DDoS Attacks Detection and Characterization

SARA BRAIDOTTI - S337844
CHIARA IORIO - S343732
MATTEO PANI - S343678
CRISTIAN SAPIA - S300461

Today, DDoS attacks are a serious threat to Internet security. They usually exploit compromised machines to prevent legitimate users from using web-based services. In this project, we first analyze a dataset reporting the flows of different types of DDoS attacks. Then we apply supervised and unsupervised Machine Learning (ML) techniques to automate the detection and characterization of this type of attack. Finally, we analyze the clusters obtained to find common patterns among the flows in the same class, trying to identify sub-attacks. The source code developed for the project can be found on GitHub. [1]

1 Data exploration and pre-processing

This section aims to analyze the given dataset, providing visualizations and statistical analysis at both the generic traffic level and the ground truth level. Then, we compute some new statistics about the different flows to better understand the characterization of the different types of traffic. Afterwards, we analyze the correlation between the features to understand whether it is necessary to standardize the data and perform dimensionality reduction via PCA and by removing the most correlated features. Finally, we examine the principal components obtained.

1.1 Dataset characterization

The provided dataset contains the results of the network analysis performed using CICFlowMeter-V3, obtained by simulating the behavior of 25 users based on the HTTP, HTTPS, FTP, SSH, and email protocols. It is made up of 64 239 rows, representing a different flow each, and 87 columns. It does not contain null values or duplicated rows. Each sample is characterized by 86 features, of which 81 are numeric, and by a label that refers to the type of traffic. In particular, the main features per flow are: the ID, the timestamp, the source/destination IP address and port, the transport protocol, the number of packets and bytes exchanged in each direction, the timing (flow duration, active time, idle time, and inter-arrival time) and the flags used.

As we continued to analyze the dataset, we discovered that some features do not contain meaningful values. In particular, few columns have null variance: some flags (PSH, FIN, ECE) are never used, and the metrics regarding the bulks are always zero. Furthermore, some features report the same information: the attributes about subflows are the exact copy of the corresponding ones about flows, and there are two columns about the header length containing the same values. Although the first is explainable considering that there could be only one subflow for each flow, the latter is more difficult to handle. Last but not least, some columns include high negative values, such as the header length and the minimum size (in bytes) of a segment, which may be due to incorrect data gathering.

As seen above, each flow is characterized by a label that specifies the type of traffic to which it belongs. In this dataset, there are twelve different labels, one referring to benign traffic, while the others identify different classes of DDoS attacks, which are NetBIOS (*ddos_netbios*), LDAP (*ddos_ldap*), MSSQL (*ddos_mssql*), UDP (*ddos_udp*), UDP-Lag (*ddos_udp_lag*), SYN (*ddos_syn*), NTP (*ddos_ntp*), DNS (*ddos_dns*), SSDP (*ddos_ssdp*), TFTP (*ddos_tftp*), and SNMP (*ddos_snmp*). The number of flows belonging to each category is shown in fig. 1. More or less all groups have the same number of samples; only *ddos_ntp* is underrepresented compared to the others.

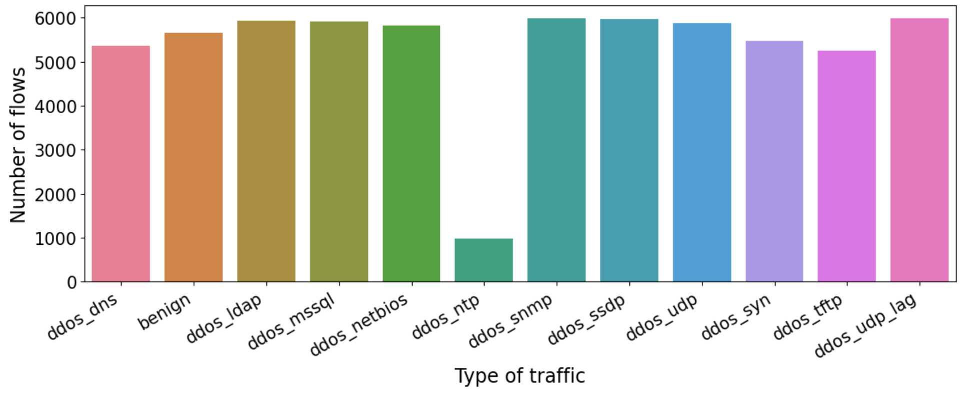
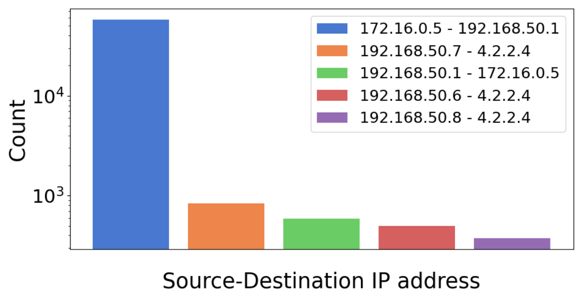
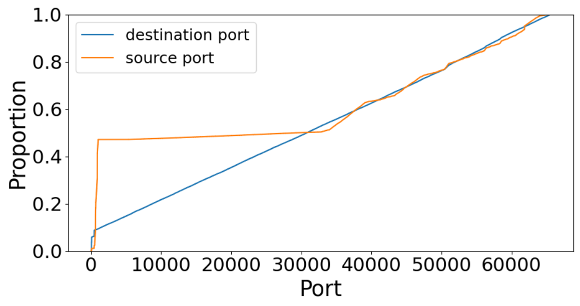


Fig. 1. Number of flows per label.



(a) Number of flows among the five most active links.



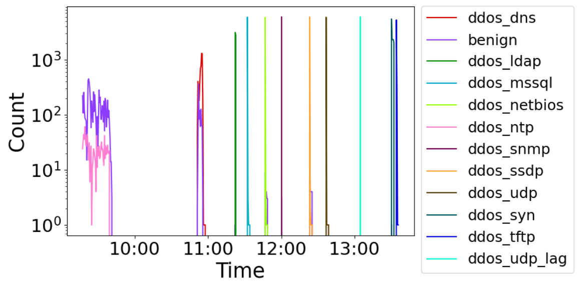
(b) Port numbers distribution.

Fig. 2. Traffic analysis based on source/destination IP address and port.

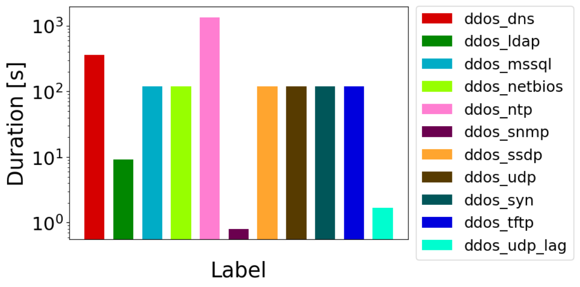
1.2 Analysis at generic traffic level

In this section, we will further analyze some of the most significant features of the provided dataset, without considering the ground truth. Three different transport protocols are used: HOPOPT (0.1 % of the flows), which is the Hop-by-Hop IPv6 extension header, TCP (23.9 %) and UDP (76 %). Furthermore, the flows can be either outbound or inbound, depending on who, between the source and the destination, initialized the connection. The vast majority of them are inbound (92 %): this means that the communication is directed towards the device. In addition to this, it is also possible to analyze the most frequent destination and source IP addresses. In fact, most inbound communications (98 %) have the same destination address (192.168.50.1), while half of outbound flows have as source address 192.168.50.7. These are local IP addresses, as these attacks have been simulated in a local network. The most active links are represented in fig. 2a. Definitely, the majority of the packets (90 %) are exchanged between 172.16.0.5 and 192.168.50.1. Furthermore, the flows in the dataset span all available destination ports (from 0 to 65 535), as shown in fig. 2b, while the low-numbered ports are especially used by the source.

It is also possible to observe that the number and length of packets sent in the forward direction are usually significantly higher (up to a few hundred packets and some thousand bytes) than those sent in the backward direction. However, the goal of some of these types of DDoS attacks is to try to overload



(a) Distribution of the traffic during the analyzed period.



(b) Total attack duration per label.

Fig. 3. Attacks time-based analysis.

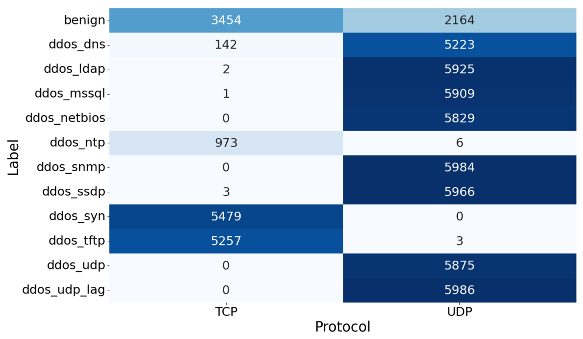
the networks by triggering big responses in the backward direction; this inconsistency may be because the reported attacks are simulated and not real ones. As expected, there is a direct proportionality between the number of packets and the total length in bytes.

According to the description of the dataset, the time features have a unit of measurement in seconds; this implies that some flows have a duration of more than three years, which is quite strange. Therefore, we will now consider them expressed in microseconds. Regardless, this assumption will not influence the subsequent steps, since the ML models do not care about the unit of measurement, as long as all the data are consistent. As a consequence, the mean flow duration is in the order of some seconds, and most of this time is idle, where the communication is not actively transferring data. On the other hand, the average inter-arrival time is in the order of a tenth of a second. There is inverse proportionality among the total number of packets and the mean IAT: therefore, the smaller this time, the more packets are sent.

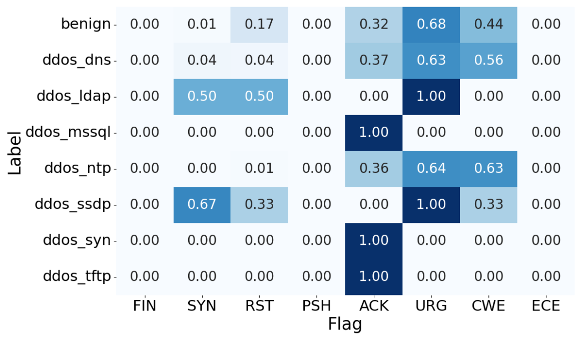
1.3 Analysis at ground truth level

The objective of this paragraph is to thoroughly examine the characteristics of flows belonging to the same class. In particular, we are going to start from the analysis of the timescale. In fig. 3a, it is possible to observe how flows are distributed during the analyzed period. Specifically, we computed the end time of each flow given its timestamp and duration, then counted the number of active flows in each interval of 30 s. The impulsive traffic peaks correspond to the different attacks, while the benign traffic is present along almost the entire timeline. This graph also gives some hint on the duration of the attacks: the *ddos_ntp* seems to be the longest. In fig. 3b the durations of the attacks are represented, considered as the difference between the timestamp of the first sent packet and the end time of the last one. This graph confirms previous intuitions: most attacks last a couple of minutes, *ddos_ntp* is the longest one (22 min) and *ddos_snmp* is the shortest (0.8 s).

We now proceed to the analysis of the protocol employed by the various attacks. As seen before, there are three different transport protocol values in the dataset, but since the HOPOPT is used rarely and mainly by benign traffic, we will concentrate on UDP and TCP. From fig. 4a, it is evident that the vast majority of attacks are based on UDP; only the *ddos_syn*, the *ddos_ntp* and *ddos_tftp* flows use TCP as a transport protocol. The benign traffic is the only one fairly split among the two of them. The traffic carried on the same protocol has some common characteristics: TCP flows are usually longer, with higher active time and IAT, a comparable number of packets in the two directions, and response packets significantly bigger than the requests; in UDP, instead, the number of forward packets is a hundred times greater than the one of the backward ones. In fig. 4b, the average number of flags per TCP flow

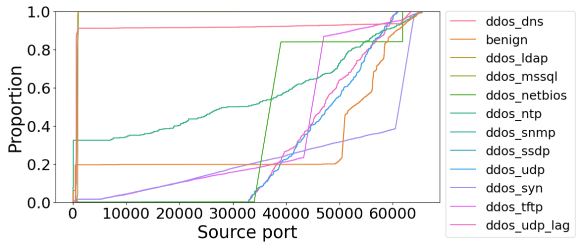


(a) Number of flows per label per protocol.

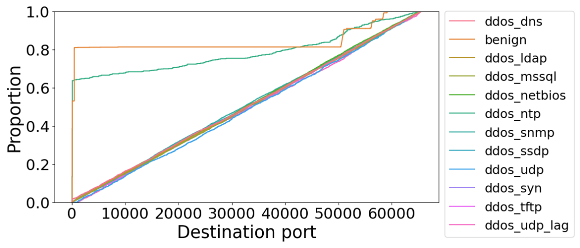


(b) Average number of flagged packets per flow in TCP.

Fig. 4. Type of protocol and flags used in the different attacks.



(a) Source ports.



(b) Destination ports.

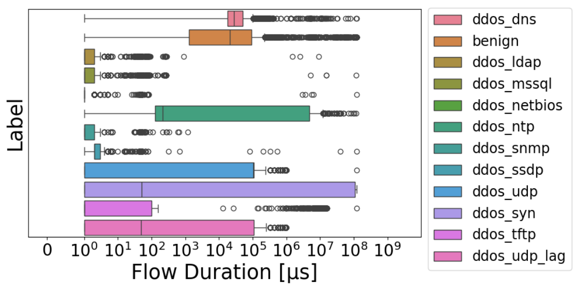
Fig. 5. Ports used by the flows of the different attacks.

per label is shown. It is evident that some flags (FIN, PSH, and ECE) are never used: this means that flows have used a reset packet (with the RST flag) to end the communication. Obviously, this analysis is mainly meaningful for the attack based on TCP: in each *ddos_syn* and *ddos_tftp* flow there is a packet containing the ACK flag, but not the SYN one, which is quite awkward. The *ddos_ntp* ones, instead, also employ the URG and CWE flags, which are used to tell the receiving station that the packet is urgent and that the routers are overloaded.

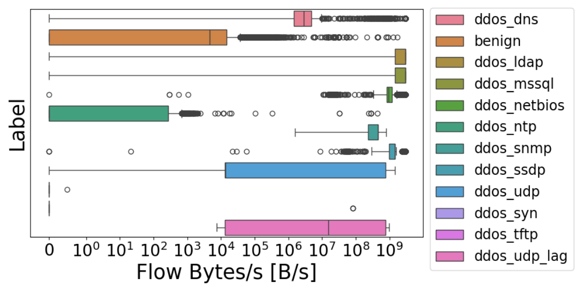
Finally, we analyze the IP addresses and ports used. All malicious flows are between the same two IP addresses: 172.16.0.5 and 192.168.50.1. Various destination port numbers are used instead by the different kinds of attacks and protocols. As illustrated in fig. 5b, during most of them packets are sent to several destination ports, likely to perform a port scan; only *benign* and *ddos_ntp* traffic usually utilize low-numbered ports. On the other hand, the source ports used are characteristics of each attack, as shown in fig. 5a. Furthermore, most of the traffic is inbound: the only two attacks that have a significant percentage of outbound flows are *ddos_dns* (5 %) and *ddos_ntp* (30 %). Outbound traffic implies that, probably, the device has been compromised and used to carry out the attack.

1.4 Additional features

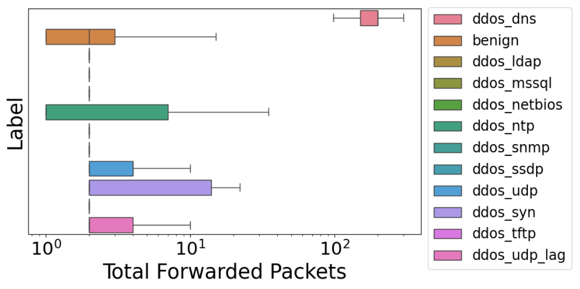
In the previous sections, we observed that the various types of attacks have different attribute values. The goal of this paragraph is to take a closer look at the most characterizing ones. To better understand



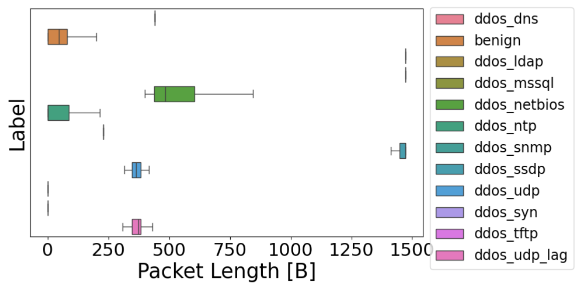
(a) Flow Duration.



(b) Flow Bytes/s.



(c) Total number of forwarded packets.



(d) Mean of packets length.

Fig. 6. Attack characterizing features.

which features could be the most useful for discerning the diverse classes of attacks, we used the `describe` function in the `pandas` library. It allowed us to compute different statistics for each feature, such as mean, maximum, minimum, quartiles, and standard deviation.

In fig. 6, some of the most defining features are shown. We decided to represent them as boxplots rather than histograms to highlight at the same time more characteristics of the feature: its mean, quartiles, and outliers. For visualization purposes only, we do not plot the outliers in fig. 6c and in fig. 6d, because they are higher orders of magnitude than most data.

In fig. 6a the flow duration feature is illustrated. In particular, some attacks have flows that can last a very different amount of time (i.e, *ddos_ntp*, *ddos_udp*, *ddos_syn*, and *ddos_udp_lag*). Some, in contrast, are usually very short (i.e., *ddos_ldap*, *ddos_mssql*, *ddos_snmp*, and *ddos_ssdp*) with outliers that last longer. *ddos_tftp* flows have a duration of up to a few hundred microseconds, while *ddos_dns* and *benign* tend to be longer, but have many more lengthy outliers. Moreover, we noticed that the inter-arrival time values follow approximately the same pattern.

In fig. 6b the flow rate is represented: some attacks, such as *ddos_ldap*, *ddos_mssql*, *ddos_netbios*, *ddos_snmp*, and *ddos_ssdp*, have a very high and consistent one, while for *ddos_syn* and *ddos_tftp* it is almost zero, with very few outliers. The flows of the other attacks have a more variable rate.

The number of forwarded packets, shown in fig. 6c, allows us to distinguish *ddos_dns* flows from all the others, since they are significantly the ones with the highest number. However, other types of traffic, such as *benign* one, have outliers with a higher number of packets sent. Backward traffic can also be helpful in classifying the types of attacks, as only *benign* and *ddos_ntp* flows send many backward packets.

Concerning fig. 6d, we can notice that each class of attack has its own specific packet length, even though there are smaller or larger outliers. *benign*, *ddos_ntp*, *ddos_syn*, and *ddos_tftp* have usually little packets (less than 100 B), while *ddos_ldap*, *ddos_mssql*, and *ddos_ssdp* ones are very big. The other attacks, finally, send packets of intermediate length.

From this analysis, it is clear that some attacks tend to have common characteristics. For instance, a group could be composed of *ddos_ldap*, *ddos_mssql*, *ddos_snmp*, and *ddos_ssdp* flows, another by *ddos_udp* and *ddos_udp_lag*, and a last one by *ddos_syn* and *ddos_tftp*. Also *benign* and *ddos_ntp* are quite similar.

We decided not to add new features based on these metrics to the dataset, otherwise we would replicate the same values across all rows classified with the same label. Hence, we would pass the label indirectly to our ML models, which would certainly lead to high accuracy but not very meaningful results. Moreover, in a real-world scenario, we would not be able to add these new features to new flows we want to classify since we do not yet know the label.

1.5 Correlation analysis

This step explores the relationships between the different numerical features. ML algorithms are sensitive to data and if two features are highly correlated, it might be that one of them is redundant, because the information provided is already replicated. If an ML model is trained using both features, it will essentially learn the same information from both, which is not only unnecessary but also increases model complexity, potentially leading to higher computational costs. To analyze the correlations, we visualize a heatmap of the correlation matrix, which highlights relationships between features. This was done using two metrics: Pearson's correlation coefficient (for linear relationships) and Spearman's rank correlation (for monotonic relationships, hence not necessarily linear). These metrics provide a quantitative assessment of the strength of the correlation and are useful in guiding feature selection decisions. We observed some slight differences among the two correlation matrices visualized: for example, according to the Pearson's correlation coefficient the flow IAT and the idle time are strongly correlated, while that is not true according to the Spearman's one. Furthermore, we can deduce that there is a non-linear relationship between the idle and the active time. On the other hand, the IAT and the flow duration are strongly correlated according both coefficients. Finally, we can notice that the different metrics about the IAT are strongly correlated. The correlation matrix computed using Pearson's coefficient is shown in appendix A.

1.6 Label encoding

Label encoding is a discretization technique that is usually applied to an attribute to convert its values from any domain to a numerical one. This is particularly useful for categorical features that, due to their nature, cannot be directly interpreted by ML models.

After careful analysis, we decided to apply label encoding exclusively to the feature *Protocol*, as the protocol associated with various flows is relevant to our task, as demonstrated in §1.3. Moreover, the *Protocol* feature contains a very small set of numerical values (only three), so we apply the label encoding because the model might misinterpret these values as having an ordinal relationship or magnitude, whereas in reality, no such order exists among the protocols. However, this encoding has little impact on the performance of the models that we trained in §2.

Regarding the choice of a label encoding technique, there are several options. For instance, one-hot encoding creates a separate binary column for each category, which can be useful but may significantly increase the number of columns, especially for features with a wide range of values. In our case, we

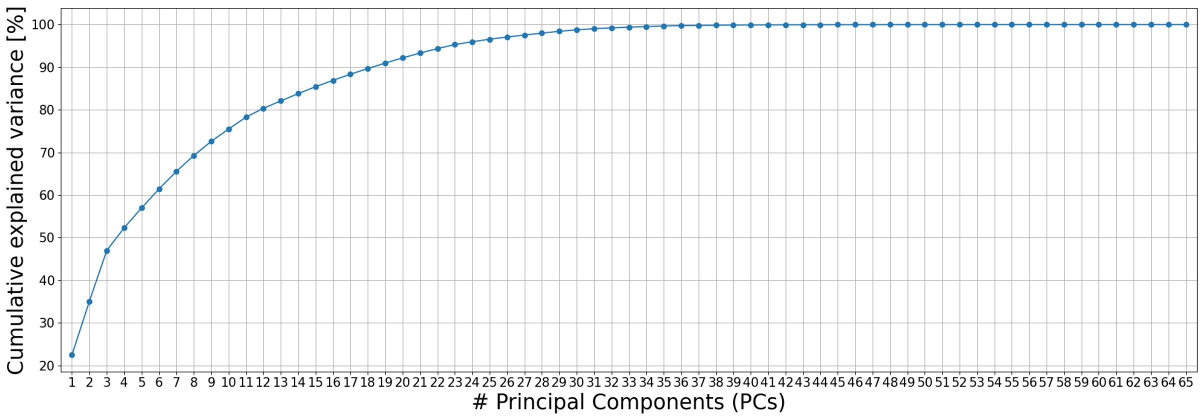


Fig. 7. PCA: elbow method.

applied this technique because the *Protocol* feature has a very small set of values. This approach provides an efficient and straightforward representation without introducing too much complexity.

On the other hand, for other categorical features such as *Source Port* or *Destination Port*, applying label encoding would not be appropriate due to the wide range of possible values. Furthermore, a wide range of values could increase the risk of overfitting and reduce the model's ability to generalize, further justifying our choice.

We faced the same issue for the IP source and destination addresses: they are categorical features that cannot be interpreted by ML models, but they seem significant to discern the traffic. Since there are too many possible values to perform label encoding, we decided to transform them into their binary representation (they are composed of four bytes). However, we experimented as the performance of ML models remained the same even removing those features.

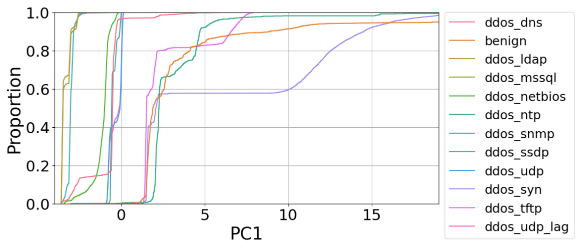
In addition, we decided to simply ignore the remaining categorical features for the following reasons:

- *Flow ID*: it does not bring any relevant information since it is composed of the IP quintuple that is already present in other features.
- *Similar HTTP*: its value is zero in more than the 99 % of the cases, so it is not very significant.
- *Timestamp*: we tried to transform it into its representation in seconds and then train the ML models described in §2. Their performance on this specific dataset definitely increased, but our models should also be applicable to discern future attacks, so the timestamp might be in another interval of time, leading to wrong classifications.

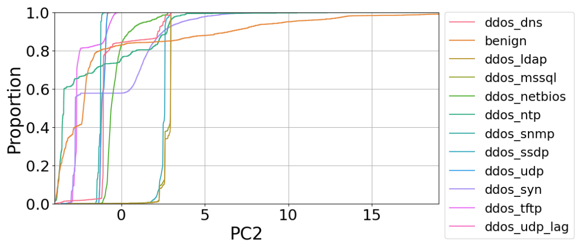
1.7 PCA

Principal Component Analysis (PCA) is a technique used to project the original dataset into a new feature space (the components). It reduces the number of variables while preserving the core characteristics and variations of the dataset as much as possible. We perform PCA because it is particularly useful for high-dimensional datasets, since it can simplify the dataset management, speed up the training process, and improve model generalization by mitigating the risk of overfitting.

The data require a standardization process, as PCA assumes that all features are on the same scale. Without standardization, features with larger magnitudes could disproportionately affect the explained variance, therefore misleading the outcome. By standardizing, we ensure that all features contribute



(a) ECDF of the first principal component.



(b) ECDF of second principal component.

Fig. 8. ECDF of the first two principal components.

equally to the variance analysis, allowing PCA to effectively identify the principal components useful for our ML task. In this case, 40 Principal Components are enough for our dataset, as they explain 99.92 % of the cumulative variance. The elbow method chart is illustrated in fig. 7.

1.8 Analysis of the features obtained via PCA

In this section, we will examine the new features obtained via PCA. As expected, the resulting components are no longer correlated with each other, which is one of the most important advantages of this technique.

Fig. 8 shows the empirical cumulative density functions (ECDF) of the first two principal components. Each class of attack has a different trend, but we can recognize the same similarities we identified at the end of §1.4. In fact, in both graphs the lines of *ddos_ldap* and *ddos_mssql*, and the ones of *ddos_udp* and *ddos_udp_lag* are almost overlapped; the ones of *ddos_syn* and *ddos_tftp*, instead, are slightly different: the former has a slower increase, which implies that, for PC1, there are no values between 2 and 10. These patterns can be found in all the generated principal components: the main difference is that, while for the first ones the range of possible values is quite wide (approximately between -5 and 20), for the last ones it is pretty narrow (between -0.5 and 0.5). As a consequence, ML models may misclassify the samples of these classes due to their very similarity. This issue will be further investigated in the following sections.

Finally, we analyzed the loading scores of the first two principal components. In particular, PC1 strongly depends on the inter-arrival and idle time, while PC2 is mainly affected by the maximum packet length.

1.9 Removing correlated features

One of the main disadvantages of PCA is that it limits the interpretability of the features obtained and, as a consequence, the results that we obtain from the ML models. For this reason, after standardizing our dataset, we decided to perform dimensionality reduction by removing the columns that have a correlation greater than 0.8 in absolute value. In doing so, we removed 36 features and kept 29 of them, without considering the ones we already dropped for the reasons explained in the previous sections. We will apply both supervised and unsupervised ML techniques to this reduced dataset, as there is no big difference in performances compared to that obtained using PCA components. Moreover, we also verified that there is no performance decrease when using the simplified dataset rather than the original one.

2 Supervised learning – classification

The goal of this part is to classify the flows into different types of attacks based on the features retained in the dataset. This section focuses on selecting different models, training and testing them with default parameters, and performing hyperparameter tuning to evaluate whether the optimization process yields improvements and significant results.

2.1 Dataset pre-processing

2.1.1 Train test split. First, it is essential to split the dataset into training and test sets. This split must be performed in a stratified manner with respect to the labels to ensure a balanced distribution of samples across all classes. This approach is crucial to achieving reliable results and mitigating the risks of overfitting or underfitting.

2.1.2 Data preparation. Then, we pre-processed our dataset as explained in the previous paragraphs. In particular, we removed inconsistent and categorical features, performed the encoding as explained in §1.6, scaled our dataset and dropped highly correlated features (§1.9). In this case, however, we fit the `StandardScaler` and computed the correlation only on the training set, then transformed both the training and the test sets: this ensures that the model's evaluation on the test set remains fair and realistic. We also transformed class labels into integers using the `LabelEncoder` since some ML algorithms work only with numerical labels.

2.2 Classification algorithms

2.2.1 ML models. At this point, we trained the models with the default parameters to pick, among the ones presented during the course, the most suitable to our dataset. The chosen ones are the following:

- k-Nearest Neighbors (k-NN): An algorithm that classifies a sample based on the majority class of its nearest neighbors.
- Logistic Regression: A linear model commonly used for binary and multiclass classification that predicts the probability that a sample belongs to a specific class and is particularly effective when the relationship between features is approximately linear.
- Random Forest: An ensemble learning method based on decision trees. It combines multiple trees to improve accuracy, making it robust for datasets with complex, nonlinear relationships.

To read more about the two discarded models (Naive Bayes and Support Vector Machines), refer to appendix B.

2.2.2 Hyperparameter tuning. Next, we performed the hyperparameter tuning to find the best combination that leads to the highest accuracy on the validation set. To this end, we used the `GridSearchCV` function provided in the `sklearn.model_selection` library. It performs stratified 5-fold cross-validation and can exploit all the cores of the PC if the *n_jobs* parameter is set to -1. As a result, the computation is much faster than with a custom implementation.

2.2.3 Preliminary remarks. We observed some common behaviors among the ML algorithms. In particular, the performance varies between the different types of attack (classes), and this can be attributed to several factors. In fact, some classes have feature values that are more distinct, such as *benign*, *ddos_dns*, *ddos_syn*, and *ddos_tftp* ones, leading to high precision, recall, and F1-scores. Other attack types, instead, have overlapping features, making it harder for the models to distinguish between such classes. For example, we saw in §1.4 and §1.8 that *ddos_ldap*, *ddos_mssql*, and *ddos_ssdp* flows have some common characteristics (e.g., flow duration, transmission rate, and packet length) and also *ddos_udp* and *ddos_udp_lag* may be difficult to discern. Furthermore, certain attacks (e.g., *ddos_mssql*, *ddos_snmp*) have features that are noisier or less discriminative, leading to poorer precision and recall.

2.3 First model: k-Nearest Neighbors (k-NN)

2.3.1 Default model performance evaluation. The model achieves higher accuracy (86 %) in the training set compared to the test set (79 %). This difference may indicate a little overfitting, meaning that the model is too tailored to the training data and does not generalize well to unseen data.

Most classes have high precision and recall in the training set, while some of them perform worse in the test set. In particular, *ddos_mssql*, *ddos_udp* and *ddos_udp_lag* have a precision higher than 70 % in the training set, which drops to around 55 % in the test. The *ddos_ldap* class is the one in which the model performs worst, suggesting that it is not able to generalize such attack.

2.3.2 Hyperparameter tuning. An effective strategy to enhance model performance is to perform hyperparameter tuning to find the best possible combination of values for the model parameters. In the k-NN model, we decided to optimize the following parameters:

i) *n_neighbors*: this is one of the most important parameters for this model. A too-low value makes the model sensitive to noise (overfitting), while a too-high value can reduce the importance of the most relevant neighbors (underfitting).
ii) *weights*: with `uniform`, each neighbor has the same weight; with `distance`, the nearest neighbors are given more weight, which can be useful for optimization.
iii) p: this specifies the type of distance used in neighbor detection: with $p = 1$, the Manhattan distance is used, while with $p = 2$, the Euclidean distance is used.

We did not test different algorithms to compute the nearest neighbors, as the `auto` option already attempts to identify the most appropriate algorithm based on the values passed to the fit method.

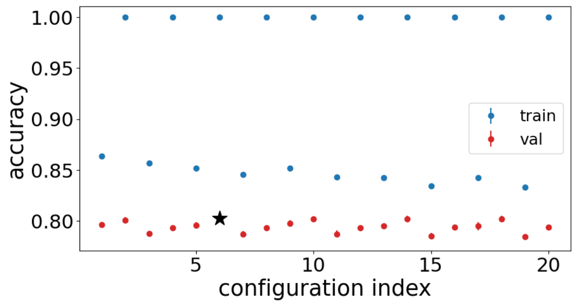
2.3.3 Results. Fig. 9a shows the accuracy in the training and validation sets with respect to different parameter configurations. In particular, we noticed that, in general, the model works better with p set to 1 (Manhattan distance). The combination of hyperparameters that leads to the best accuracy in the validation set (80 %) is *n_neighbors* = 6, *weights* = `distance`, and $p = 1$. As a result, we obtain an accuracy of 100 % in the training set: our model is probably overfitting. However, we obtained similar accuracy values in the validation set even with other combinations of hyperparameters (using *weights* = `uniform`) that lead to a lower accuracy (around 85 %) in the training set.

Several factors can contribute to overfitting. For example, k-NN, especially with a small number of neighbors considered, tends to memorize the training data, making it highly sensitive to noise and outliers. However, this is probably not our case since we considered 6 of them. Additionally, high dimensionality also influences the results: if the dataset has many features, the "curse of dimensionality" makes distance calculations less meaningful. This affects the ability of k-NN to classify accurately.

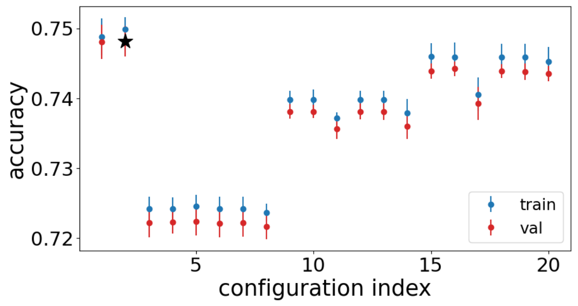
Regarding the training cost of the model, we believe that the time spent performing the hyperparameter tuning is not really a meaningful metric to compare the different models because it may vary according to the number of values tested. On the other hand, we can consider the time to perform the fit on the training set and the predict on the whole dataset as a more relevant indicator of the model's cost, even if it is machine-dependent. The fit of the k-NN model takes 0.01 s, while the predict takes 11.25 s. In fact, it does not construct a general internal model, but simply stores the instances of training data. Then, during the predict phase, it computes the distances between the records to find the closest neighbors.

2.4 Second model: Logistic Regression

2.4.1 Default model performance evaluation. Logistic Regression assumes that the data is linearly separable, so if the features of some attacks are not, the model struggles to classify them (e.g., *ddos_mssql*). Moreover,



(a) Grid search k-NN.



(b) Grid search Logistic Regression.

Fig. 9. Hyperparameter tuning k-NN and Logistic Regression models.

logistic regression cannot model complex relationships between features, meaning that its simplicity can result in poor classification performance when there are intricate patterns in certain attacks.

The model exhibits an accuracy (74 %) in both the training and test sets. Since the performance is acceptable and the two sets have similar accuracy, we cannot detect overfitting or underfitting. Using the default parameters, the model is not able to converge, which means that it was still learning when it reached the maximum number of iterations.

2.4.2 Hyperparameter tuning. We optimized the following parameters:

i) *solver*: algorithm to use in the optimization problem. For multiclass problems, we can use the `lbfgs`, `newton-cg` and `saga` solver. Since the model did not reach convergence even after a high number of iterations with the last solver, we decided to use the first two.
ii) *penalty*: it specifies the norm of the penalty. If set to `None` no penalty is added; while with `l2` a L2 penalty term is added.
iii) *C*: inverse of regularization strength; as in support vector machines, smaller values specify stronger regularization.
iv) *tol*: tolerance for stopping criteria.

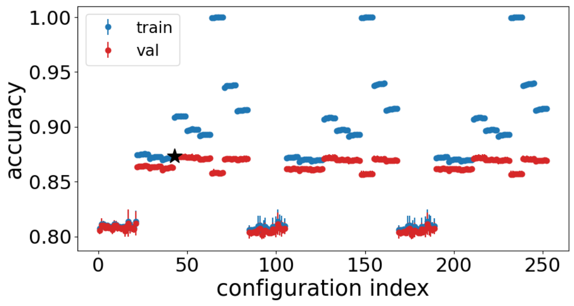
C and *tol* can only be set if *penalty* is set to `l2`.

2.4.3 Results. The combination of hyperparameters that leads to the highest accuracy in the validation set (75 %) is *solver*=`newton-cg` and *penalty*=`None`. Since the accuracy in the training set is very similar, there is no overfitting. Fig. 9b shows the results of the performed grid search. In particular, we can notice that some combinations, with *penalty*=`l2` and a low value of *C*, lead to lower accuracy than others.

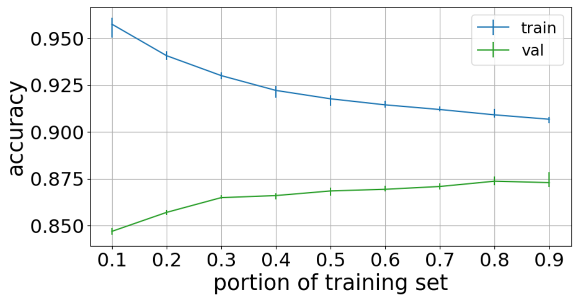
Among the three models analyzed here, Logistic Regression is the one with the highest cost: the fit phase takes 46.91 s, since it has to build the model. The predict, instead, is very short, just 0.01 s.

2.5 Third model: Random Forest

2.5.1 Default model performance evaluation. The model achieves excellent performance (100 %) in the training set. Although the performance on the test set is high (86 % accuracy), the gap between training and test accuracy suggests that the model is memorizing the training data rather than generalizing well to unseen data, which is a clear indication of overfitting. In particular, looking at the default parameters, we discovered that the tree nodes are expanded until all leaves are pure, so it may be too strongly affected by the training data. We will address this issue in the hyperparameter tuning phase. Furthermore, all



(a) Grid search Random Forest.



(b) Learning curve Random Forest.

Fig. 10. Hyperparameter tuning on Random Forest model.

metrics (precision, recall, F1-score) are 1.00 for every class in the training set, but some classes, such as *ddos_mssql*, *ddos_udp*, and *ddos_udp_lag*, show a marked drop in performance in the test set.

2.5.2 Hyperparameter tuning. For the Random Forest model, we optimized the following parameters:

i) *n_estimators*: this represents the number of trees in the forest. A value that is too low may result in underfitting, while an excessively high value increases training time without significant performance improvements.
ii) *max_depth*: the maximum depth of the trees. Excessive depth can lead to overfitting, while insufficient depth may cause underfitting.
iii) *min_samples_leaf*: the minimum number of samples required to be at a leaf node. A number that is too low may cause overfitting.
iv) *criterion*: the function used to measure the quality of a split. Optimizing this parameter can enhance the model's performance.

2.5.3 Results. The results of the grid search performed are shown in fig. 10a. It is evident that with some combinations we obtained lower accuracies, around 80 %: these cases are the ones in which the depth of the trees is up to 5, which is definitely not enough. Moreover, increasing the number of trees does not bring any advantage in terms of performances, only more costs. In general, the best performances are obtained using *max_depth* = 15: it is better not to completely expand all the nodes, as it may lead to overfitting. Trying to run the hyperparameter tuning multiple times, we obtained different best combinations, since the performance variations are very slight and they may be affected by the split selected. For this reason, we chose one of the least computationally expensive, since it is based on a low number of estimators.

The combination of hyperparameters that leads to the best accuracy in the validation set (87 %) is: *n_estimators* = 50, *max_depth* = 15, *min_samples_leaf* = 1, and *criterion* = `gini`. In this case, we obtained an accuracy of 90 % in the training set: since both sets have a similar accuracy value, there is no more overfitting. Our model still performs better in some classes than in others, having a precision of only 62 % in the test set in *ddos_udp* and *ddos_udp_lag*.

Regarding the costs, we measured that the fit takes 1.38 s, while the predict phase 0.22 s. During the fit phase, in fact, the model creates the decision trees, while during the predict phase it simply compares the features values with the ones in the nodes to choose the best path for that sample.

2.6 Selection of the best model

This final step involves selecting the model that shows the best performance in multiple metrics. By analyzing the classification report of all the models in both the training and test sets, it is evident that the Random Forest and k-NN models outperform Logistic Regression. Furthermore, Random Forest is generally more effective than k-NN in several metrics, making it the better choice:

i) Higher test accuracy: Random Forest achieves 87 % accuracy, compared to 80 % for k-NN.
ii) Better Macro and Weighted Averages: Random Forest has a macro average F1-score of 0.88, while k-NN achieves 0.81.
iii) Consistent Precision and Recall: Random Forest maintains higher and more consistent precision and recall values in the test set for most classes, ensuring fewer false positives and false negatives.

As for the costs, it is clear that Logistic Regression is the least efficient. Random Forest is slower during the fit phase than during the predict one, while for k-NN, it is the opposite. However, in a real-world scenario, we would like to train our model and then efficiently and rapidly predict labels for new data. For this reason, we would prefer Random Forest rather than k-NN.

In conclusion, we do believe that Random Forest is the best model because it generalizes better to unseen data, provides higher accuracy, and demonstrates superior precision, recall, and F1-score across all classes and a lower cost compared to k-NN.

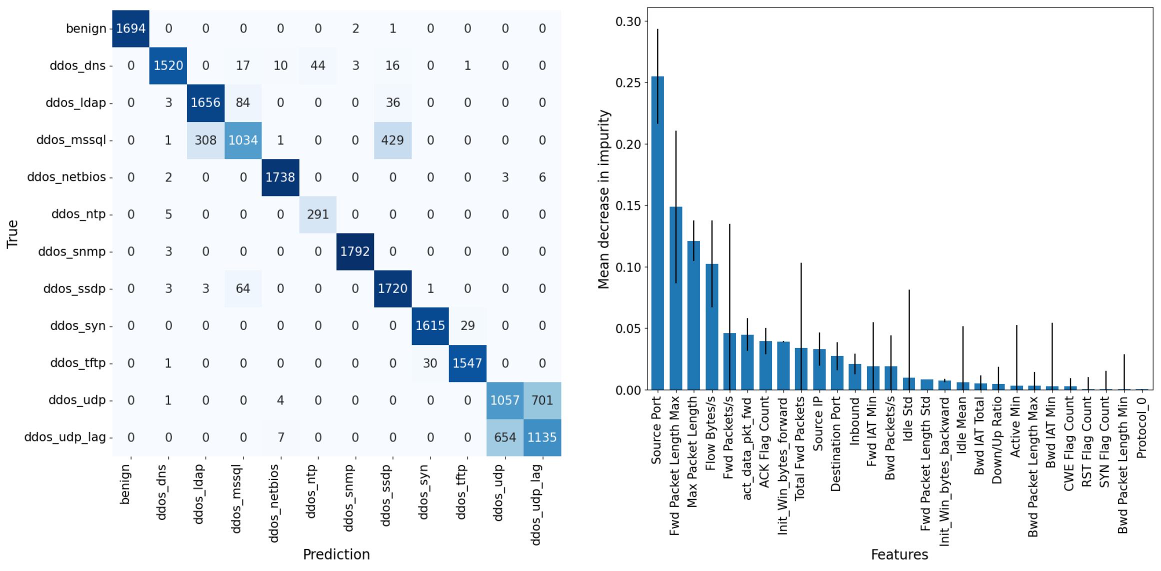
2.7 Learning curve

To determine the optimal proportion of the dataset for training the selected model, we plot the learning curve. This helps evaluate how the model's performance on the validation set varies as the training size increases. The goal is to identify the minimum training size required to achieve the desired validation accuracy and to assess whether the collection of additional data would significantly improve performance. For each portion, we used random stratified sampling (with $k = 5$).

Observing fig. 10b, we can conclude that a 70 % training set proportion is quite good, as it results in an accuracy of 87 % (with a small margin error, as indicated by the vertical band). This is quite close to the training curve (the blue one), which shows an accuracy of 91 %. It can also be useful to note that a portion of 90 % approximately leads to the same accuracy. However, it is more efficient to use the smaller portion, as using fewer samples for training likely results in a faster process.

2.8 Result investigation

2.8.1 Performance evaluation on the test set. After performing the hyperparameter tuning we obtain an accuracy of 87 % on the test set using the Random Forest model. We can observe the most misclassified attacks in the confusion matrix shown in fig. 11a. In particular, our model tends to confuse *ddos_ldap* and *ddos_mssql*, *ddos_udp* and *ddos_udp_lag*, *ddos_ssdp* and *ddos_mssql*. While for the first two cases there is more or less the same amount of misclassified objects in the two directions, this is not true for the last one: the model tends to predict *ddos_ssdp* for *ddos_mssql* much more than vice versa. Furthermore, these groups reflect those we already identified in §1.4: since these attacks have similar features values, the model struggles to discern them. Although the *ddos_ntp* class is underrepresented in comparison to the others, the model correctly classifies almost all of its samples. Finally, we can also observe that the *ddos_dns* class tends to have some false negatives, which are distributed between different attacks. Fig. 11b highlights the most important features for our model. In particular, it is evident that its decisions are particularly based on the source port, the maximum packet length, and the transmission rates. The least useful, instead, are the number of CWE, RST and SYN flags and the usage of the protocol HOPOPT. The confused classes have similar values for the most important features.



(a) Confusion matrix.

(b) Most important features for classification.

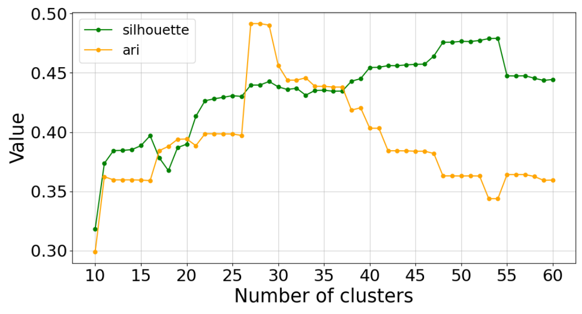
Fig. 11. Metrics about the optimized Random Forest.

2.8.2 TP, TN, FP and FN. In conclusion, we can analyze what the different metrics mean in the context of DDoS attacks. The true positive (TP) and true negative (TN) samples are those correctly classified by our model. Both false positives (FP) and false negatives (FN) are misclassified samples. In the FP case, we focus on a specific predicted class: they are given by the sum of all the elements in the column of that predicted label except for the TP. Our model does not classify malicious traffic as benign. This is really important in a real-world scenario because otherwise we would have undetected attacks. FN, instead, focus on a given actual class: they are given by the sum of all the elements on the same row except the TP. We can observe that only three samples have been predicted as malicious attacks, even though they belong to benign traffic. This is again fundamental because if the model generates too many alarms, the security systems may be overwhelmed, reducing their operational efficiency.

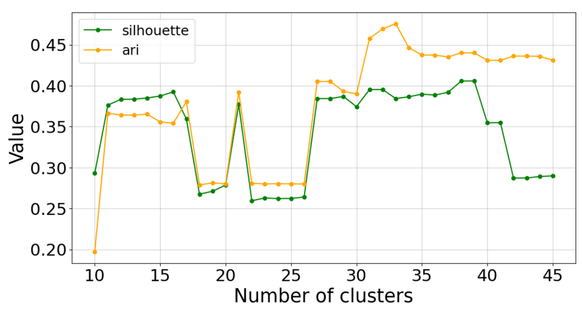
2.8.3 Precision and recall. The precision metric measures the number of objects correctly assigned to a class (TP) over the number of objects assigned to that class (TP and FP). Having a high precision is crucial, as missed attacks represent significant threats. Recall is the number of objects correctly assigned to a class (TP) over the number of objects belonging to that class (TP and FN). Obtaining a high recall is important to minimize false alarms and maintain reliability in the system.

3 Unsupervised learning – clustering

The goal of this section is to group network flows into clusters that exhibit similar, correlated, or coordinated patterns. A cluster is defined as a group of discrete items that are close to each other. Therefore, the objective of clustering is to identify groups of similar data points that are in some sense homogeneous. Unlike Section 2, this analysis is performed in an unsupervised manner, independently of the attack labels. In §4, we will investigate whether distinct 'families' or 'groups' of attacks naturally emerge based on the features retained in the dataset.



(a) Silhouette k-Means.



(b) Silhouette GMM.

Fig. 12. Hyperparameter tuning k-Means and GMM.

3.1 Clustering algorithms

3.1.1 Algorithms description. We have decided to perform our analysis using three different algorithms:

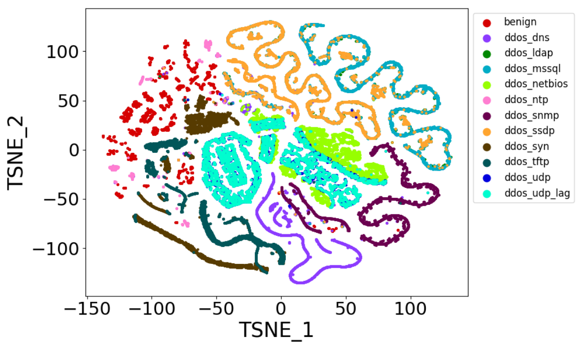
- k-Means: a hard clustering algorithm, in which each data point is assigned to a single cluster (among the k available clusters). In this algorithm, each cluster is associated with a point (called *mean* or *centroid*).
- Gaussian Mixture Model (GMM): a soft clustering algorithm. In this technique, each data is associated with the degree of belonging to each cluster, which can be interpreted as the probability that the i-th data-point belongs to the cluster c. In GMM, each cluster is represented as a random multidimensional Gaussian distribution with its own mean, standard deviation, and covariance.
- DBSCAN: a hard clustering algorithm based on connectivity clustering; data points need to be connected to core points defined as the ones having a minimum number of neighbors.

3.1.2 Performance metrics and indicators. In order to determine the best algorithm, the best hyper-parameters and the number of clusters, we used several metrics:

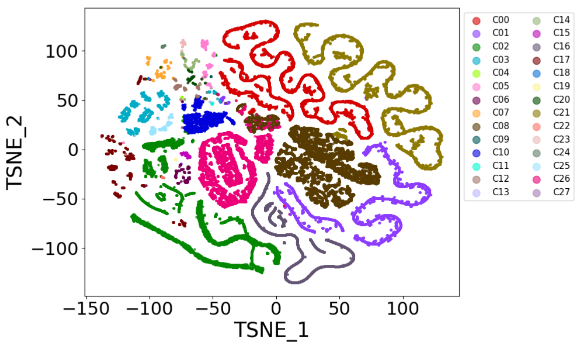
- Silhouette: measures the consistency within clusters of data, indicating how similar a data point is to its own cluster (cohesion) compared to other clusters (separation). It is defined for each sample by using two values: the mean distance between the sample and all the other points in the cluster, and the mean distance between the samples and all the other points in the next nearest cluster.
- Rand Index (RI): measures the similarity between two assignments.
- Adjusted Rand Index (ARI): corrects the RI ensuring to obtain values close to 0.0 for a random labelling.

3.2 First algorithm: k-Means

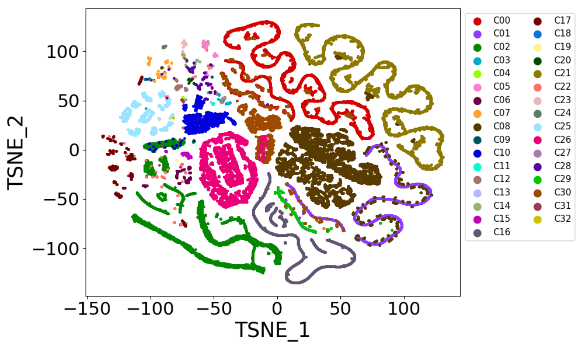
3.2.1 Initial parameters and performance. For the k-Means algorithm, we have only analyzed the *n_clusters* parameter, which indicates the number of clusters to be created. Regarding the *init* hyper-parameter, we left the default option `k-means++` because it makes centroids to be equally distant from each other, thus leading to better results. Initially, we set $n\_clusters = 12$, which is the number of labels. From this initial analysis, we obtained a silhouette score of 0.38, a RI of 0.81 and an ARI of 0.36, which can be considered as an acceptable starting point: clusters are fairly separated, and they somehow reflect the GT labels. The RI value is not really meaningful, since it can look good simply by random chance. Three of the detected clusters have almost no samples inside, likely representing outliers.



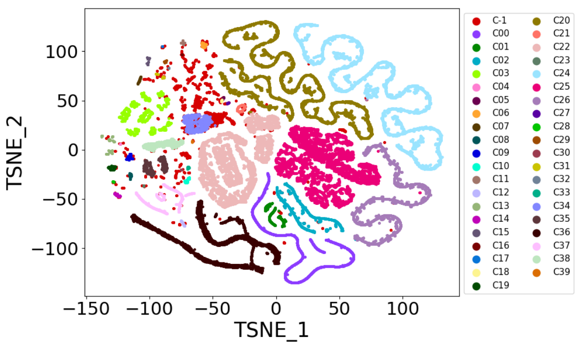
(a) t-SNE Projection original labels (GT).



(b) t-SNE Projection Plot k-Means 28 clusters.



(c) t-SNE Projection Plot GMM 33 clusters.

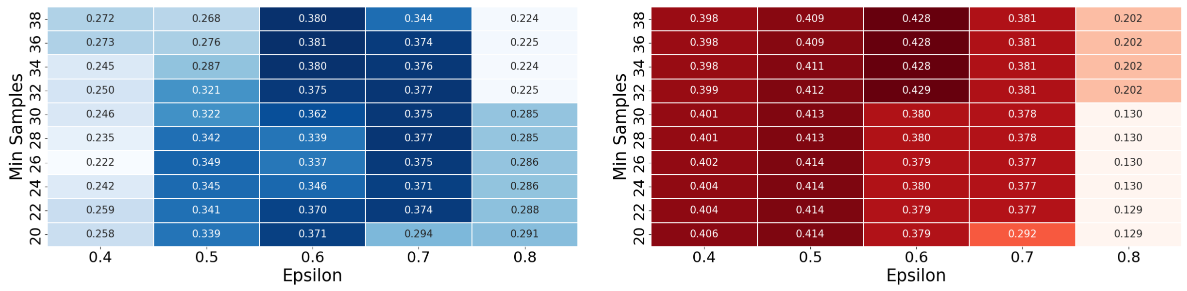


(d) t-SNE Projection DBSCAN best parameters.

Fig. 13. t-SNE Projection Plots.

3.2.2 Hyper-parameter tuning. Next, we want to see if the previous results can be improved. By varying *n_clusters*, as shown in fig. 12a, we observed that the best possible value in terms of the silhouette is 0.48, which can be obtained with *n_clusters* = 54. The values for the other metrics are: Clustering Error = 162121, RI = 0.91 and ARI = 0.34. It is evident that the silhouette score notably increased, hence the clusters are now quite separated. On the other hand, increasing the number of clusters does not boost the ARI: in particular, the highest value (0.49) is obtained using 28 clusters, with a silhouette score of 0.44. The clustering error, used only for the k-Means analysis, favors high values of *n_clusters*: by increasing the number of clusters the average distance from the points to the corresponding cluster means (which defines the clustering error) cannot decrease. Therefore, it is better to focus on the other metrics when evaluating the performance of this algorithm.

3.2.3 Cluster Visualization and further analysis. To visualize the obtained clusters we use t-SNE (t-distributed Stochastic Neighbor Embedding), which is a powerful technique for dimensionality reduction and data visualization. The *perplexity* hyperparameter is important for the t-SNE algorithm, as it controls the effective number of neighbors that each point considers during the dimensionality reduction process. We ran a loop to calculate the Kullback-Leibler (KL) Divergence metric between the high and low-dimensional probability distributions, on various *perplexity* values from 5 to 120 with a 5-point gap, noticing that it becomes quite constant after 100. We also tried to visualize clusters using different



(a) Silhouette DBSCAN.

(b) ARI DBSCAN.

Fig. 14. Hyperparameter tuning DBSCAN.

perplexity values, and we observed that with 50 and 115 the results were pretty similar. We selected a *perplexity* value of 50, as the documentation suggests that typical values are between 5 and 50. At this point, we can visualize the 28 clusters obtained in fig. 13b. This is the clustering that best reflects the labels, leading to the highest ARI. It is evident that the clusters reflect the classes quite well, even though the model groups together *ddos_netbios* and *ddos_udp_lag* samples and it does not discern *ddos_udp* and *ddos_udp_lag*, as well as *ddos_mssql* and *ddos_ssdp*. The *benign* traffic is scattered among many different clusters. A more in-depth analysis will be conducted in section §4.

3.3 Second algorithm: Gaussian mixture model (GMM)

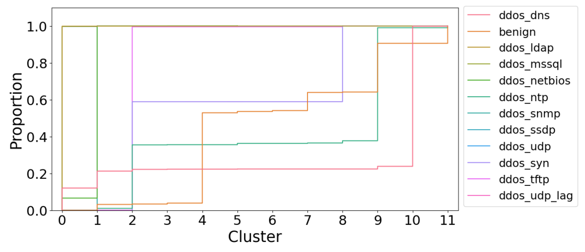
3.3.1 Initial parameters and performance. As regards the GMM algorithm, we analyzed the *n_components* parameter, which indicates the number of clusters to be used. We set *n_components* = 12, which is the number of labels. From this initial analysis, we obtained a silhouette score of 0.38, a RI of 0.81, and an ARI of 0.36. As with k-Means, three clusters are nearly empty, while the first two contain different types of traffic.

3.3.2 Hyper-parameter tuning. By varying *n_components*, as shown in fig. 12b, we observed that the best possible value in terms of silhouette is 0.405, which can be obtained with *n_components* = 39. The other metrics are RI = 0.87 and ARI = 0.44, that are slightly lower than those obtained with k-Means. The best ARI (0.476) is achieved using 33 clusters, with a silhouette score of 0.38. In fig. 13c we visualize the clusters obtained with GMM using *n_components* = 33: the clustering obtained is quite similar to the one obtained with k-Means, but *ddos_snmp*, *ddos_tftp* and *ddos_ssdp* are divided into more clusters.

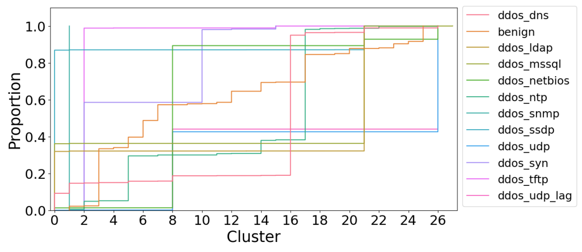
3.4 Third algorithm: DBSCAN

3.4.1 Initial parameters and performance. The DBSCAN algorithm autonomously detects the optimal number of clusters, but we need to optimize two parameters by performing a Grid Search: *eps*, which represents the maximum distance within which two points are considered connected, and *min_samples*, which defines the minimum number of neighbors required for a point to be qualified as a core point. Using default parameters, we obtained a low silhouette score (0.12) and a low ARI (0.26), with 150 clusters and 2144 noise points.

3.4.2 Hyper-parameter tuning. Initially, we optimized the silhouette score individually for the two parameters, keeping all other parameters at their default values. We focused on smaller values of *eps*, as larger values significantly slow down the analysis. Subsequently, we conducted a more detailed search

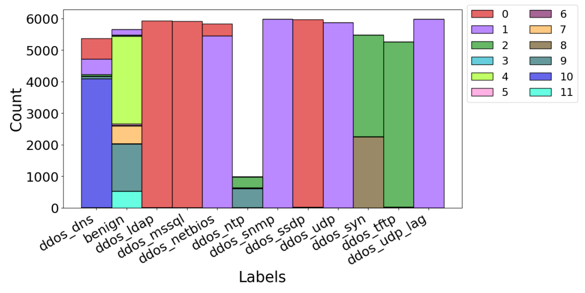


(a) k-Means with 12 clusters.

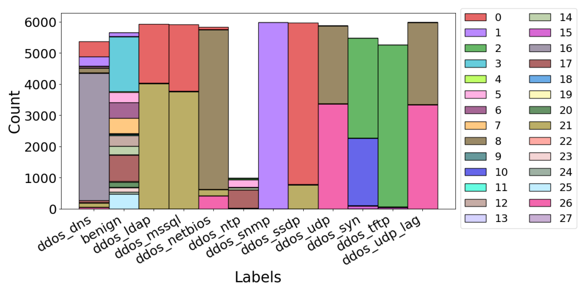


(b) k-Means with 28 clusters.

Fig. 15. Number of clusters assigned to each class.



(a) k-Means with 12 clusters.



(b) k-Means with 28 clusters.

Fig. 16. Attacks distribution between clusters.

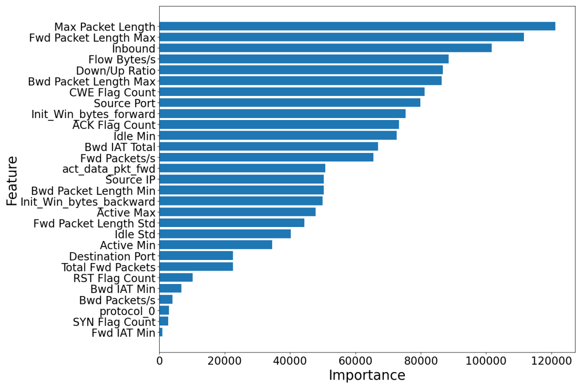
around the identified optimal values for *eps* and *min_samples*. The results are presented in fig. 14 using two heatmaps that illustrate the variation in silhouette scores and ARI. Overall, the best parameter values we obtained are $eps = 0.6$ and $min\_samples = 36$, leading to a silhouette score of 0.38, ARI = 0.43, 41 clusters detected, and 4575 noise points. We visualize the resulting clusters in fig. 13d using t-SNE: the outcome is similar to those obtained with the other models, but part of *benign* and *ddos_syn* traffic is classified as noise.

4 Clusters explainability and analysis

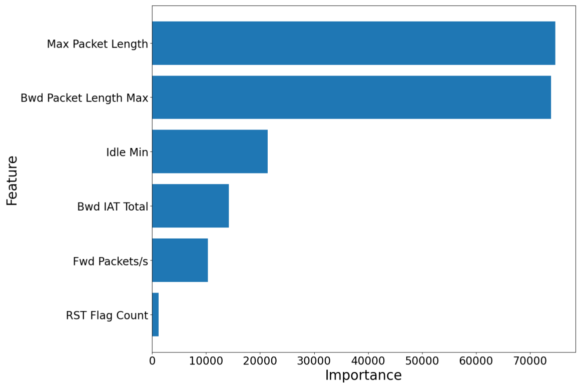
From the previous section, we can conclude that the k-Means algorithm delivers the best performance. Consequently, we will focus our further analysis on this model, using both 12 and 28 clusters, which correspond to the number of labels and to the option that leads to the best ARI, respectively. In particular, throughout this section we will provide a deeper analysis of the clusters detected, using also the GT labels. Furthermore, we will detect the most important features for our model using permutation importance. Finally, we will identify the sub-attacks and investigate why some attacks are confused by our model.

4.1 Cluster distribution of traffic types

We may think that the detected clusters reflect more or less the labels of our samples. However, real traffic is more complex and, as already seen in §2, models struggle to classify certain types of attacks. Fig. 15 represents the ECDF of number of clusters assigned to each class, while fig. 16 visualizes how samples of each attack are distributed among clusters. In particular, from fig. 15a and fig. 16a, it is



(a) Permutation feature importance.



(b) Permutation feature importance correlated features.

Fig. 17. Feature importance on k-Means with 28 clusters.

evident that approximately all samples of some attacks are assigned to the same cluster: *ddos_netbios*, *ddos_snmp*, *ddos_udp*, and *ddos_udp_lag* flows are all included in cluster 1, while those of *ddos_ldap*, *ddos_mssql* and *ddos_ssdp* belong to cluster 0. All *ddos_tftp* samples and half of the *ddos_syn* ones are in cluster 2. On the other hand, *benign* traffic is scattered among different clusters, meaning that it presents many different patterns: clusters 4, 7 and 11 are characteristic of harmless flows, while more than a thousand samples are assigned to cluster 9, which also contains *ddos_ntp* traffic. Hence, they may have some common elements, since also the SVM model tends to confuse them (appendix B.2). *ddos_dns* is probably quite different from the others attacks, since the majority of its samples are assigned to cluster 1, that contains only this type of traffic. Looking at fig. 15b and fig. 16b, we can notice that the model still confuses the same classes of before, but the samples of many attacks are now divided into two or more clusters, suggesting the presence of sub-attacks. For instance, *ddos_ldap*, *ddos_mssql* and *ddos_ssdp* are split into a couple of clusters and the first two also in the same proportion. Same thing for *ddos_udp*, *ddos_udp_lag* and *ddos_netbios*. On the other hand, *ddos_snmp* flows now belong to a completely different cluster. The *benign* class is still scattered among different clusters, but mantains cluster 17 in common with *ddos_ntp*. Moreover, *ddos_syn* flows are still split into two clusters, so it has two main sub-attacks. Finally, *ddos_tftp* traffic is still clustered together: its traffic is very homogeneous.

4.2 Feature importance analysis

In this section, we attempt to detect the most important features in the trained model. For this purpose, we used the permutation feature importance: this technique consists in taking a feature and randomly shuffling its values, breaking any kind of relationship between that feature and the result. Then, it applies the already trained model to the dataset with these shuffled features: if performance decreases significantly compared to the one with the original dataset, it means that that feature is really important to our model. Fig. 17a shows the outcome of this technique applied to the k-Means model with 28 clusters. The most important features are *Max Packet Length* and *Fwd Packet Length Max*, while the least ones are *protocol_0* and *Fwd IAT Min*. *Source IP*, *Source Port* and *Destination Port* are not really important to our model, meaning that it may be able to generalize other data. We also applied this technique to the k-Means model with 12 clusters, obtaining similar results. The *Max Packet Length* was already an important feature for the Random Forest model, as analyzed in §2.8.

Even though we already performed dimensionality reduction during the preprocessing phase, there are still quite high-correlated features. When features are collinear or highly correlated, permuting one feature has little effect on the models' performance because it can get the same information from a correlated feature. One way to address this is by performing hierarchical clustering on the Spearman rank-order correlations, picking a threshold, and keeping a single feature from each cluster. Finally, we performed the permutation feature importance using the dataset containing only the selected features. The result is shown in fig. 17b and it almost reflects the one obtained before.

4.3 Attack pattern investigation

In the previous paragraphs, an in-depth analysis has been performed to explain how some attacks might contain sub-attack patterns, as *ddos_syn*. We can hypothesize that the most important features identified in §4.2 are those that make two samples being clustered together or not. Hence, to complete our analysis, we wrote a custom function to detect the most diverse features among two given clusters containing the same label, with the aim of understanding which features contribute to uniquely identifying some particular attack patterns. For instance, we discovered that the most discriminative features for *ddos_syn* between cluster 2 and cluster 10 are *Destination Port*, *Fwd Packets/s*, and *Flow Bytes/s*. By applying the same function to other clusters and classes, we noticed that in most cases these three features are the most discriminative, along with *Total Fwd Packets*, *Bwd IAT Total* and *Active Max*. This means that the different sub-attacks have different values for those features, so they are classified into different clusters.

On the other hand, the attacks clustered together have similar values for those features. Therefore, we tried to find the most diverse features between two classes in the same cluster, in order to better understand why flows belonging to different classes are grouped together. In general, results show that even the most different features have quite similar values among the confused classes, meaning that the model struggles to discern them. *Fwd Packets/s*, *Flow Bytes/s*, *Fwd Packet Length Max*, *Source Port*, and *Destination Port* are the most distinctive features, but still too similar.

In conclusion, we analyze similarities between attacks from the perspective of their definition or attack methodology. The *ddos_ssdp*, *ddos_ldap* and *ddos_mssql* attacks are amplification and reflection based: the attacker sends a small command to the accomplices resulting in a large amount of traffic being sent to their victim. Hence, he needs fewer accomplices to overwhelm the victim. Both *ddos_udp* and *ddos_udp_lag* attacks are UDP-based and their goal is to cause a performance degrade. In particular, in UDP flood attack this is achieved by sending numerous UDP packets to random ports at a very high rate, while UDP-LAG attack is mostly used in online gaming where the attacks want to slow down or interrupt the movement of other players to outmaneuver them. Finally, both *ddos_syn* and *ddos_tftp* attacks are performed over TCP (fig. 4a) and aim to consume resources on the victim server.

5 Conclusions

In conclusion, supervised and unsupervised ML models allow us to automate the detection and analysis of flows generated during DDoS attacks. In particular, they are able to distinguish benign from malicious traffic, as well as identify the class of attack. However, they are not able to perfectly discern all of them, since some exhibit similar patterns. Using supervised learning techniques, we can predict the correct label with high precision on unseen data. This means that the trained model could be used to classify traffic from future attacks in the same context. On the other hand, unsupervised learning models allow us to group the traffic into different categories that somehow reflect the original labels, although we identified some confused classes and sub-attacks. Therefore, these results can be used to develop effective defenses against DDoS attacks, as well as to study their characteristics and patterns more in depth.

References

[1] Sara Braidotti, Chiara Iorio, Matteo Pani, and Cristian Sapia. Ddos attacks detection and characterization (source code). https://github.com/iochia02/ML4N_project/, 2025.

[2] Luca Vassio, Gabriele Ciravegna, Zhihao Wang, and Tailai Song. Course material, 2024.

[3] NumPy Developers. NumPy reference; NumPy v2.2 Manual — numpy.org. https://numpy.org/doc/stable/reference/, 2024.

[4] Pandas Developers. API reference; pandas 2.2.3 documentation — pandas.pydata.org. https://pandas.pydata.org/docs/reference/index.html, 2024.

[5] scikit-learn developers. sklearn — scikit-learn.org. https://scikit-learn.org/stable/api/sklearn.html, 2024.

[6] Utkarsh Kant. Complete Guide to Categorical Features Encoding Guide — kantschants.com. https://kantschants.com/complete-guide-to-encoding-categorical-features, 2024.

[7] Mohammed M. Alani. Handling IP Addresses in Machine Learning Datasets. https://www.mohammedalani.com/tutorials/handling-ip-addresses-in-machine-learning-datasets/, 2024.

[8] Eugenio "Jay" Zuccarelli. Performance Metrics in Machine Learning — Part 3: Clustering. https://towardsdatascience.com/performance-metrics-in-machine-learning-part-3-clustering-d69550662dc6?gi=57c437b7e456, 2021.

[9] Kemal Erdem (burnpiro). t-SNE clearly explained — towardsdatascience.com. https://towardsdatascience.com/t-sne-clearly-explained-d84c537f53a?gi=3e26dc9f50b7, 2020.

[10] Introduction to LDAP Injection Attack — cobalt.io. https://www.cobalt.io/blog/introduction-to-ldap-injection-attack#:~:text=Denial%2Dof%2DService%20(DOS,to%20crash%20or%20become%20unresponsive.), 2023.

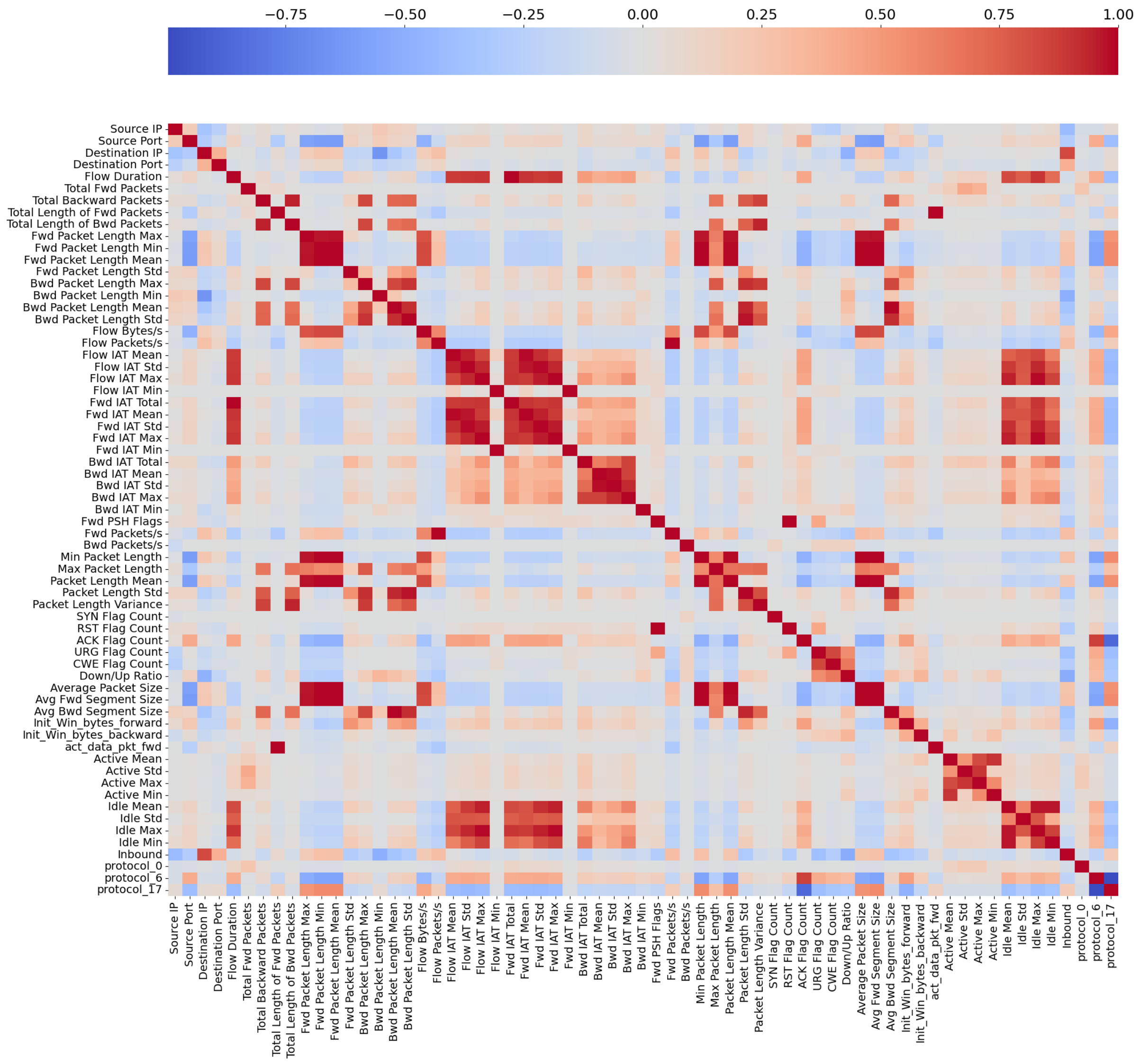
[11] Inc Cloudflare. SSDP DDoS attack. https://www.cloudflare.com/it-it/learning/ddos/ssdp-ddos-attack/, 2025.

[12] Inc Cloudflare. SYN Flood attack. https://www.cloudflare.com/it-it/learning/ddos/ssdp-ddos-attack/, 2025.

[13] Akamai Technologies. UDP Flood DDoS. https://www.akamai.com/glossary/what-is-udp-flood-ddos-attack#:~:text=UDP%20flood%20is%20a%20type,require%20few%20resources%20to%20execute., 2025.

[14] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A. Ghorbani. DDoS 2019. https://www.unb.ca/cic/datasets/ddos-2019.html, 2019.

A Correlation matrix using the Pearson correlation coefficient



B Other supervised learning ML models

The objective of this section is to investigate the two techniques of supervised learning that we decided not to talk about in §2.

B.1 Naive Bayes Classifier

Naive Bayes Classifier (Gaussian NB) is a probabilistic model based on Bayes' theorem, assuming that the features follow a Gaussian (normal) distribution.

B.1.1 Default model performance evaluation. As Logistic Regression, the Naive Bayes Classifier works in the space of linear maps. This probabilistic classifier assumes that the features follow a Gaussian distribution, supposing them to be independent from each other: its goal is to estimate the mean and standard deviation of these distributions. However, the features in our dataset are not completely independent and do not follow a perfect Gaussian distribution; moreover, the model is sensitive to noise and outliers.

The model exhibits an accuracy of 56 % in both the training and test sets, suggesting underfitting. We will address this issue during the hyperparameter tuning phase. Additionally, metrics such as precision, recall, and F1-score are quite poor in some classes. In particular, they are zero on both the training and test set for the *ddos_udp* class, while *ddos_udp_lag* has low precision by high recall, since many samples tend to be recognized as this attack by the model (a large amount of false positives impacts precision but not recall). Furthermore, the model struggles to correctly classify *ddos_ldap*, *ddos_mssql*, and *ddos_tftp* samples.

B.1.2 Hyperparameter tuning and results. In the Naive Bayes Classifier we tried to optimize the *var_smoothing* parameter, which indicates the largest variance of all features that are added to variances for calculation stability. Using *var_smoothing* $= 10^{-1}$ we obtained an accuracy of 65 % in both the training and the validation set: after performing the hyperparameter tuning, the class *ddos_udp* still has zero precision, recall, and F1-scores, while the performances on some other classes, such as *ddos_mssql* and *ddos_tftp*, improved. This is the model in which the accuracy improved the most after hyperparameter tuning.

Although the performances of this model are not excellent, it has multiple pros. The main one is its efficiency: both the fit phase took only 0.01 s, while the predict 0.05 s.

B.2 Support Vector Machines

Support Vector Machines (SVM) refer to a class of powerful algorithms that find the optimal hyperplane to separate data points of different classes. SVM is particularly effective in high-dimensional spaces and when there is a clear margin of separation between classes. The choice of kernel (e.g., linear, polynomial, or RBF) plays a crucial role in its performance, allowing it to handle both linear and non-linear classification tasks. However, the C-Support Vector Classification algorithm provided in the `sklearn.svm` library does not seem to be the best choice for our dataset, since the fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. Indeed, for large datasets it is suggested to use `LinearSVC`.

B.2.1 Default model performance evaluation. In linear SVC, multiclass support is handled according to a one-vs-the-rest scheme. It tries to find a linear hyperplane that separates the data. Using the default parameters, it has an accuracy of 70 % in both the training and the test sets, suggesting underfitting. In addition to the misclassifications already described in §2.2.3, it performs very poorly in the *ddos_mssql* class, since it often predicts its samples as belonging to *ddos_ldap*. It also assigns to *ddos_netbios* class *ddos_udp* and *ddos_udp_lag*. Moreover, it tends to assign malicious traffic (*ddos_ntp*) to the *benign* class.

B.2.2 Hyperparameter tuning. We tried to optimize:

i) *tol*: the tolerance for the stopping criteria.
ii) C: a regularization parameter; it is inversely proportional to the size of the margin. By using a smaller value and obtaining a larger margin, the classifier becomes more flexible and allows more classification mistakes. A high value may lead to overfitting.

B.2.3 Results. The combination of hyperparameters that leads to the highest accuracy in the validation set (70 %) is: $C = 100$ and $tol = 10^{-5}$. The model performance did not improve much after the hyperparameter tuning phase, which means that it is probably not very suitable for classifying our dataset. However, we can highlight that it tends less to classify the *ddos_ntp* flows as *benign*.

This model has the highest cost among the analyzed models: in particular, the fit phase takes 393.81 s for Linear SVC.