DDoS Attack Detection and Classification

Abstract

This report presents a comprehensive project in the field of machine learning, centred around the analysis and mitigation of Distributed Denial of Service (DDoS) attacks. The project starts with the exploration and visualization of a provided DDoS dataset, aiming to gain insights into the characteristics and patterns of the network traffic associated with such attacks. The visualization and exploration phase leverages data visualization techniques to unveil key features and trends within the dataset.

Subsequently, the project delves into the application of supervised learning techniques for classification purposes. Multiple machine learning models are employed to classify network traffic instances: KNN, Support Vector Machines, Gaussian Naive Bayes and Random Forest. The evaluation of these models involves metrics such as accuracy, precision, recall, and F1 score, providing a comprehensive assessment of their performance in identifying DDoS attacks.

The project incorporates clustering techniques to uncover hidden structures such as similar “families” of attacks within the dataset. By applying clustering algorithms such as K-means, DBSCAN and Gaussian Mixture; the project aims to group similar network traffic instances together. Evaluation of clustering results is performed using several unsupervised and supervised metrics.

Finally, feature importance is retrieved from the clustering algorithm with the combination of the clustering labels and a supervised tree-based classifier. This process aims to provide explainability of the decision process involved in machine learning algorithms such as the clustering ones.

Section 1- Data exploration and pre-processing

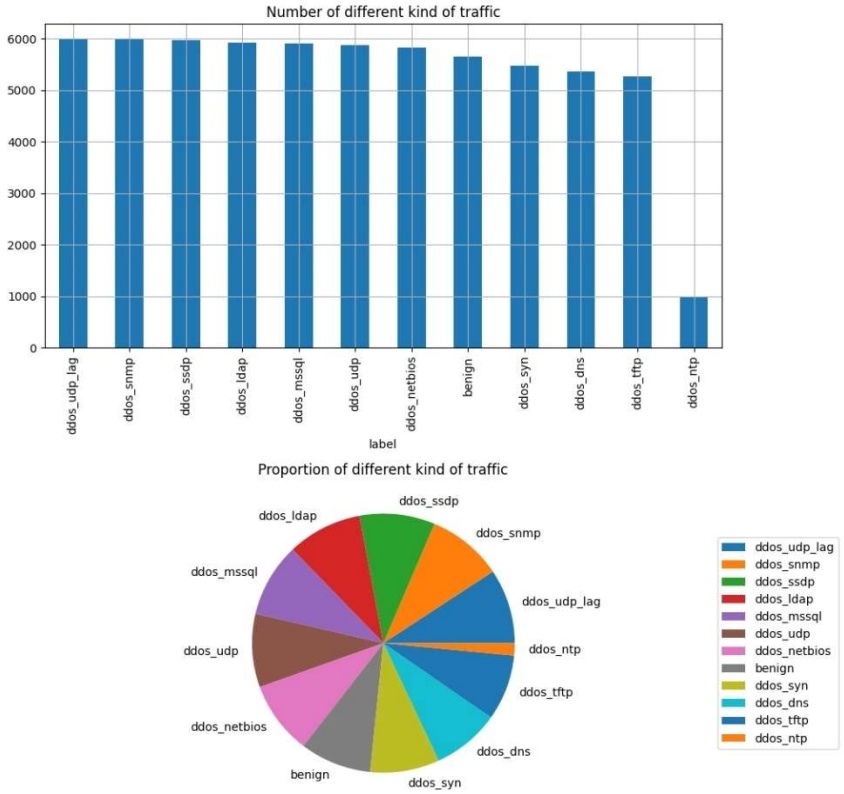
Throughout this exploration, we will leverage statistical analyses, visualizations, and correlation assessments trying to retrieve information from the dataset. As we navigate the intricacies of the dataset, it is imperative to recognize that this analysis represents a snapshot in time, and the cybersecurity landscape is dynamic. Therefore, the insights gleaned from this exploration should be considered within the context of contemporary cyber threats.

The dataset is characterized by 12 different labels which are briefly described as follows:

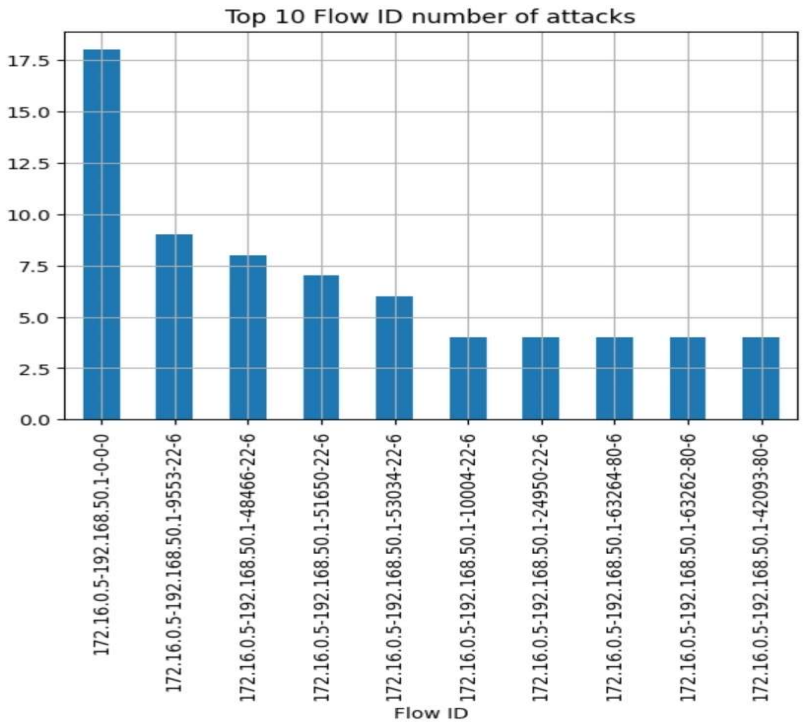
DDoS DNS	A DDoS DNS attack is a type of denial-of-service (DoS) attack that floods a DNS server with requests, making it unavailable to legitimate users. This can prevent users from accessing websites or other online services that rely on DNS.
DDoS LDAP	A DDoS LDAP attack is a type of DoS attack that floods an LDAP server with requests, making it unavailable to legitimate users. This can prevent users from accessing directory services, such as Active Directory.
DDoS MSSQL	A DDoS MSSQL attack is a type of DoS attack that floods an MSSQL server with requests, making it unavailable to legitimate users. This can prevent users from accessing databases or other online services that rely on MSSQL.
DDoS NetBIOS	A DDoS NetBIOS attack is a type of DoS attack that floods a NetBIOS server with requests, making it unavailable to legitimate users. This can

	prevent users from accessing NetBIOS-based services, such as file sharing or printing.
DDoS NTP	A DDoS NTP attack is a type of DoS attack that floods an NTP server with requests, making it unavailable to legitimate users. This can disrupt time synchronization for devices that rely on NTP.
DDoS SNMP	A DDoS SNMP attack is a type of DoS attack that floods an SNMP server with requests, making it unavailable to legitimate users. This can prevent network administrators from managing their networks.
DDoS SSDP	A DDoS SSDP attack is a type of DoS attack that floods an SSDP server with requests, making it unavailable to legitimate users. This can prevent devices from discovering and connecting to each other on a network.
DDoS UDP	A DDoS UDP attack is a type of DoS attack that floods a target server with UDP packets. UDP is a connectionless protocol, so attackers can easily spoof the source IP address of their packets, making it difficult to identify the source of the attack.
DDoS SYN	A DDoS SYN attack is a type of DoS attack that floods a target server with SYN packets. This causes the server to keep track of all the pending connections, which can eventually exhaust its resources and make it unavailable to legitimate users.
DDoS TFTP	A DDoS TFTP attack is a type of DoS attack that floods a TFTP server with requests. TFTP is a simple file transfer protocol that is often used to transfer firmware updates to devices.
DDoS UDP lag	A DDoS UDP Lag attack is a type of DoS attack that floods a target server with UDP packets with an invalid checksum. This can cause the server to spend time processing the invalid packets, which can slow down or even crash the server.
Benign	Traffic which is not malicious.

The aim of the following plots is to focus on differences between malicious and benign traffic throughout the analysis of the features of the samples. These features provide information about statistics of packets and flow which are meaningful in a network communication scenario.



By analysing of above graphs, it's possible to show the proportion in which the dataset is partitioned highlighting a similar number of samples for all kinds of traffic except for ddos_ntp which is the less represented in the dataset.



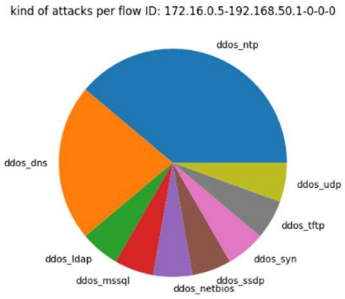
The number of attacks associated with the top 10 flow IDs in a dataset (Flow ID: a unique identifier for each flow of network traffic in the dataset). We've taken into consideration only the malicious traffic applying a mask to the dataset DataFrame.

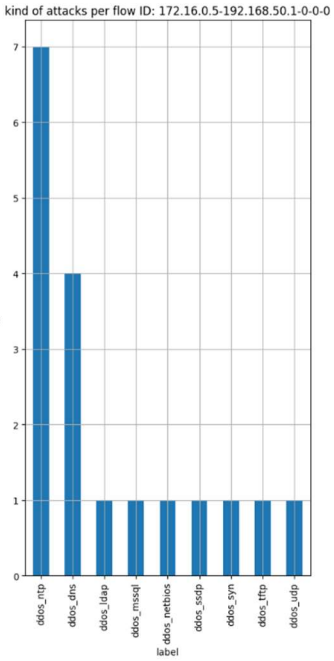
```
mask_bening = operational_df['label'] != 'benign'
```

The graph indicates that the flow ID with the highest number of attacks is 172.16.0.5-192.168.50.1-0-0-0 with 18 attacks which put a stress on the fact that there is a flow ID much more involved in generating malicious traffic. Analysing in depth flow ID's attacks:

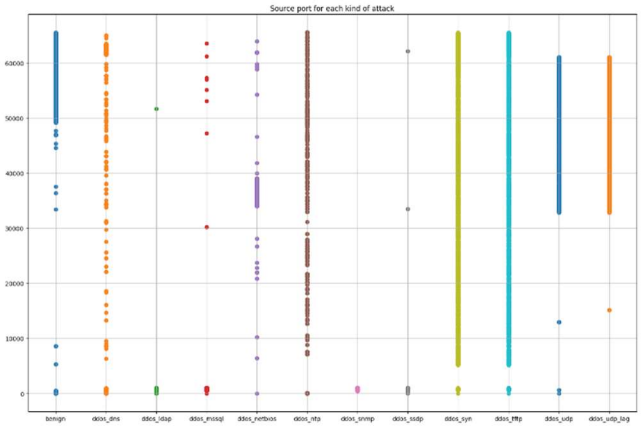
flow ID - 172.16.05-192.168.50.1-0-0-0 performs several kinds of attacks, in particular ddos_ntp and ddos_dns. Other attacks are performed, but only one instance of them is recorded.

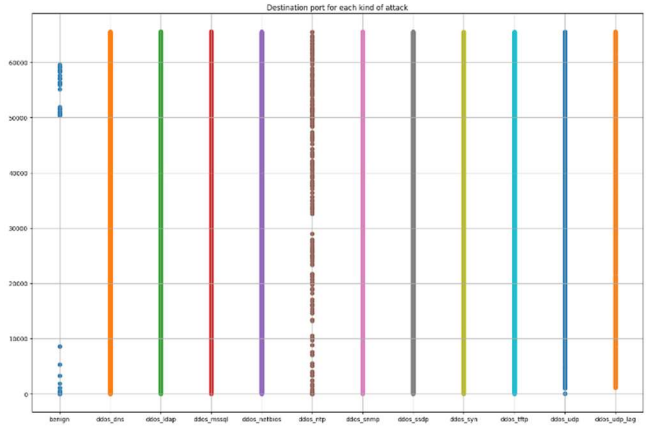
From further analysis, it is the only flow ID that perform various kind of attacks in respect of other flows that performs only one unique attack.

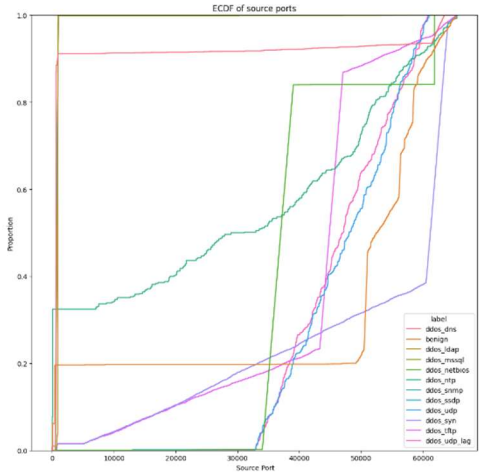


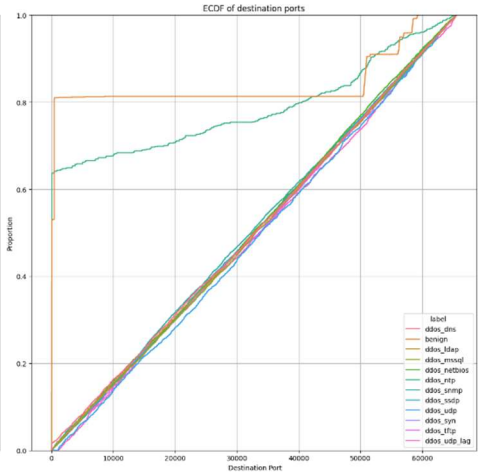


The analysis of source port and destination port of each kind of traffic provided the following results:



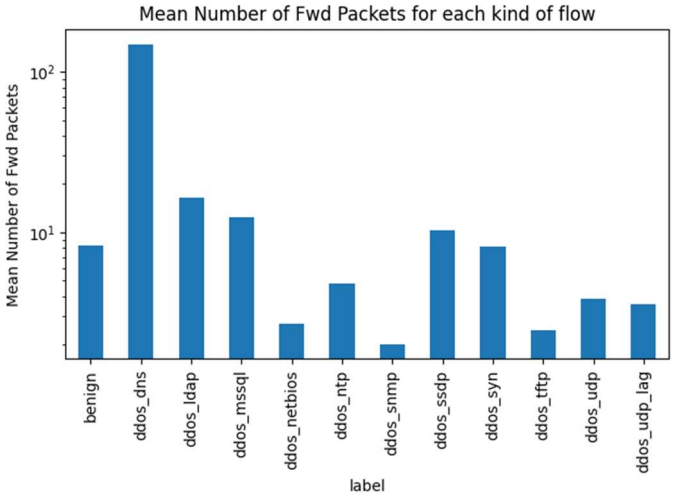


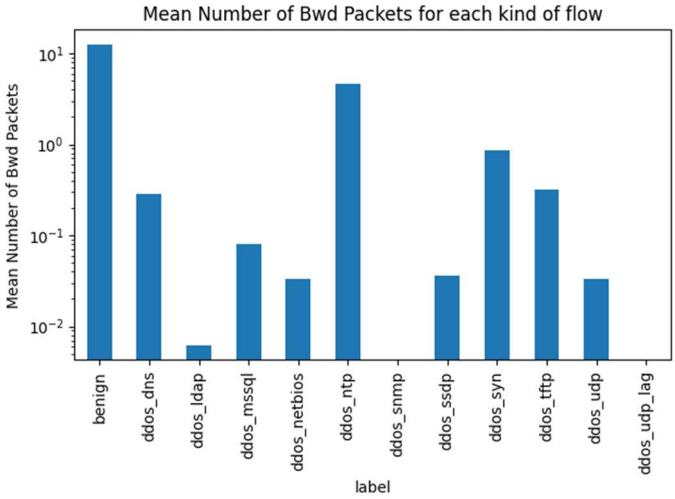




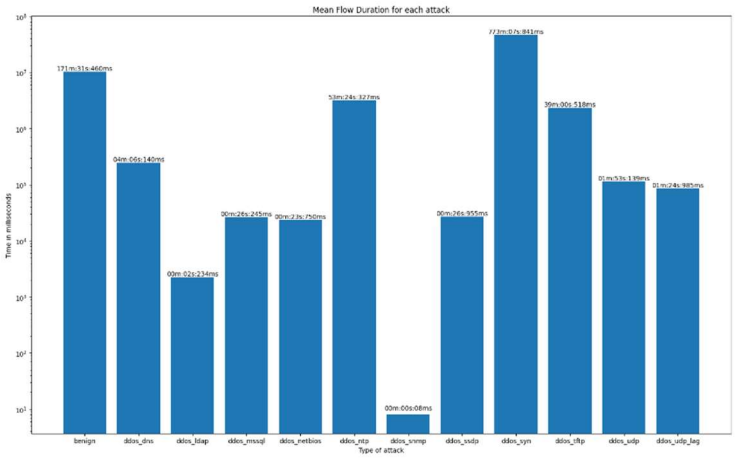
The scatter plot shows that the source ports used vary based on the kind of traffic. On the other hand, the destination ports are specific for benign traffic, while, for the malicious traffic, almost all of them are used.

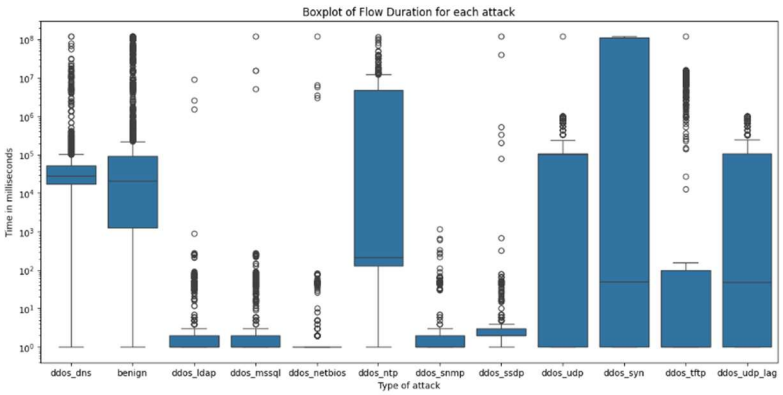
ECDF plot of the source port distribution shows that most of DDoS attack traffic tends to be concentrated on a small number of source ports. This makes it important for network defenders to be aware of these ports and to take steps to mitigate DDoS attacks that target them.



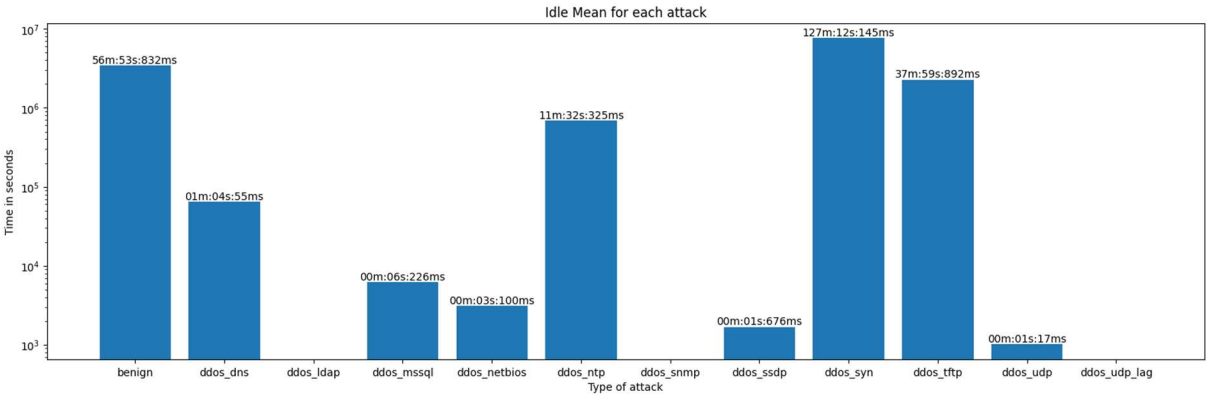


There is a significant difference among the different flows evaluating the mean number of packets in the forward and backward direction of a communication. DDoS_snmp and DDoS_UDP_lag show a mean number in the backward direction that is equal to 0, basically there is no response. Bwd Packets y axis shows, in a logarithmic scale, that some mean has a value less than 1 showing many total counts of Bwd packets equal to 0, because of a lack of response for that specific attack.

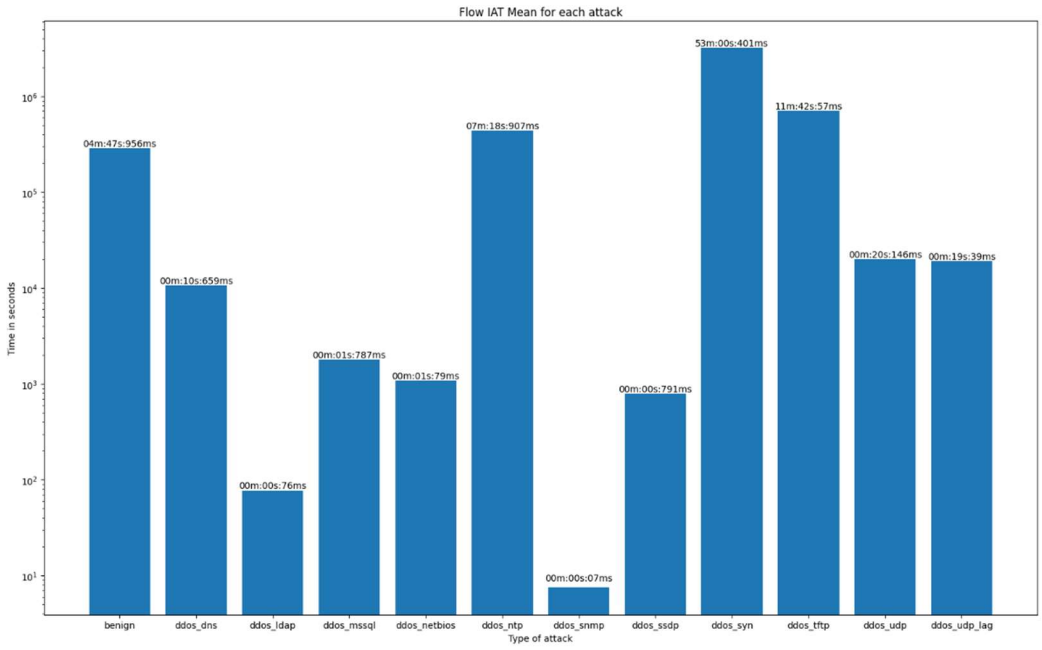




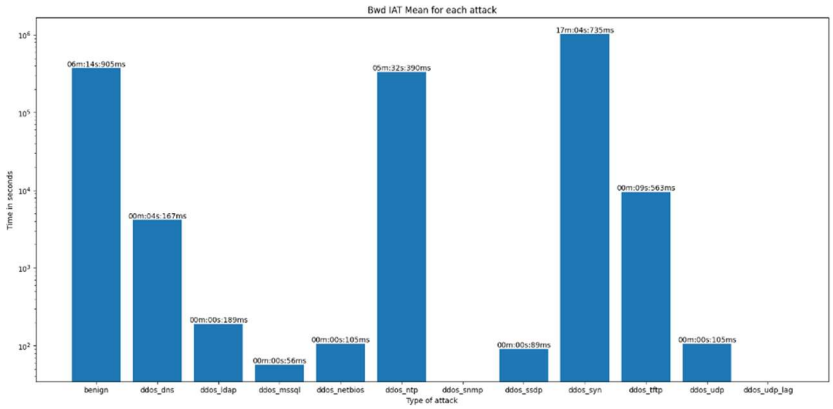
The plots show the mean flow duration for each packet computed displayed in milliseconds on the y axis. We can observe that ddos_syn has the highest mean flow duration of packet because SYN floods involve sending many SYN packets to target servers. It is followed by benign being also relatively long-lasting, typically lasting for a few minutes.
Attack like ddos_ldap , ddos_mssql , ddos_netbois, ddos_snmp , ddos_ssdp tend to be shorter-lived compared to other DDoS attack types. The observed differences in flow duration across attack types suggest that the nature and complexity of the attack techniques influence the duration of network flows.

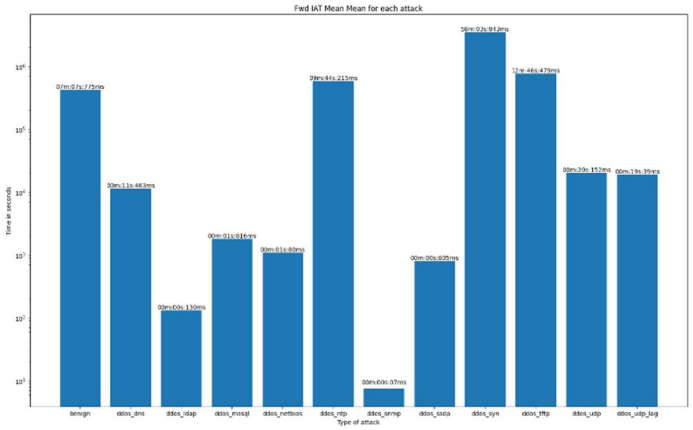


The above plot is a bar graph that shows the idle mean duration for each type of network attack. The idle mean duration is the average amount of time that a network flow belonging to that attack type was idle. Idle time refers to the time when no data was transmitted or received over a connection.
The highest idle mean duration is ddos_syn followed by benign. On the other hand, there are DDoS_UDP_lah, DDoS_SNMP and DDoS_LDAP that are characterized by an Idle Mean of 0.

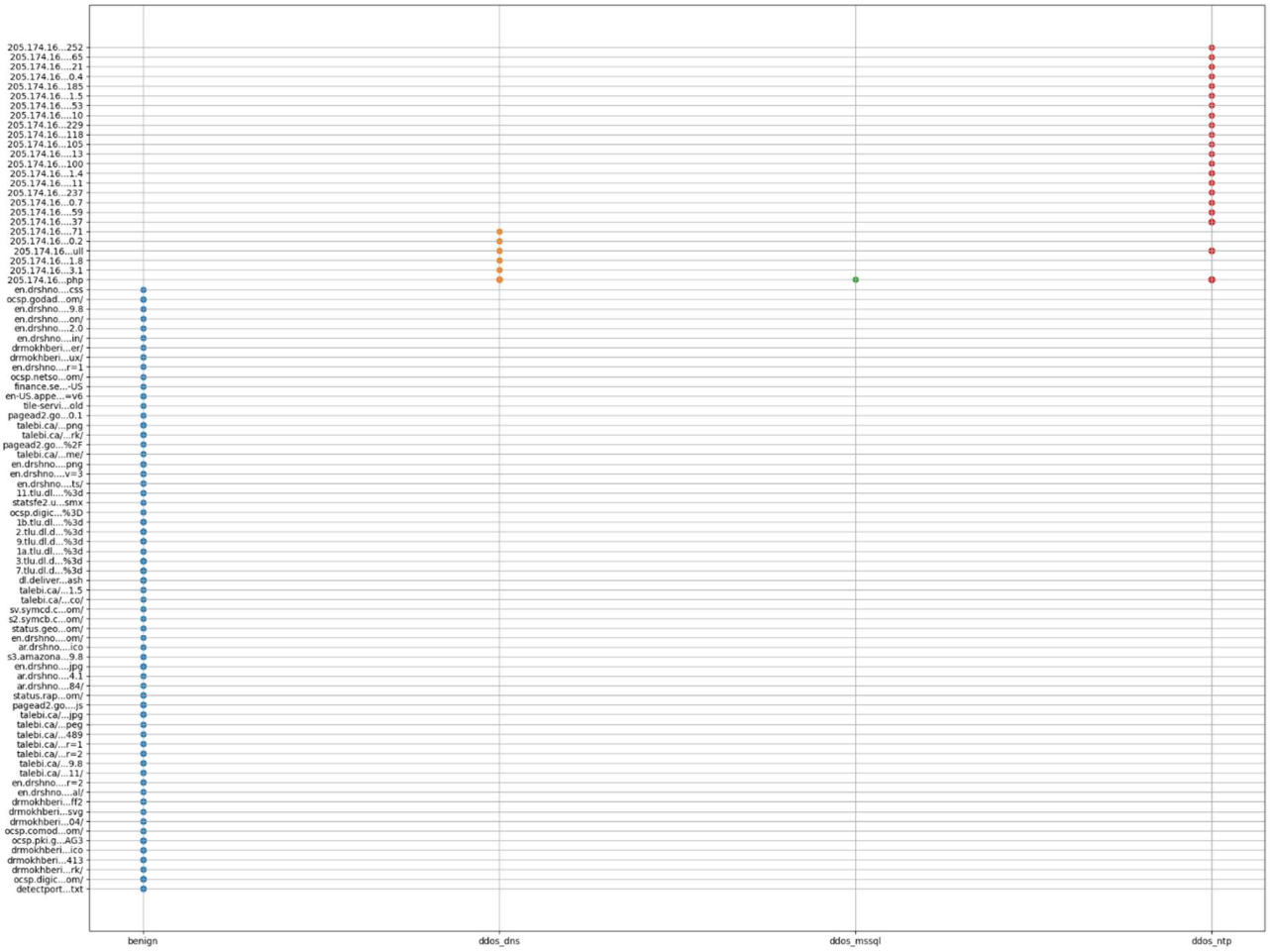


The bar graph that shows the mean inter-arrival time (IAT) for each type of network attack. The IAT is the average amount of time between two packets from the same source IP address. The x-axis of the graph lists the different types of network attacks, and the y-axis shows the mean IAT in milliseconds.
The highest mean IAT is ddos_syn followed by ddos_tftp. The long mean IAT for DDoS tftp attacks is also notable.





The forward and backward inter-arrival time (IAT) are two important metrics used to analyse network traffic patterns. The forward IAT is the average time between consecutive packets traveling in the same direction, while the backward IAT is the average time between consecutive packets traveling in the opposite direction. The results are similar to the one provided by the plot from mean number of packets for each flow for forward and backward communication.



The scatter plot shows, for each traffic that has a value of SimillarHTTP != ‘0’, which kind of protocol is used in the communication. SimillarHTTP feature in the dataset represents A measure of the similarity of the network communication to the HTTP protocol. These plots suggest that the use of HTTP-like protocols can be a useful indicator of whether traffic is benign or malicious due to the strict distinctions of values used by attacks and benign flows.

Pre-processing

Since the ambiguity of the SimillarHTTP values of the dataset (it is not understandable which kind of protocol or what the values represents), but the strong relationship in the partitioning of benign or malicious traffic it has been binary encode. One-hot or dummy encoding has been excluded due to the large number of different values of SimillarHTTP feature that mean that too many additional features were needed for one of the two previously cited encoding approaches. Therefore, the values are represented by a binary value: 0 if a SimillarHTTP protocol is not used, 1 otherwise:

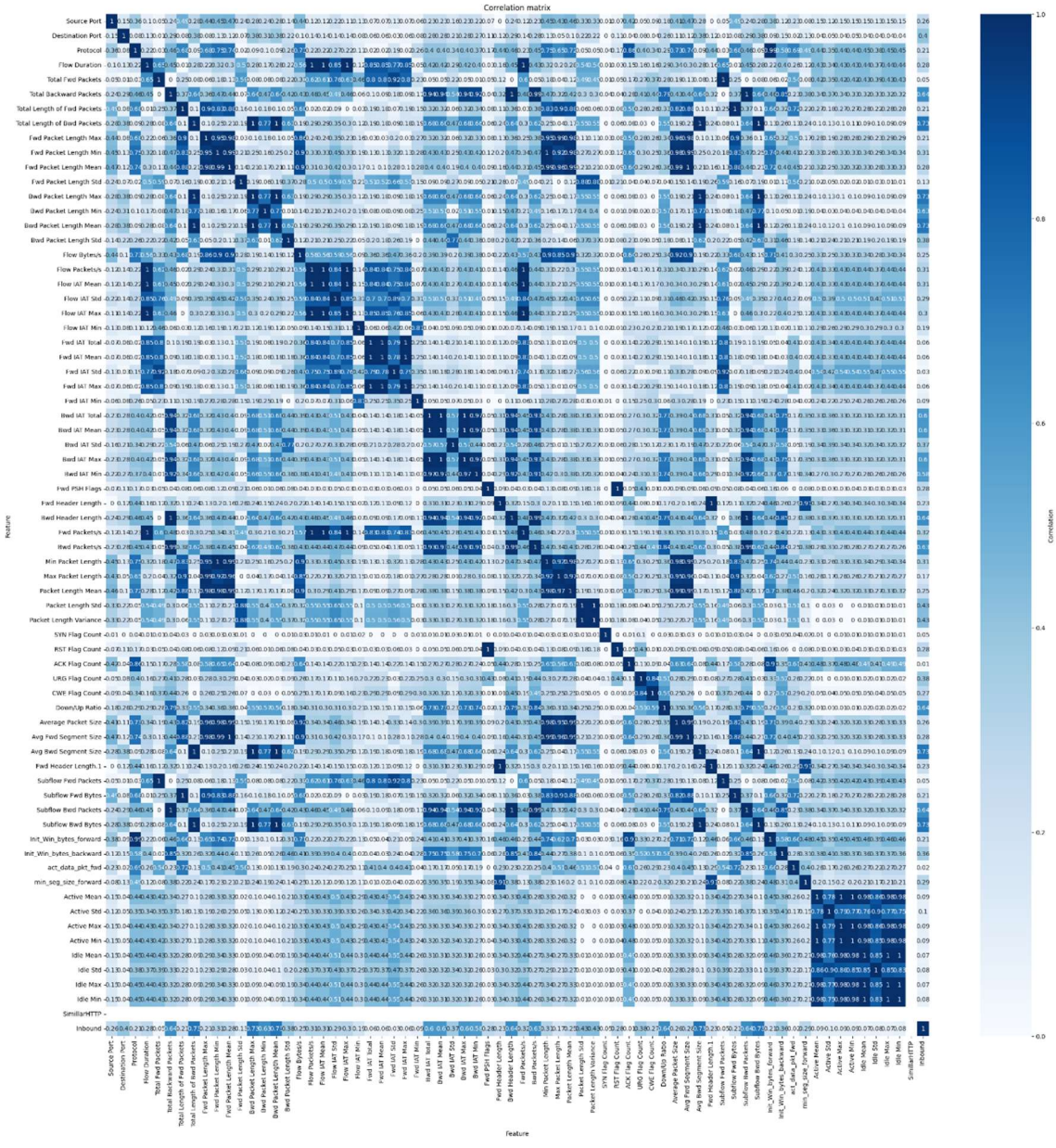
```
mask = df['SimillarHTTP'] == '0'
df.loc[mask,'SimillarHTTP'] = 0
mask = df['SimillarHTTP'] != '0'
df.loc[mask,'SimillarHTTP'] = 1
```

There are other features that assume only values equal to zero which are removed due to their lack of information:

' Bwd PSH Flags', ' Fwd URG Flags', ' Bwd URG Flags', 'FIN Flag Count', ' PSH Flag Count', ' ECE Flag Count', 'Fwd Avg Bytes/Bulk', ' Fwd Avg Packets/Bulk', ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk', ' Bwd Avg Packets/Bulk', 'Bwd Avg Bulk Rate'.

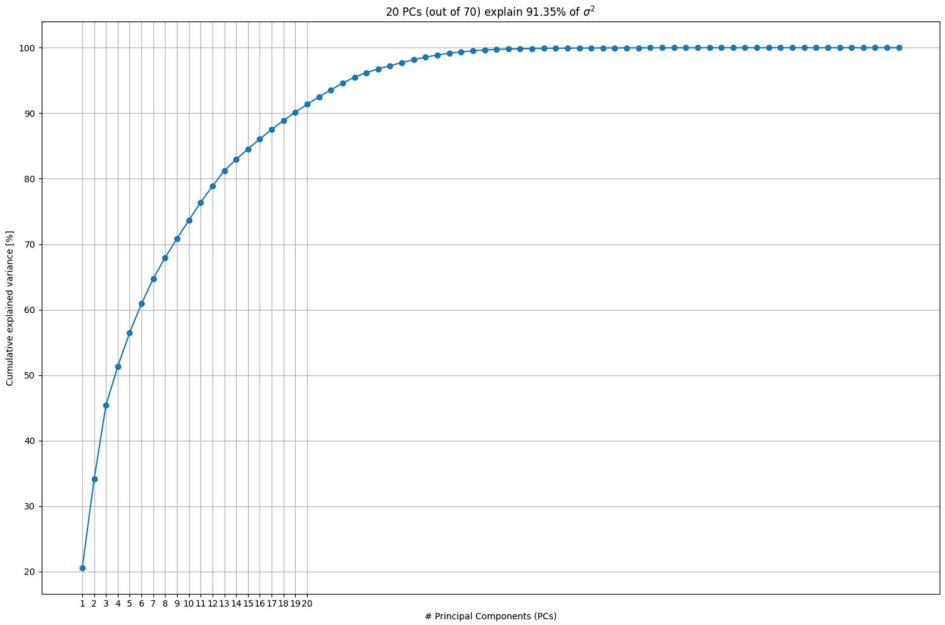
From this point on, even the **Timestamp**, **Source IP**, **Destination IP** and **Flow ID** features have been removed since they are meaningless in a context of a classification task to achieve general understanding of the problem.

The dataset has been standardised using the scikit-learn StandardScaler() class. Using correlation analysis, it is possible to show the correlation among the dataset's features:

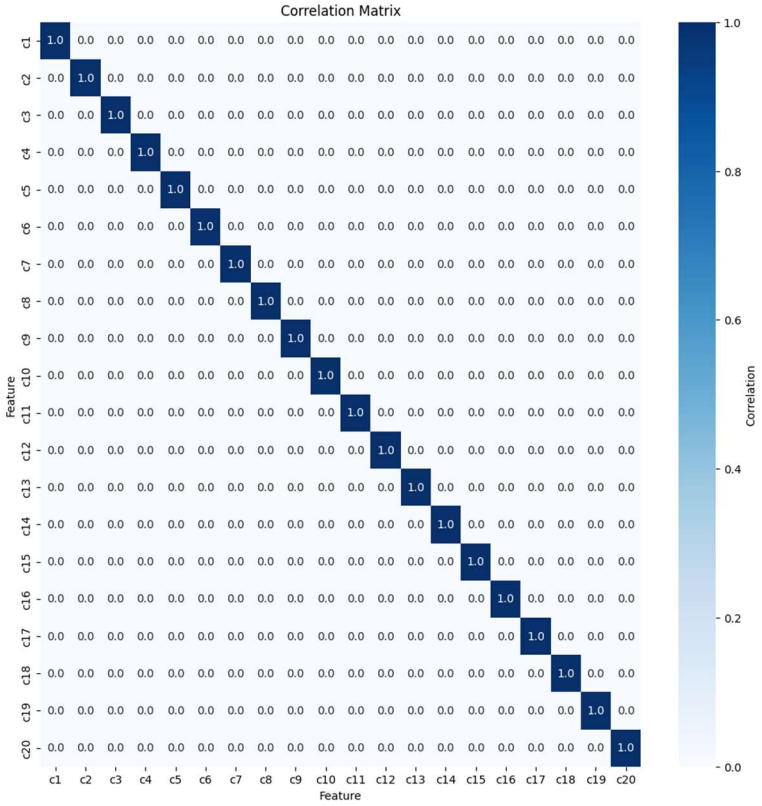


Many features are highly correlated, so to remove redundancy and reducing the feature space retaining the information, PCA has been chosen as pre-processing process to produce a set of uncorrelated features. PCA has been applied to the scaled dataset to reduce its dimensions to better retrieve patterns of high dimensional data and to improve the computation time of the following sections of the project.

From the 73 original features, 20 PCA transformed features have been chosen explaining the 91.35% of the total variance. This number of features has been chosen to retain the most of the variance in the dataset, with a significant decrease of the feature space.



The correlation matrix of the new feature space displays the achieved goal of a set of uncorrelated (Pearson correlation) features:



In summary, the following pre-processing steps have been performed:

1. SimillarHTTP encoding
2. Null features removed
3. Dataset scaling process
4. Apply PCA to the dataset
    a. The choice of number of features is based on the need to retain the most of the variance of the dataset

Section 2- Supervised learning – classification

To solve the multilabel classification problem, four supervised ML classifiers have been evaluated: K-Nearest-Neighbours-Classifier (KNN), Random Forest Classifier (RF), Support Vector Machine (SVC) and Gaussian-Naive-Bayes (GNB). The choice of models is based on the intention of evaluating those which are different in terms of complexity to evaluate their performance on a quite complex dataset such as the one taken into consideration on DDoS attacks. All of them can find non-linear decision boundaries and have different tolerances in terms of noise and outliers.

The models' implementation is the one from the scikit-learn python library. The following are the metric used for model evaluation:

- Accuracy: $\frac{Number\ of\ correctly\ classified\ objects}{Number\ of\ classified\ objects}$
- Precision: $\frac{Number\ of\ objects\ correctly\ assigned\ to\ c}{Number\ of\ objects\ assigned\ to\ c}$
- Recall: $\frac{Number\ of\ objects\ correctly\ assigned\ to\ c}{Number\ of\ objects\ belonging\ to\ c}$
- F1-score: $\frac{2\,rp}{r+p}$ $r = recall, p = precision$

K Neighbors Classifier

K-Nearest Neighbors (KNN) is an instance-based learning algorithm used for classification and regression tasks. In KNN, the training phase involves storing all training examples in memory. When making predictions for new data, the algorithm identifies the k-nearest neighbors from the training set based on a distance metric, typically Euclidean distance. For classification, the algorithm assigns the class label most frequently occurring among the k-nearest neighbors. Key parameters include 'k' (the number of neighbors) and choosing an appropriate value for k is crucial. KNN can be sensitive to noise and outliers.

Random Forest Classifier

Random Forest is a versatile machine learning algorithm widely used for both classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. In the training phase, a set of decision trees is built using random subsets of the data and random subsets of the features. Each tree contributes to the final prediction, and the ensemble nature of Random Forest enhances its robustness and reduces overfitting. Random Forest is known for its high accuracy and ability to handle large datasets with many features. It can capture complex relationships in the data and is less prone to overfitting compared to individual decision trees. Key parameters include the number of trees in the forest and the depth of each tree. Tuning these parameters is crucial to achieving optimal performance. This model is robust to outliers and noise in the data.

Support Vector Classifier (SVC)

Support Vector Machine (SVM) is a powerful and widely used machine learning algorithm for both classification and regression tasks. It works by finding the optimal hyperplane that best separates data points belonging to different classes in a high-dimensional space. In the context of classification, the Support Vector Classifier (SVC) aims to find a hyperplane that maximizes the

margin, which is the distance between the hyperplane and the nearest data points from each class. The data points that lie on the margins or violate the margin are referred to as support vectors. SVC is particularly effective in scenarios where the data is not linearly separable. To handle non-linear relationships, kernel tricks can be applied, transforming the input space into a higher-dimensional space, where a hyperplane can effectively separate the data. Key parameters in SVC include the choice of the kernel (linear, polynomial, radial basis function, etc.) and regularization parameters. These parameters influence the flexibility of the decision boundary and the model's generalization capability. SVC is robust to outliers.

Gaussian Naïve Bayes

Gaussian Naive Bayes is a probabilistic machine learning algorithm used for classification tasks. It is based on Bayes' theorem and the assumption of independence among features, which simplifies the computation of probabilities. In this algorithm, the term "Gaussian" indicates that it assumes the features follow a normal distribution (Gaussian distribution). Despite its simplicity and the assumption of feature independence, Gaussian Naive Bayes often performs surprisingly well in practice. The algorithm calculates the probabilities of a given instance belonging to each class by modelling the distribution of each class using the mean and standard deviation of the features. It then assigns the class with the highest probability as the predicted class for that instance. Gaussian Naive Bayes is particularly useful for datasets with continuous features, and it is less sensitive to irrelevant features. It works well in situations where the independence assumption is reasonable, even if it doesn't strictly hold.

Experimental setting

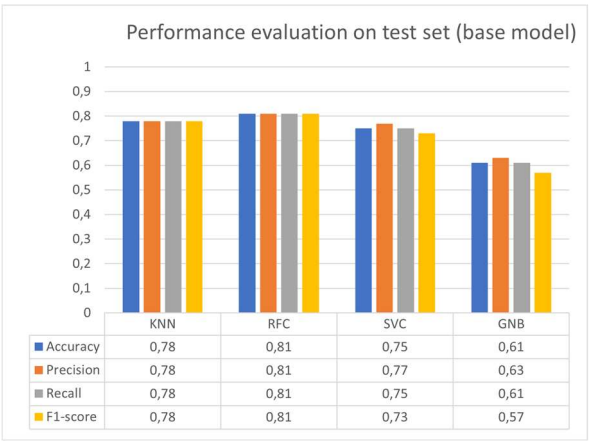
Firstly, the dataset (pca_dataframe.csv saved from the pre-processing section) is split in training and test set in a stratified way in respect of the labels and they are used to train and evaluate the models respectively.

```
X_train, X_test, y_train, y_test = train_test_split(
    pca_df,
    df['label'],
    stratify=df['label'],
    train_size=0.7,
    random_state=15
)
```

Firstly, the models are trained and evaluated with their default hyperparameters; subsequently a hyperparameter tuning process, the evaluation is repeated, and the results are compared to the default models' results.

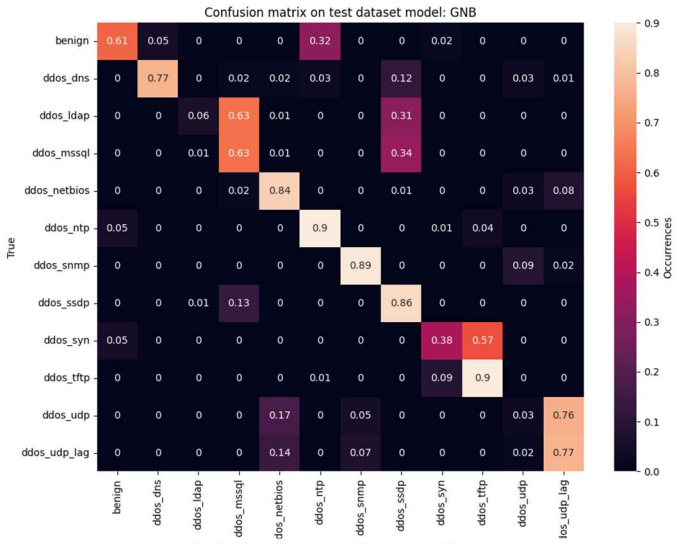
Default hyperparameters results

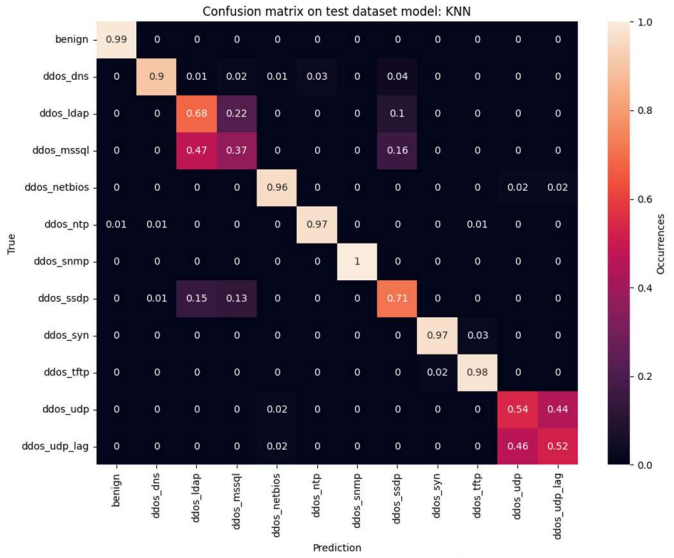
	KNN	RFC	SVC	GNB
Accuracy	0,85	1	0,75	0,61
Precision	0,86	1	0,78	0,63
Recall	0,85	1	0,75	0,61
F1-score	0,85	1	0,73	0,57

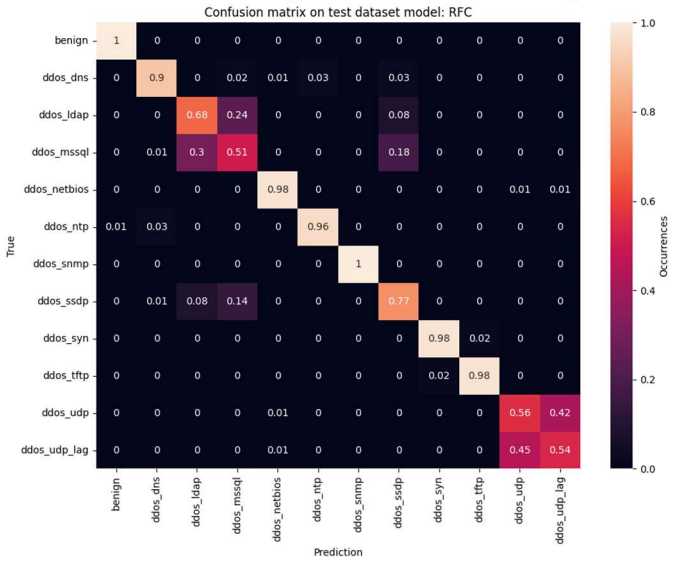


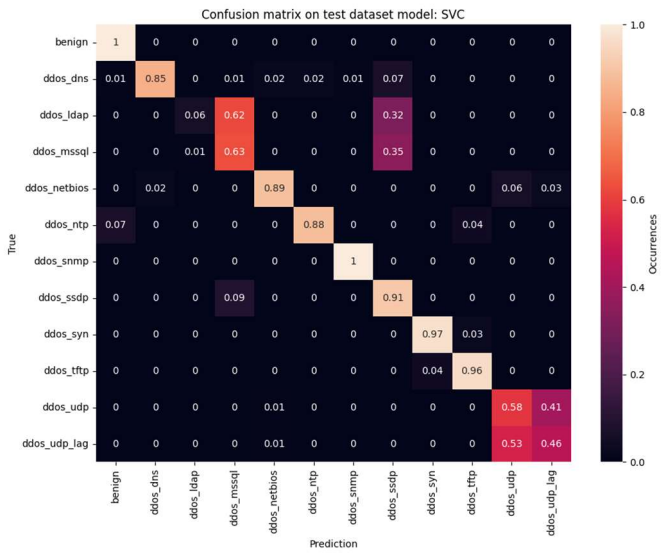
	KNN	RFC	SVC	GNB
Accuracy	0,78	0,81	0,75	0,61
Precision	0,78	0,81	0,77	0,63
Recall	0,78	0,81	0,75	0,61
F1-score	0,78	0,81	0,73	0,57

KNN, RFC and SVC have similar results on the test set with KNN and RFC that reach a score above 80% for all the metrics evaluated. GNB, since it starts from the assumption of Gaussian distribution of data and related probability independence among features, is the worst with scores around 60% for the metrics involved in the evaluation. Considering the confusion matrix on test set, it is possible to visually highlight which class of traffic are misclassified.









KNN and RFC there are few attacks that are classified erroneously (<90% of correct predictions on test dataset):

- ddos_ldap (true) misclassified with: ddos_mssql, ddos_ssdp
- ddos_mssql (true) misclassified with: ddos_ldap, ddos_ssdp
- ddos_ssdp (true) misclassified with: ddos_mssql, ddos_ldap, ddos_dns
- ddos_udp (true): misclassified with: ddos_udp_lag (>40% erroneous classifications), ddos_netbios
- ddos_udp_lag (true): misclassified with: ddos_udp_lag (>40% erroneous classifications), ddos_netbios

ddos_udp and ddos_udp_lag are the ones that are misclassified the most among each other. It is understandable from the nature of this kind of flows which are strictly related. The first one is an actual DDoS attack that exploits the vulnerabilities of UDP protocol sending broadcast UDP echo request using a reflector, while the other (UDP DDoS lag) a type of DoS attack that floods a target server with UDP packets with an invalid checksum. This can cause the server to spend time processing the invalid packets, which can slow down or even crash the server.

SVC, unlike RFC and KNN, can correctly classify more than 90% of DDoS SDDP. However, it has a much worse performance in DDoS LDAP classification with a correct prediction of only 0.06% of the samples.

GNB has a similar behaviour of the previously described model, in except of correct prediction rate of the benign traffic of only 64% of the samples in the test dataset. That flow is confused with DDoS NTP and it's the only model to misclassify benign traffic in a sensible way.

Hyperparameters Tuning

After the first model evaluation, we proceeded with hyperparameters tuning for each model to try to increase their performances on our DDoS dataset. We choose to use a Grid Search algorithm that takes care of performing cross validation trying to reach more reliable performance estimates, reduce overfitting, and contributing to a better understanding of a model's generalization capabilities. sklearn.GridSearchCV()is a part of the model selection module and is designed for hyperparameter tuning with dataset cross-validation. It performs an exhaustive search over a specified parameter grid, training and evaluating a model for each combination of hyperparameters to find the best set of hyperparameters that maximizes a specified scoring metric.

```
gsCV = GridSearchCV(estimator=model,
                    param_grid=parameters,
                    scoring='accuracy',
                    cv=5,
                    return_train_score=True)

gsCV.fit(X_train, y_train)
```

The approach used for the hyperparameter tuning is explained by the function's input parameters:

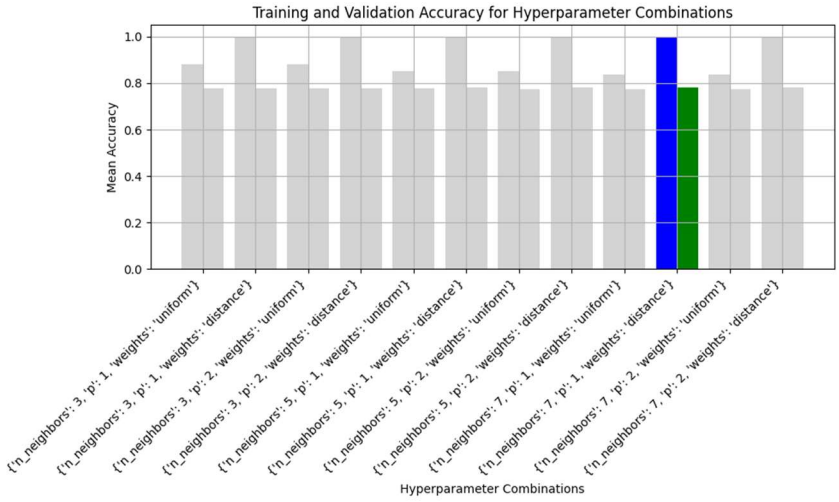
- estimator: model to be tuned
- param_grid: Dictionary with parameters names (str) as keys and lists of parameter settings to try as values
- scoring: Strategy to evaluate the performance of the cross-validated model on the test set.
- cv: determines the cross-validation splitting strategy

The following are the hyperparameters and their values tuned in the process which involved sklearn.GridSearchCV():

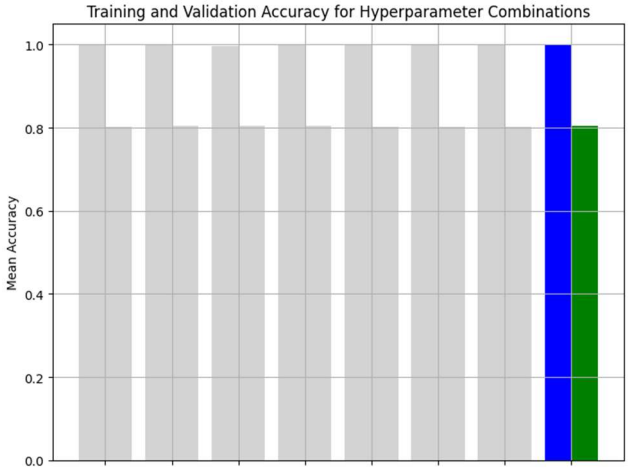
- KNN:
  - “n_neighbours”: [3, 5, 7]
  - “weights”: [“uniform”, “distance”]
  - “p”: [1, 2]
- RFC:
  - “criterion”: [gini, entropy]
  - “n_estimators”: [50, 100]
  - “max_depth”: [None, 10]
  - “min_samples_split”: [2, 3]
- SVC:
  - “C”: [0.1, 1, 10]
  - “kernel”: [rbf, poly]
- GNB:
  - “var_smoothing” = $e^{-11}, e^{-10}, e^{-9}, e^{-8}, e^{-7}$

For each model, as a result, the best set of hyperparameters are:

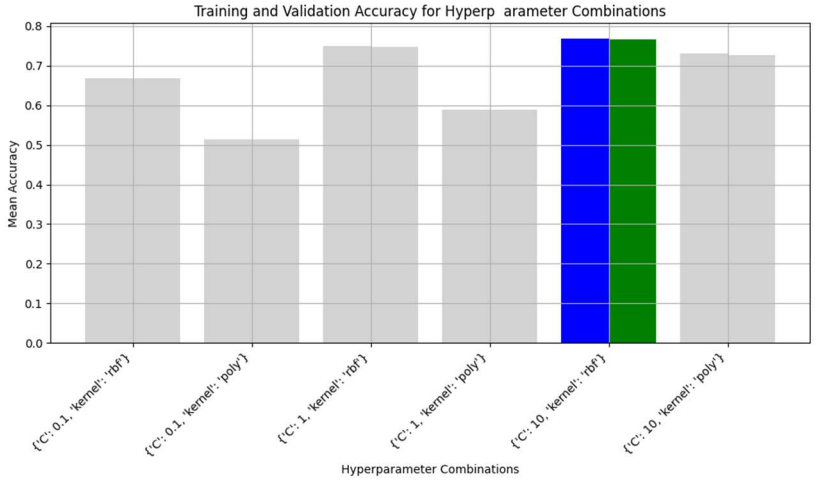
- KNN:
  - “n_neighbours” = 7
  - “p” = 1
  - “weights” = “distance”
  - Validation score: 0.783



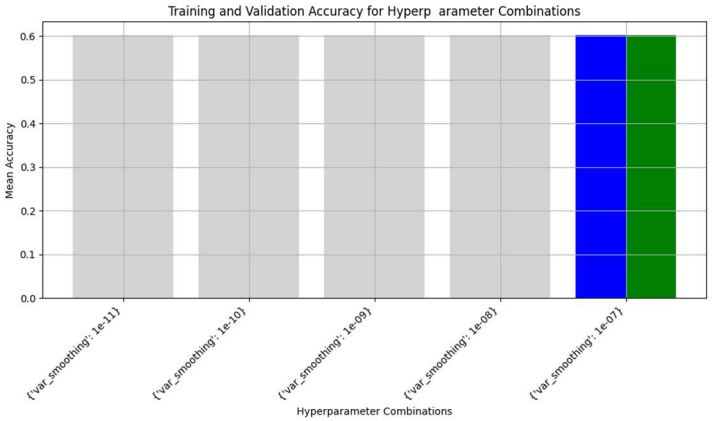
- RFC:
  - “criterion” = “entropy”
  - “max_depth” = None
  - “min_samples_split” = 3
  - “n_estimators” = 100
  - Validation score: 0.804



- SVC:
  - “C” = 10
  - “kernel” = “rbf”
  - Validation score: 0.766

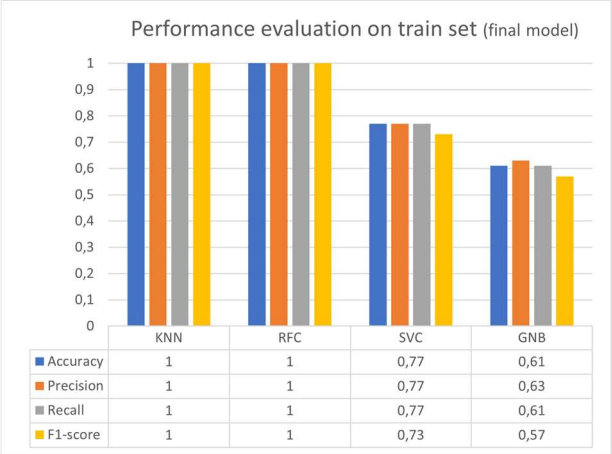


- GNB:
  - “var_smoothing” = e^{-7}
  - Validation score: 0.602

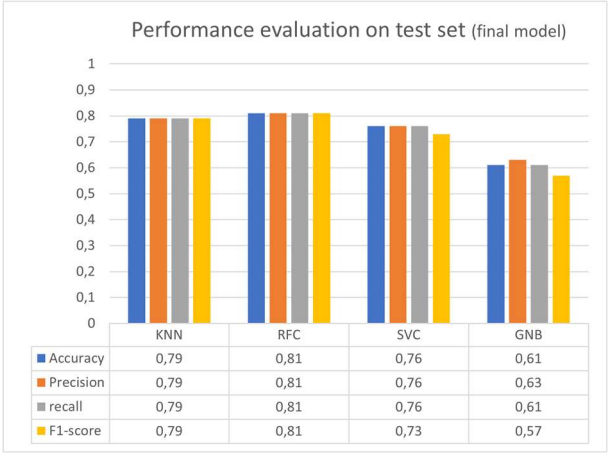


Tuned Hyperparameters results

The models initialized with their best hyperparameters are evaluated on the dataset producing the following results:

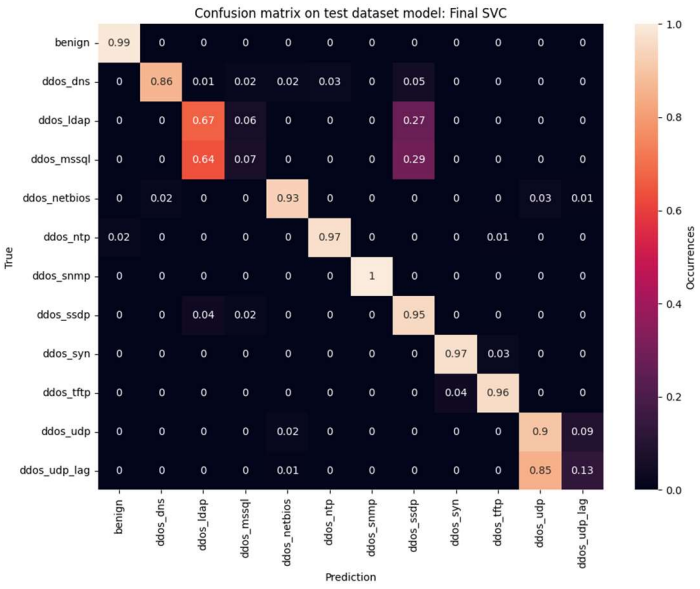


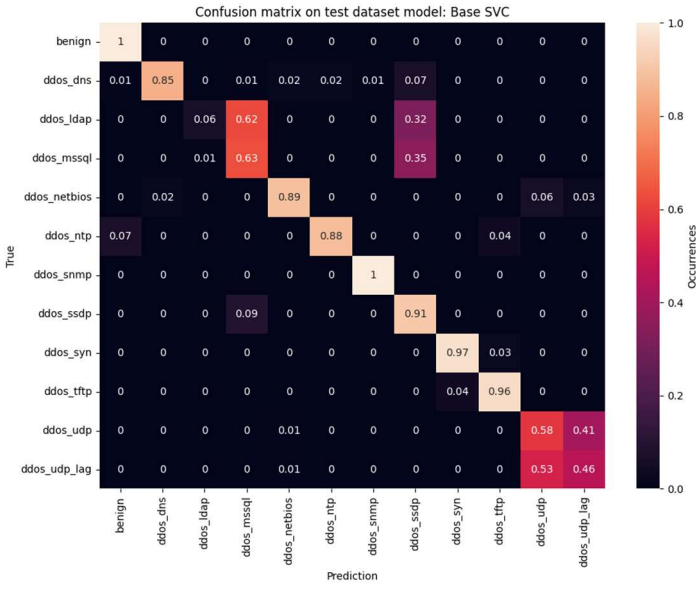
	KNN	RFC	SVC	GNB
Accuracy	1	1	0,77	0,61
Precision	1	1	0,77	0,63
Recall	1	1	0,77	0,61
F1-score	1	1	0,73	0,57



	KNN	RFC	SVC	GNB
Accuracy	0,79	0,81	0,76	0,61
Precision	0,79	0,81	0,76	0,63
recall	0,79	0,81	0,76	0,61
F1-score	0,79	0,81	0,73	0,57

There is little or no significant improvement for all the three models in the test set predictions, in respect of the base model. It was predictable from the plots during the hyperparameter tuning, since the validation score changed slightly with different hyperparameter initialization. This kind of behaviour can be caused from the complex dataset nature, so models are not able to achieve a higher level of performance on the evaluated metrics. Consequently, even the tuned models' confusion matrix remained quite the same, with the same class misclassification. While RFC, KNN and GNB had minor changes in correct predictions percentage, SVC model had an interesting change in the confusion matrix:





The final model (on the left) had meaningful rise in DDoS UDP correct classification rate on the detriment of DDoS UDP lag correct predictions that fall to 0.13%. So, even though the overall average score is unchanged, it is not the same for the kind of flows misclassified.

All the models, except from the GNB, showed a perfect or almost perfect benign traffic detection, and a good percentage of malicious attack classification. However, taken into consideration performance on

benign traffic, all or almost all the malicious traffic, even though not always correctly classified, is detected. Moreover, the majority amount of erroneous detection is among very similar kind of attacks (e.g. DDoS UDP and DDoS UDP LAG).

Section 3- Unsupervised learning – clustering

Clustering analysis is performed by applying to the dataset the following algorithms characterised by different approaches: K-Means (hard-clustering), DBSCAN (Density Based clustering) and Gaussian Mixture Model (soft-clustering). As an unsupervised task, these models are chosen to see how different clustering approaches (soft-clustering, hard-clustering) behave with big complex datasets.

KMeans

KMeans is an iterative partitioning algorithm used for cluster analysis in machine learning and data mining. Operating on a dataset with 'n' observations, the algorithm aims to group these observations into 'k' distinct clusters based on their feature similarities. The process begins by randomly initializing 'k' cluster centroids, typically using the data points themselves. Subsequently, each observation is assigned to the cluster whose centroid is closest, based on a chosen distance metric, commonly Euclidean distance. In the iterative update step, the centroids of the clusters are recalculated as the mean of all the points assigned to that cluster. This process repeats until convergence, where the assignment of data points to clusters remains stable across iterations or reaches a predefined convergence criterion. KMeans minimizes the within-cluster sum of squared distances, essentially optimizing the compactness of clusters. The algorithm's objective function, known as the inertia or within-cluster sum of squares, quantifies the quality of the clustering. One crucial consideration in employing KMeans is the need to predefine the number of clusters, 'k,' which can significantly impact the results.

Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a probabilistic model used for clustering. It assumes that the data is generated by a mixture of several Gaussian distributions with unknown parameters. Unlike KMeans, which assigns data points to hard clusters, GMM assigns each data point a probability of belonging to each cluster. The model represents the probability density function as a weighted sum of Gaussian distributions, where each Gaussian distribution corresponds to a cluster. The weights indicate the likelihood of a data point belonging to a particular cluster, and the Gaussian distributions capture the shape and spread of the data within each cluster. The key parameters of a GMM include the mean, covariance matrix, and weight for each Gaussian component. The Expectation-Maximization (EM) algorithm is commonly used to iteratively estimate these parameters. The E-step calculates the probability that each data point belongs to each cluster based on the current parameter estimates, while the M-step updates the parameters to maximize the likelihood of the data given the current cluster assignments. GMMs are flexible and capable of modelling complex data distributions, making them suitable for applications where clusters may have different shapes and sizes.

DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a clustering algorithm designed for discovering clusters with varying shapes and densities within a dataset. Unlike traditional methods that require the user to predefine the number of clusters, DBSCAN identifies clusters based on the density of data points in the feature space. The core idea behind DBSCAN is to classify each data point as a core point, border point, or noise point. Core points are those with a minimum number of neighbouring points within a specified radius, indicating regions of high density. Border points, while not meeting the density criteria themselves, are reachable from core points and contribute to the cluster. Noise points do not satisfy the

density conditions and are typically considered outliers. The algorithm proceeds by iteratively exploring the neighbourhoods of each core point, expanding the cluster by connecting core points and incorporating border points. This process continues until all reachable points are assigned to a cluster. Unvisited points that do not meet the density criteria remain labelled as noise. DBSCAN's strength lies in its ability to identify clusters of arbitrary shapes and handle outliers effectively. It is particularly useful when dealing with datasets where clusters exhibit varying densities. Additionally, DBSCAN inherently handles the challenge of determining the number of clusters, a common limitation in other clustering algorithms.

Performance evaluations

The number of clusters, for KMeans and GMM, is retrieved by evaluating the silhouette score variation based on the related parameter of the python implementation. On the other hand, DBSCAN compute the number of clusters implicitly based on the value of other hyperparameters (epsilon and MinPts) that has been tuned based on silhouette score value.

The number of clusters parameter, for KMeans and GMM, varies within a range from $0.5 \times number\ of\ labels$ to $1.5 \times number\ of\ labels$.

Note: the knowledge of the effective number of labels from the Ground Truth is used to estimate a range of the number of cluster parameters.

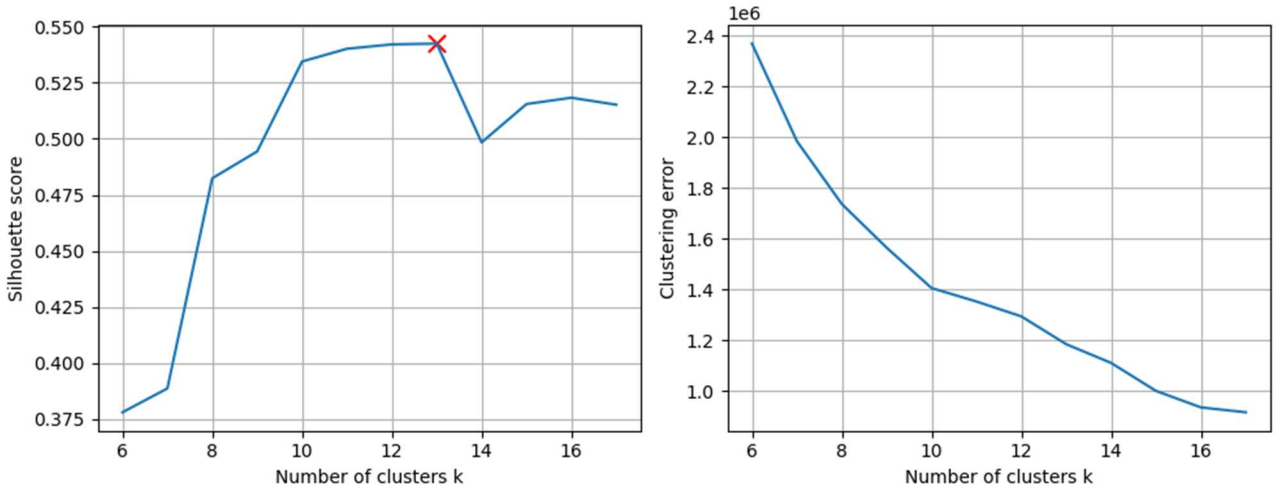
After the tuning phase, the clustering assignments will be compared taking in consideration the following metric:

- Silhouette Score
    - it measures consistency within clusters of data, in other words, how similar a data point is to its own cluster (cohesion) compared to other clusters (separation)
    - It ranges from -1 to +1 where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters

Hyperparameter Tuning

GMM and KMeans needs the number of clusters to retrieve, so we could perform different iterations of the algorithms and choose the best number based on the highest silhouette score leaving other parameters with default values (except for random state that has been set to produce a reproducible output across multiple function calls). Then other parameters will be tuned with iterating the process varying them within the appropriate range for each parameter.

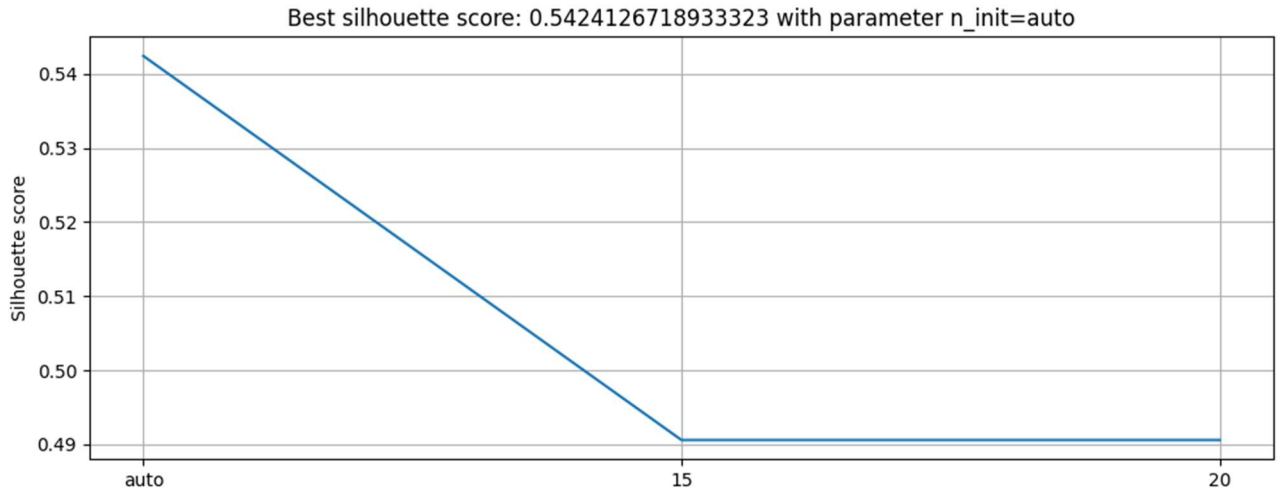
KMeans



As a result, the best number of clusters is 13 with a corresponding silhouette score value of 0.54. Accordingly with the increase of number of clusters, the clustering error decreased its value. After the choice of the parameter n_clusters, it is possible to proceed with other hyperparameter tuning, n_init; it represents the number of times the k-means algorithm is run with different centroid seeds.

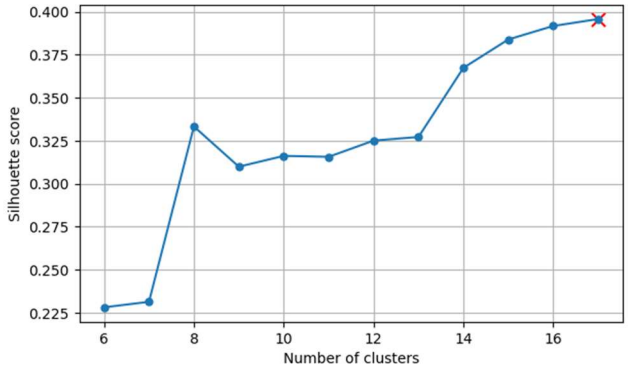
- n_init: [‘auto’, 15, 20]

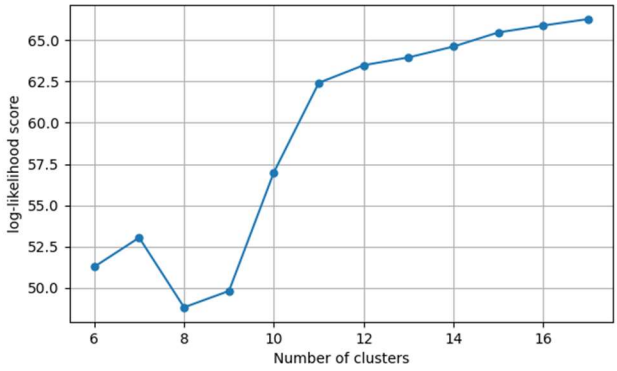
Note: ‘auto’ is the default parameter that is equal to 10



The best value chosen for “n_init” is “auto” since the silhouette score decreases with other values. Therefore, the default KMeans model is the best according to the results.

GMM



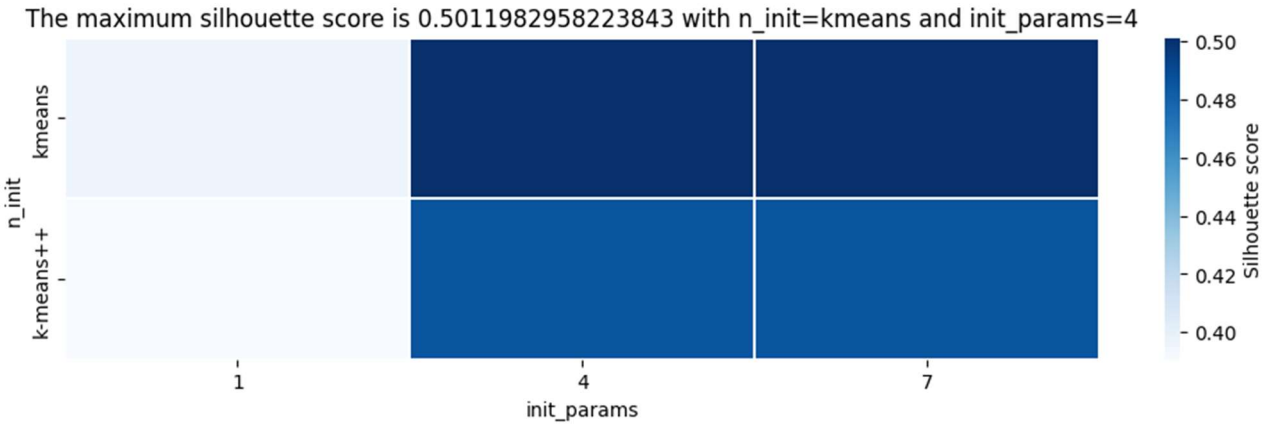


Log-likelihood metric, which is a GMM specific one, measures how well the model explains the observed data. The goal is to maximize the log-likelihood, meaning finding the parameters (cluster means, covariances, and weights) that make the observed data most probable under the model. The more the number of clusters increase, the more the silhouette and log-likelihood rise.

The best number of clusters is 17 with a silhouette score of 0,3957.

Once the n_components is found, it’s time to tune other hyperparameters. n_init and init_params will pass through the tuning process. The former is the number of initializations to perform, while the latter represents the method used to initialize the weights, the means and the precision. They will vary within the following values:

- n_init: [1, 4, 7] (default=1)
- init_params: [‘k-means’, ‘k-means++’ (default=‘kmeans’)

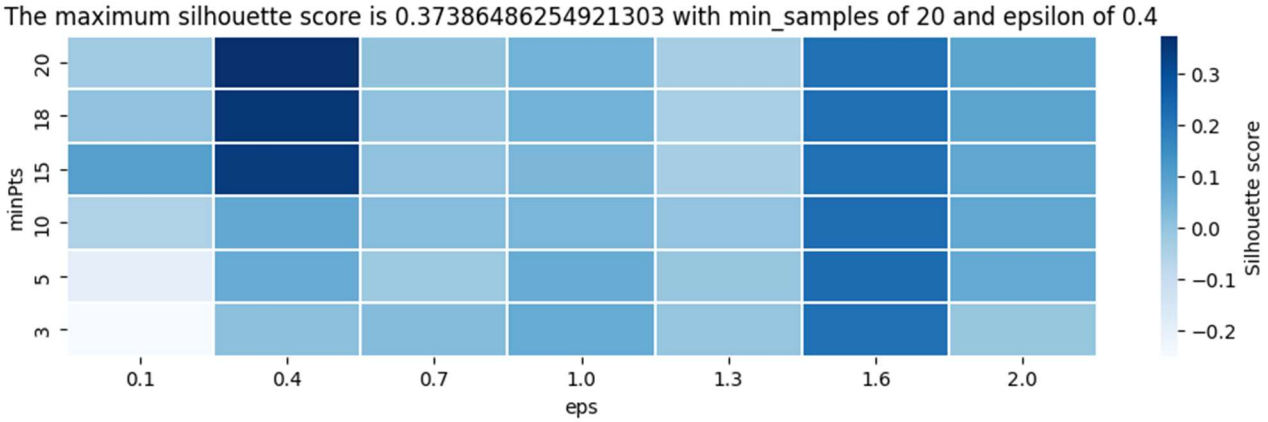


The final silhouette score for the best hyperparameters, n_init=kmeans and init_params=4, is 0.5011 reaching a similar score of KMeans.

DBSCAN

The parameters to validate for this algorithm are min_samples and eps which represent the minimum number of points to be a core point and the maximum distance to be connected respectively. The values vary in the following ranges:

- min_samples: [3, 5, 10, 15, 18, 20]
- eps: [0.1, 0.4, 0.7, 1, 1.3, 1.6, 2]

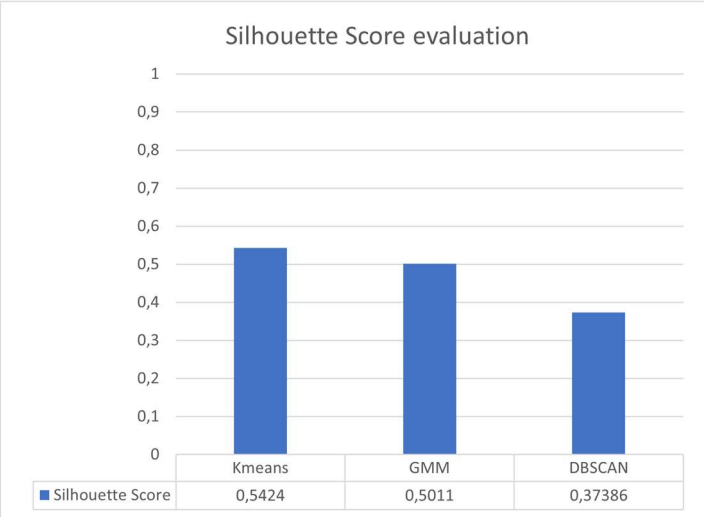


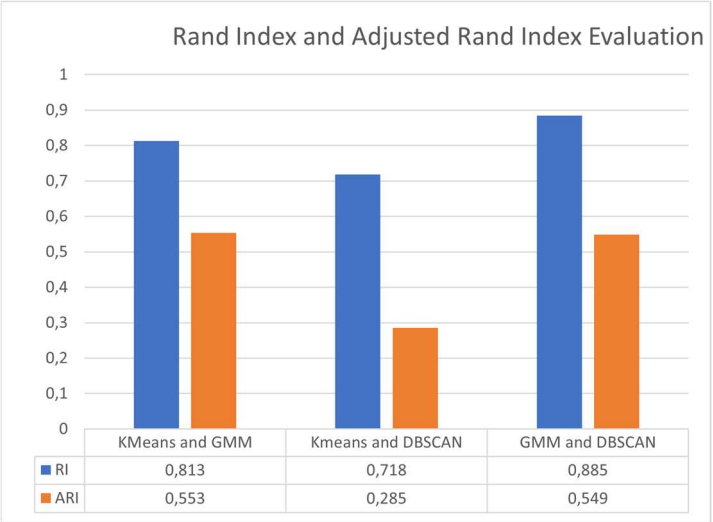
The best hyperparameters are min_samples=20 and epsilon=0.4 with a silhouette score of 0.373. DBSCAN is shown to be the worst clustering algorithm according to silhouette score evaluation among all the evaluated.

Clustering results

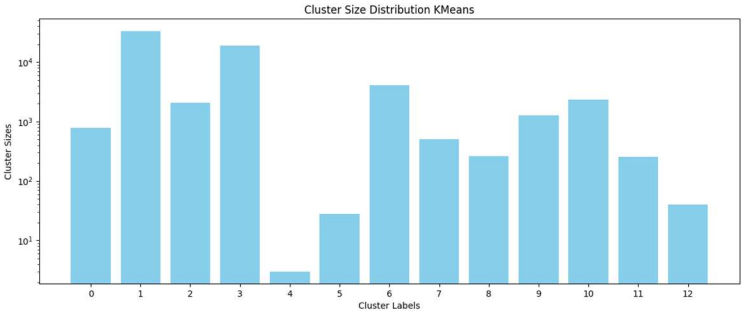
The evaluation of the clustering algorithms is based on the silhouette score, since it is an unsupervised task, and we don't have access to the GT. Therefore, in this context the silhouette score values are considered along with Rand Index and Adjusted Rand Index metrics are used to assess similarities among clusters. In summary, we use these metrics to assess how different algorithms produce similar clustering assignments.

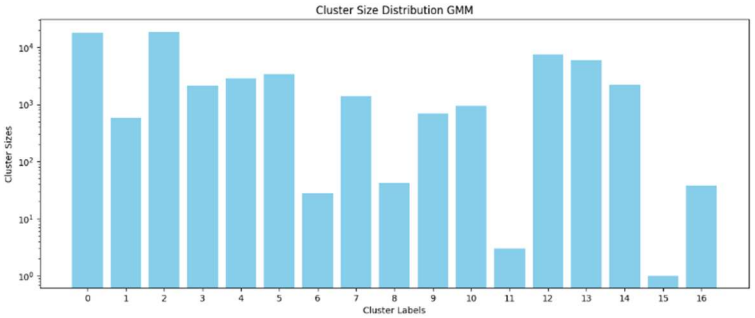
- Rand Index: measures the similarities of two assignments (clustering1 labels, clustering2 labels)
- Adjusted Rand Index: It is an adjustment of the Rand Index because it does not ensure to obtain a value close to 0.0 for a random labelling

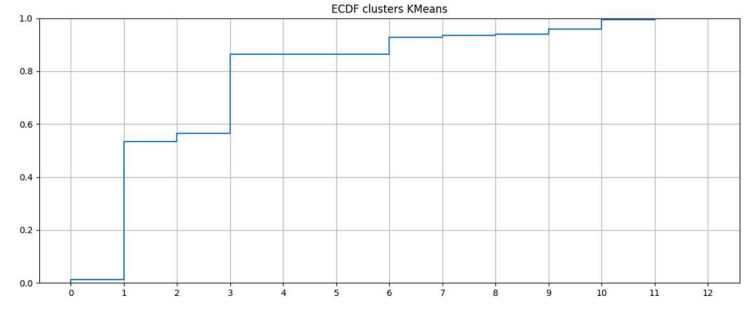


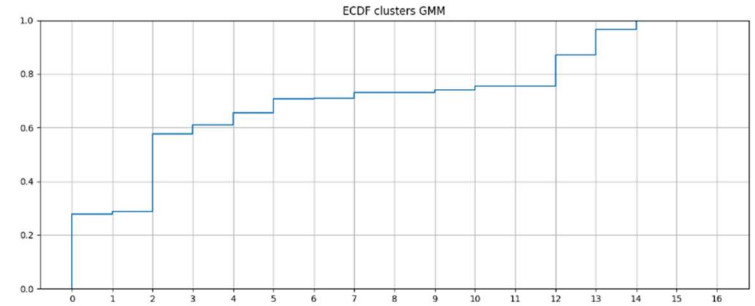


According to the results, KMeans and DBSCAN are the ones that differ the most among them. On the other hand, GMM shares assignment similarities between the other two algorithms acting like a sort of trade off between the other two algorithms.

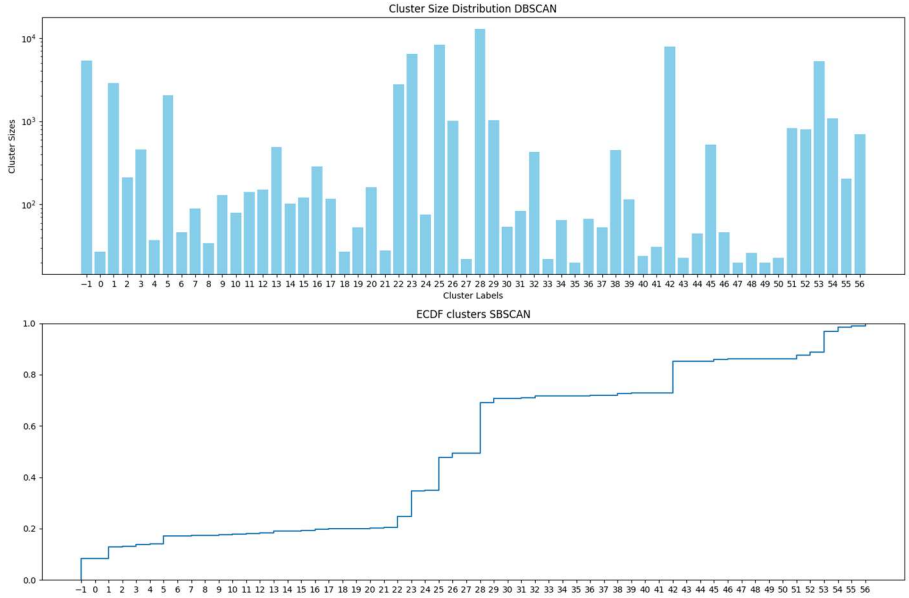








Comparing KMeans and GMM's clustering size distribution, it is shown how in both the cluster size varies among different order of magnitude. Even the ecdf has some shape similarities such as the presence of several small clusters.



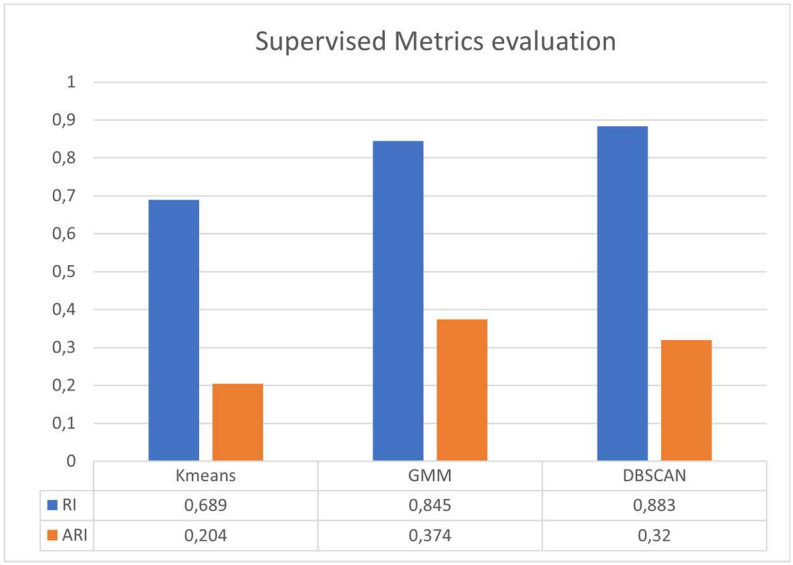
DBSCAN, since its different method to retrieve number of clusters, has assigned much more clusters in respect of KMeans and GMM. Trying to find any similarities to the previous algorithms' distributions, the same pattern of very different clusters' size assignments is repeated. Particularly, there are several clusters composed by a small number of samples in respect of other one that are composed by thousands of them.

Note: cluster -1 represents the sample that are considered as noise

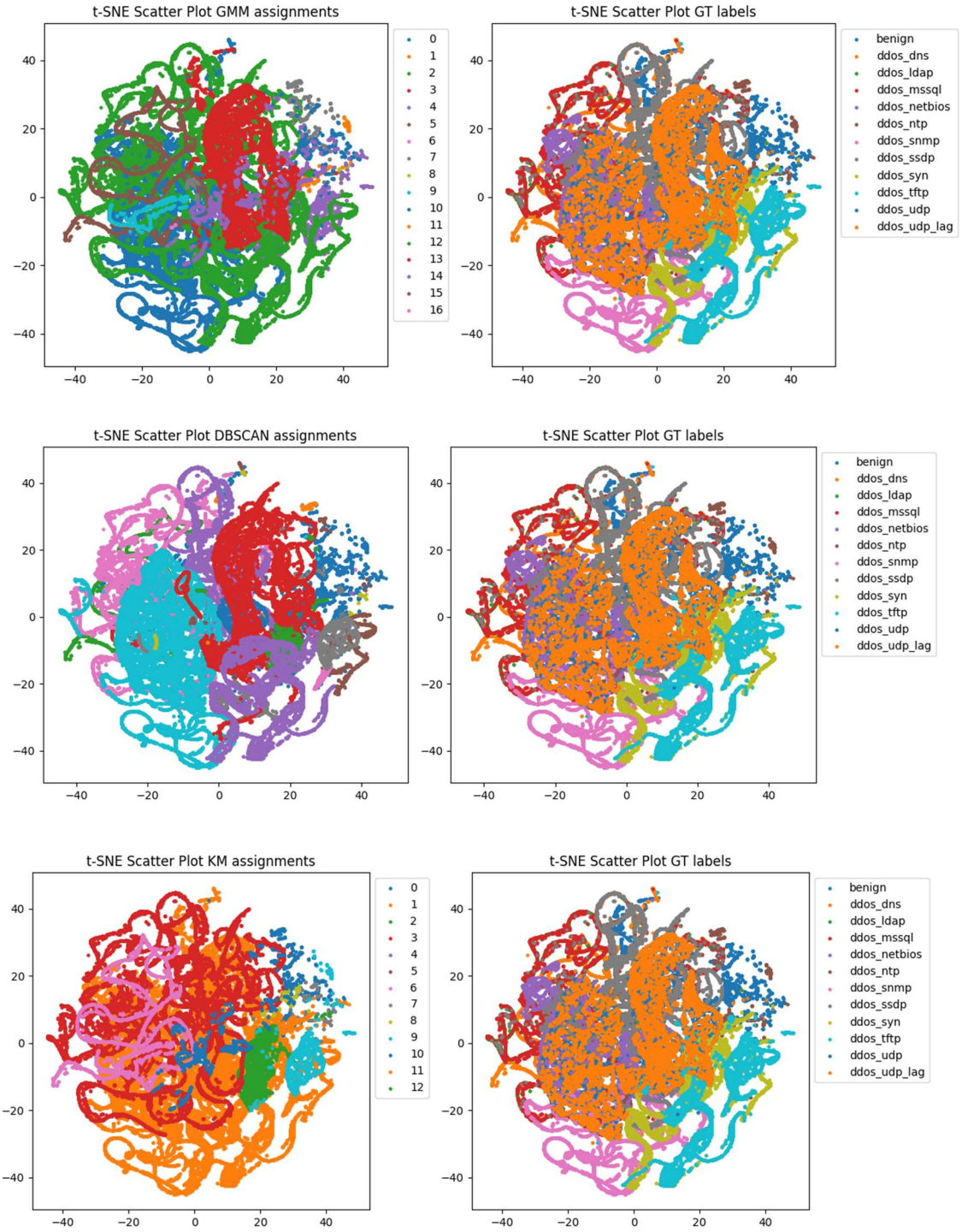
Section 4- Clusters explainability and analysis

In this section, we will delve into the significance of ground truth comparison in evaluating clustering algorithms. Through a comprehensive analysis of ground truth comparison, we seek to contribute to a deeper understanding of relationships clustering assignments and GT. In addition, we tried to explain which feature contributed the most into the decision-making process of clustering assignments. To evaluate the clustering performance in respect of GT two metrics have been evaluated Rand Index and Adjusted Rand Index already described in the previous section.

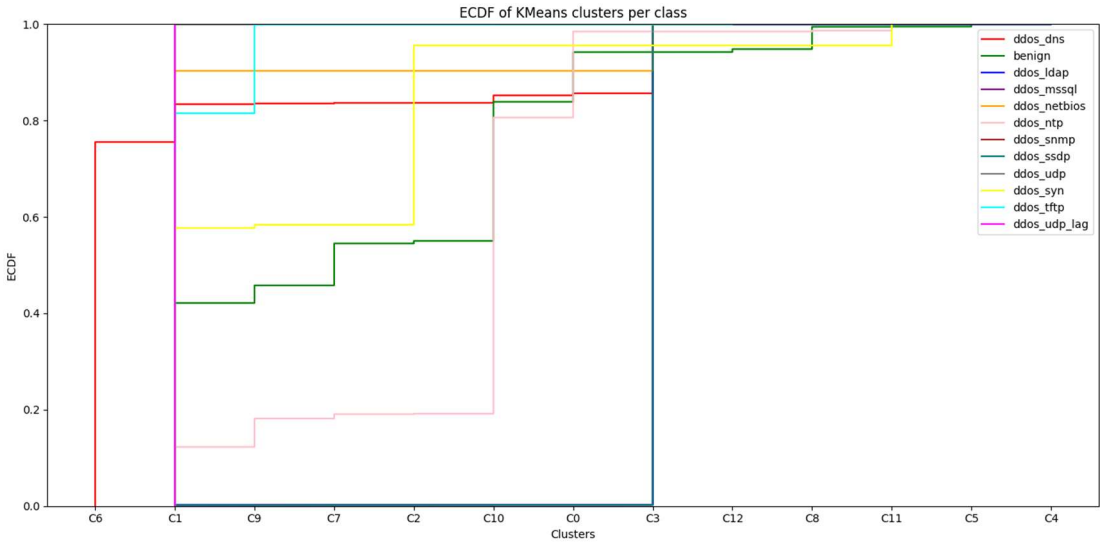
From the results, GMM algorithm is the one that performs better, since has higher adjusted rand index value while the worst one is KMeans with a lower score for each metric. Introducing Ground Truth knowledge, the algorithm performance evaluation changed completely the view on the models' performance since KMeans was the best according to silhouette score.

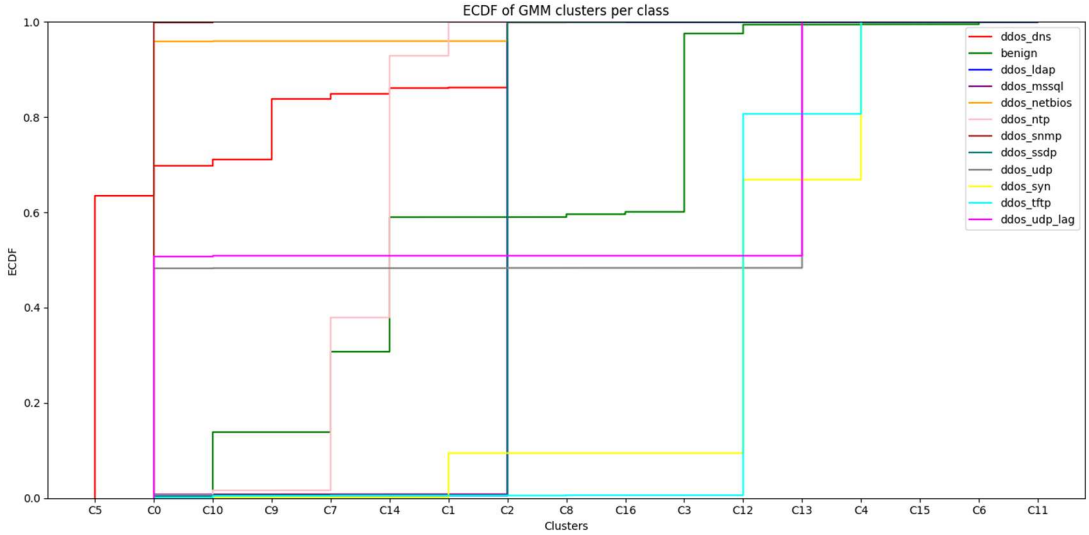


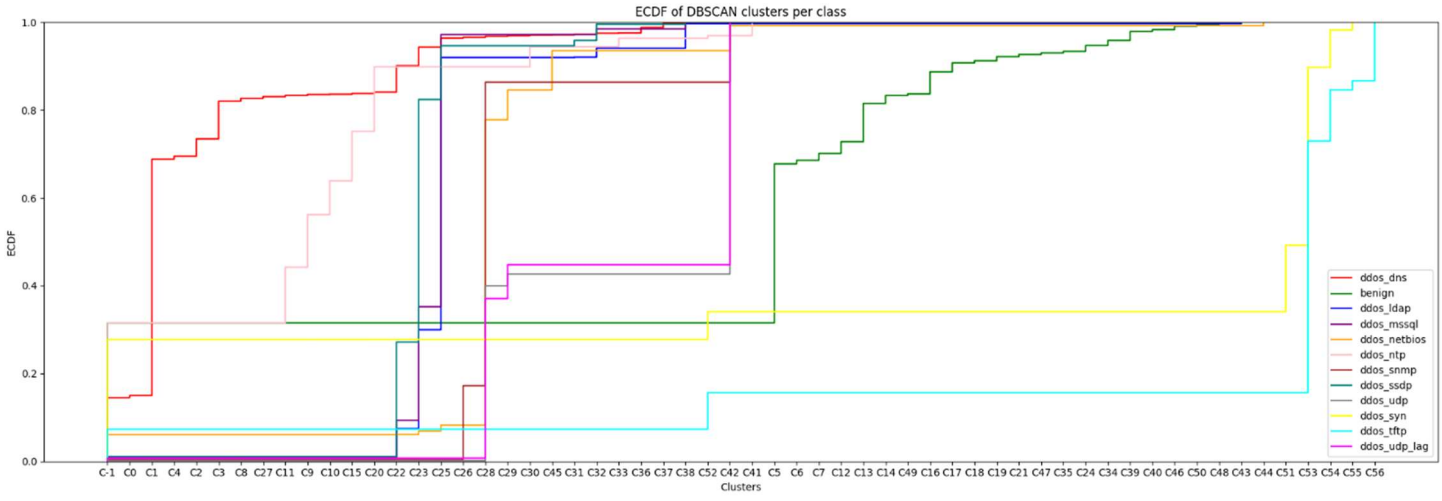
To better understand the complex shape of this kind of dataset, it is useful to visualize a t-SNE reduced version of the dataset and compare the samples assigned from each clustering algorithm in respect of GT labels.



Some sort of relationship between clustering assignments and ground truth are displayed by ECDF of clusters per class. Therefore, we can spot which kind of flow is grouped together, and even if there is some cluster that represents only one kind of flow resulting in a “correct” clustering assignment.







KMeans ECDF analysis

The plot shows that DDoS UDP lag (cluster C1) attacks are assigned to a single cluster, and it is the only algorithms that achieve this performance. There is another flow that is mostly correctly assigned, and it is DDoS LDAP (cluster C3). However, Cluster C1 label is assigned to different attacks showing similar pattern according to KMeans; these flows are:

- DDoS netbios (mostly)
- DDoS tftp
- DDoS ssdp

It is important to mention that DDoS ntp and benign flows follow a very similar pattern among the different clusters they are assigned to.

GMM ECDF analysis

In this case, the attacks that are mostly assigned to a single cluster are DDoS mssql (cluster C0) and DDoS ldap (cluster C2). There are two labels that has the exact same cluster assignment (assigned to two clusters):

- DDoS UDP lag
- DDoS UDP

It could be a consequence of the actual similarity of the two attacks, and a similar confusion happened in the supervised classification task. Even with GMM, as well as KMeans, benign and ddos_ntp share similarities in clustering showing that the two flows have some intrinsic connection. In addition, also a considerable amount of DDoS tftp and DDoS SYN are assigned to the same cluster: Cluster C12.

DBSCAN ECDF analysis

From the ecdf an important information is retrieved. Benign traffic (green line) is assigned to several clusters, but these clusters are composed only by benign flow highlighting the fact that benign flow is in some way different from the malicious one (recall to the supervised results where the benign flow was completely or at least 99% correctly classified).

In summary, it is shown how KMeans and GMM clustered correctly some kind of attacks, while DBSCAN, even assigning more clusters for the benign flow (maybe different kind of benign flows), correctly distinguished most of the benign traffic from the malicious one.

Feature importance

Two approaches are compared to retrieve feature importance from the obtained clusters:

- XGBClassifier's feature importance:
  - XGBClassifier is a class in the XGBoost library specifically designed for classification tasks. The `XGBClassifier` class is an implementation of the XGBoost algorithm tailored for classification problems. XGBClassifier is a tree-based model and its implementation allows to extract the feature importance by an inner function that returns the features with their importance score respect the whole dataset, so it represents a score of global importance
- SHAP:
  - SHAP (SHapley Additive exPlanations) values are a concept from cooperative game theory, and in the context of machine learning, they provide a way to fairly allocate the contribution of each feature to the prediction of a model. The SHAP library in Python is specifically designed for interpreting the output of machine learning models.

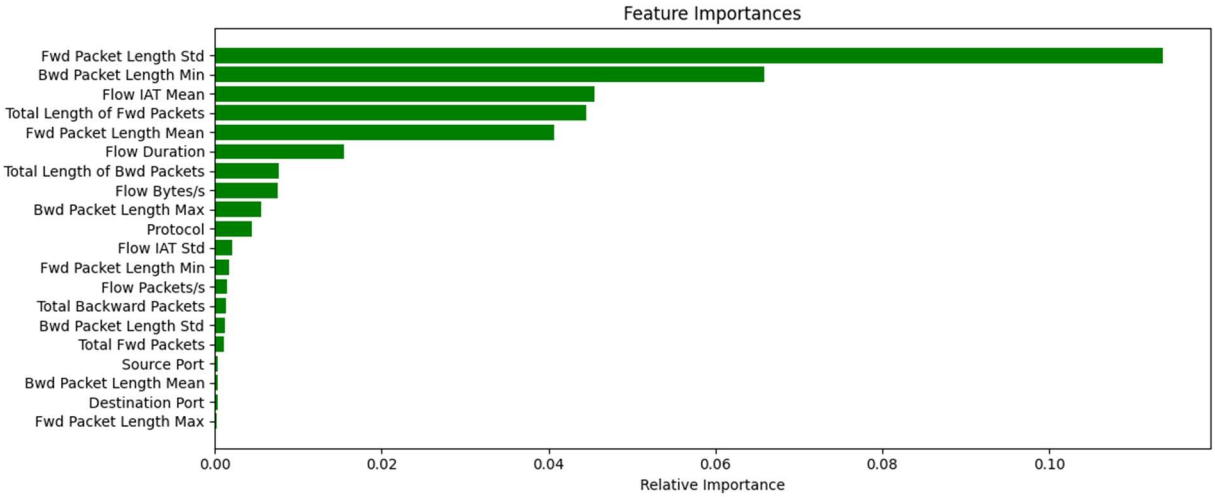
The approach is divided into several steps:

1. Clusters are retrieved by applying the clustering algorithm to the PCA-reduced dataset
2. XGBClassifier is trained with the original scaled dataset (non PCA-reduced to retrieve original feature importance) along with the labels provided by the clustering assignments
3. Feature importance is extracted from the model using its inner function (XGBClassifier.feature_importances_)
4. Retrieve SHAP values using the SHAP library and plotting them using the shap.summary_plot function

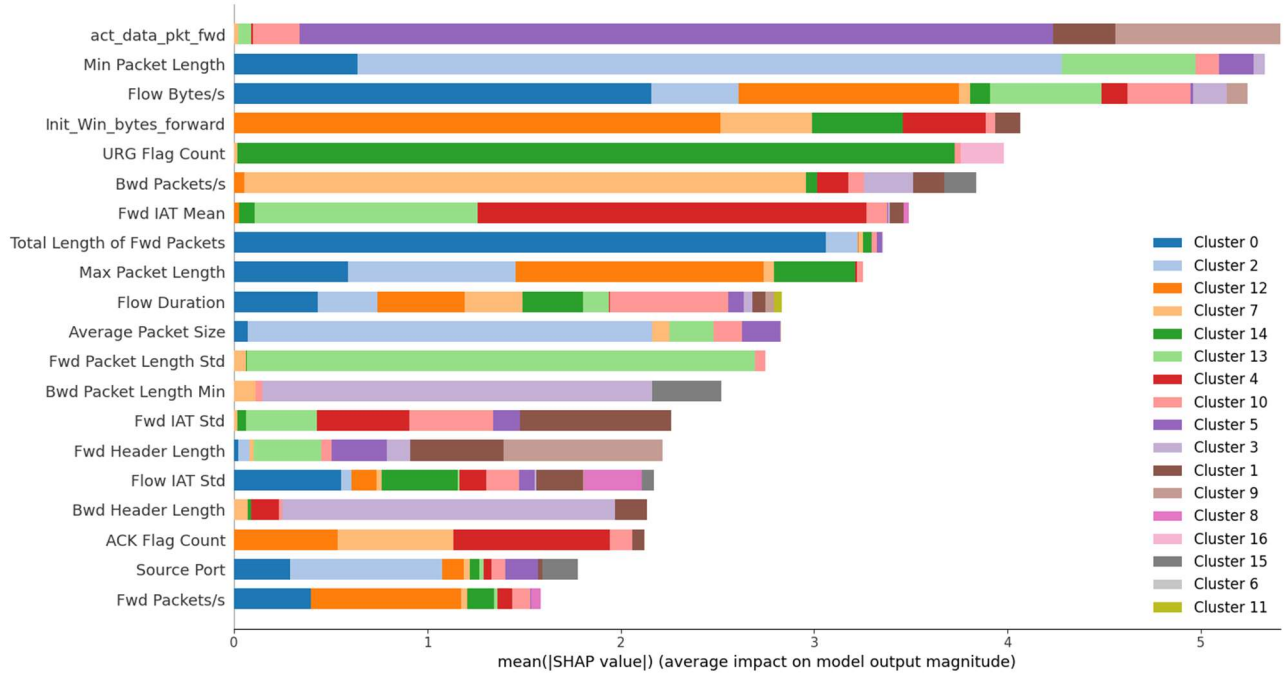
```
explainer = shap.TreeExplainer(model, feature_names=features)
shap_values = explainer.shap_values(df)
shap.summary_plot(shap_values, df, class_names=clusters, feature_names=features,
                  sort=True)
```

Results

The following considerations are based on the results provided by GMM clustering algorithm, since it was chosen due to its higher Adjusted Rand Index score. Obviously, the following reasoning process can be iterated with other clustering results.



Feature importance in the context of tree-based classifiers, such as decision trees, random forests, and gradient boosting machines (like XGBoost), refers to the measure of the contribution of each feature to the predictive performance of the model. It helps identify which features are the most influential in making predictions. The importance is typically assessed based on the contribution of each feature to the reduction in impurity or error during the construction of decision trees.



In this plot, the impact of a feature on the classes is stacked to create the feature importance plot. In other words, the summary plot for multiclass classification can show you what the machine managed to learn from the features. It is shown how different features are used to define certain clusters. It is possible to show even feature importance for each Cluster n with the following code:

```
shap.summary_plot(shap_values[n], feature_names=df.columns, sort=True)
```

For example, recalling the clusters similar flow that shared similar clusters assignments in the ecdf analysis, there were some couples of flows that were assigned to the same clusters:

- DDoS UDP and DDoS UDP lag: assigned to Cluster 0 and Cluster 13 and the most important features for the two assignments are respectively:
    - “Total Length of Fwd Packets”
    - “Fwd Packet Length Std”
- DDoS SYN and DDoS tftp: share Cluster 12 and Cluster 4 assignments with most important features respectively:
    - “Init_Win_bytes_forward”
    - “Fwd IAT Mean”
- Benign and DDoS ntp: share Cluster 7 and Cluster 14 assignments with most important features respectively:
    - “Bwd Packets/s”
    - “URG Flag Count”

In summary, the previously reported features are responsible for clustering assignments similarities between the flows considered. This analysis could be useful to enhance the nature of certain attacks and put a stress on the features considered, for example, to build a classifier for particular kind of flows.