Politecnico di Torino

Department of Control and Computer Engineering

Master's degree in Cybersecurity

DDoS Attacks Detection and Characterization

Pietro Armenante, Christian Coduri, Eliana De Giuseppe, Luca Serafini

Academic year 2023/2024

1 Internet Security and Distributed-Denial-of-Service

Internet security is a branch of computer security that contains the Internet, browser security, web site security, and network security. Its objective is to establish rules and measures to use against attacks over the Internet. The Internet is an inherently insecure channel for information exchange, with high risk of intrusion or fraud, such as phishing, online viruses, trojans, ransomware and worms.

In computing, a denial-of-service attack (DoS attack) is a cyber-attack in which the attacker seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to a network. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

In a **distributed denial-of-service attack** (DDoS attack), the incoming traffic flooding the victim originates from many different sources. A DDoS attack uses more than one unique IP address or machines, often from thousands of hosts infected with malware.

The types of DDoS attacks that will be analysed, along with the benign traffic, are:

- DDoS UDP LAG, a large number of UDP packets are sent to a targeted server;
- DDoS SNMP, it uses a spoofed IP address to create requests, triggering a flood of SNMP responses;
- DDoS SSDP, it exploits the Universal Plug and Play (UPnP) network protocols;
- DDoS LDAP, the attacker sends small requests to a publicly available vulnerable LDAP server;
- DDoS MSSQL, it is based on tampering with the Microsoft SQL permission protocol to launch a reflection attack;
- DDoS UDP, the victim server receives a large number of UDP packets from a large number of IP addresses;
- DDoS NETBIOS, it makes the victim system unavailable to communication other NetBIOS hosts;
- DDoS SYN, the attacked server receives spoofed SYN requests containing a spoofed source IP address at high speed;
- DDoS DNS, the attacker forms a query in response to which the DNS server returns as much data as possible;
- DDoS TFTP, it makes a default request for a file, and the victim TFTP server returns data to the requesting target host as a result of this request regardless of a file name mismatch;
- DDoS NTP, the attacker organizes a flood of fake NTP requests from a large number of IP addresses.

2 Data exploration and pre-processing

Data exploration and pre-processing are essential steps in the data analysis pipeline, aimed at understanding and preparing raw data for further analysis. These stages are crucial for extracting meaningful insights, ensuring data quality, and enhancing the performance of machine learning models.

2.1 Data cleaning

The provided dataset, named *df* in the Python source code, contains traffic traces collected during the execution of some of the most common DDoS attacks on the 1^{st} of December 2018 from $09:17:11.18$ to $13:34:27.98$.

There are 64249 rows (objects) with 88 columns (features). Each row is referred to a flow (sequence of packets exchanged between a source and a destination IP). The provided dataset is classified in 12 labels: 11 related to DDoS attacks. The last one is benign and specifies that the labelled flow does not belong to any attack.

Analysing the dataset, it can be seen that there are no null features and there are 12 features that have a unique value. Those features will be removed as they do not contribute in anyway to the analysis of the dataset: their standard deviation is equal to 0. Furthermore, the features in the dataset are divided into categorical and numerical, respectively 4 and 83.

2.2 Label analysis

Following a comprehensive dataset analysis, attention has been directed towards characterizing the behavior of the labels. Upon aggregating the dataset into two categories, malicious and benign traffic, it becomes evident that the malicious category constitutes a predominant portion, accounting for 91.2%, while benign traffic represents 8.8% of the dataset. Subsequently, a focused examination of attack frequencies has been conducted, revealing that the most prevalent attacks include *ddos_udp_lag*, *ddos_snmp*, and *ddos_ssdp*.

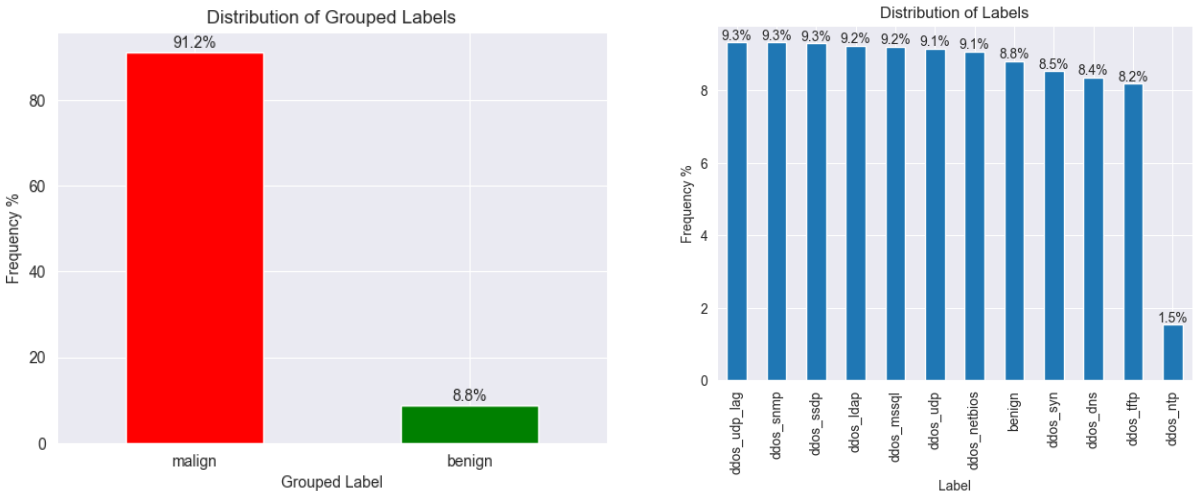

Figura 1: Distribution of malicious and benign traffic

2.3 Traffic level

In order to better understand how the traffic is distributed within the dataset, a comprehensive analysis was undertaken, focusing on different aspects such as protocols, flow volumes over time, and the distribution of source and destination addresses.

To enhance interpretability, a *protocol_mapping* was employed, translating numerical protocol codes into their corresponding protocol names. Subsequently, the distribution of protocols was visually represented, revealing that UDP is the most frequently used protocol, constituting 76.1% of the total.

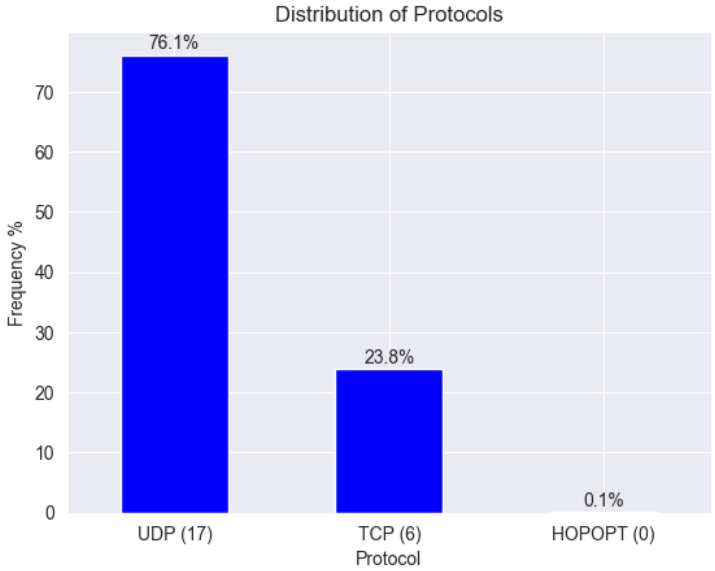

Figura 2: Distribution of protocols

A detailed examination of the number of flows over time was conducted by partitioning time into 30-second slots. The resulting plot displayed multiple peaks, indicative of time slots characterized by a significant surge in flow volume, aligning with the modus operandi of DDoS attacks involving numerous flows over short durations.

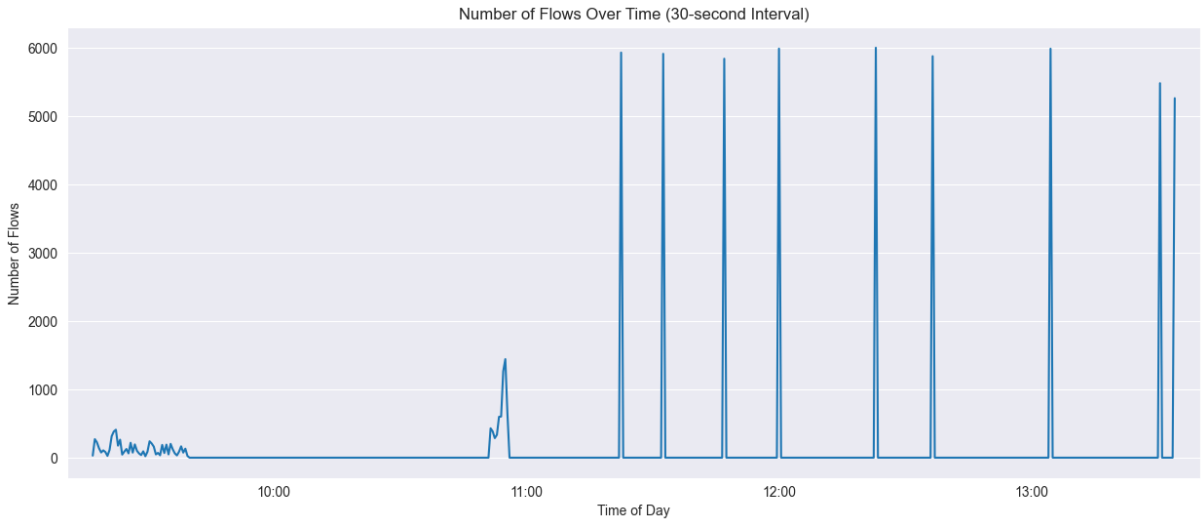

Figura 3: Number of flows over time

The investigation extended to the analysis of IP addresses, with histograms depicting the distribution of source and destination addresses. 172.16.0.5 emerged as the most frequently used source IP address, constituting 90.28%, while 192.168.50.1 dominated as the most prevalent destination IP address, comprising 90.29%. Why are those values so high compared to the others?

In order to answer to this question, an analysis on the malign traffic, with a focus on the destination and source IP addresses, was done. It is possible to see that the source IP associated to the malign traffic is 172.16.0.5, while the destination IP is 192.168.50.1. So, it can be expected that the first is the attacker and the second is the victim.

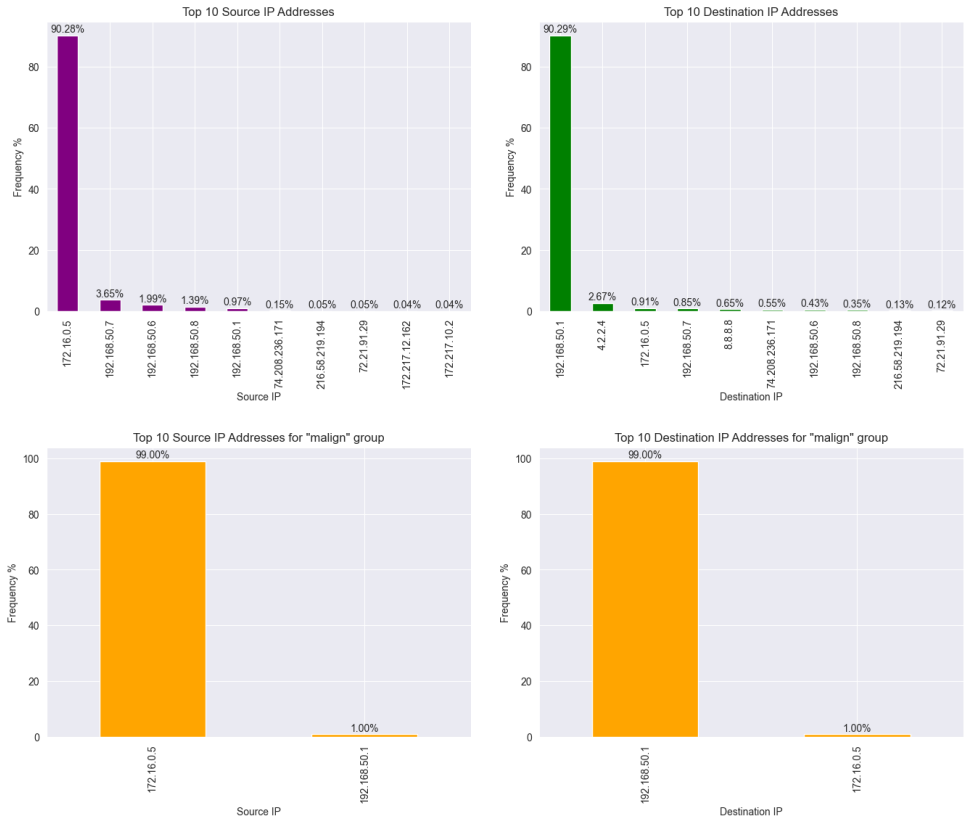

Figura 4: Distribution of source and destination IP addresses

Upon examinating the malign traffic, it became apparent that the source IP associated with malicious activities is 172.16.0.5, while the destination IP is 192.168.50.1. This observation suggests a potential adversarial context, where 172.16.0.5 may be identified as the attacker, and 192.168.50.1 as the victim.

To better prove this hypothesis, a renewed examination of the flow distribution over time was undertaken, specifically focusing on flows related to these two addresses. At this point it is interesting to understand if those peaks are related on the attacks performed.

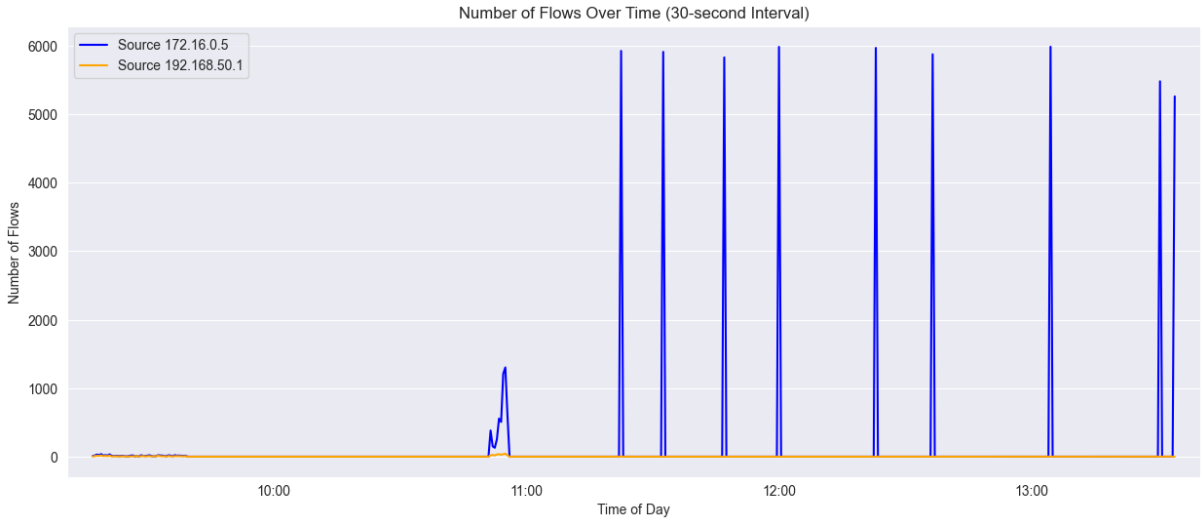

Figura 5: Number of flows over time related to the two addresses

The peaks in flow distribution were found to distinctly align with the different attacks, thus confirming the hypothesis as can be seen in the figure below.

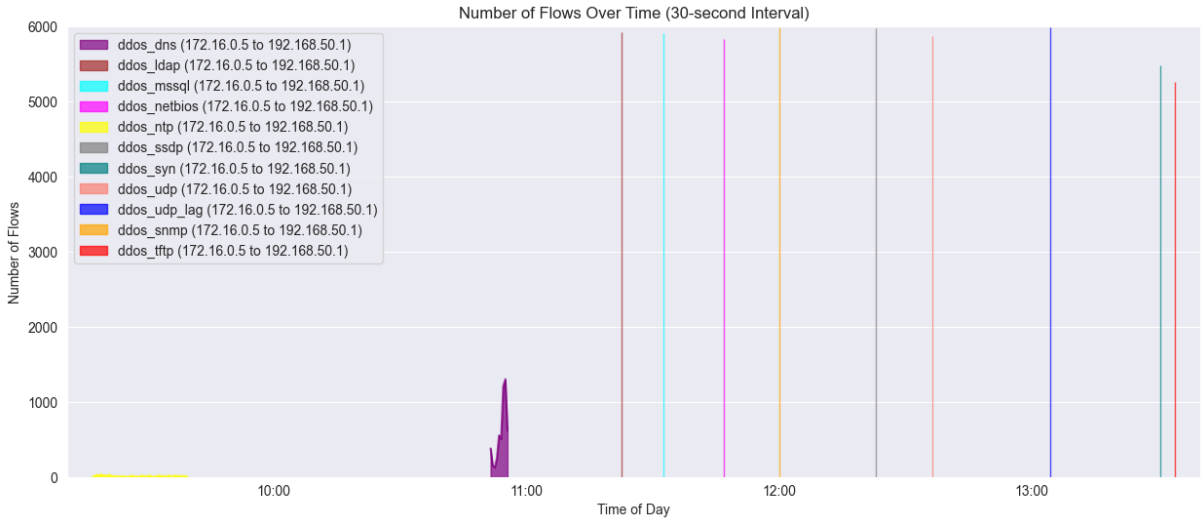

Figura 6: Number of flows over time considering the attacks

2.3.1 Flow analysis

An examination of packet distribution within a specific timeframe highlights the prominence of a particular flow identified by the *Flow_ID* $172.16.0.5 - 192.168.50.1 - 0 - 0 - 0$. Further analysis of this flow reveals its importance, by exhibiting a notably higher packet count. A subsequent in-depth analysis discerns that this particular flow is exclusively associated with malign activity. This observation underscores the significance of *Flow_ID* $172.16.0.5 - 192.168.50.1 - 0 - 0 - 0$ within the dataset, emphasizing its role as a prominent contributor to the overall malicious traffic.

	Flow_ID	Total_Fwd_Packets	Timestamp
5984	172.16.0.5-192.168.50.1-0-0-0	85894	2018-12-01 11:22:40.254769
11913	172.16.0.5-192.168.50.1-0-0-0	60959	2018-12-01 11:32:32.915441
35635	172.16.0.5-192.168.50.1-0-0-0	49476	2018-12-01 12:23:13.663425
47603	172.16.0.5-192.168.50.1-0-0-0	4360	2018-12-01 13:30:30.750958
17832	172.16.0.5-192.168.50.1-0-0-0	4114	2018-12-01 11:47:08.465544
5972	172.16.0.5-192.168.50.1-0-0-0	2768	2018-12-01 10:55:44.590386
1588	172.16.0.5-192.168.50.1-0-0-0	1942	2018-12-01 10:53:40.214163
3433	172.16.0.5-192.168.50.1-0-0-0	1290	2018-12-01 10:54:47.671297
17	172.16.0.5-192.168.50.1-0-0-0	1014	2018-12-01 10:51:40.212271
26719	172.217.9.230-192.168.50.7-443-50904-6	916	2018-12-01 09:26:27.501955

Figura 7: Top 10 Flow_ID with most Total_Fwd_Packets

Flow_ID	Grouped_Label	Total_Fwd_Packets
172.16.0.5-192.168.50.1-0-0-0	malign	212815
172.217.9.230-192.168.50.7-443-50904-6	benign	926
192.168.50.253-224.0.0.5-0-0-0	benign	684
192.168.50.7-125.56.201.90-50700-80-6	benign	673
172.217.12.198-192.168.50.7-443-50529-6	benign	553
192.168.50.7-125.56.201.112-50765-80-6	benign	537
192.168.50.7-8.253.140.119-50687-80-6	benign	537
192.168.50.254-224.0.0.5-0-0-0	benign	513
192.168.50.7-216.58.219.194-50944-443-6	benign	499
172.217.3.102-192.168.50.7-443-50604-6	benign	446

Figura 8: Top 10 Flow_ID grouped by label with most Total_Fwd_Packets

2.3.2 Port analysis

Upon an extensive analysis of the dataset, a focus was directed towards comprehending the behavior of source ports. Utilizing a violin plot it is revealed that the preeminent source ports predominantly fall within the numerical range of 0 to approximately 10000. Remarkably, this range aligns with the well-known ports, signifying that the most frequently employed source ports in the dataset correspond to those conventionally reserved for standard services and applications.

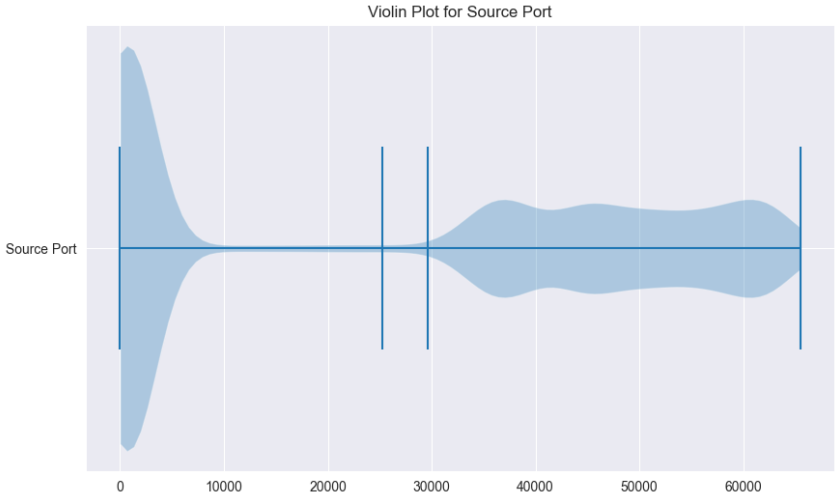

Figura 9: Violin plot for source port

At this point, an analysis on the distribution on the source ports in the different attacks can be done using a boxplot. It shows that the median port number for benign traffic is much higher than the median port number for any of the DDoS attack types. This suggests that benign traffic is more likely to use a wider range of port numbers, while DDoS attacks tend to use a smaller number of ports.

The DDoS attack types that use the highest port numbers are *ddos_udp_lag*, *ddos_udp*, and *ddos_syn*. These attacks are all UDP-based DDoS attacks, which are a type of attack that floods a target with UDP packets. The DDoS attack types that use the lowest port numbers are *ddos_dns*, *ddos_ldap*, *ddos_mssql*, and *ddos_netbios*. These attacks are all TCP-based DDoS attacks, which are a type of attack that floods a target with TCP SYN packets.

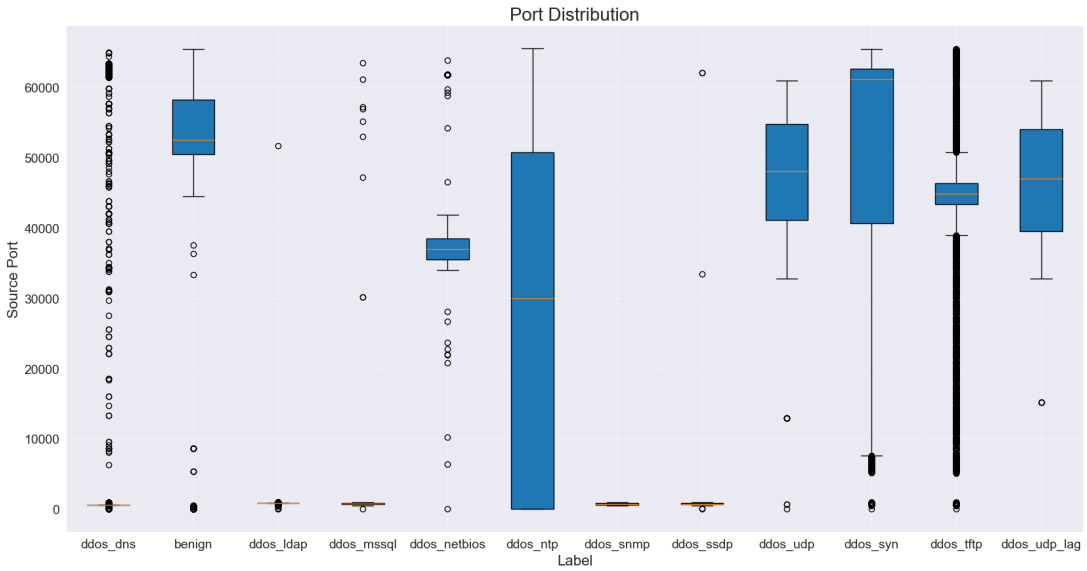

Figura 10: Boxplot for source port distribution

In contrast to the distinct pattern observed in the distribution of source ports, the analysis of destination ports reveals a more uniform distribution (Figure 11). The resulting graph, particularly depicted through a boxplot in Figure 12, underscores the lack of discernible patterns that could aid in the recognition of different attack types.

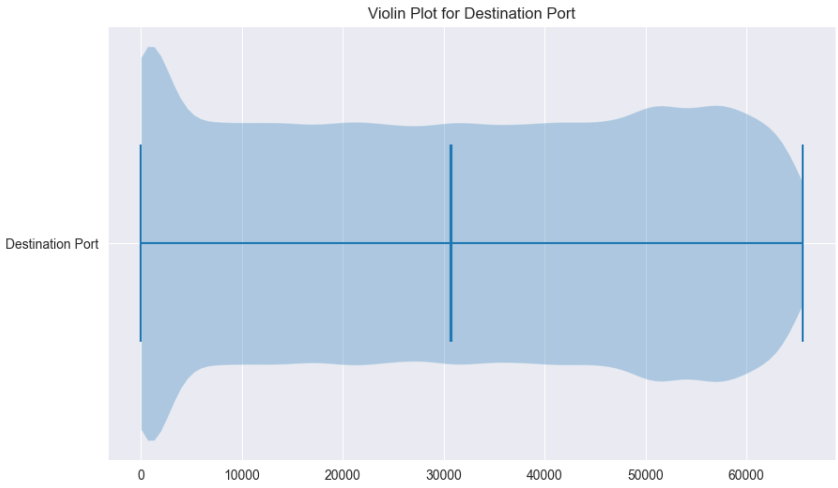

Figura 11: Violin plot for destination port

Furthermore, the boxplot analysis draws attention to an intriguing aspect: benign traffic demonstrates an association with lower destination ports. This behavior aligns with the broader observation that benign activities tend to utilize ports within a specific range. Notably, a similar pattern is discerned in the case of the *ddos_ntp* attack, highlighting the notion that this particular attack type exhibits a similar port-related behavior as benign traffic.

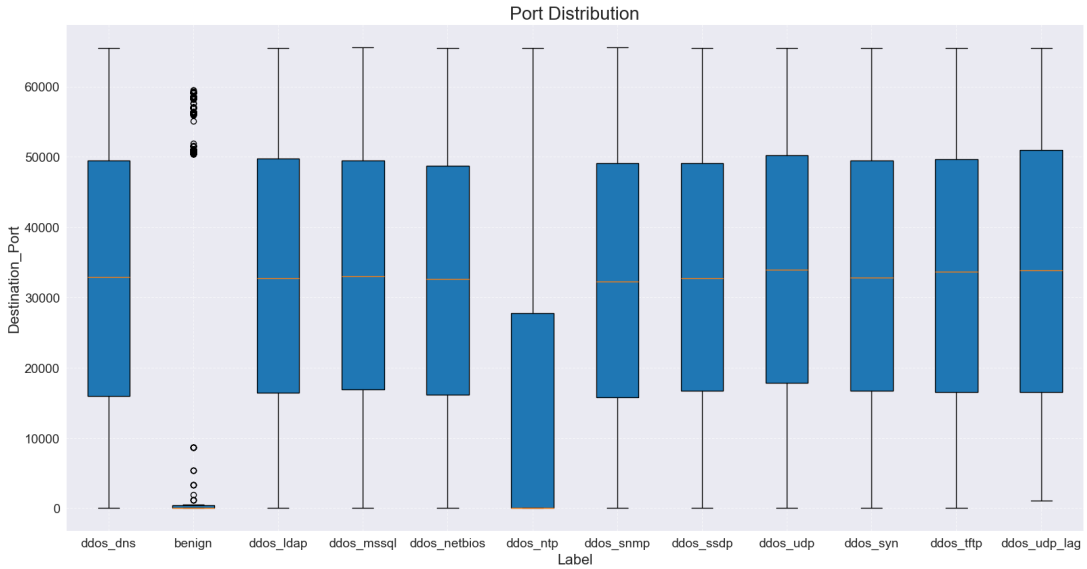

Figura 12: Boxplot for destination port distribution

To enhance the comprehension of each flow's behavior within the dataset, a set of new features has been introduced, increasing the total number of columns to 94. These newly incorporated features provide statistical insights into key attributes, including 'Total_Fwd_Packets', 'Total_Backward_Packets', 'Flow_Duration', and 'Flow_Packets/s'. The additional features encompass the mean value, maximum value, minimum value, and quantile information for each of these attributes.

2.4 Correlation analysis

Following the analysis of network traffic, the subsequent step involves conducting a correlation analysis on the dataset features. To facilitate this analysis, preliminary steps include the encoding of categorical features by converting them into numerical values. The encoded labels are meticulously stored in a dedicated DataFrame denoted as *df_label* in order to better administrate them in the following steps, while the other categorical features remain within the original dataset, referred to as *df*. Additionally, for a standardized representation of time, the timestamp data has been transformed into integer values, representing time in seconds relative to a predefined reference date ('$2018-12-01, 10:51:39.813448$').

In preparation for the standardization process, the distinct measurement units associated with various features necessitate a preliminary step. The labels have been removed from the dataset. Subsequently, the standardization procedure has been executed, resulting in the creation of a new DataFrame designated as *feature_new*.

The creation of a correlation matrix (Figure 13) has shown that certain features exhibit a high degree of correlation with one another, even considering its extensive array of features. This observation underscores the interdependencies and relationships that exist among specific attributes.

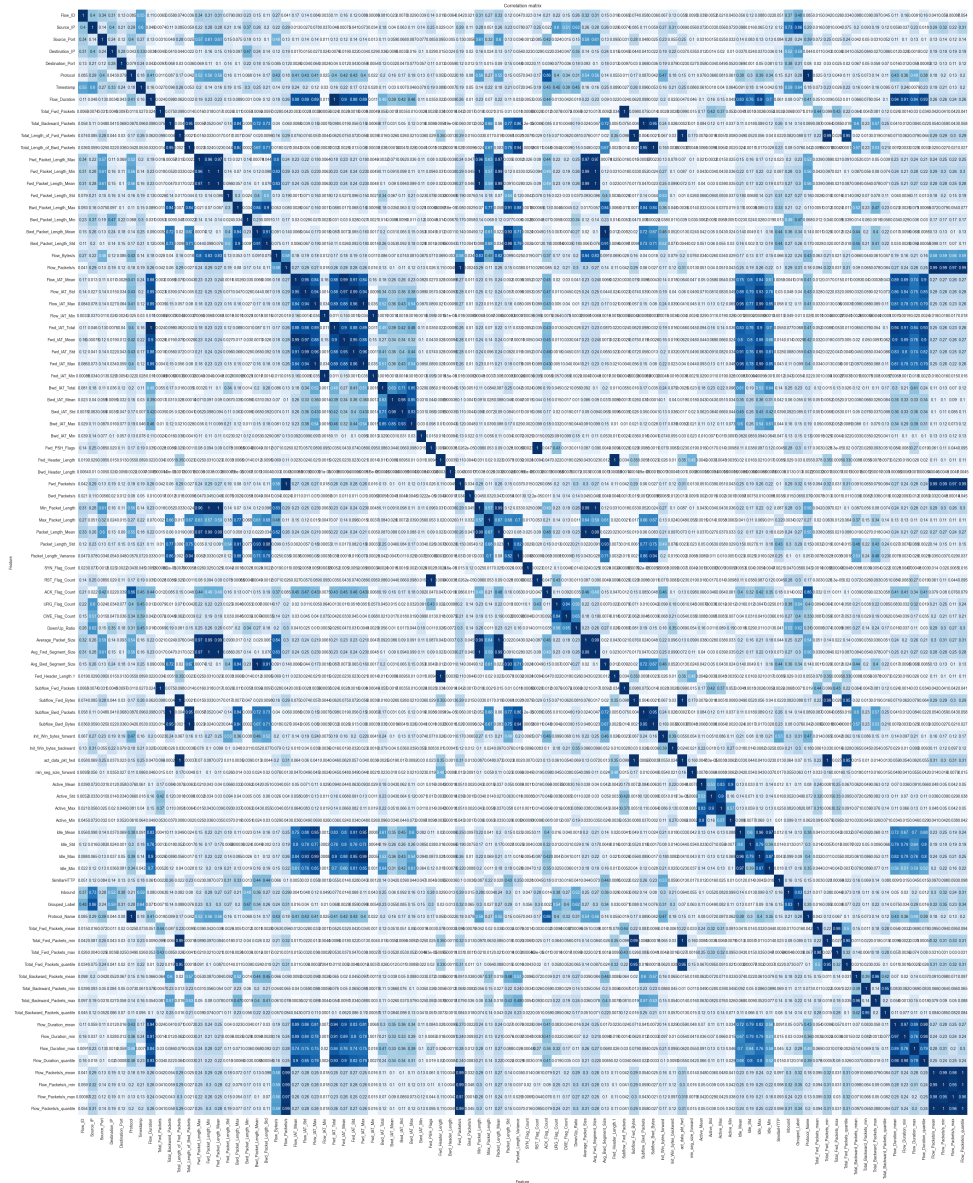

Figura 13: Correlation matrix

Upon analysing the correlation matrix, it has been decided to removing features exhibiting a high correlation coefficient, specifically those exceed the threshold of 0.8. This thresholding process resulted in the elimination of 53 features. Consequently, a refined DataFrame, denoted as *newdf*, has been created, encompassing only the remaining features that satisfy to the defined correlation threshold. This strategic reduction not only optimizes computational efficiency but also preserves the most informative and distinct attributes. This new dataset will be composed of 39 features.

Based on the obtained DataFrame, a new correlation matrix can be computed (Figure 14).

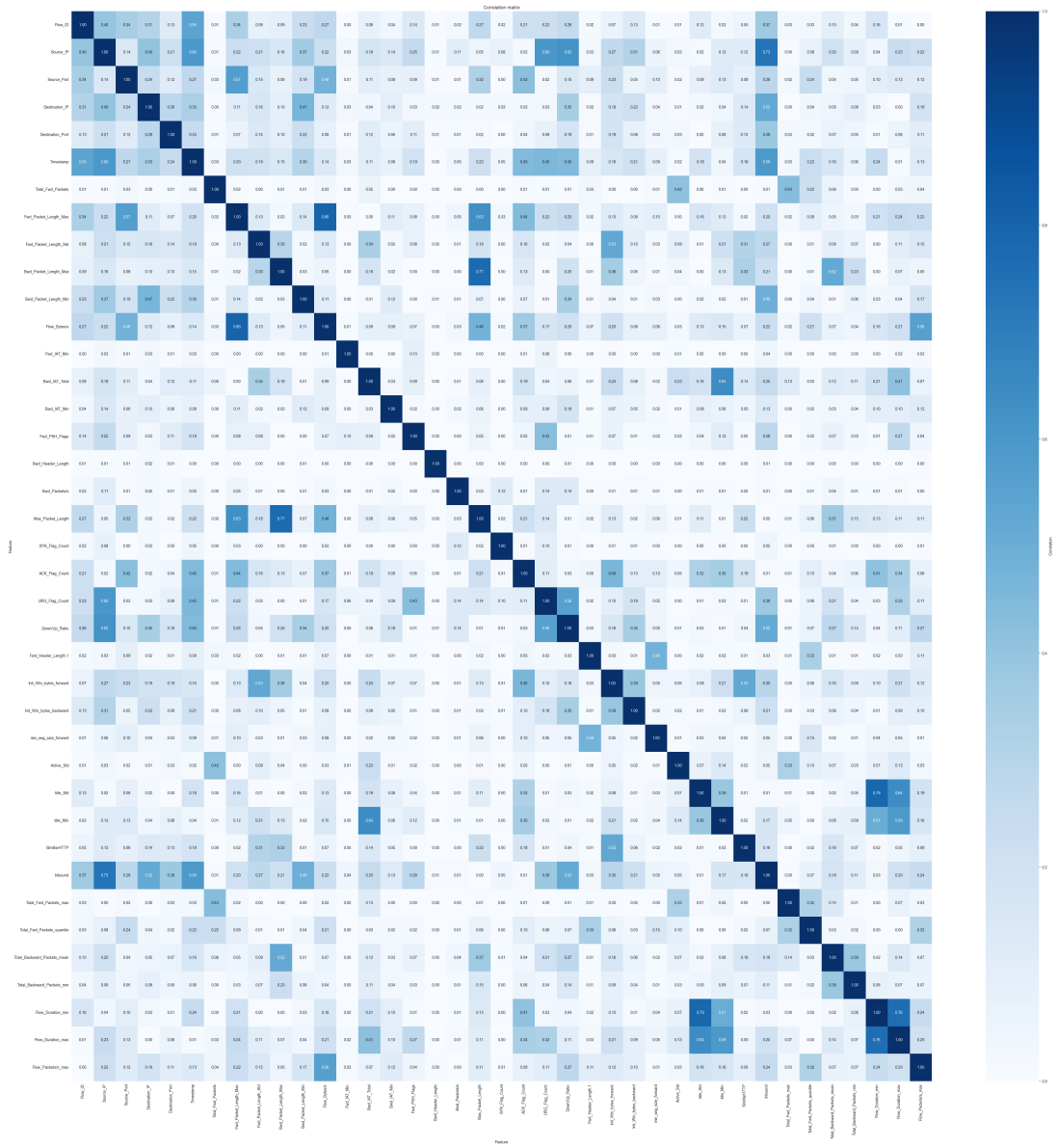

Figura 14: Correlation matrix of the reduced DataFrame

2.4.1 Principal Component Analysis

Principal Component Analysis (PCA) emerges as a pivotal technique in the realm of data analysis, offering a powerful means to split complex information and uncover latent structures within a dataset. The primary objective of PCA is to transform a high-dimensional dataset into a lower-dimensional representation, known as principal components, while retaining as much of the original variance as possible.

The PCA will be performed on the scaled dataframe (*newdf*) and Kaiser's rule will be used to detect the right number of principal components. The Kaiser's rule is an heuristic guideline that suggests

retaining principal components with eigenvalues greater than 1 and discarding those with eigenvalues less than or equal to 1. The idea is that eigenvalues less than 1 contribute less variance than a single original variable, and by retaining components with eigenvalues greater than 1, it is preserved more information than any individual variable. According to the described method, the number of PCs selected is 13.

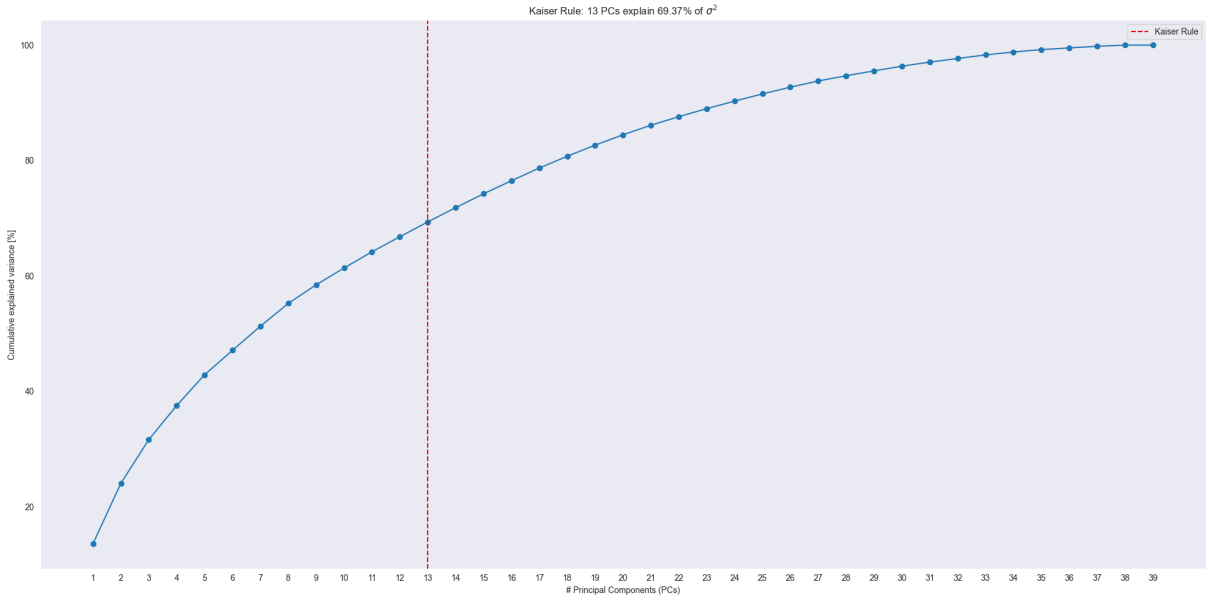

Figura 15: Principal Component Analysis

Since PCs are linear combination of the original features and the first are the most relevant ones, it is interesting to show how they are composed. In the tables below it is possible to see the most important features involved along with their respective weights.

Top 5 features for PC0:

	PC0_Loading
Inbound	0.338132
Source_IP	0.328715
Down/Up_Ratio	0.274838
Init_Win_bytes_forward	0.235626
Flow_Duration_max	0.221426

Top 5 features for PC1:

	PC1_Loading
Timestamp	0.350500
Flow_Duration_min	0.322639
ACK_Flag_Count	0.316398
Flow_ID	0.301980
Idle_Std	0.280771

Figura 16: Top 5 for PC0 and PC1

Following careful consideration, a decision has been made to employ the reduced DataFrame (*newdf*) rather than the Principal Components (PCs) in the upcoming analytical steps. This choice comes from the will to enhance the interpretability and clarity of the results. While PCs encapsulate key patterns within the data, their representation might not lend itself to a straightforward and meaningful visualization.

3 Supervised learning - Classification

In this section, a supervised learning procedure will be performed in order to classify the different flows according to the attacks.

The preliminary step involves the splitting of the whole dataset into training (X) and testing (X_test) set. This stratified partitioning is executed with respect to the labels, ensuring that the distribution of attack types is representative in both the training and testing subsets. The division of the dataset follows a predetermined ratio, allocating 70% of the data to the training set (X) and the remaining 30% to the testing set (X_test). In addition, the training set is further split into training (X_train) and validation (X_val) set.

```
X, X_test, y, y_test = train_test_split(newdf, y, stratify=y, train_size=0.7,
↪  random_state=15)

X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    stratify = y,
    train_size = 0.5/0.7,
    random_state = 15
)
```

Figura 17: Script for the splitting of the dataset

3.1 Supervised learning algorithms

The classification will be performed by means of three different supervised learning algorithm:

- Random Forest (RF)
- Support Vector Machine (SVM)
- K-Nearest Neighbour (K-NN)

For each algorithm, default parameters for the classifiers are used and performances are evaluated by means of confusion matrix, precision, recall and F1 score.

3.1.1 Random Forest

Random Forest is a learning algorithm used for classification and regression tasks. It builds multiple decision trees during training and merges their predictions to improve accuracy and prevent overfitting.

```
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
y_train_pred_rf = rf_clf.predict(X_train)
y_val_pred_rf = rf_clf.predict(X_val)
```

Figura 18: Creation of RandomForestClassifier

After the fit and predict operations, the results are shows by the following classification report. The graph shows that the metrics of the RandomForestClassifier are almost perfect.

Training set

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2829
1	1.00	1.00	1.00	2684
2	1.00	1.00	1.00	2963
3	1.00	1.00	1.00	2956
4	1.00	1.00	1.00	2915
5	1.00	1.00	1.00	493
6	1.00	1.00	1.00	2992
7	1.00	1.00	1.00	2985
8	1.00	1.00	1.00	2740
9	1.00	1.00	1.00	2631
10	1.00	1.00	1.00	2938
11	1.00	1.00	1.00	2993
accuracy			1.00	32119
macro avg	1.00	1.00	1.00	32119
weighted avg	1.00	1.00	1.00	32119

Validation set

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1132
1	0.99	1.00	1.00	1074
2	1.00	1.00	1.00	1186
3	1.00	1.00	1.00	1182
4	1.00	1.00	1.00	1166
5	0.99	0.98	0.99	197
6	1.00	1.00	1.00	1197
7	1.00	1.00	1.00	1194
8	1.00	1.00	1.00	1096
9	1.00	1.00	1.00	1052
10	1.00	1.00	1.00	1175
11	1.00	1.00	1.00	1197
accuracy			1.00	12848
macro avg	1.00	1.00	1.00	12848
weighted avg	1.00	1.00	1.00	12848

Figura 19: Precision, recall and F1 score for RF

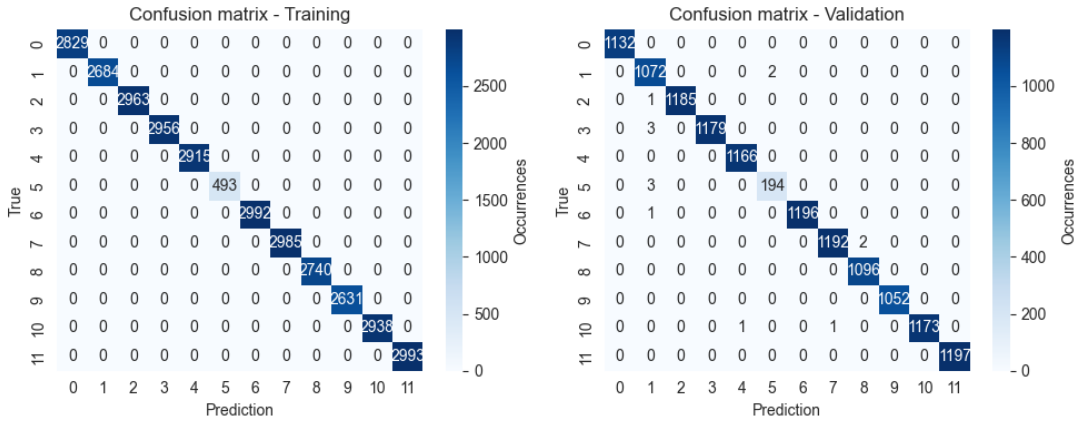

Figura 20: Confusion matrix for RF

3.1.2 Support Vector Machine

Support Vector Machines (SVM) is an algorithm used for classification and regression. It seeks an optimal hyperplane to separate data points, maximizing the margin between classes.

```
svm_classifier = SVC()
svm_classifier.fit(X_train, y_train)
svm_train_predictions = svm_classifier.predict(X_train)
svm_val_predictions = svm_classifier.predict(X_val)
```

Figura 21: Creation of SVC

After the fit and predict operations, the results are shown by the following classification report. The graphs show that the metrics of the SVC are good but not like the RF.

Training set

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2829
1	0.99	0.98	0.99	2684
2	0.96	0.99	0.98	2963
3	0.99	0.97	0.98	2956
4	1.00	1.00	1.00	2915
5	0.97	1.00	0.98	493
6	1.00	1.00	1.00	2992
7	1.00	1.00	1.00	2985
8	0.96	0.98	0.97	2740
9	0.98	0.95	0.97	2631
10	1.00	1.00	1.00	2938
11	1.00	1.00	1.00	2993
accuracy			0.99	32119
macro avg	0.99	0.99	0.99	32119
weighted avg	0.99	0.99	0.99	32119

Validation set

	precision	recall	f1-score	support
0	0.99	1.00	1.00	1132
1	0.99	0.98	0.98	1074
2	0.97	0.99	0.98	1186
3	1.00	0.97	0.99	1182
4	1.00	1.00	1.00	1166
5	0.96	0.97	0.97	197
6	1.00	1.00	1.00	1197
7	1.00	1.00	1.00	1194
8	0.95	0.98	0.96	1096
9	0.98	0.95	0.96	1052
10	1.00	1.00	1.00	1175
11	1.00	1.00	1.00	1197
accuracy			0.99	12848
macro avg	0.99	0.99	0.99	12848
weighted avg	0.99	0.99	0.99	12848

Figura 22: Precision, recall and F1 score for SVM

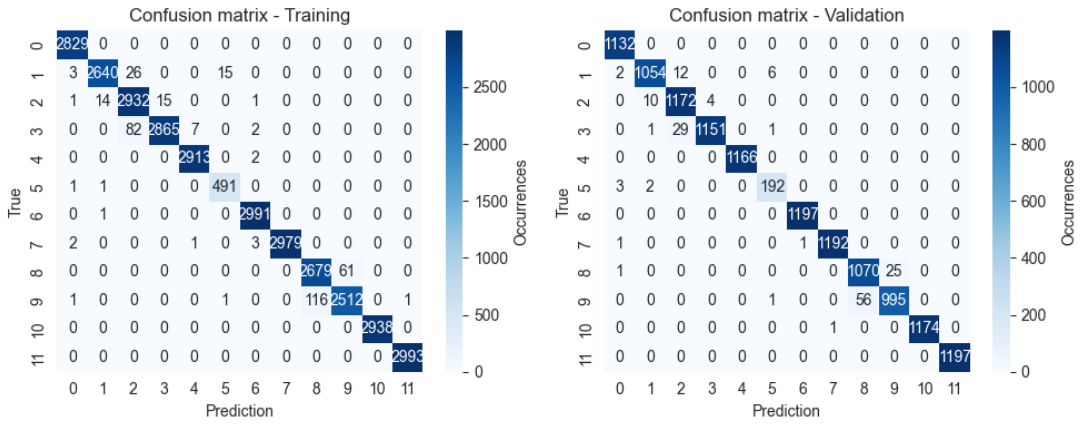

Figura 23: Confusion matrix for SVM

3.1.3 K-Nearest Neighbour

k-Nearest Neighbors (KNN) is an algorithm for classification and regression tasks. It operates on the principle of proximity, making predictions based on the majority class or average of nearby data points in the feature space.

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
y_train_pred_knn = knn_clf.predict(X_train)
y_val_pred_knn = knn_clf.predict(X_val)
```

Figura 24: Creation of KNeighborsClassifier

After the fit and predict operations, the results are shown by the following classification report. The graphs show that the metrics of the KNeighborsClassifier are good but not like the others.

Training set

	precision	recall	f1-score	support
0	0.99	1.00	1.00	2829
1	0.99	0.98	0.98	2684
2	0.98	0.98	0.98	2963
3	0.98	0.98	0.98	2956
4	1.00	1.00	1.00	2915
5	0.95	0.97	0.96	493
6	1.00	1.00	1.00	2992
7	1.00	1.00	1.00	2985
8	0.99	0.99	0.99	2740
9	0.99	0.99	0.99	2631
10	1.00	1.00	1.00	2938
11	1.00	1.00	1.00	2993
accuracy			0.99	32119
macro avg	0.99	0.99	0.99	32119
weighted avg	0.99	0.99	0.99	32119

Validation set

	precision	recall	f1-score	support
0	0.99	1.00	1.00	1132
1	0.99	0.97	0.98	1074
2	0.97	0.96	0.97	1186
3	0.96	0.97	0.97	1182
4	1.00	1.00	1.00	1166
5	0.92	0.96	0.94	197
6	1.00	1.00	1.00	1197
7	1.00	0.99	1.00	1194
8	0.99	0.99	0.99	1096
9	0.98	1.00	0.99	1052
10	1.00	1.00	1.00	1175
11	1.00	1.00	1.00	1197
accuracy			0.99	12848
macro avg	0.98	0.99	0.98	12848
weighted avg	0.99	0.99	0.99	12848

Figura 25: Precision, recall and F1 score for K-NN

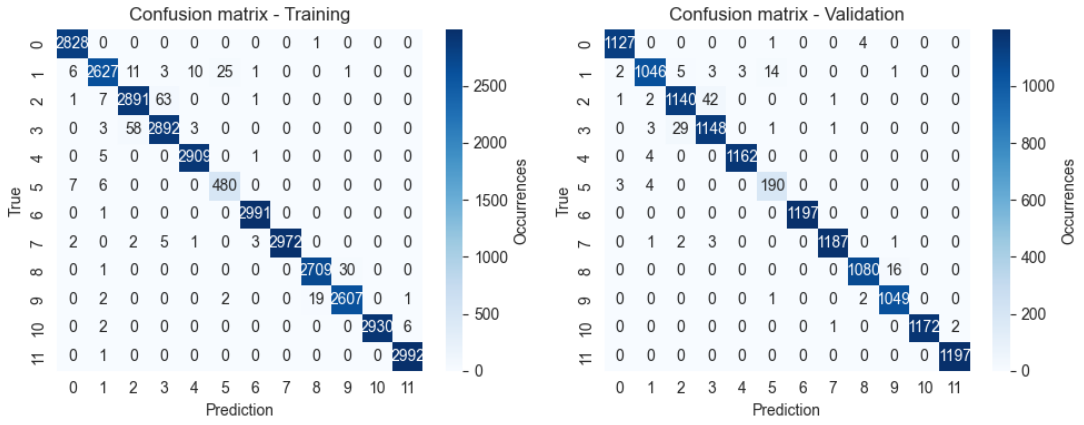

Figura 26: Confusion matrix for K-NN

3.2 Hyper-parameters tuning

Analysing the confusion matrices and classification report, it is possible to understand if underfitting or overfitting occur. Underfitting is characterized by low accuracy, recall, and F1-score on both training and validation sets, accompanied by high off-diagonal elements in the confusion matrix for the training set. On the other hand, overfitting manifests as high accuracy, precision, recall, and F1-score on the training set but lower values on the validation set, accompanied by high diagonal elements in the confusion matrix for the training set.

Based on those statements, it is possible to declare that neither underfitting nor overfitting occur.

Further analysis can be done by tuning the hyper-parameters in order to get the most efficient configuration for the three models previously introduced. The validation curves, depicted in Figure 27, align with the earlier assessments. Notably, both the training score and validation score do not exhibit characteristics indicative of underfitting or overfitting. The absence of low training scores implies a well-fitted model to the training data, while the high validation scores signify robust generalization to unseen data.

The performance indicators exhibit a balanced representation, suggesting that the model adequately captures patterns within the training data while generalizing effectively to the validation set.

Following the previous analysis, it is possible to identify the best model as the Random Forest. The value chosen for the Random Forest hyper-parameter is 10. It is the one that will be used in the following steps.

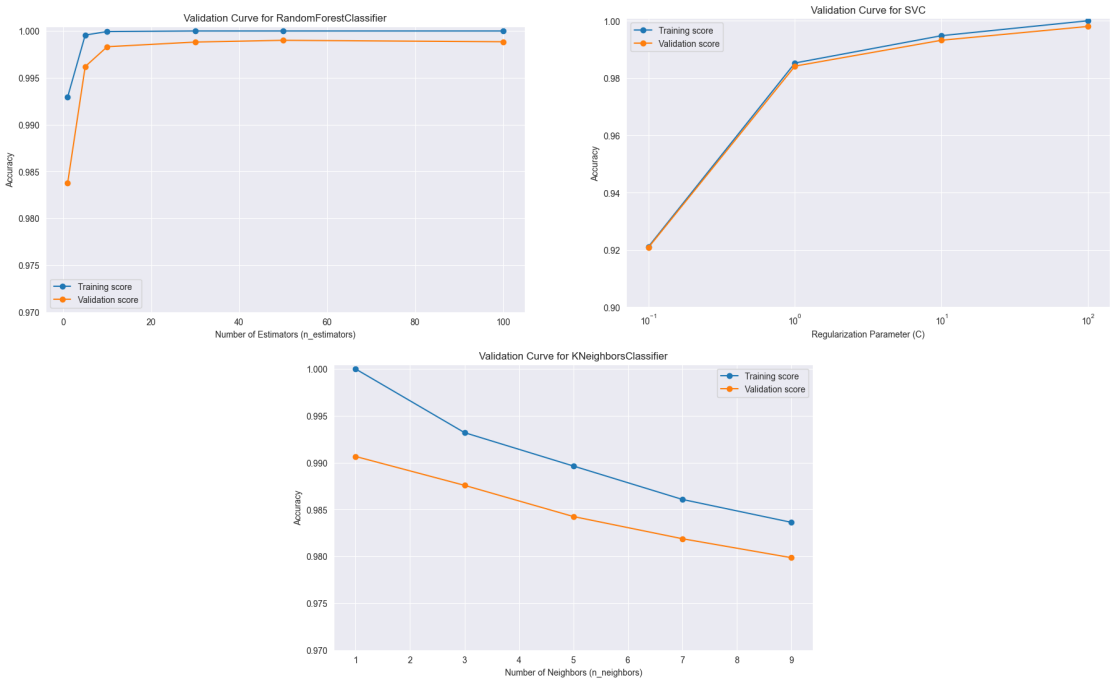

Figura 27: Validation curve for RF, SVM and KNN

3.3 False Positive and False Negative inspection

An examination of misclassifications has been conducted, involving the assessment of False Positive and False Negative assignments for each class. Remarkably, the findings reveal an infrequent occurrence of misclassifications across all classes. The model demonstrates a high level of accuracy in distinguishing between true positive instances and minimizing false assignments.

This robust performance in terms of misclassifications further reinforces the effectiveness of the model. The scarcity of False Positives and False Negatives underscores the model's precision and recall capabilities, indicating its proficiency in correctly identifying instances and minimizing errors.

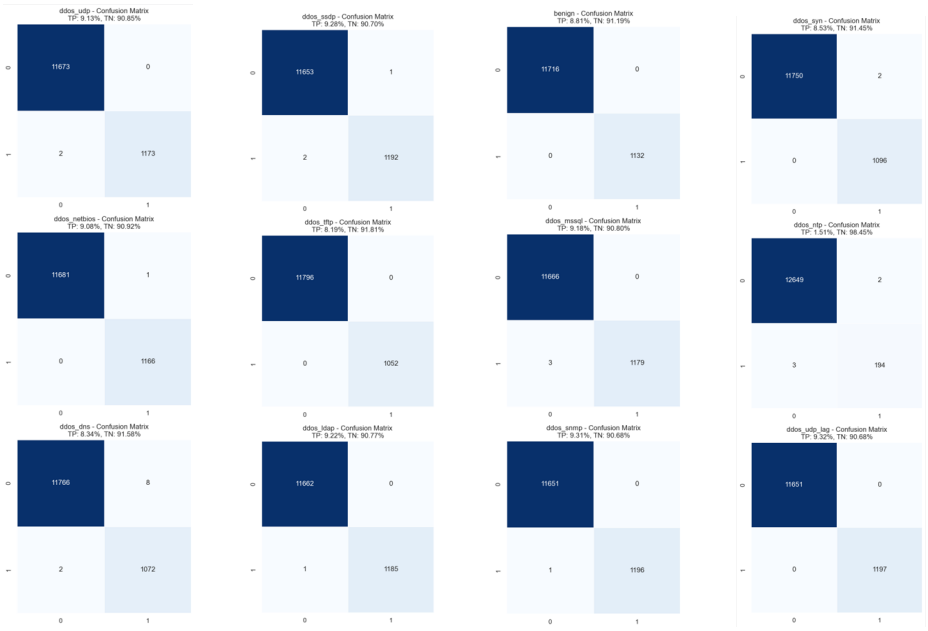

Figura 28: Confusion matrix in terms of labels

The ROC curve, shown in Figure 29, serves as further confirmation of the model's robust performance. The curve's trend to the upper-left corner, together with a high Area Under the Curve (AUC) score, attests to the model's ability to discriminate between classes effectively.

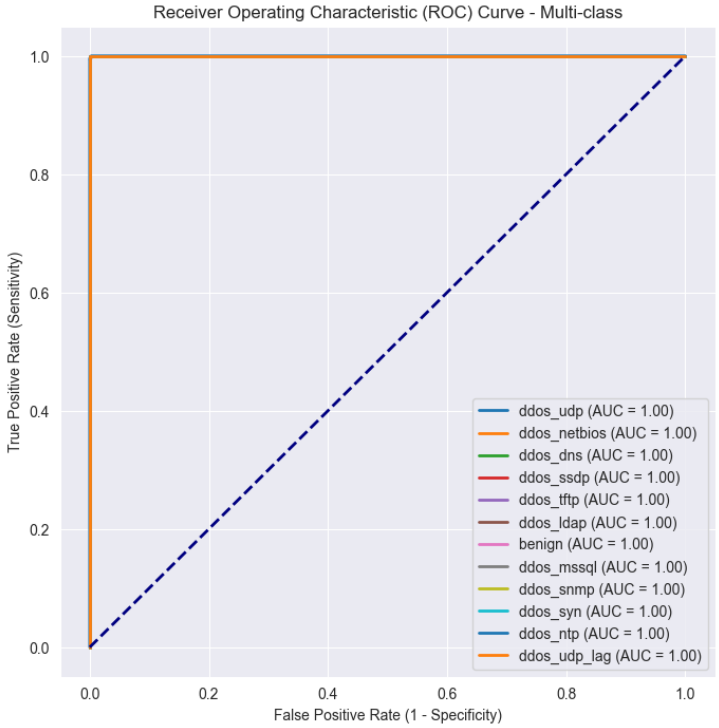

Figura 29: ROC for Random Forest

Finally, the feature importance analysis indicates that the feature contributing the most to misclassification is the *Timestamp*. This suggests that the model may not effectively utilize temporal information to distinguish between the two types of flows.

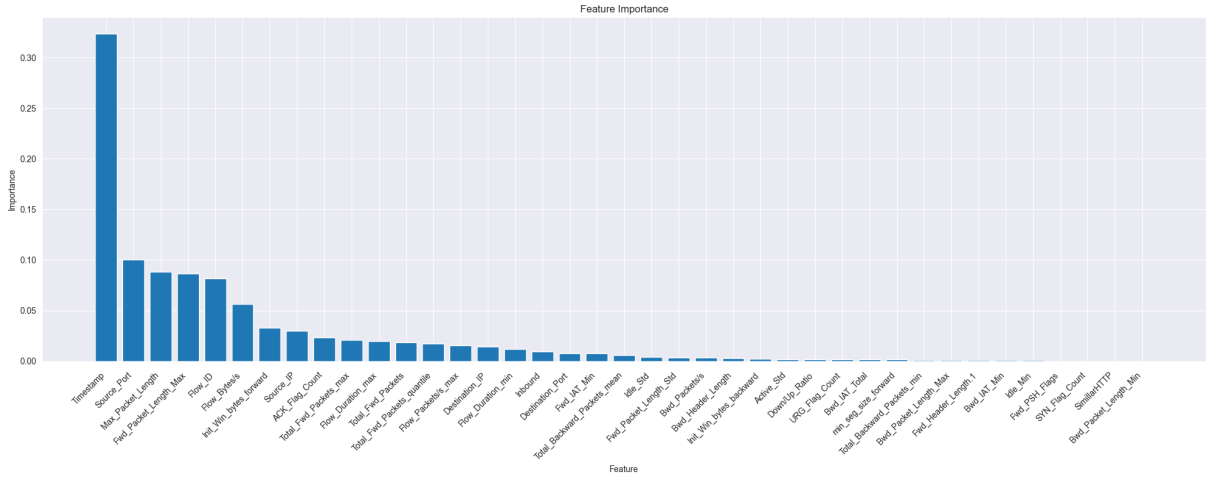

Figura 30: Feature importance analysis related to missclassifications

```python
feature_names = newdf.columns

feature_importances = rf_clf.feature_importances_

sorted_feature_indices = np.argsort(feature_importances)[::-1]

sorted_feature_names = feature_names[sorted_feature_indices]

plt.figure(figsize=(20, 8))
plt.bar(range(X_train.shape[1]), feature_importances[sorted_feature_indices])
plt.xticks(range(X_train.shape[1]), sorted_feature_names, rotation=45,
 ↪  ha='right')
plt.tick_params(axis='x', which='major', pad=8)

plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance')
plt.tight_layout()
```

Figura 31: Code for missclassification

3.4 Testing performance

Following the training phase with the Random Forest model, a testing phase was conducted using the selected number of estimators (*n_estimators = 10*). The outcomes of the testing phase are summarized in the following visuals. The testing results confirm the performance observed during the training phase.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1697
1	1.00	1.00	1.00	1611
2	1.00	1.00	1.00	1779
3	1.00	1.00	1.00	1773
4	1.00	1.00	1.00	1749
5	0.99	1.00	0.99	296
6	1.00	1.00	1.00	1795
7	1.00	1.00	1.00	1791
8	1.00	1.00	1.00	1644
9	1.00	1.00	1.00	1578
10	1.00	1.00	1.00	1763
11	1.00	1.00	1.00	1796
accuracy			1.00	19272
macro avg	1.00	1.00	1.00	19272
weighted avg	1.00	1.00	1.00	19272

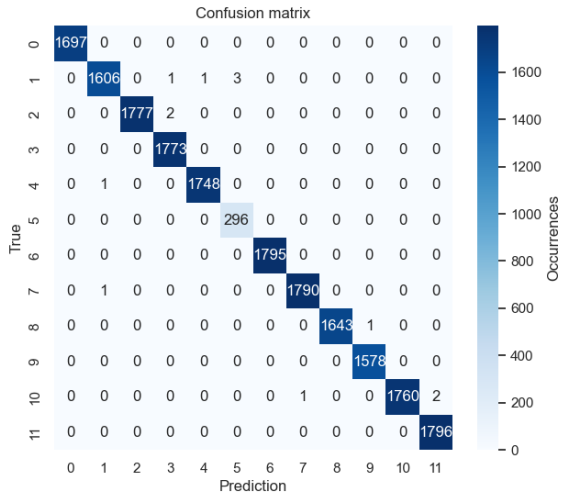

Figura 32: Test performance

Such high performance could be due to the use of categorical features such as Flow_ID. In fact, since a few ports dominate over the entire distribution of senders and receivers, it is enough to know that a Flow_ID has made malicious attacks to always categorize it in the same way. While this information is interesting because it means that having a memory of past attacks significantly improves classification, it can also be considered as overfitting, because the model is not actually extracting information from the data but is only storing it. This kind of overfitting is not visible from the split you made because there should only be new flow-ids in the test set.

4 Unsupervised learning - Clustering

In this section, alternative machine learning methodologies will be employed that operate independently from the labels. Such methodologies are commonly categorized as *unsupervised* learning techniques, designed to cluster datasets based solely on their inherent structure and characteristics.

From now on, the dataset previously subjected to dimensionality reduction and scaling, denoted as *newdf*, will be referred as *newdf_cluster* for the purpose of clustering analysis.

At first glance, it is interesting to visualize the distribution of data along the axes of the two most significant principal components.

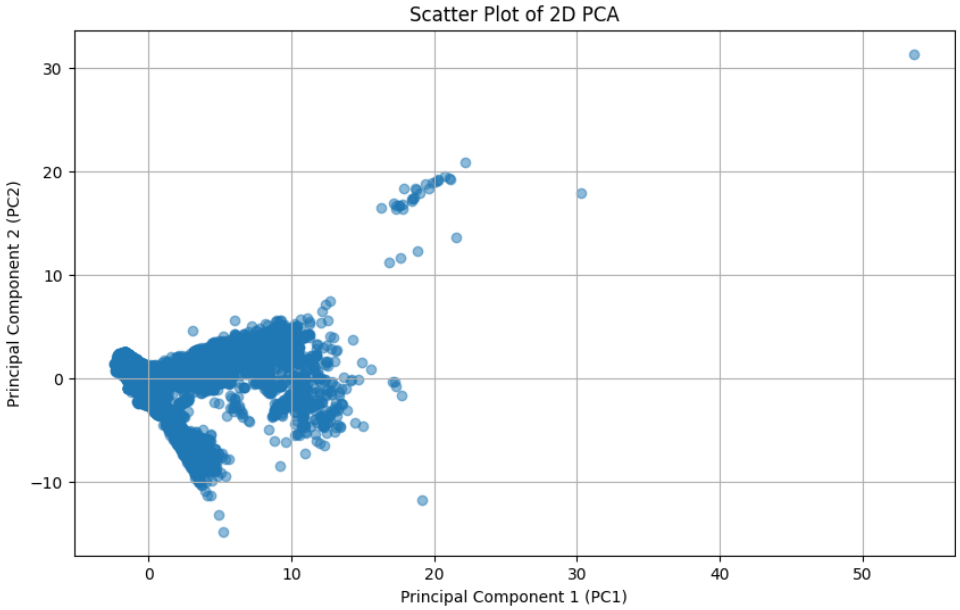

Figura 33: Data distribution of samples by means of 2 PCs

From the scatter plot, it is evident that the structure delineated by the primary two principal components is non-Euclidean in nature. Consequently, traditional clustering algorithms such as k-means and Gaussian Mixture Models (GMM) are not the best choice for this dataset. A valid alternative could be DBSCAN; however, its implementation entails considerable computational time, which may be prohibitive for practical use. For this reason K-means and GMM will be used.

4.1 K-means

K-means represents a hard clustering technique wherein each sample is exclusively associated with a single cluster. This method partitions a set of m observations into k clusters, where each observation is assigned to the cluster whose mean (cluster centroid) is closest to it.

As the initial step, the behavior of K-means was examined by dividing the data into 12 clusters, corresponding to the number of labels present, using default hyperparameters. This process involved first computing the centroids and subsequently assessing several metrics such as *silhouette score*, *adjusted rand index*, and *rand index*.

```
k-Means with 12 clusters
Size of each cluster:  [  603 18690  8646  2316  2632   586    18 27457     1    26  2255  1009]
k_means clustering error: 1198275.0
Silhouette: 0.31
RI: 0.77
ARI: 0.3
```

Figura 34: K-means with random parameters

Looking at the results in Figure 34, this initial configuration is not optimal due to the low silhouette score and high clustering error observed. Hence, it becomes imperative to determine the most appropriate number of clusters and the optimal hyperparameter values.

4.1.1 Determining the number of clusters via Silhouette Analysis and Elbow Method

Silhouette analysis assesses the similarity of an object to its own cluster (cohesion) relative to other clusters (separation). The silhouette score ranges from -1 to 1, where higher values indicate more distinct clusters. Meanwhile, the elbow method entails executing the KMeans algorithm across a range of cluster numbers and plotting the sum of squared distances (inertia) against the number of clusters. The point at which the decrease in inertia decelerates (forming an "elbow" in the plot) is often considered a reliable estimate for the optimal number of clusters.

Both methods can be utilized to choose the appropriate number of clusters for a given dataset.

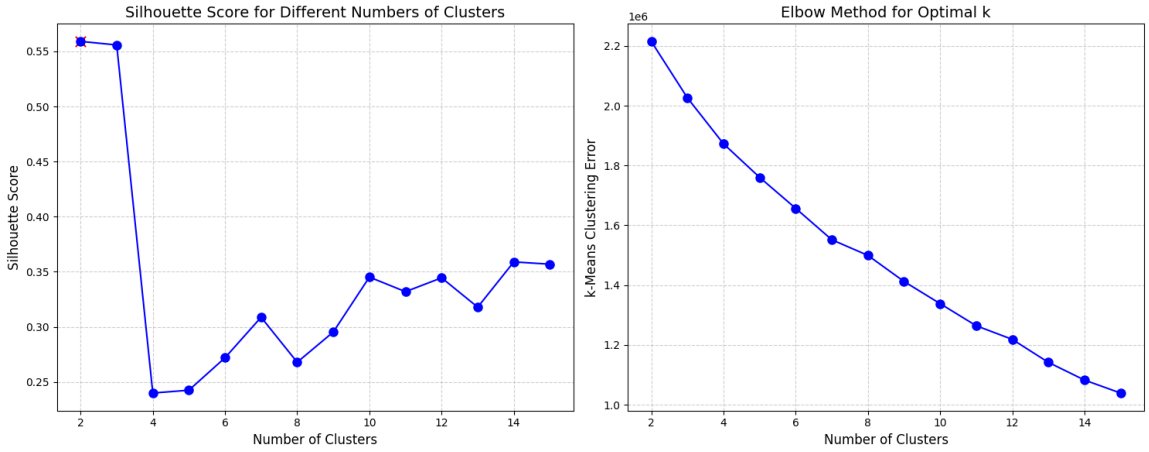

Figura 35: Tuning the number of clusters

The silhouette plot suggests two as the optimal value. However, to strike a balance with the clustering error, three clusters were chosen as it exhibits lower clustering error and a similar silhouette score.

The following graph illustrates a comparison of various data distributions based on their number of clusters against the ground truth.

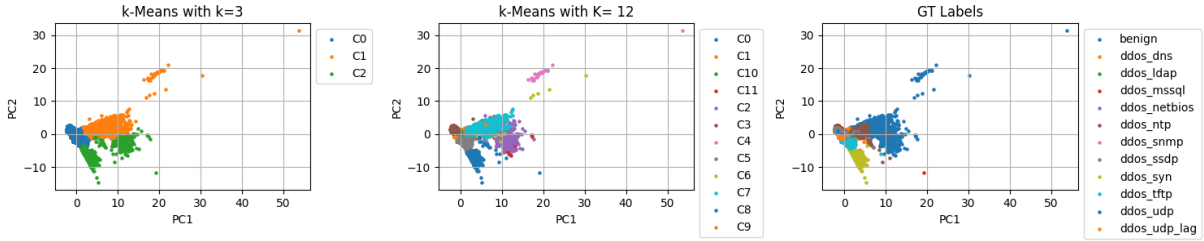

Figura 36: Comparison of data distributions

4.1.2 Hyper-parameter tuning

The K-means method requires the usage of hyper parameter such as *init* and *random_state*. The parameter *init* can take on the values of *kmeans++* and *random*, while *random_state* selects from 0, 15, 42, and 99 (random numbers). Their selection is performed through a grid search, which explores all possible combinations of specified values. The combination yielding the highest silhouette score is chosen as the optimal configuration. In the analyzed scenario, the recommended parameters correspond to those illustrated in Figure 37.

```
best_init: random
best_random_state: 15
```

Figura 37: Result of hyper-parameter tuning

After determining the correct number of clusters and the hyperparameter values, the K-means algorithm is re-run with these values, and the following results are obtained.

```
KMeans
KMeans(init='random', n_clusters=3, n_init=10, random_state=15)

Sum of squared distances of the samples from their centroid:  2026510.2984517028
Mean squared distances of the samples from their centroid:  31.54641726134751
Silhouette score of the samples:  0.5558096048426602
```

Figura 38: The new recomputed silhouette score

As shown in the Figure 38 the Silhouette score is better then the previously measured in Figure 34.

4.2 Gaussian Mixture Modelling (GMM)

Gaussian Mixture Models (GMM) contrast with K-means as a soft clustering technique, wherein each sample is probabilistically associated with multiple clusters. GMM assumes that data points are generated from a mixture of several Gaussian distributions. This probabilistic approach allows GMM to capture complex data distributions and account for overlapping clusters.

Looking at the results in Figure 39, this initial configuration is not optimal due to the low silhouette score and high clustering error observed. So, it is needed to determine the most appropriate number of clusters and the optimal hyperparameter values.

As a first step, the behavior of GMM is examined by assigning 12 clusters, mirroring the approach previously employed with K-means.

```
GMM with 12 clusters
Total log-likelihood score:  150.73381537297757
Silhouette score of the samples:  0.2819014064128538
RI between clustering and given classes is  0.7939825913412533
ARI between clustering and given classes is  0.3192320589602637
```

Figura 39: GMM with random parameters

As expected, the Silhouette value is not optimal, so an investigation about the number of cluster and hyper parameters values is needed.

4.2.1 Determining the number of clusters via Silhouette, Log-likelihood Analysis and Adjusted Rand Index

The log-likelihood is a measure that assesses how well the model explains the observed data. It quantifies the probability of observing the given data under the parameters of the GMM. Maximizing the log-likelihood during training is a common objective, as it leads to a better fit of the model to the data.

ARI measures the similarity between the true class assignments and the assignments predicted by the model, while correcting for chance. ARI ranges from -1 to 1, where 1 indicates perfect clustering, 0 suggests random clustering, and negative values mean the clustering is worse than random. A higher ARI implies better agreement between the true and predicted clusters, considering the baseline of random clustering.

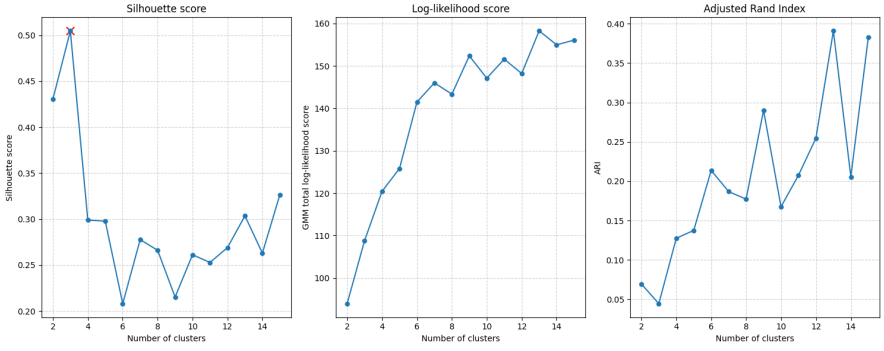

Figura 40: Tuning the number of clusters

As depicted in the Figure 40, the optimal value for the silhouette metric is observed at 3. However, the values for log-likelihood and ARI suggest considering higher values. Specifically, a compromise has been chosen, and the value 7 has been selected. This decision involves a trade-off: the silhouette metric is lowered, while both log-likelihood and ARI are increased. This choice is made with the intention of balancing the metrics and achieving a more satisfactory overall performance in the given context.

The Figure 41 illustrates a comparison of various data distributions based on their number of clusters against the ground truth.

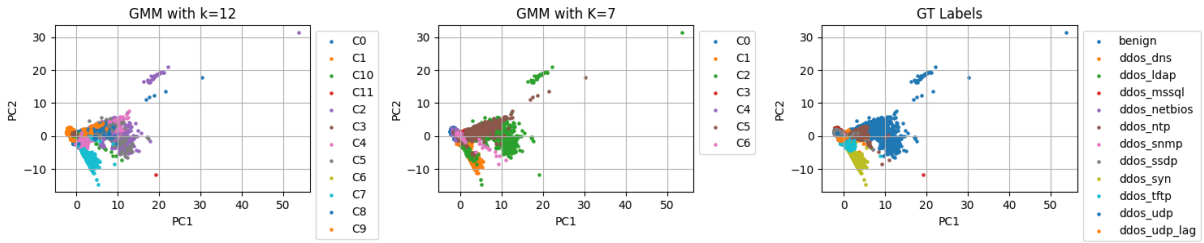

Figura 41: Comparison of data distribution

4.2.2 Hyper-parameter tuning

As for K-means, GMM requires the usage of hyper-parameters like *init* e *random_state*. Their selection is performed through a grid search. The combination yielding the highest silhouette score is chosen as the optimal configuration. In the analyzed scenario, the recommended parameters correspond to those illustrated in Figure 42.

```
best_init: random
best_random_state: 0
```

Figura 42: Result of hyper-parameter tuning

After determining the correct number of clusters and the hyperparameter values, the GMM algorithm is re-run with these values, and the following results are obtained.

```
GaussianMixture
GaussianMixture(init_params='random', n_components=7, n_init=10, random_state=0)

Sum of squared distances of the samples from their centroid:  -139.88599896805707
Mean squared distances of the samples from their centroid:  -0.0021775868081392465
Silhouette score of the samples:  0.41501047234039046
```

Figura 43: The new recomputed silhouette score

As shown in the Figure 42 the Silhouette score is better then the previously measured in Figure 39.

Finally, the following graphs depict the contribution of each cluster to the final silhouette score, which defines the quality of the utilized algorithm.

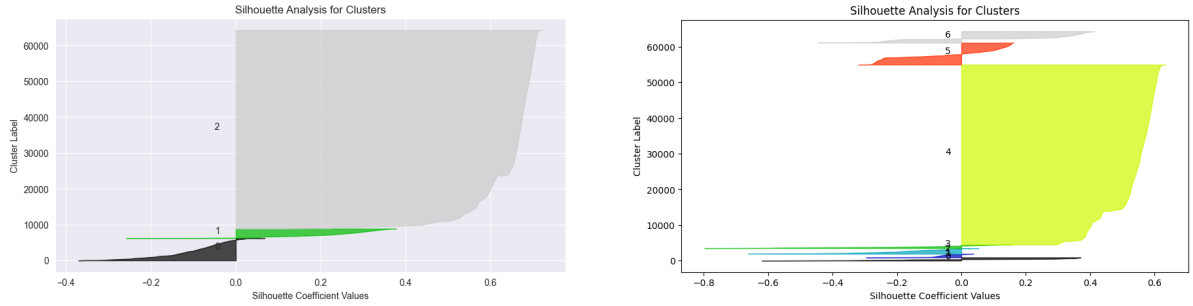

Figura 44: Silhouette analysis for clusters in K-means and GMM

5 Clusters explainability and analysis

In this section, an analysis of the identified clusters will be conducted, accompanied by an interpretation at the ground truth level. However, prior to proceeding, it is imperative to select only one of the clustering techniques developed earlier.

As shown in the Figure 45, it seems that the K-means technique is better than GMM.

```
Adjusted Rand Index (k=3): 0.2222877931236337
Normalized Mutual Information (k=3): 0.5040190314678777
```

```
Adjusted Rand Index (k=7 GMM): 0.1737531966074104
Normalized Mutual Information (k=7 GMM): 0.45936213634985373
```

Figura 45: Comparison between K-means and GMM

Upon examinating Figure 46, it becomes evident that the ground truth does not align with the predicted clusters.

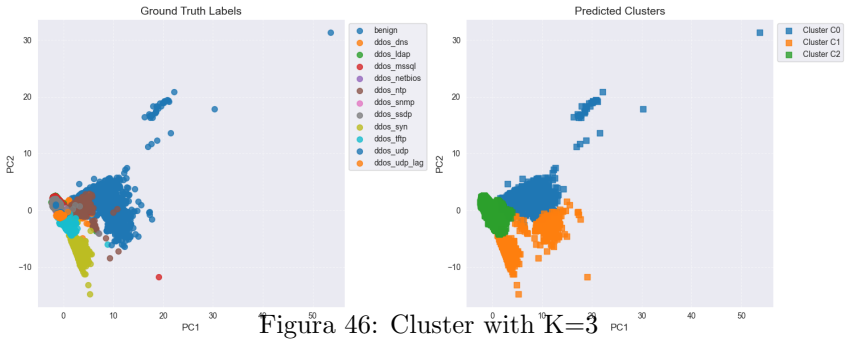

Figura 46: Cluster with K=3

It is time to investigate how the clusters are composed and their intrinsic characteristics.

The graphs in Figure 48 show how datapoints in a certain class are distributed over the various clusters. Specifically it is possible to deduce how many clusters are needed to reach the 100% occurrencies. It, also, shows which cluster most contributes to this percentage For example:

- The benign class reaches the 94% with Cluster0, then 98% with Cluster1 and 100% with Cluster2.
- The *ddos_syn* class reaches the 0% with Cluster0, then the 42% with Cluster1 and 100% with Cluster2
- The *ddos_udp_lag* class reaches the 0% with Cluster0, then the 0% with Cluster1 and 100% with Cluster2
- And so on...

To enhance the visualization of data point distribution and Empirical Cumulative Distribution Functions (ECDF), the table in Figure 47 illustrates the occurrence count for each class assigned to respective clusters.

cluster	0	1	2
y			
benign	5297	271	90
ddos_dns	100	5	5264
ddos_ldap	2	1	5925
ddos_mssql	1	1	5909
ddos_netbios	0	1	5829
ddos_ntp	815	14	157
ddos_snmp	0	0	5984
ddos_ssdp	3	1	5966
ddos_syn	0	2318	3162
ddos_tftp	0	14	5247
ddos_udp	0	1	5875
ddos_udp_lag	0	0	5986

Figura 47: Class Cluster Counts Analysis

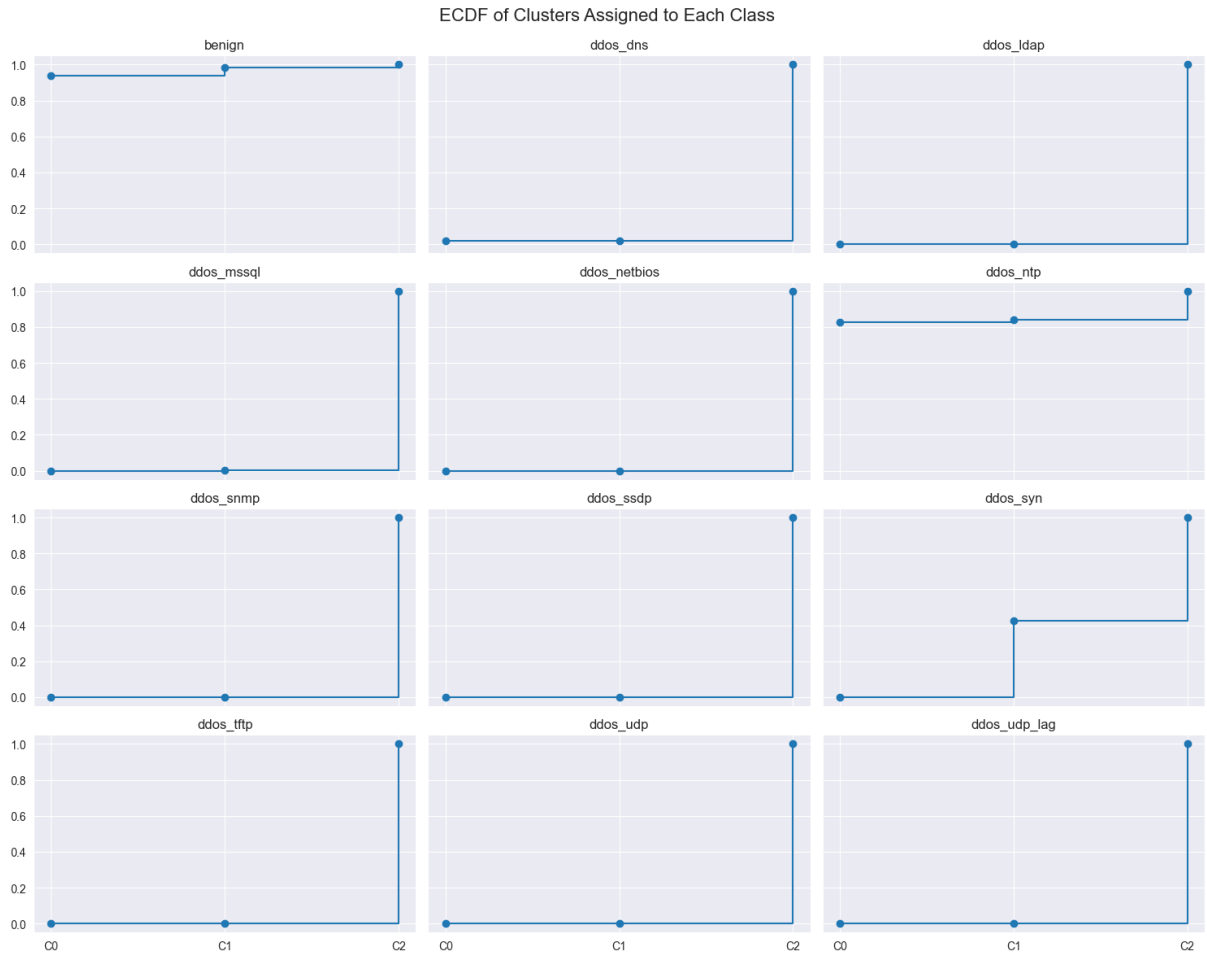

Figura 48: ECDF of clusters assigned to each class

Upon a closer examination of the table content, it is evident that there are no pure clusters, as none consist solely of one class. Nevertheless, a notable dominance of two classes within Cluster 0 and 1 is observed. Cluster 0 predominantly contains a high percentage of benign occurrences, while Cluster 1 is primarily composed of *ddos_syn attacks*. Furthermore, an interesting observation is the behavior of the *ddos_ntp* class, which exhibits similarities to benign flows. Similarly, though in smaller quantities, even the benign class displays behaviors similar to malicious activities. The *ddos_syn* attack class is nearly evenly distributed between Cluster 1 and Cluster 2, suggesting the difficulty in categorizing such attack types distinctly. Lastly, Cluster 2 exhibits a high concentration of malicious attacks.

From this point, the clustered data will be treated as a new variable named *clustered_data*, which is a copy of the *newdf_cluster*.

It is interesting to understand what are the most important features in each obtained cluster. What are the most significant features which permit to assign a datapoint to a given cluster? Feature importance analysis can be performed.

In order to compute feature importance, it is mandatory to import those libraries.

```
1  ! git clone https://github.com/YousefGh/kmeans-feature-importance.git
2  ! mv "./kmeans-feature-importance/kmeans_interp/" "."
3  ! pip install -r "kmeans-feature-importance/requirements.txt"
```

Figura 49: Feature importance import

To perform feature importance, a *K-means interpreter* is initialized with the following code:

```
kms = KMeansInterp(n_clusters=best_n,
                   random_state=15,
                   init = 'random',
                   n_init = 10,
                   ordered_feature_names = clustered_data.columns,
                   feature_importance_method='wcss_min',
                  ).fit(clustered_data)

labels = kms.labels_
clustered_data['cluster']=labels
```

Figura 50: Feature importance code

The *KMeansInterp* has been initialized by means of the same hyper-parameters used in the previous section. Moreover, it requires the usage of other parameters such as:

- *ordered_fearure_names*, it consists of the list of features;
- *feature_importance_method*, it consists in the method used to compute the figure importance. The feature importance is determined based on Within-Cluster Sums of Squares: it implies that the algorithm assesses how much each feature contributes to minimizing the WCSS. Features that lead to more compact and well-defined clusters are considered more important in this context.

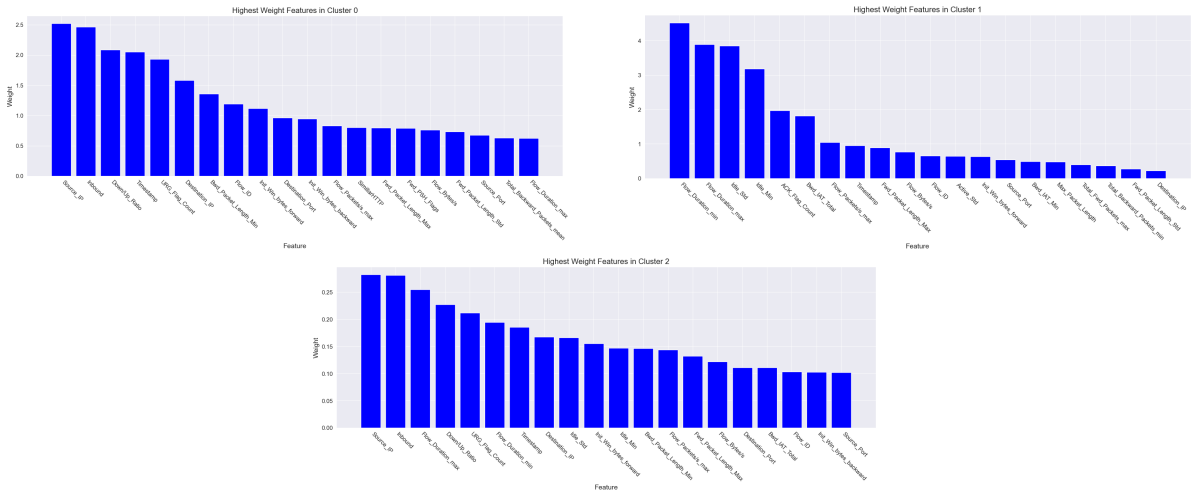

Figura 51: Feature importance analysis

Regarding cluster 0 and cluster 2

- *Source_IP* is determinant in distinguishing between benign and malicious traffic. Certain source IPs may be associated with known benign entities or trusted sources, while others might be indicative of malicious activities. As anticipated in first section (at Figure 4), the most of the attacks are associated to specific source IP. This helps to tag a flow as malign.
- *Inbound* suggests that the directionality of network communication (i.e., whether it is directed towards the device or initiated by the device) plays a crucial role in distinguishing between benign and malicious traffic.

For what concerns cluster 1, it can be seen that is predominantly composed of the features *flow_duration_min*, *flow_duration_max* and *idle_std*.

- The importance of flow duration suggests that the duration of the traffic flow plays a key role in identifying *ddos_syn* attacks within this specific cluster.
- The significance of *idle_std* indicates that the variability or spread in the idle time of network communication is crucial for distinguishing *ddos_syn* attacks within this cluster.

5.1 Sub-attack analysis

Considering that Cluster 2 encompasses various attacks, a decision was made to perform a secondary clustering within this cluster. The objective is to discern whether some of the attacks may belong to different sub-clusters. The identification of the same attack across different clusters could imply the existence of multiple variants exhibiting diverse characteristics.

From this point onward, it will exclusively utilized a subset of the clustered data, consisting of data points belonging to Cluster 2 (*cluster_subAttack*). The following graph represents the distribution of data for Cluster 2.

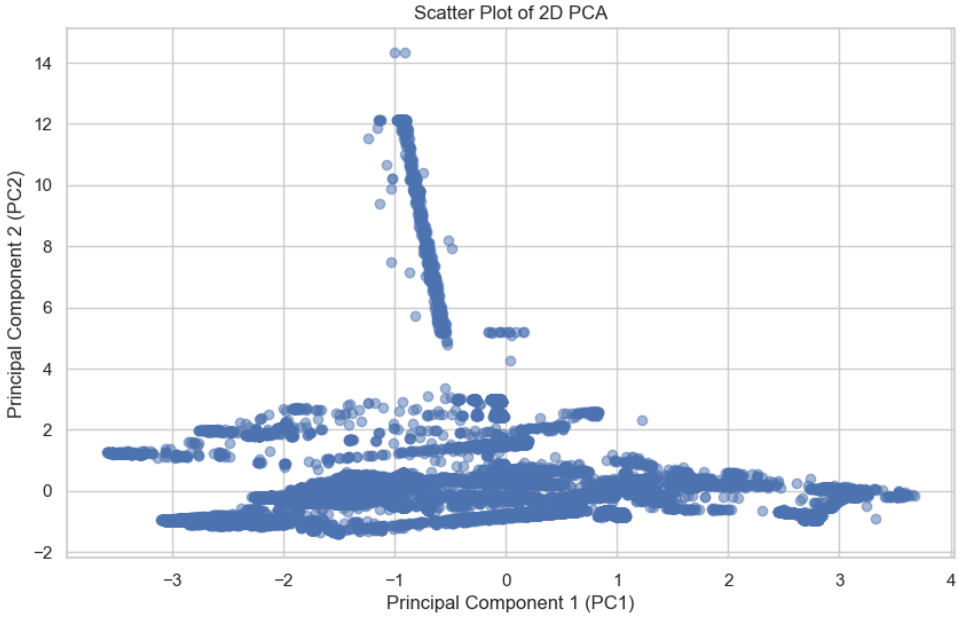

Figura 52: Scatter plot of *cluster_subAttack*

As done in previous sections, following an analysis of silhouette scores and clustering errors, a decision has been made to employ 12 as the number of sub-clusters within Cluster 2.

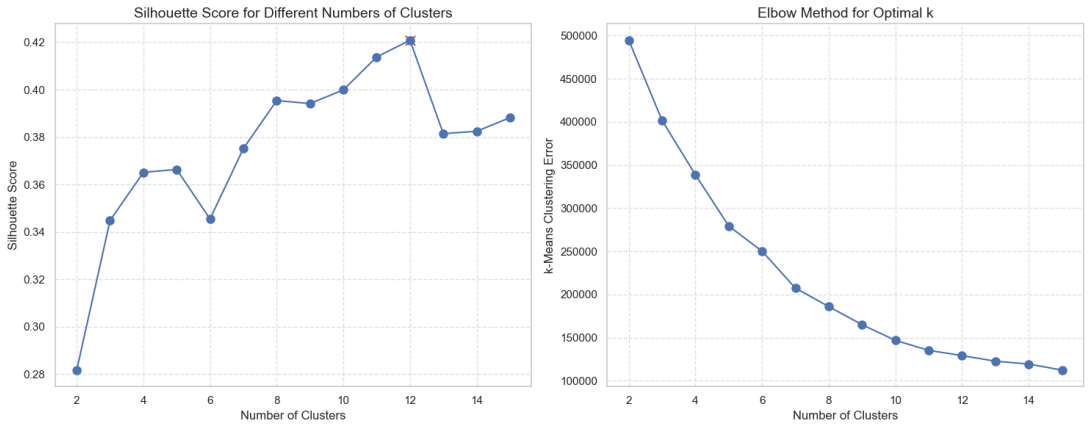

Figura 53: Metrics for *cluster_subAttack*

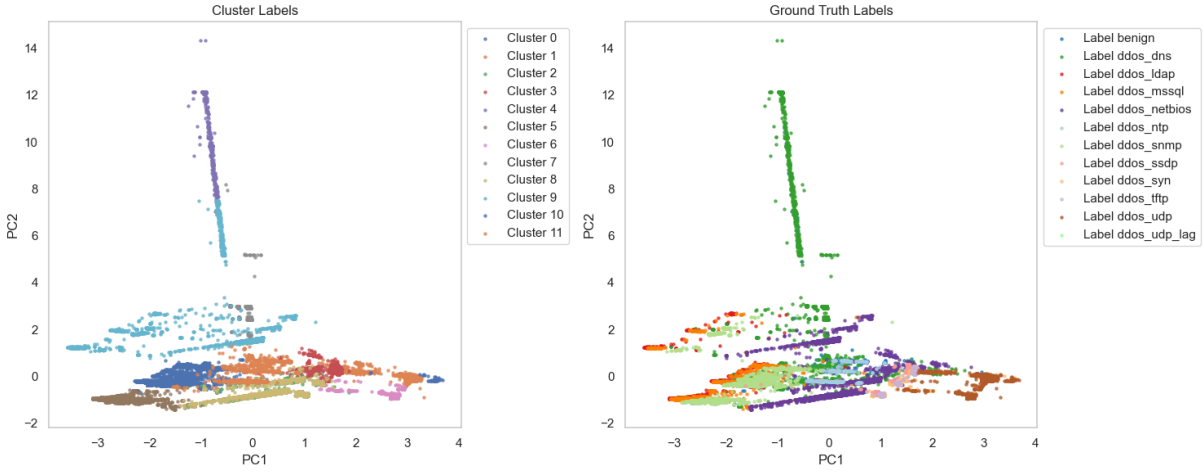

Figura 54: Scatter plot for sub-clusters

sub-cluster y	0	1	2	3	4	5	6	7	8	9	10	11
benign	0	0	0	0	0	4	85	1	0	0	0	0
ddos_dns	3400	556	31	84	531	29	221	245	0	0	113	54
ddos_ldap	0	1723	0	0	0	0	11	500	0	0	3691	0
ddos_mssql	0	2044	0	0	0	0	9	173	0	0	3683	0
ddos_netbios	0	19	365	2355	0	0	1	717	0	0	87	2285
ddos_ntp	0	0	4	56	0	94	2	0	0	1	0	0
ddos_snmp	0	0	23	98	0	0	5790	0	0	0	0	73
ddos_ssdp	0	4876	1	3	0	0	7	349	0	0	730	0
ddos_syn	0	0	0	0	0	826	0	0	18	2318	0	0
ddos_tftp	0	0	0	1	0	2062	0	0	54	3128	0	2
ddos_udp	0	2	3410	1145	0	0	0	0	0	0	0	1318
ddos_udp_lag	0	0	3479	1171	0	0	0	0	0	0	0	1336

Figura 55: Numerical table for the scatter plot

The tables enable the identification of various patterns within the same attack category. In essence, it becomes possible to discern different typologies within a given attack. An attack exhibiting diverse characteristics is segmented into different clusters, similar to having two or more distinct "species" of attacks. For instance, the analysis reveals that the *ddos_udp_lag* and *ddos_udp* attacks exhibit three distinct forms, each further categorized into sub-clusters.

On the other hand, attacks like *ddos_dns* share common traits with other attacks, making them easily mistakabled. Furthermore, the *ddos_dns* class contributes to the formation of two pure clusters, denoted as 0 and 4. This suggests that the "species" within these clusters are easily distinguishable compared to other species within the same class.

Analyzing the feature importance associated with the assignment of attacks to their respective clusters is an intriguing task. As an example, one may consider the distribution of the *ddos_udp* and *ddos_udp_lag* attacks, which are allocated across clusters 2, 3, and 11.

To better understand the distinguishing features among the three different species of UDP lag and UDP, a decision was made to perform feature importance analysis. From the obtained results, it is evident that both present three sub-attacks, two of which are similar and one is not.

- For *ddos_udp*, Cluster 0 and Cluster 1 are similar, as illustrated in the graph 56, sharing the same top three feature importances; the other, though similar, differs in the *destination_port*.

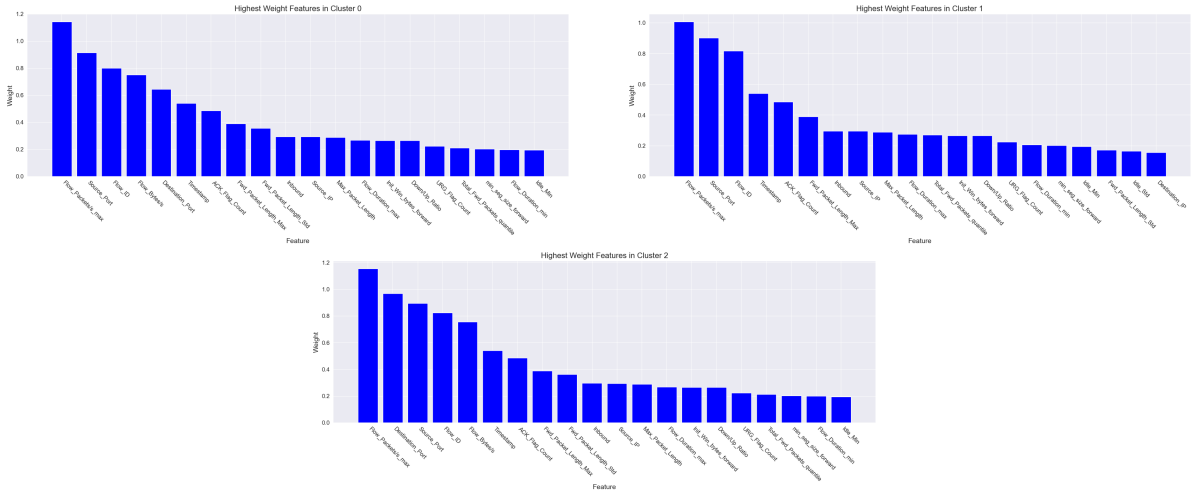

Figura 56: Feature importance for ddos_udp

- Regarding *ddos_udp_lag*, Cluster 0 and Cluster 2 are similar, as shown in the graph 57, sharing the same top four feature importances; the other, though similar, differs in the *destination_port*.

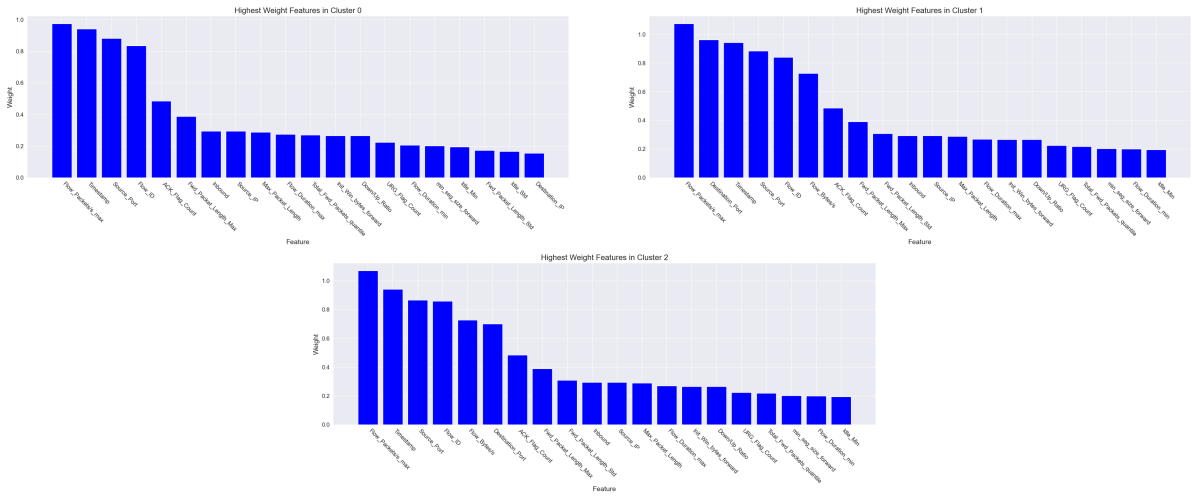

Figura 57: Feature importance for ddos_udp_lag

In conclusion, the analysis of these feature importances allows for the distinction between different variants of the same attack. This type of analysis could be applied to any type of attack present in the dataset, bearing in mind that in some cases, distinguishing between variants might be more complex.