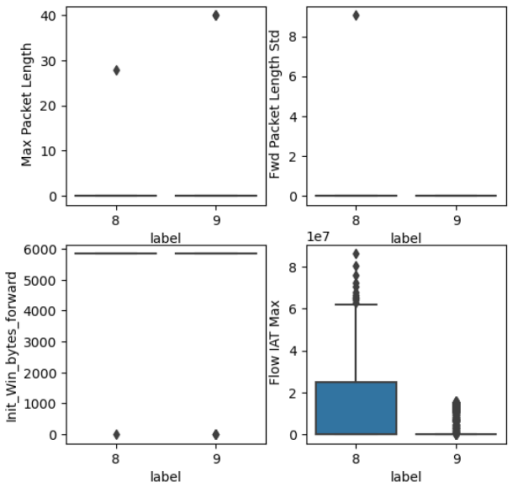


Figure 64: Boxplots For Label 8 and 9

The same approach was applied to labels 3 and 7.

```
Feature Importance in ascending order:
 Destination Port_53: 0.0
 Bwd Packet Length Min: 0.0
 Down/Up Ratio: 0.0
 Source Port_61850: 0.0
 Source Port_80: 0.0
 Source Port_443: 0.0
 Fwd Packet Length Std: 6.503143203045644e-06
Fwd PSH Flags: 8.113682880127953e-06
 Bwd IAT Min: 1.2929768276616036e-05
 SYN Flag Count: 1.4741421254234874e-05
 Destination Port_0: 2.1999596795338914e-05
 Destination Port_80: 2.342420335991465e-05
 Bwd Packets/s: 2.6073207133456813e-05
 Bwd Header Length: 3.263570558211809e-05
 Packet Length Std: 3.459369372030932e-05
 Bwd IAT Std: 4.0158108159364376e-05
Init_Win_bytes_forward: 4.8931697294809034e-05
 URG Flag Count: 5.430351332632074e-05
Protocol_17: 5.765162497470771e-05
 Idle Std: 6.601178638309936e-05
 Init_Win_bytes_backward: 6.662835001528519e-05
```

Figure 65: Feature Importance for Random Forest with Label 3 and 7

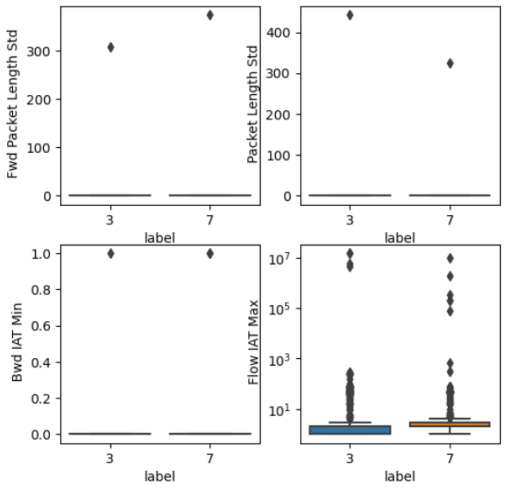


Figure 66: Boxplots For Label 3 and 7

5 Unsupervised learning – clustering

In this chapter we will group flows that produce similar/correlated/coordinated patterns. We decide to use three clustering algorithms: Kmeans, Gaussian Mixture and DBSCAN.
Dealing with real-world data is not a trivial task, in the ever-growing world of cyber attacks having a small set of already recognized threats can be limiting in terms of security analysis, defence systems development and so on. For this reason we developed an unsupervised classification solution, better known as clustering algorithms.
After taking into consideration various metrics, approaches to the problem and computational load we settled on three different algorithms: K-means, Gaussian Mixture Model and DBSCAN. Their substantial differences can give us an insight on the applicability of unclassified clustering algorithms to our problem.

5.1 K-means

K-means is a popular clustering algorithm used in machine learning and data mining. The goal of the K-means algorithm is to partition a set of data points into clusters such that each data point belongs to the cluster with the nearest mean. It is used in various scenarios across different domains where there is a need to group data points into distinct clusters based on their similarity or proximity.
In our case, we wanted to investigate if there are any notable differences between the labeled attack types and, most importantly, if there is some kind of more granular clustering inside of them.
Applying K-means with the default scikit-learn parameters and a number of clusters equal to the ground truth labels, commonly referred as GT, yields poor but still promising results. It is pretty clear that the number of clusters, K, has to be tuned to get better results in term of clustering error while also granting a certain level of variety in the cluster assinings. We will take into consideration the following hyper-parameters:

1. **n_clusters**: number of clusters to form;
2. **n_init**: number of initializations to perform;
3. **init**: best_centroids;

By looking at the plots 67, it is possible to note an elbow, this suggests that 50 clusters is probably a good compromise.

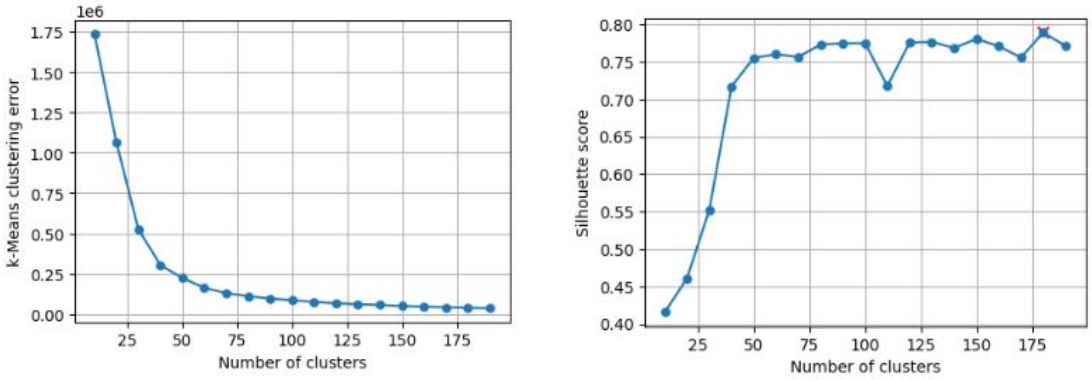


Figure 67

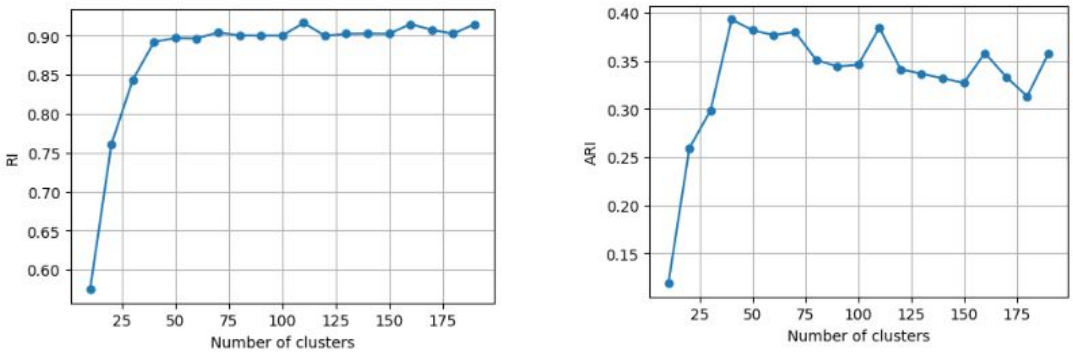


Figure 68

Another important parameter to tune when dealing with kmeans is the centroids initialisation array. As previously mentioned, KMeans works by assigning the data points to the nearest n-dimensional center. These points are called centroids. In our case there are 50 centers, with 43 components each. The algorithm works iteratively to assign each data point to the nearest centroid and then updates the centroids based on the mean of the points assigned to each cluster. This process continues until the centroids no longer change significantly, or until a specified number of iterations is reached. If the initial n-dimensional points are badly positioned the resulting cluster assignments can be worse than expected or even misleading in some cases. For this reason we tried one hundred random clusters initializations and selected the best in terms of Silhouette, Inertia, Rand Index and Ari.

```
k-Means with 50 clusters and best centroids
Size of each cluster:  [  68 1025 7585  489  142 3397 1638 1928 3496  156  165  811 3951 1201
  683 1311  373   65  231    3    1 4165    8 1049 1617  329 2905    1
  409  417 6084 6365    1    1 2134  117  454 1074    1 1060 2012    1
  512   23    7    3  117  509 4142    3]
k_means clustering error: 569696.29
Silhouette: 0.68
RI: 0.92
ARI: 0.4
```

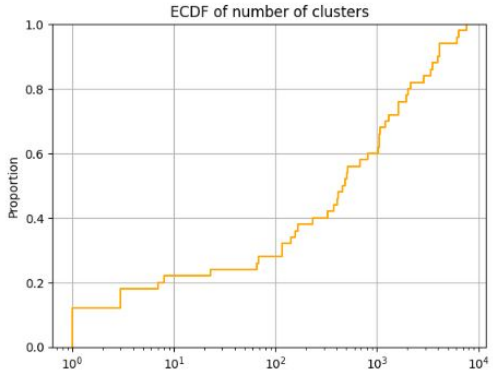


Figure 69

We now finally have our fine-tuned model, in the figures above the output produced with the metrics, the cluster cardinality and the ECDF of the latter for better clarity.
We can appreciate the homogeneous distribution of data points among the clusters, which indicates good centroid placement.
We can also see some low cardinality ones, meaning the presence of possible outliers in our dataset.
A better visualization can be achieved with the tSNE technique in figure 70. We can clearly

see the correct identification of the main clusters and the subdivision of some GT point into sub-clusters.

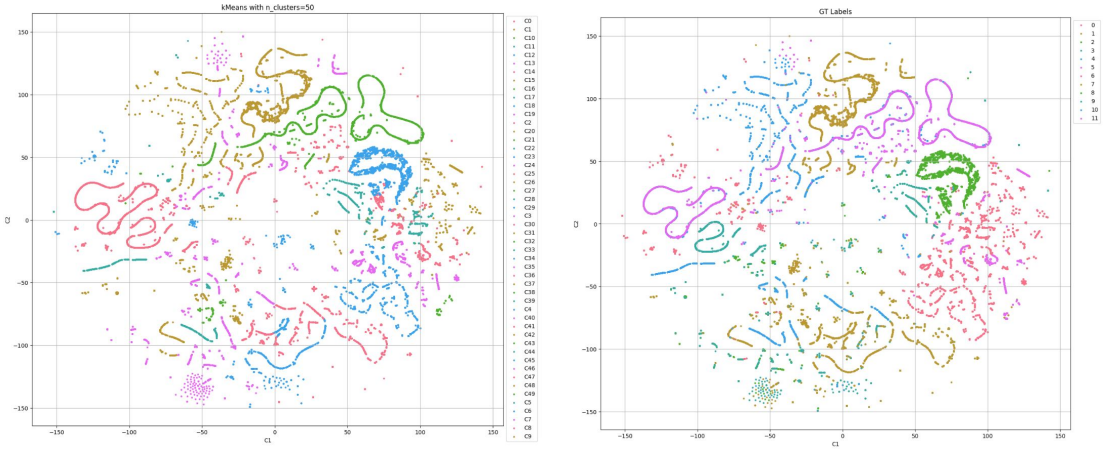


Figure 70

5.2 Gaussian Mixture

Gaussian Mixture Model is a probabilistic model composed of multiple Gaussian distributions, also known as components or clusters. Each Gaussian component is characterized by its mean, covariance matrix, and weight. The mean represents the center of the cluster, the covariance matrix describes the shape and orientation of the cluster, and the weight represents the importance of the cluster in the overall mixture. The first step is to determine the number of clusters to use in the model. In order to do it we have to look at the performance that this model has on our dataset by trying with different number of clusters, in particular by doing a silhouette analysis and an adjusted-rand index analysis. For the silhouette analysis we perform the gaussian mixture model with a number of cluster from 3 to 90, in three different steps as in Figure 71. We notice that from a certain point, around 70/75 clusters, the ongoing of the silhouette stops to increase, so we decide to look for a good ARI below 75 clusters (Figure 73). Comparing the two type of plots what we can notice is that for the silhouette we have the greatest pick with 72 clusters, on the contrary with the ARI we have the greatest pick with 22 clusters. The cons to choose 22 clusters is that the silhouette is really low, meanwhile taking 72 clusters it's the ARI to be low. So we decide to take a compromise: 38 clusters. This choice it's made because it has a good ARI and also a good silhouette, even if it's not the best. Moreover for this kind of problem we don't want to take a huge number of clusters.

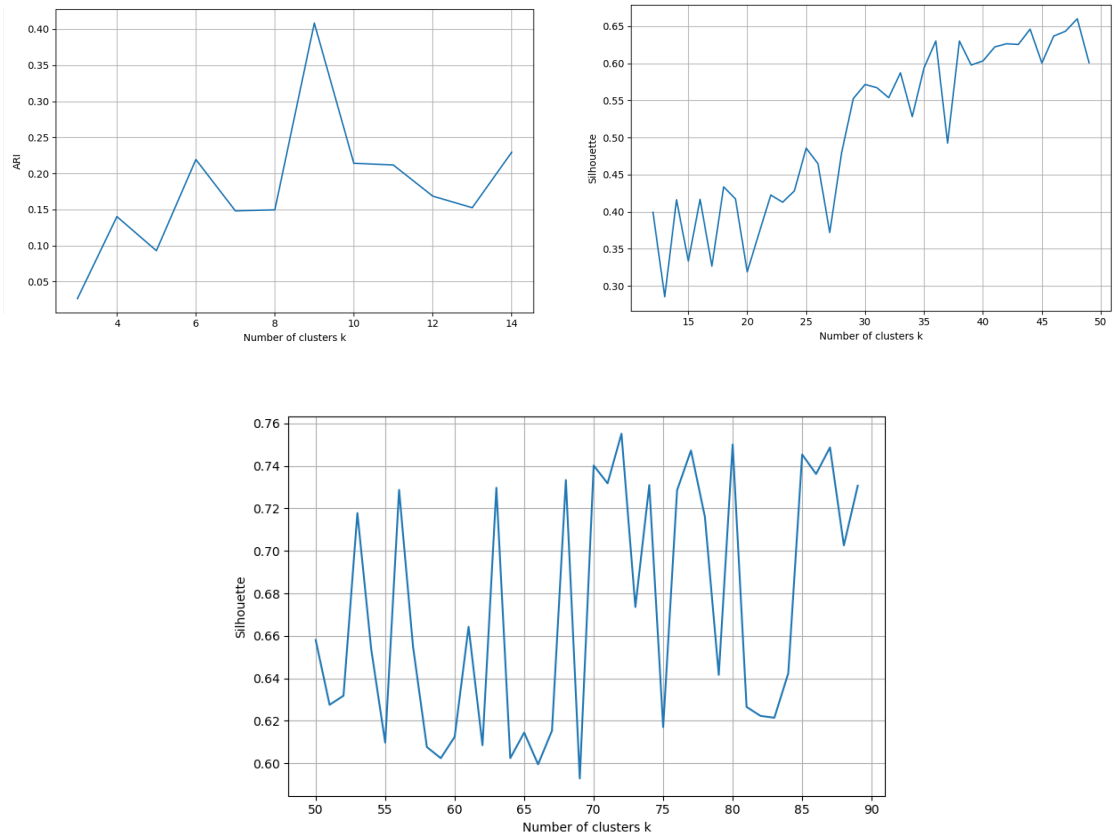


Figure 71: silhouette analysis

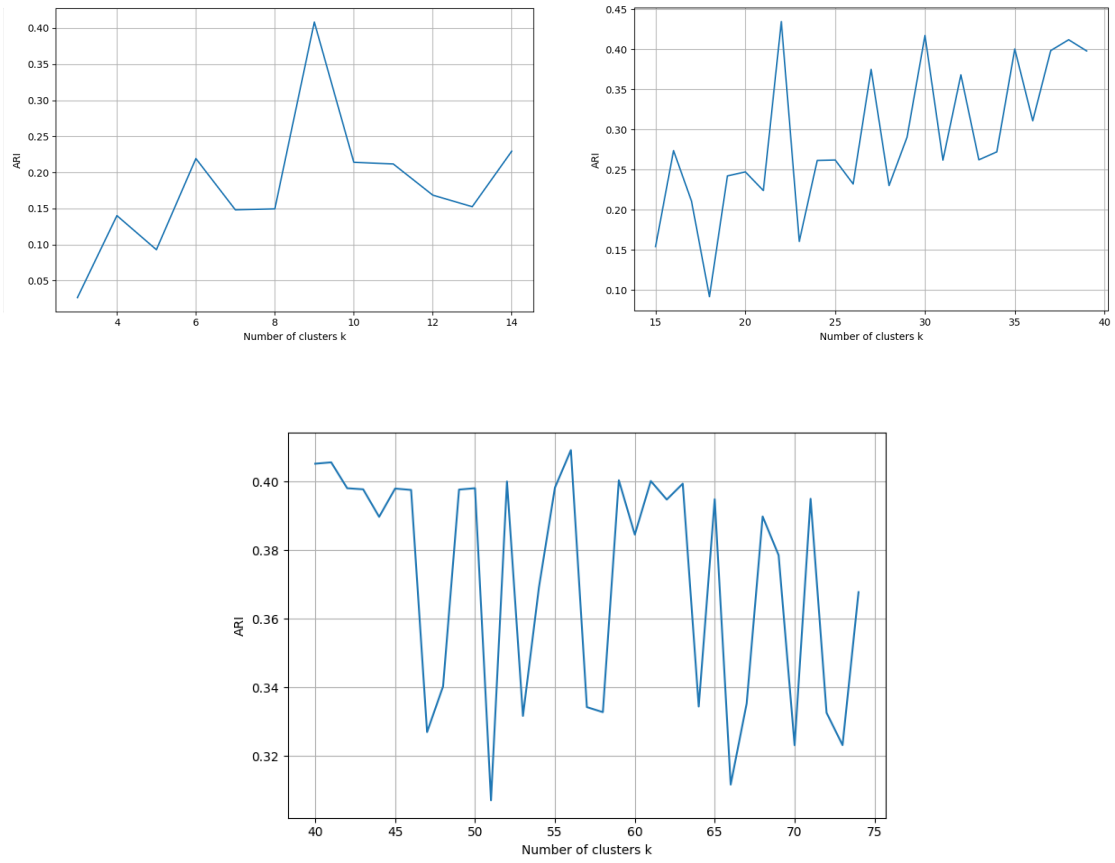


Figure 73: ARI analysis

Regarding the hyper-parameters we take as 'covariance_type' the 'full' one, since we want to have a covariance matrix for each component that will be helpful in the next section. Then we perform an hyper-parameters tuning, based on the best ARI, by looking at three parameters:

1. max_iter: the number of EM iterations to perform;
2. n_init: the number of initializations to perform;
3. init_params: the method used to initializate the weights;

Hyper-parameters	Best-ones
max_iter	300
n_init	1
init_param	kmeans

Table 4: Table of best hyper-parameters

Then we evaluate the clusters with the best hyper-parameters , in which the rand index is 0.89, the adjusted-rand index is 0.43, the silhouette is 0.63 and the log-likelihood is 59.93 (it measures of how well the model explains the observed data). Afterwards we get the ECDF of number of flows per cluster (Figure 74). Finally in order to have a better visualization of the detected clusters with respect to the labels we perform a PCA with two components (Figure 75) and a t-SNE with two components(Figure 76).

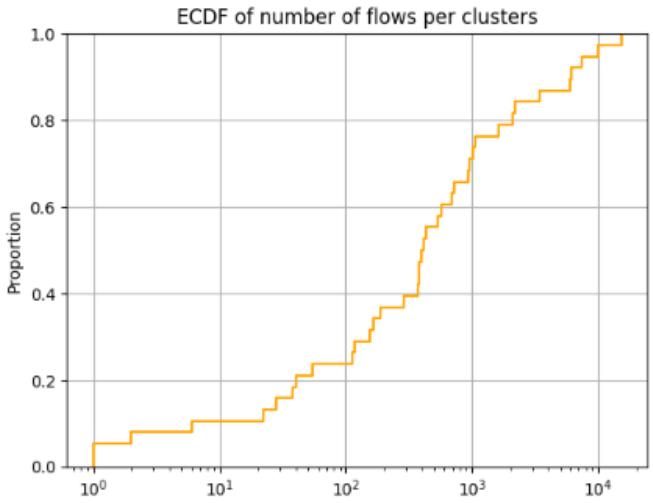


Figure 74

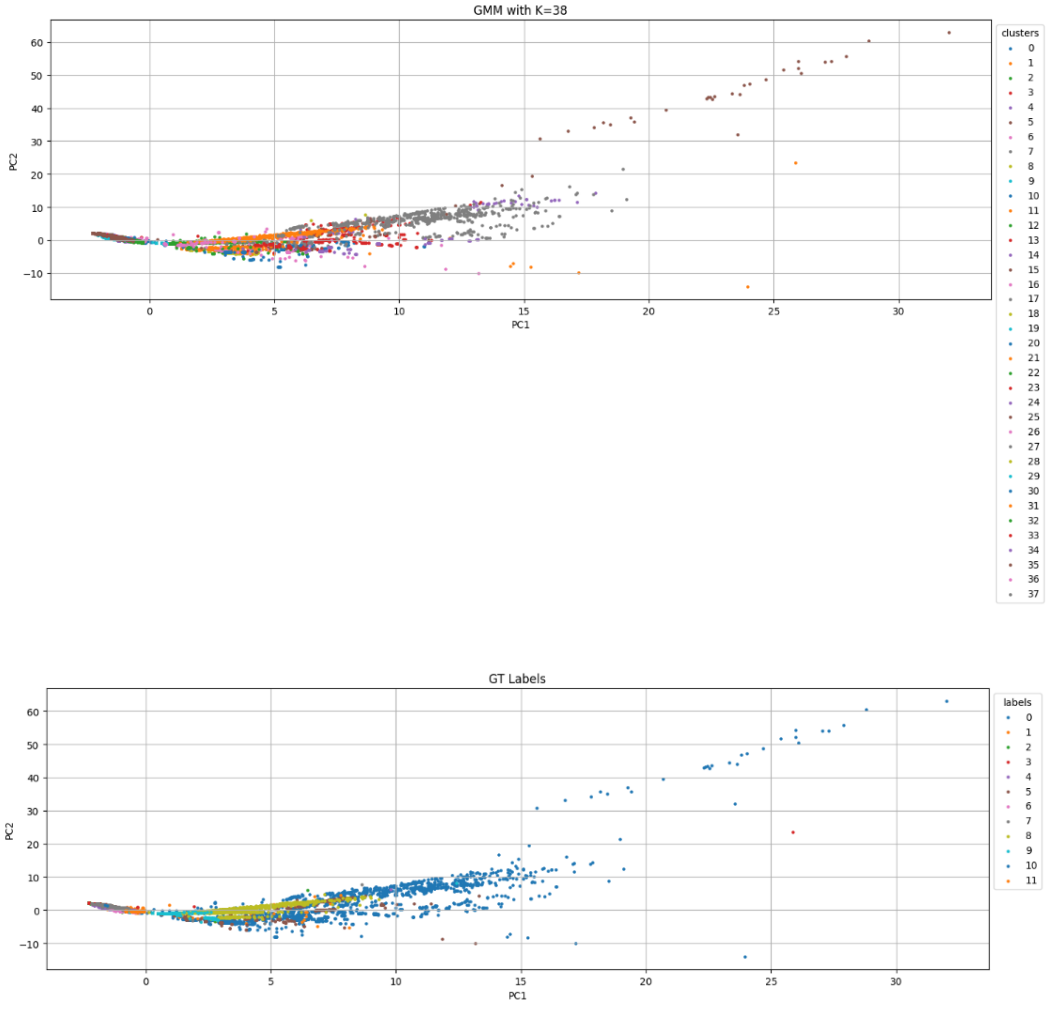
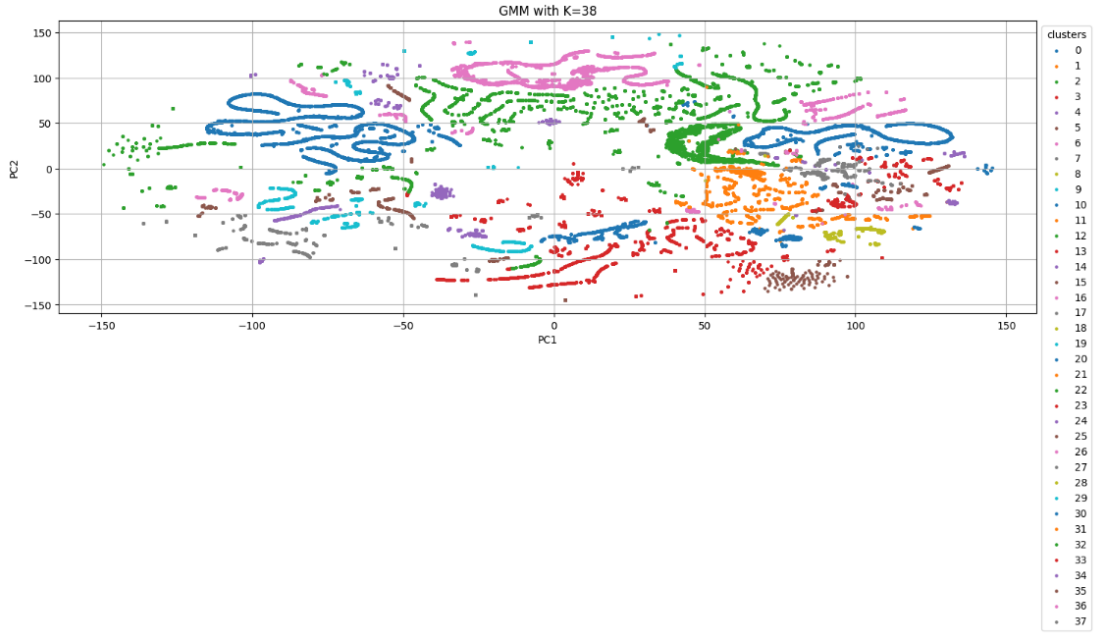


Figure 75: PCA



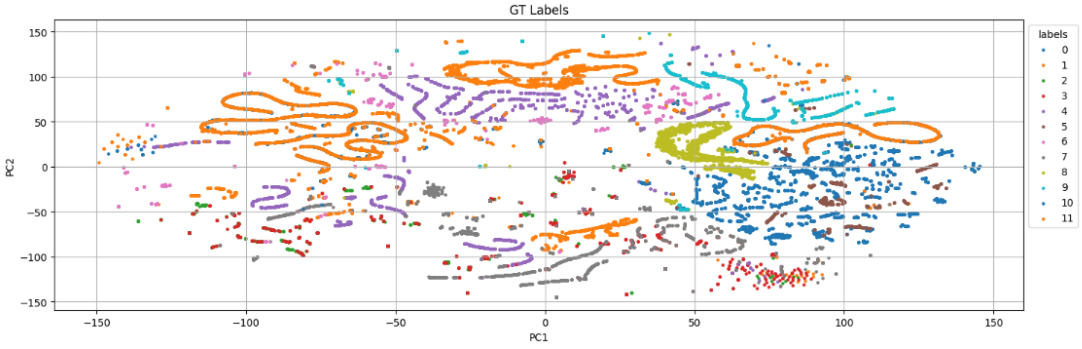


Figure 76: t-SNE

5.3 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is another clustering algorithm commonly used in machine learning and data mining. Unlike K-Means, which partitions the dataset into a pre-defined number of clusters, DBSCAN is designed to identify clusters of arbitrary shapes in spatial data based on the density of data points.
DBSCAN has two main hyper-parameters:

1. eps: the distance at which two points are considered neighbours;
2. min samples: the number of neighbours needed for a point to be considered a core one;

First of all we test the parameters to find a suitable range for a grid search.
The statistics taken into account are: silhouette, ri, ari and the number of clusters, which has to be kept at bay to avoid having many micro-clusters.

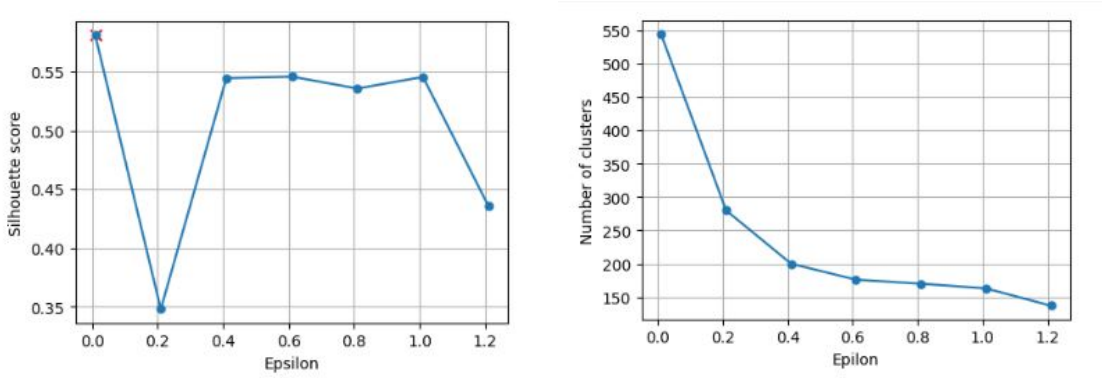


Figure 77

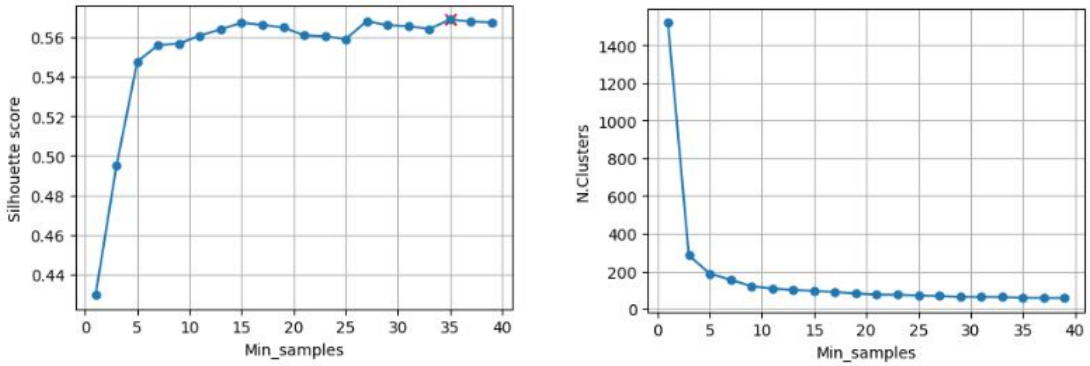


Figure 78

By looking at the plots in Figure 78 we can notice a stable silhouette score past Epsilon=0.4 and Min_samples=5 while there is a steep descent of the number of clusters for both the hyper-parameters. IN the table below the chosen ranges.

Hyper-parameters	Range
eps	[0.6,0.9]
min_samples	[30,40)

Table 5: Table of hyper-parameter ranges

We can now proceed to run a grid search of the best hyper-parameters in the specified ranges. From the silhouette-based heatmap in Figure 79 we can infer the best combination of them, as shown in the table below.

Hyper-parameters	Best-ones
eps	0.7
min_samples	36

Table 6: Table of best hyper-parameters

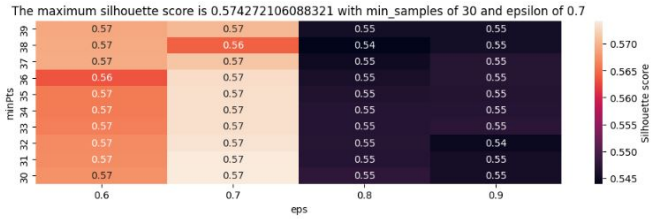


Figure 79: ARI analysis

We can now evaluate the performance of the tuned DBSCAN algorithm. By looking at the cluster distributions and the relative ECDF, we can spot two fairly big clusters. This behaviour is not optimal because it can lead to misclassifications of benign traffic or vice-versa.
For this reason and for the sub-par metrics obtained, we deemed this algorithm not suitable for our research and, therefore, discarded.

```
DBSCAN
Size of each cluster:  [ 3316   130  2795    95    60   232   448    57  2056    48   139 15654
   139   151    65    63   310   103   274   131   304   116    53   813
   226    41    93    35    41   160 15353    58    54    63   628   478
    57    37    65   128   115    89  4134  2009   328   328   161   831
   123   662   132    91    46    36    30    36  2307   702    47  1679
  5236    48]
Number of clusters:  62
Silhouette: 0.57
RI: 0.85
ARI: 0.23
```

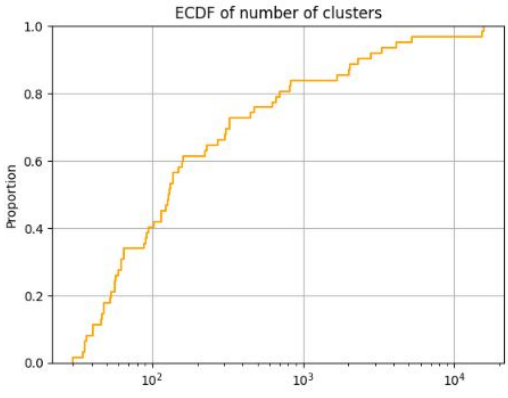


Figure 80

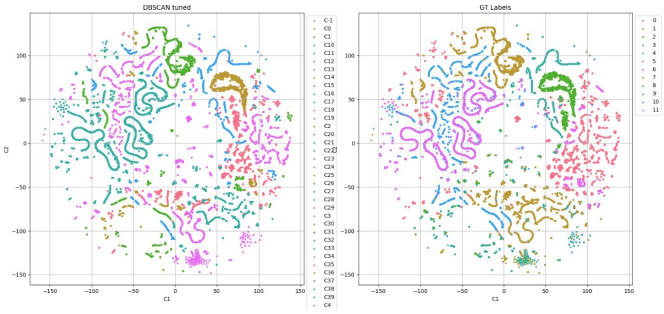


Figure 81: Comparison of GT and DBSCAN

6 Clusters explainability and analysis

In this section, we examine the interpretability and analysis of clusters by describing them in relation to the distribution of features and patterns of activity, providing insights and reflections on how these findings contribute to the understanding of network traffic.

6.1 K-means

For the K-means clustering method the number of clusters is higher than the number of labels, in fact, starting from 12 labels of the GT, the optimal number of cluster for this algorithm is 50.

In machine learning, K-means clustering reveals intriguing insights when the optimal number of clusters, often exceeding predefined labels, is explored. With a ground truth dataset of 12 labels, K-means consistently suggests 50 clusters, uncovering nuanced relationships and hidden patterns.
This report digs into these implications, offering a concise exploration of the algorithm's adaptability and its ability to unveil inherent dataset complexities.
The figure 82 shows from which cluster a label belongs, in particular label 1, which corresponds to benign traffic, has the highest number of representative clusters, while 'ddos udp lag' (label 11) is composed by only 4 clusters.

Label	Clusters
0	{2, 6, 7, 8, 10, 13, 14, 17, 21, 22, 24, 27, 28, 29, 34, 37, 39, 42, 43, 44, 47, 49}
1	{0, 2, 3, 4, 5, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 22, 24, 25, 26, 28, 30, 31, 36, 37, 39, 42, 45, 46}
2	{32, 40, 9, 42, 12, 48, 17, 16, 23, 24, 25, 30}
3	{0, 1, 4, 7, 40, 9, 8, 12, 46, 15, 48, 16, 14, 20, 23, 24, 25, 30}
4	{35, 7, 8, 11, 12, 15, 19, 24, 25, 30, 31}
5	{33, 2, 34, 37, 38, 39, 7, 10, 42, 13, 14, 47, 45, 17, 49, 21, 22, 29}
6	{0, 1, 4, 5, 7, 8, 9, 40, 46, 16, 48, 23, 31}
7	{0, 1, 4, 7, 40, 41, 9, 8, 12, 10, 46, 15, 16, 17, 23, 24, 25, 30}
8	{2, 6, 39, 11, 14, 47, 29}
9	{2, 5, 39, 17, 19}
10	{1, 7, 8, 19, 21, 31}
11	{8, 31, 21, 7}

Figure 82

On the contrary, the table 84 shows which of the GT labels are covered by each cluster. It is important to notice a good amount of single-labeled clusters that can be associated to noise/outliers.

In addition, Figures 83 and 85 present stacked bar plots that provide a detailed breakdown of the number of labels associated with each of the 50 clusters, on different scales. The vertical axis of the stacked bar plots represents the total count of labels, while the horizontal axis depicts the individual clusters. Each bar in the plot is divided into segments, representing the specific count of labels attributed to the corresponding cluster.
This visualization offers a comprehensive overview of label distribution across the entire dataset, allowing for a more nuanced understanding of the cluster-label affiliations.

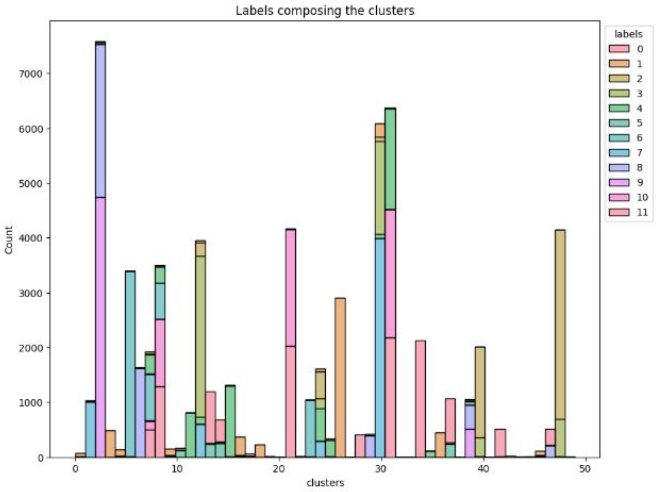


Figure 83

Cluster	Labels
0	{1, 3, 6, 7}
1	{10, 3, 6, 7}
2	{0, 1, 5, 8, 9}
3	{1}
4	{1, 3, 6, 7}
5	{1, 6, 9}
6	{0, 8}
7	{0, 1, 3, 4, 5, 6, 7, 10, 11}
8	{0, 1, 3, 4, 6, 7, 10, 11}
9	{1, 2, 3, 6, 7}
10	{0, 1, 5, 7}
11	{8, 4}
12	{1, 2, 3, 4, 7}
13	{0, 1, 5}
14	{0, 1, 3, 5, 8}
15	{1, 3, 4, 7}
16	{1, 2, 3, 6, 7}
17	{0, 1, 2, 5, 7, 9}
18	{1}
19	{9, 10, 4}
20	{3}
21	{0, 10, 11, 5}
22	{0, 1, 5}
23	{2, 3, 6, 7}
24	{0, 1, 2, 3, 4, 7}
25	{1, 2, 3, 4, 7}
26	{1}
27	{0}
28	{0, 1}
29	{0, 8, 5}
30	{1, 2, 3, 4, 7}
31	{1, 4, 6, 10, 11}
32	{2}
33	{5}
34	{0, 5}
35	{4}
36	{1}
37	{0, 1, 5}
38	{5}
39	{0, 1, 5, 8, 9}
40	{2, 3, 6, 7}
41	{7}
42	{0, 1, 2, 5}
43	{0}
44	{0}
45	{1, 5}
46	{1, 3, 6, 7}
47	{0, 8, 5}
48	{2, 3, 6}
49	{0, 5}

Figure 84

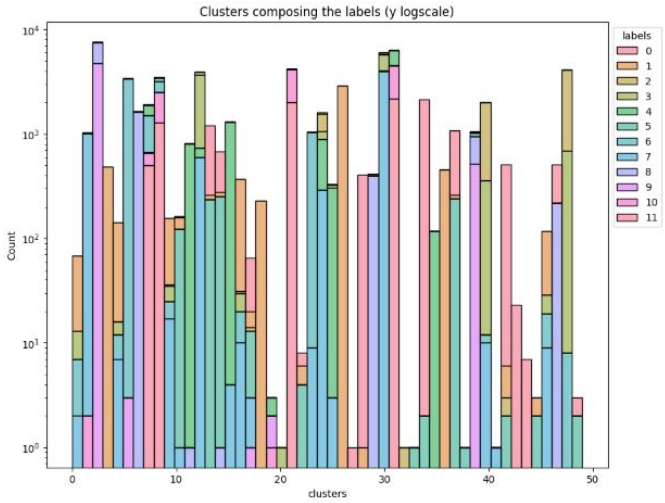


Figure 85

Upon close examination of the two graphical representations and the accompanying table, a discernible pattern emerges.
Some clusters exhibit a diverse affiliation with multiple labels, suggesting a complex relationship, whereas others, exemplified by clusters 3, 18, and 20, distinctly represent a singular label. This discrepancy in label distribution across clusters raises the intriguing possibility that clusters 3, 18, and 20 may serve as notable outliers or noise within the dataset.
This observation prompts further investigation into the nature of these clusters and their potential significance in understanding the underlying structure and characteristics of the data.
For enhanced clarity and visibility regarding how clusters are associated with each label, the following graph provides a visual representation.
This graph serves to reflect the previously described pattern, by highlighting the diverse participation of certain clusters with multiple labels. the aim is to facilitate a more intuitive understanding of the intricate relationships between clusters and labels within the dataset.

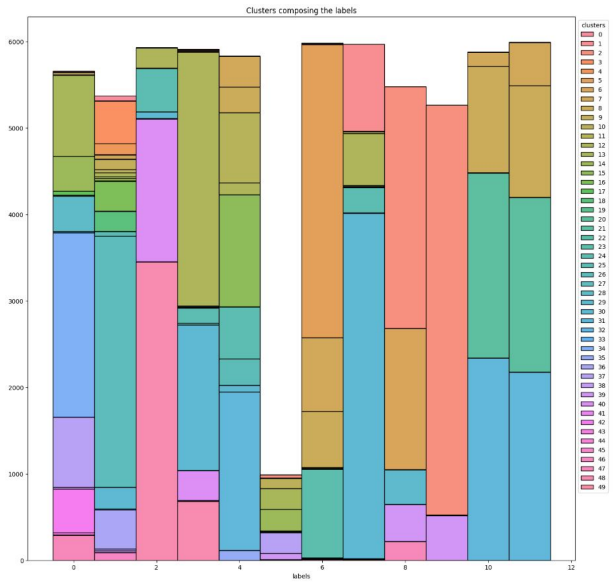


Figure 86

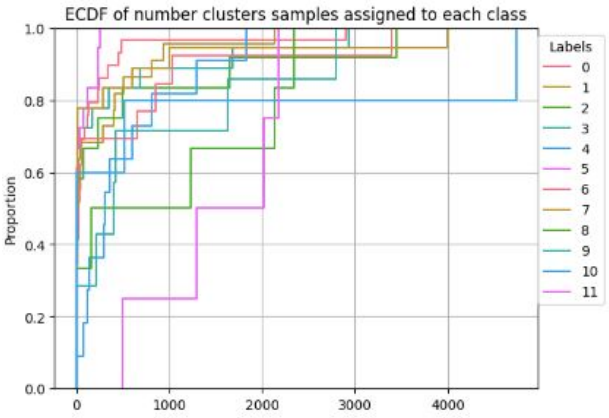
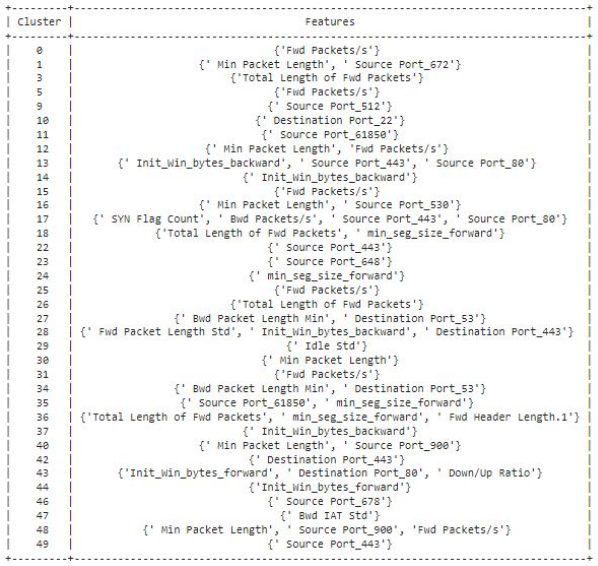


Figure 87

A potential visualization for Figure 89 is the Empirical Cumulative Distribution Function (ECDF). Both these graphs illustrate the prevalence of specific clusters for certain labels. Notably, large bars in the first graph and significant "jumps" in the second imply the dominance of some clusters within that specific label. These findings are highly promising as they indicate that a small number of clusters effectively describe the majority of samples belonging to a particular label. This insight underscores the significance of these key clusters in characterizing and capturing the essence of the associated labels in our dataset.
For our research it is also important to analyse the behaviour of a benign user and the possible actions that can be classified as malicious traffic.
An important point regarding the previous figures and Figure 6 is the presence of the "Benign" label (Label 1) sharing clusters with attack associated labels. This phenomenon may be attributed to outliers, where a benign user's traffic mimics a DDoS attack or, more likely, an attacker trying to appear as a benign user.
In summary, there are identifiable features with common characteristics between benign and malicious flows.
In particular, it is possible to observe from Figures 1,2 and 4 how, for example, cluster 7 is present in labels 0, 1, 3, 4, 5, 6, 7, 10, or cluster 24 is present in labels 0, 1, 2, 3, 4, 7 (where 1 is the benign label, while all the others are different attacks). The only clusters that represents only the label 1 are 3, 18, 26, 36.
This means that benign traffic can be wrongly associated to malicious behaviours.
Another important step for cluster analysis is feature characterization.
Having applied the k-means algorithm we are able to get the most important features, in terms of descriptive value, by analyzing the centroids of said algorithm after it has been trained on the entire dataset.
Centroids are points in the n-dimensional space of the dataset which are obtained by computing a mean between the points belonging to the same cluster. By leveraging this property it is possible to search for outliers in the values for the same component (feature) in each centroid.
If present, this outliers can be used tho determine the importance of a feature for a cluster. In fact, abnormal values of a centroid coordinate can be seen as the value that distinguishes a certain cluster from the others, in other words, this feature is important for that cluster.
In our case we checked for outliers in every column of the 43X50 matrix obtained from the optimized k-means model, if a value is greater than a threshold ($2*\sigma$), it will be flagged as outlier. So, we obtain the following features:



Cluster	Features
0	{'Fwd Packets/s'}
1	{' Min Packet Length', ' Source Port_672'}
3	{'Total Length of Fwd Packets'}
5	{'Fwd Packets/s'}
9	{' Source Port_512'}
10	{' Destination Port_22'}
11	{' Source Port_61850'}
12	{' Min Packet Length', 'Fwd Packets/s'}
13	{' Init_Win_bytes_backward', ' Source Port_443', ' Source Port_80'}
14	{' Init_Win_bytes_backward'}
15	{'Fwd Packets/s'}
16	{' Min Packet Length', ' Source Port_530'}
17	{' SYN Flag Count', ' Bwd Packets/s', ' Source Port_443', ' Source Port_80'}
18	{'Total Length of Fwd Packets', ' min_seg_size_forward'}
22	{' Source Port_443'}
23	{' Source Port_648'}
24	{' min_seg_size_forward'}
25	{'Fwd Packets/s'}
26	{'Total Length of Fwd Packets'}
27	{' Bwd Packet Length Min', ' Destination Port_53'}
28	{' Fwd Packet Length Std', ' Init_Win_bytes_backward', ' Destination Port_443'}
29	{' Idle Std'}
30	{' Min Packet Length'}
31	{'Fwd Packets/s'}
34	{' Bwd Packet Length Min', ' Destination Port_53'}
35	{' Source Port_61850', ' min_seg_size_forward'}
36	{'Total Length of Fwd Packets', ' min_seg_size_forward', ' Fwd Header Length.1'}
37	{' Init_Win_bytes_backward'}
40	{' Min Packet Length', ' Source Port_900'}
42	{' Destination Port_443'}
43	{'Init_Win_bytes_forward', ' Destination Port_80', ' Down/Up Ratio'}
44	{'Init_Win_bytes_forward'}
46	{' Source Port_678'}
47	{' Bwd IAT Std'}
48	{' Min Packet Length', ' Source Port_900', 'Fwd Packets/s'}
49	{' Source Port_443'}

Figure 88

Having observed that some of the features are significant for more than one cluster, we now shift our interest in finding possible sub-attacks for classification purposes.
By looking at this plot (86) it is clear that the attacks are further divided into sub-sets of various sizes.
As always, we want to get the features useful for determining these differences. The problem is subdivided by label. For each macro attack a decision tree is fitted having the cluster identifiers as labels. For example, if attack 1 is represented by clusters 0 and 3, we will gather all the samples belonging to attack 1 that are identified within clusters 0 and 3, and setting them as labels for the decision tree to predict.
By looking at the visual representation and data retrieved from the model we are able to determine the different important features for the sub attacks.
For clarity, when we refer to 'sub-attack', it signifies that within a specific type of attack, there can be subtle variations reflected by its distinctive features. These variations are manifested through the formation of different clusters, each representing a nuanced aspect or sub-type of the overarching attack category.
In essence, the features associated with an attack may exhibit minor differences, resulting in the identification of diverse clusters that capture the variability within the attack type.

```
Most important features for label 1:  ('Total Length of Fwd Packets', 0.37874311263290367) (' Fwd Header Length.1', 0.273695015673972) (' Source Port_530', 0.08808245326920014)
Most important features for label 0:  (' Destination Port_53', 0.34852724046854155) ('Total Length of Fwd Packets', 0.18892114946405325) (' Source Port_-1', 0.16399369572621028)
Most important features for label 2:  ('Fwd Packets/s', 0.5680625948638092) (' min_seg_size_forward', 0.25424418517308817) (' Source Port_900', 0.1739593509672761)
Most important features for label 3:  ('Fwd Packets/s', 0.4537042524349131) (' Source Port_-1', 0.41582237709400555) (' min_seg_size_forward', 0.043154403287079354)
Most important features for label 4:  ('Total Length of Fwd Packets', 0.2968643992424803) (' Fwd Header Length.1', 0.206968778457079) (' Flow IAT Max', 0.1811735783113039)
Most important features for label 5:  ('Init_Win_bytes_forward', 0.2826192240805534) (' Source Port_-1', 0.2642952203232215) (' Destination Port_80', 0.1997975746113851)
Most important features for label 6:  ('Fwd Packets/s', 0.4188136462905484) (' Source Port_648', 0.3492770850122022) (' Flow IAT Max', 0.19700113653297613)
Most important features for label 7:  (' Source Port_672', 0.4553913084557603) (' Flow IAT Max', 0.3179726767668135) (' Fwd Header Length.1', 0.16116006321206844)
Most important features for label 10:  ('Fwd Packets/s', 0.6110548499493128) (' Fwd Packet Length Std', 0.38742059488961256) (' Source Port_672', 0.0010125065096088486)
Most important features for label 8:  ('Fwd Packets/s', 0.5304016303252373) (' Idle Std', 0.16750976468353496) (' Bwd Packets/s', 0.13539516980381888)
Most important features for label 9:  (' Down/Up Ratio', 0.9473807788743535) (' Bwd IAT Min', 0.042060765444709505) ('Init_Win_bytes_forward', 0.006863709542418267)
Most important features for label 11:  ('Fwd Packets/s', 0.6384002000826084) (' Packet Length Std', 0.3615997999173915) (' Total Fwd Packets', 0.0)
```

Figure 89

Moreover, the decision tree comes into play when we want to determine different but 'similar' groups of clusters.
Therefore, our objective is to understand which features contribute less into discriminating one cluster from another.
To achieve this, we feed the entire dataset into the decision tree, with each sample labeled according to its respective cluster. By examining the decision tree, the features present in the features that the algorithms uses to take decisions near the leaves are the ones that contribute less to the discrimination.

Consequently, we can assert that these features exhibit similar trends among the leaves beneath them. Thus, these clusters can be considered similar. So the main points of this process are:

1. Training Dataset: Utilize the entire dataset labeled based on clusters as input for the decision tree.

2. Learning Phase: The decision tree learns features contributing less to discriminate among different cluster groups.

3. Decision Analysis: Examine decisions near the leaves of the decision tree to identify less influential features in discrimination.

4. Interpretation: Features with similar behaviors near the leaves indicate similar trends among corresponding clusters.

5. Definition of Similarity: Based on feature behavior, establish similarity between clusters.

In examining the decision tree model (for the full decision tree rendering, check the attachment) employed for cluster discrimination, a good example is the utilization of the "max packet length" feature.
This feature, when integrated into the decision tree, showcases a notable success in achieving a Gini of 0.066. This success is particularly evident in the discrimination between clusters 13 and 40 from cluster 4, where the model demonstrates effectiveness.
However, the intricacies become apparent when attempting to distinguish clusters 13 and 40 from each other.
In this scenario, the Gini rises significantly to 0.444, indicating substantial difficulty in the discrimination task. The challenge appears to be more pronounced when dealing with these specific clusters, suggesting a nuanced relationship that the decision tree struggles to capture accurately.
A noteworthy observation is the limited number of samples associated with clusters 13 and 40, amounting to 2 and 1, respectively.
This raises the question of whether these clusters could be considered outliers. The influence of such outliers becomes evident in the discriminatory challenges faced by the model, as the sparse data points contribute to the complexity of distinguishing between clusters.
Expanding our analysis to clusters 40, 16, and 4, we find that the decision tree resorts to creating intricate decision paths rather than straightforward exclusions to achieve discrimination.
This complexity is indicative of the interwoven nature of these clusters, requiring the model to navigate through dense decision-making processes. (Those clusters define labels 3, 6, 7)
To sum up, we can assert that a deep decision tree, as in our case, with numerous nodes near the leaves, reflects the difficulty of distinguishing certain clusters from others.
This complexity underscores the importance of a careful evaluation of features and underlying data to further refine our model and enhance its effectiveness in cluster discrimination.

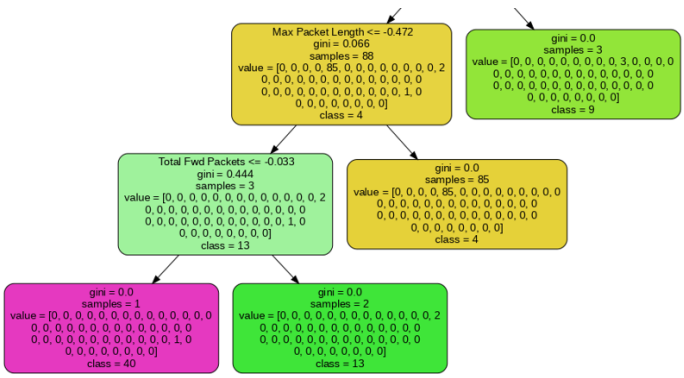


Figure 90

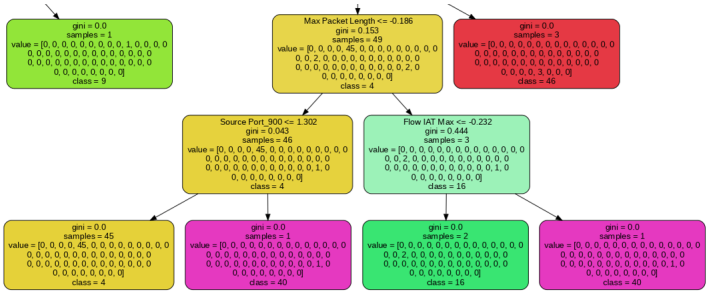
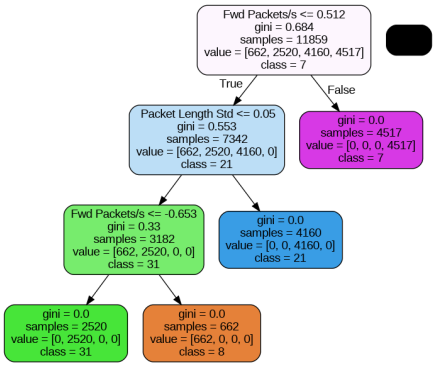


Figure 91

As for the supervised learning approach, we encountered extreme similarities in the 10 and 11 ('tftp ddos' and 'udp lag ddos' respectively) attacks.
This is visually verifiable by looking at the plot (Figure 86). This is further confirmed by the centroid-based analysis (Figure 82) by comparing the clusters in common between 10 and 11.
A Decision tree approach is taken again to determine the feature importance.



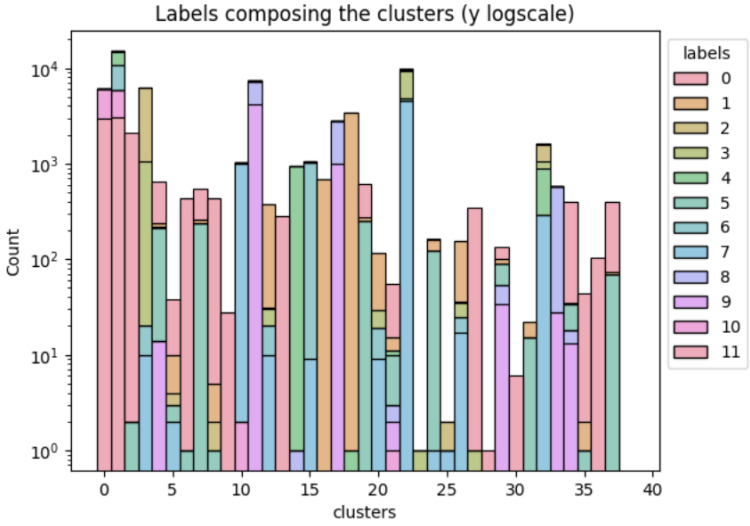

```
Feature Importance:
Fwd Packets/s: 0.6284899571552169
 Packet Length Std: 0.3715100428447831
 Total Fwd Packets: 0.0
 Total Backward Packets: 0.0
Total Length of Fwd Packets: 0.0
 Fwd Packet Length Std: 0.0
 Bwd Packet Length Min: 0.0
 Flow IAT Max: 0.0
 Flow IAT Min: 0.0
Bwd IAT Total: 0.0
 Bwd IAT Std: 0.0
 Bwd IAT Min: 0.0
Fwd PSH Flags: 0.0
```

Figure 92

As evident from the very small decision tree for these four clusters, there is a notable challenge for the decision tree in distinguishing them.
In fact, out of the over 40 features in our dataset, only two actively contribute to the decision-making process.
This underscores and confirms the issues identified in the supervised context for the distinction of these two attacks and extends to the challenges faced in the unsupervised scenario as well, where clusters are assigned either to 10 or 11 labeled data points.

6.2 Gaussian Mixture

Given the assignments of samples to clusters, we observe through the histogram in Figure 93 that there are clusters belonging only to one class and others belonging to multiple classes. In particular thanks to the table in Figure 93 we can see that there are clusters that represents only one label, that are clusters number: 9, 13, 16, 23, 28, 30, 36. On the contrary clusters with the majority of labels are: 0, 1, 21, 32.



Cluster	Labels
0	{0, 3, 4, 6, 7, 10, 11}
1	{0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11}
2	{0, 5}
3	{2, 3, 6, 7}
4	{0, 1, 3, 5, 9}
5	{0, 1, 2, 5, 7}
6	{0, 5}
7	{0, 1, 5}
8	{0, 1, 2, 5}
9	{0}
10	{10, 3, 6, 7}
11	{0, 1, 5, 8, 9}
12	{1, 2, 3, 6, 7}
13	{0}
14	{8, 4}
15	{2, 3, 6, 7}
16	{1}
17	{0, 8, 9}
18	{1, 4}
19	{0, 1, 5}
20	{1, 3, 6, 7}
21	{0, 1, 4, 5, 8, 9, 10}
22	{1, 2, 3, 4, 7}
23	{3}
24	{0, 1, 5, 7}
25	{2, 7}
26	{1, 2, 3, 6, 7}
27	{0, 3}
28	{0}
29	{0, 1, 5, 8, 9}
30	{0}
31	{1, 5}
32	{0, 1, 2, 3, 4, 7}
33	{0, 8, 9}
34	{0, 1, 5, 8, 9}
35	{0, 1, 5}
36	{0}
37	{0, 1, 5}

Figure 93

An insight of how the samples of a class are divided is given by the Empirical Cumulative Distribution Function in Figure 94.

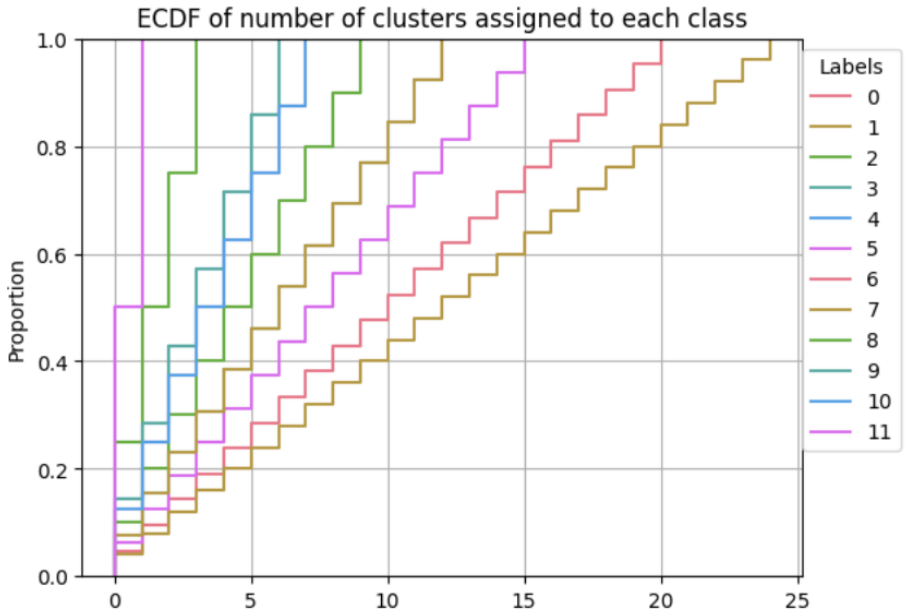


Figure 94

In the Figure 95 the steps are proportional to the number of objects in cluster x that belong to the class associated with the color.

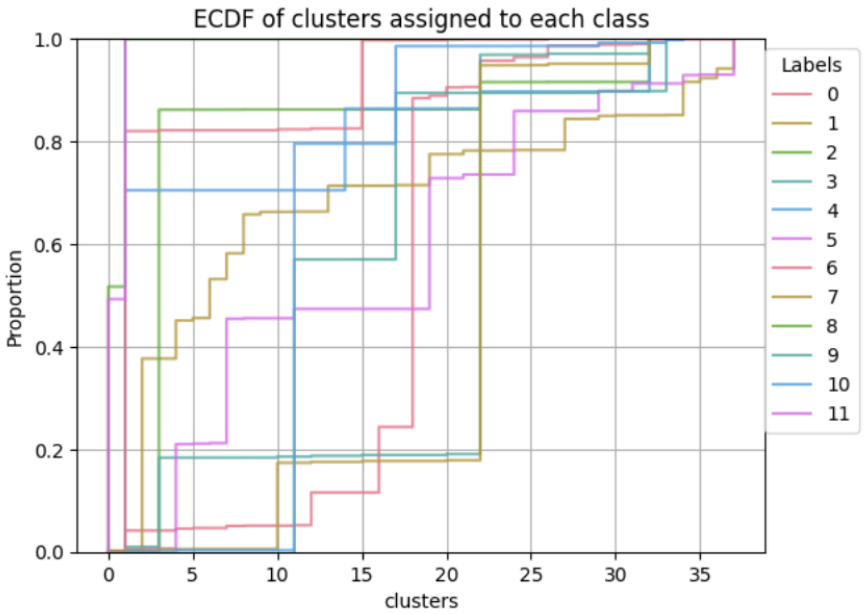


Figure 95

The size of the clusters varies greatly; for example, Cluster 1 contains almost sixteen thousand elements, Cluster 22 nearly ten thousand, however some clusters have few elements. This could indicate that these elements are outliers, as they are distant in space from the rest of the points. (Figure 96)

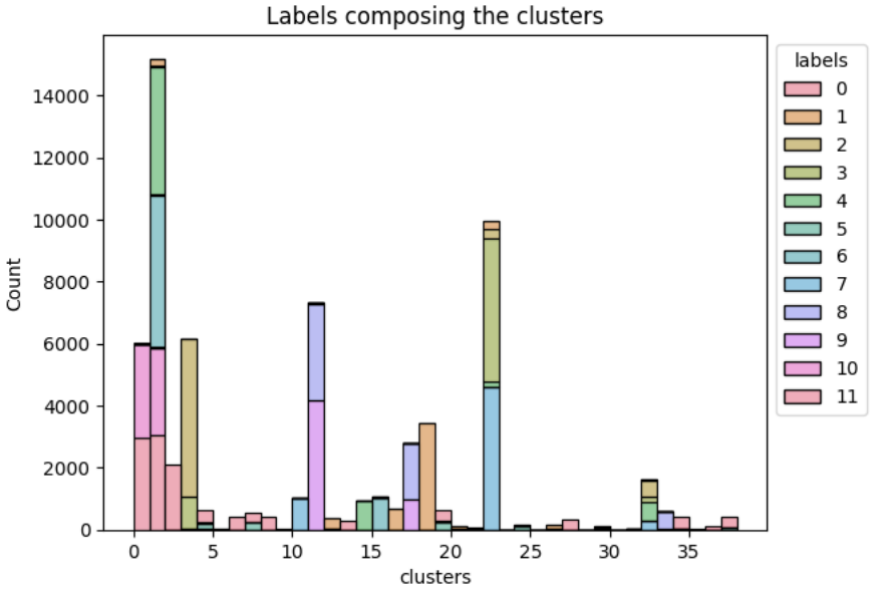


Figure 96

Many instances of benign traffic (Label 1), as shown in the figure 97, have been assigned by the model to clusters that contain also instances of malicious traffic, demonstrating that a subset of benign traffic exhibits characteristics similar to malicious traffic.

Label	Clusters
0	{0, 1, 2, 4, 5, 6, 7, 8, 9, 11, 13, 17, 19, 21, 24, 27, 28, 29, 30, 32, 33, 34, 35, 36, 37}
1	{1, 4, 5, 7, 8, 11, 12, 16, 18, 19, 20, 21, 22, 24, 26, 29, 31, 32, 34, 35, 37}
2	{32, 1, 3, 5, 8, 12, 15, 22, 25, 26}
3	{0, 1, 32, 3, 4, 10, 12, 15, 20, 22, 23, 26, 27}
4	{32, 1, 0, 14, 18, 21, 22}
5	{1, 34, 35, 4, 37, 6, 7, 5, 8, 2, 11, 19, 21, 24, 29, 31}
6	{0, 1, 3, 10, 12, 15, 20, 26}
7	{0, 1, 32, 3, 5, 10, 12, 15, 20, 22, 24, 25, 26}
8	{33, 34, 11, 14, 17, 21, 29}
9	{33, 34, 1, 4, 11, 17, 21, 29}
10	{0, 1, 10, 21}
11	{0, 1}

Figure 97

In a Gaussian Mixture Model (GMM), the covariance matrix is a crucial component that characterizes the shape of each cluster. The variance of each feature influences the spread or dispersion of data along that feature's axis.
Here's a more detailed explanation:

- **High Variance:** Features contribute more to the overall spread of the cluster. Therefore, during the training process, the GMM places more weight on features with higher variances when estimating the parameters of each component. In other words, it suggests that the feature has the potential to discriminate or capture the variability within that particular cluster.
- **Low Variance:** The feature may not contribute significantly to the variability within the cluster. They might not play a crucial role in defining the cluster's shape.

We extract from each cluster the covariance matrix and then the five greatest variance values, the Figure 98 shows the results.

Cluster	Features
Cluster 0	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' Packet Length Std', ' Fwd Packet Length Std'}
Cluster 1	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' Source Port_-1', 'Fwd Packets/s'}
Cluster 2	{' Down/Up Ratio', ' Destination Port_80', ' Packet Length Std', ' Bwd Packet Length Min', ' Destination Port_53'}
Cluster 3	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' Total Fwd Packets', 'Fwd Packets/s'}
Cluster 4	{'Init_Win_bytes_forward', ' Destination Port_80', ' Destination Port_443', ' Fwd Packet Length Std', 'Bwd IAT Total'}
Cluster 5	{' Destination Port_443', ' Source Port_443', ' Destination Port_-1', ' Bwd Packets/s', ' Destination Port_22'}
Cluster 6	{' Flow IAT Max', ' Fwd Packet Length Std', 'Fwd PSH Flags', 'Bwd IAT Total', ' Idle Std'}
Cluster 7	{' Down/Up Ratio', ' Init_Win_bytes_backward', ' Bwd IAT Min', ' Fwd Packet Length Std', 'Fwd PSH Flags'}
Cluster 8	{'Init_Win_bytes_forward', ' Flow IAT Max', ' Bwd IAT Std', ' Destination Port_443', 'Fwd Packets/s'}
Cluster 9	{' Max Packet Length', ' Total Backward Packets', ' Down/Up Ratio', ' Packet Length Std', 'Bwd IAT Total'}
Cluster 10	{' Fwd Header Length.1', ' Max Packet Length', ' Min Packet Length', ' min_seg_size_forward', 'Fwd Packets/s'}
Cluster 11	{'Init_Win_bytes_forward', ' Down/Up Ratio', ' Bwd IAT Min', 'Fwd Packets/s', ' Bwd Packets/s'}
Cluster 12	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' min_seg_size_forward', 'Fwd Packets/s'}
Cluster 13	{'Init_Win_bytes_forward', ' Down/Up Ratio', ' Flow IAT Min', ' Fwd Packet Length Std', 'Fwd PSH Flags'}
Cluster 14	{' Max Packet Length', 'Protocol_17', ' Min Packet Length', ' min_seg_size_forward', 'Fwd Packets/s'}
Cluster 15	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' Total Fwd Packets', 'Fwd Packets/s'}
Cluster 16	{' Fwd Header Length.1', 'Total Length of Fwd Packets', ' Min Packet Length', ' Source Port_-1', ' Fwd Packet Length Std'}
Cluster 17	{' Active Max', ' Flow IAT Max', ' Flow IAT Min', ' Bwd IAT Min', ' Idle Std'}
Cluster 18	{'Total Length of Fwd Packets', ' Flow IAT Max', ' Min Packet Length', ' Source Port_-1', ' Fwd Packet Length Std'}
Cluster 19	{' Down/Up Ratio', ' Destination Port_80', ' Packet Length Std', ' Destination Port_-1', 'Bwd IAT Total'}
Cluster 20	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' min_seg_size_forward', 'Fwd Packets/s'}
Cluster 21	{' Active Max', ' Flow IAT Max', ' Bwd IAT Std', 'Bwd IAT Total', ' Idle Std'}
Cluster 22	{' Max Packet Length', 'Total Length of Fwd Packets', ' Flow IAT Max', ' Min Packet Length', 'Fwd Packets/s'}
Cluster 23	{' Packet Length Std', ' Destination Port_443', ' SYN Flag Count', 'Fwd PSH Flags', ' URG Flag Count'}
Cluster 24	{'Init_Win_bytes_forward', ' Active Max', ' Down/Up Ratio', ' Fwd Packet Length Std', ' URG Flag Count'}
Cluster 25	{' Total Backward Packets', ' Flow IAT Max', ' Bwd IAT Min', ' Total Fwd Packets', 'Bwd IAT Total'}
Cluster 26	{' Max Packet Length', 'Total Length of Fwd Packets', ' Min Packet Length', ' min_seg_size_forward', 'Fwd Packets/s'}
Cluster 27	{'Init_Win_bytes_forward', ' Down/Up Ratio', ' Packet Length Std', ' Fwd Packet Length Std', ' URG Flag Count'}
Cluster 28	{' Packet Length Std', ' Destination Port_443', ' SYN Flag Count', 'Fwd PSH Flags', ' URG Flag Count'}
Cluster 29	{' Destination Port_80', ' Bwd IAT Min', ' Bwd Packet Length Min', ' URG Flag Count', ' Destination Port_53'}
Cluster 30	{'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' Bwd IAT Min', ' Source Port_443', ' Source Port_80'}
Cluster 31	{' Max Packet Length', ' Flow IAT Max', ' Min Packet Length', ' Flow IAT Min', 'Fwd PSH Flags'}
Cluster 32	{' Max Packet Length', ' Min Packet Length', ' Source Port_-1', 'Fwd Packets/s', ' Source Port_900'}
Cluster 33	{' Active Max', ' Bwd IAT Std', ' Bwd IAT Min', 'Bwd IAT Total', ' Idle Std'}
Cluster 34	{' Init_Win_bytes_backward', ' Source Port_443', ' Source Port_80', ' Bwd Packets/s', ' URG Flag Count'}
Cluster 35	{'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' Source Port_443', ' Fwd Packet Length Std', 'Fwd PSH Flags'}
Cluster 36	{' Active Max', ' Down/Up Ratio', ' Destination Port_443', ' Source Port_443', ' URG Flag Count'}
Cluster 37	{'Init_Win_bytes_forward', ' Down/Up Ratio', ' Init_Win_bytes_backward', ' Destination Port_80', ' Destination Port_443'}

Figure 98

As can be clearly seen in the Figure 99, some features play a significant role in characterizing or distinguishing a specific cluster such as **Min Packet Length**, which aligns with our previous analyses.

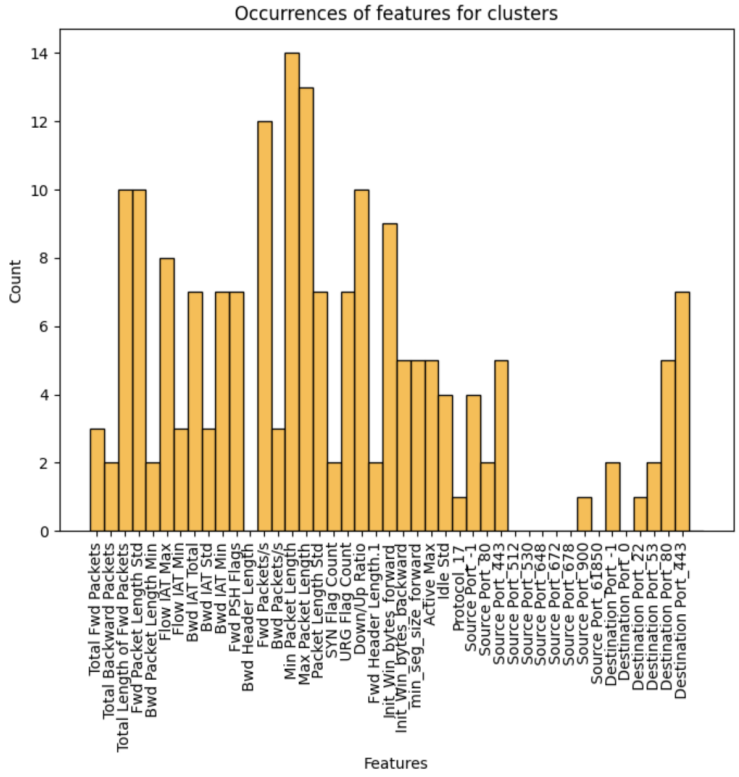


Figure 99

In the Figure 100 is possible to spot sub-attacks, variations within a specific DDoS attack, represented by different clusters within an attack class.

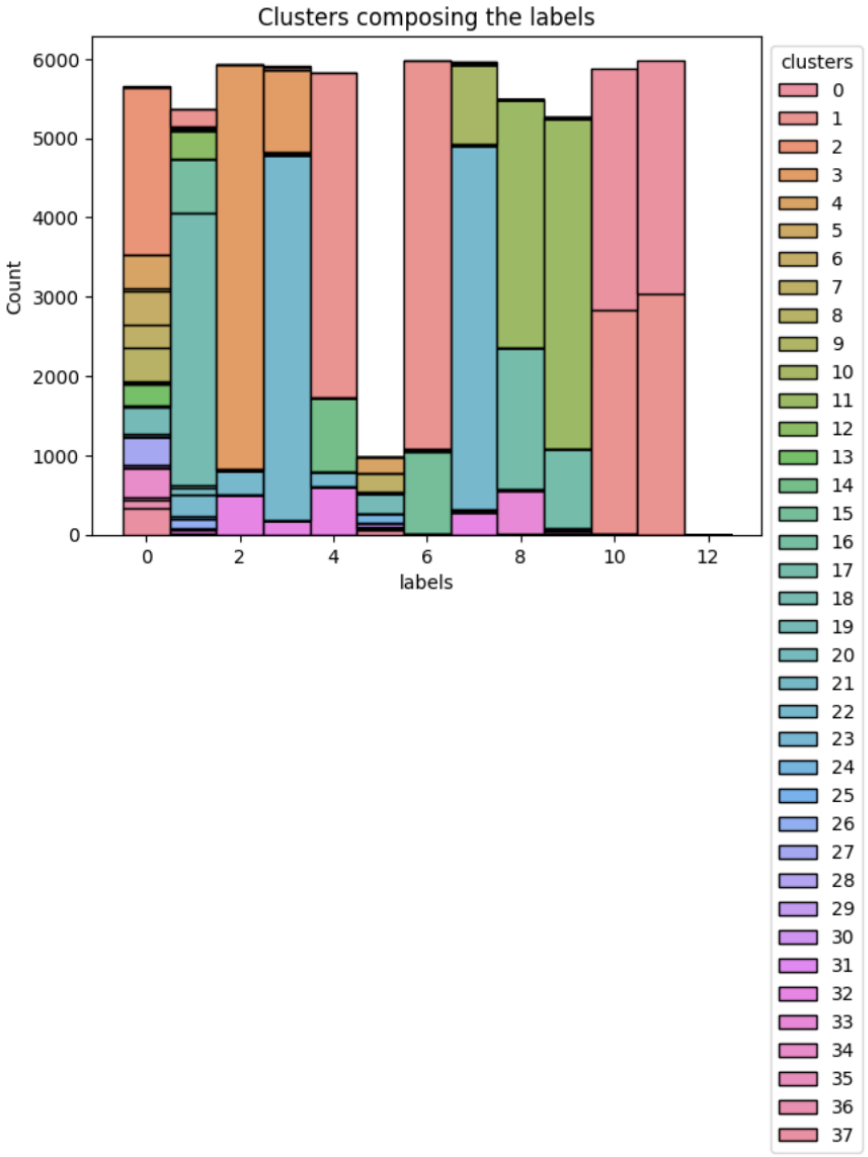


Figure 100

To extract the features distinguishing the sub-attacks, we employed a Decision Tree model with the input being data belonging to class i, but with the labels being the identifiers of the clusters to which the points of class i were assigned.
Decision trees often provide a built-in feature importance score. Most machine learning libraries, such as scikit-learn in Python, offer a **feature_importances_** attribute after fitting a decision tree model. This attribute contains the relative importance scores of each feature:

- **High Importance Scores** Features contribute more to the decision-making process of the decision tree for a specific cluster. Variations in these features strongly influence the decision tree's ability to differentiate points in that cluster from points in other clusters.
- **Low Importance Scores** Features are less influential in distinguishing between clusters. This doesn't mean these features are not important for the overall dataset; but only within the context of the sub-attacks.

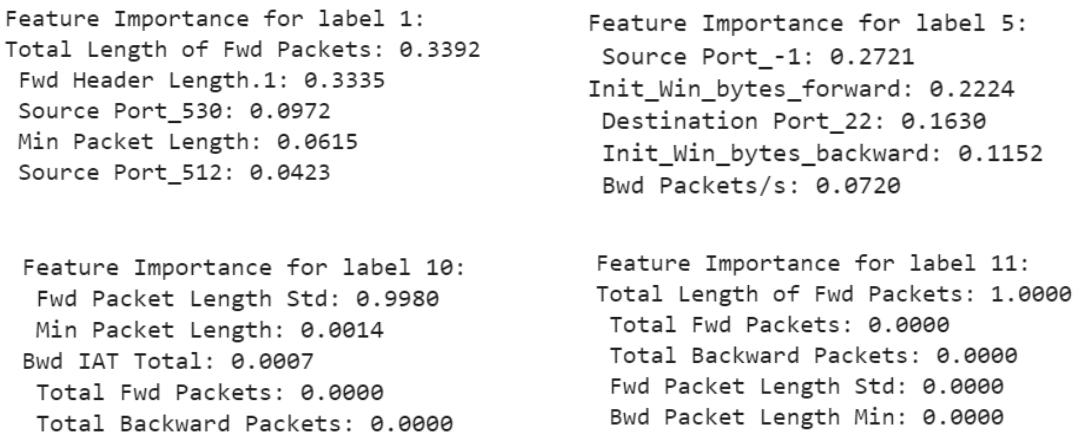


Figure 101

Given our previous analysis on the most important features for the obtained clusters and the earlier graphical representation of cluster points in space (Figure 75 and Figure 76), it is interesting to observe how some clusters exhibit similarity. This similarity is attributed to the proximity in space and the overlap in the top features across clusters. Features that consistently hold importance across multiple clusters may signify similarities, while distinct features unique to each cluster may imply differences.
These tables display the clusters with the 2, 3 and 4 most important features in common.

Features	Similar clusters
Fwd PSH Flags, Destination Port_443	28, 23
Fwd Packets/s, min_seg_size_forward	10, 26, 14
Min Packet Length, Fwd Packets/s	1, 3, 22, 15
Min Packet Length, min_seg_size_forward	12, 20
Init_Win_bytes_forward, Down/Up Ratio	27, 37
Source Port_443, Bwd Packets/s	34, 5

Table 7: Similar clusters with 2 features

Features	Similar clusters
Fwd Packets/s, Min Packet Length, min_seg_size_forward	12, 20, 10, 26, 14
Fwd Packets/s, Min Packet Length, Max Packet Length	3, 22, 15
URG Flag Count, Fwd PSH Flags, Destination Port_443	28, 23

Table 8: Similar clusters with 3 features

Features	Similar clusters
Fwd Packets/s, Min Packet Length, min_seg_size_forward, Total Length of Fwd Packets	3, 15
Fwd Packets/s, Min Packet Length, Max Packet Length, min_seg_size_forward	12, 20, 10, 26, 14
SYN Flag Count, URG Flag Count, Fwd PSH Flags, Destination Port_443	28, 23

Table 9: Similar clusters with 4 features

In order to see the attacks that are more similar, we can take a look at the Figure 100. We can notice that various labels are composed by similar clusters, in particular: 2 and 3, 4 and 6, 8 and 9, 10 and 11. Taking the largest common clusters, namely those that characterize both labels the most, we can perform a cross-search to find the most important shared features within those clusters. The results are shown in the table below.

Labels	Similar features
2-3	Fwd Packets/s, Min Packet Length, Max Packet Length'
4-6	Total Length of Fwd Packets, Min Packet Length, Max Packet Length
8-9	Bwd IAT Min
10-11	Total Length of Fwd Packets, Min Packet Length, Max Packet Length

Table 10: Similar features for attacks

7 Conclusions

In conclusion, as seen in the previous sections, the most critical attacks to analyze and, above all, to distinguish were those corresponding to DDoS attacks that utilize "UDP lag" and TFTP. This is something that could have been predictable.
Indeed, both attacks involve the use of the UDP protocol. TFTP uses UDP as the transport protocol for file transmission, and a DDoS attack targeting TFTP could involve saturating the bandwidth with unwanted UDP packets. In the case of "UDP lag", latency is caused by a high volume of unwanted UDP packets that can overwhelm the network or the destination server. Furthermore, both a DDoS attack on TFTP and one causing "UDP lag" can lead to service interruptions. In fact resource saturation may prevent the system from handling legitimate requests properly, compromising access to the service for authorized users.
Therefore, these similarities primarily lie in the objective of overloading the UDP protocol and system resources to generate latency issues or service interruptions.
The main difference lies in the specific goal: a DDoS attack on TFTP aims to compromise the file transfer service, while a DDoS attack causing "UDP lag" may have a more general impact on latency.