# **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 Forests. 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.

Furthermore, 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.

## **Section 1**

## **Section 2**

In this project, 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.

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

* Accuracy:
* Precision:
* Recall:
* F1-score:

### **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, while for regression, it calculates the average (or another aggregation) of the target values of the k-nearest neighbors.

Key parameters include 'k' (the number of neighbors) and choosing an appropriate value for k is crucial.

### **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.

### **Support Vector Machine (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.

### **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 modeling 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.

Firstly, the dataset (pca\_dataframe.csv) 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,

operational\_df['label'],

stratify=operational\_df['label'],

train\_size=0.7,

random\_state=15

)

### **Default hyperparameters results**

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.

Immagine che contiene testo, schermata, quadrato, diagramma

Descrizione generata automaticamenteImmagine che contiene testo, schermata, quadrato, diagramma

Descrizione generata automaticamenteImmagine che contiene testo, schermata, quadrato, diagramma

Descrizione generata automaticamenteImmagine che contiene testo, schermata, quadrato, diagramma

Descrizione generata automaticamente

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 addition to correct prediction 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 confuse benign traffic.

### **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. The following are the hyperparameters 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” =

The hyperparameter tuning process has been performed with sklearn.GridSearchCV() function. It 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.

The models’ best parameters are chosen based on the ones that guarantee the best score on validation set. For each model, as a result, the best set of hyperparameters are:

* KNN:
  + “n\_neighbours” = 7
  + “p” = 1
  + “weights” = “distance”
  + Validation score: 0.783

Immagine che contiene testo, schermata, diagramma, linea

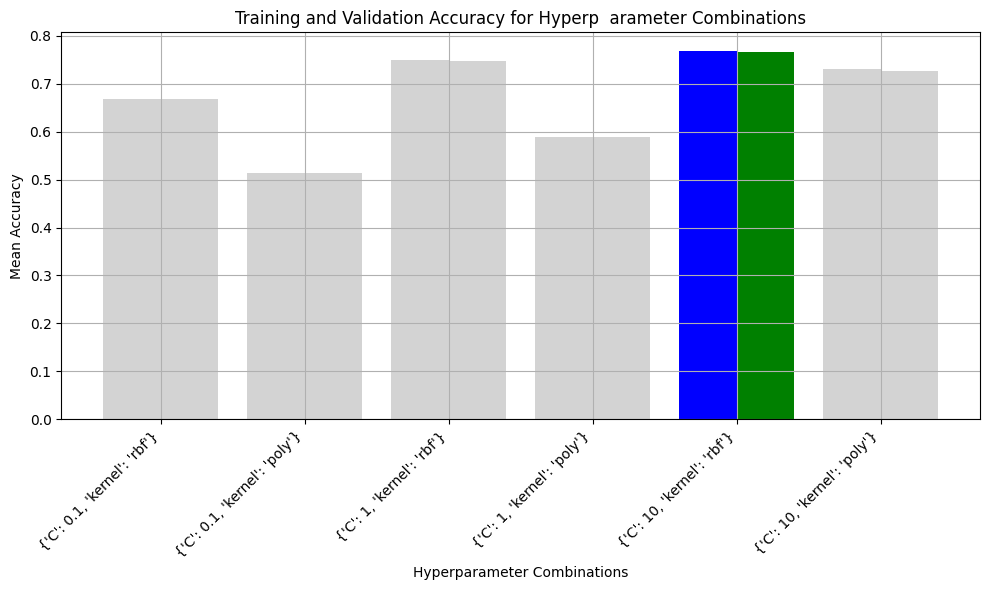
Descrizione generata automaticamente

* RFC:
  + “criterion” = “entropy”
  + “max\_depth” = None
  + “min\_samples\_split” = 3
  + “n\_estimators” = 100
  + Validation score: 0.804

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

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

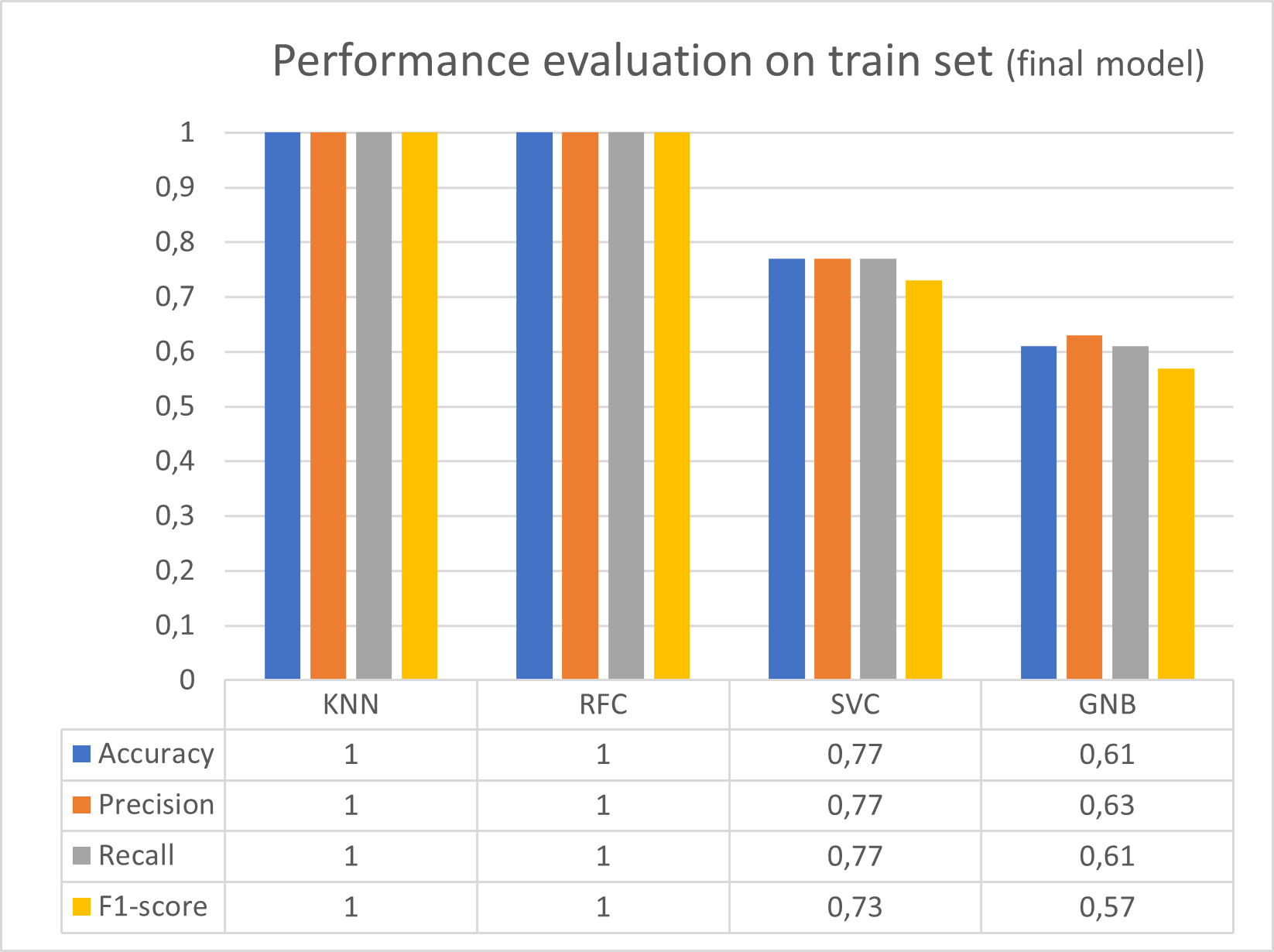
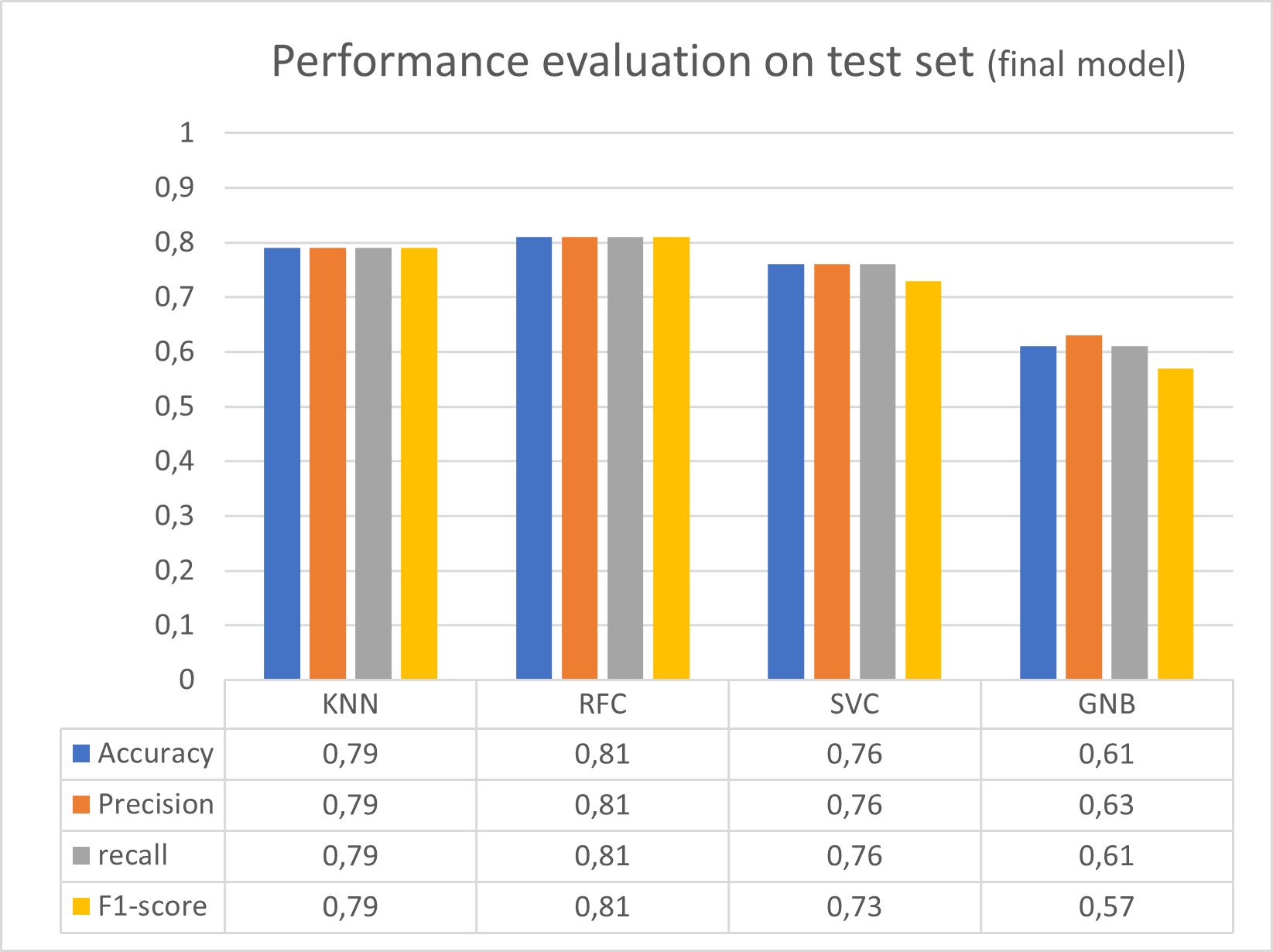


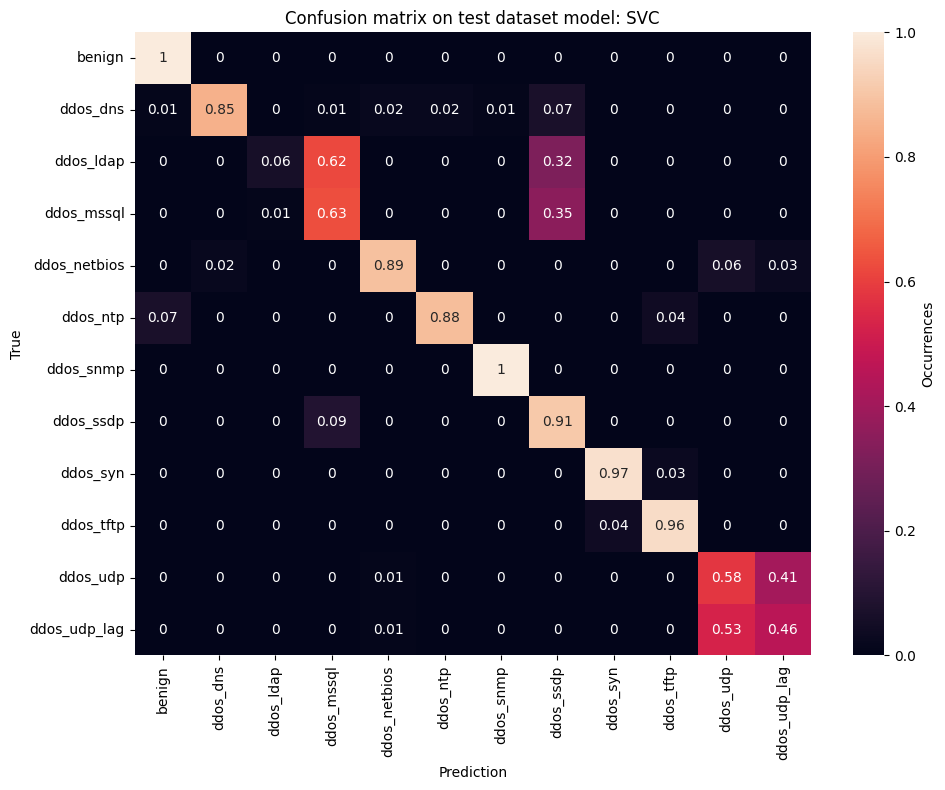
* GNB:
  + “var\_smoothing” =
  + Validation score: 0.602

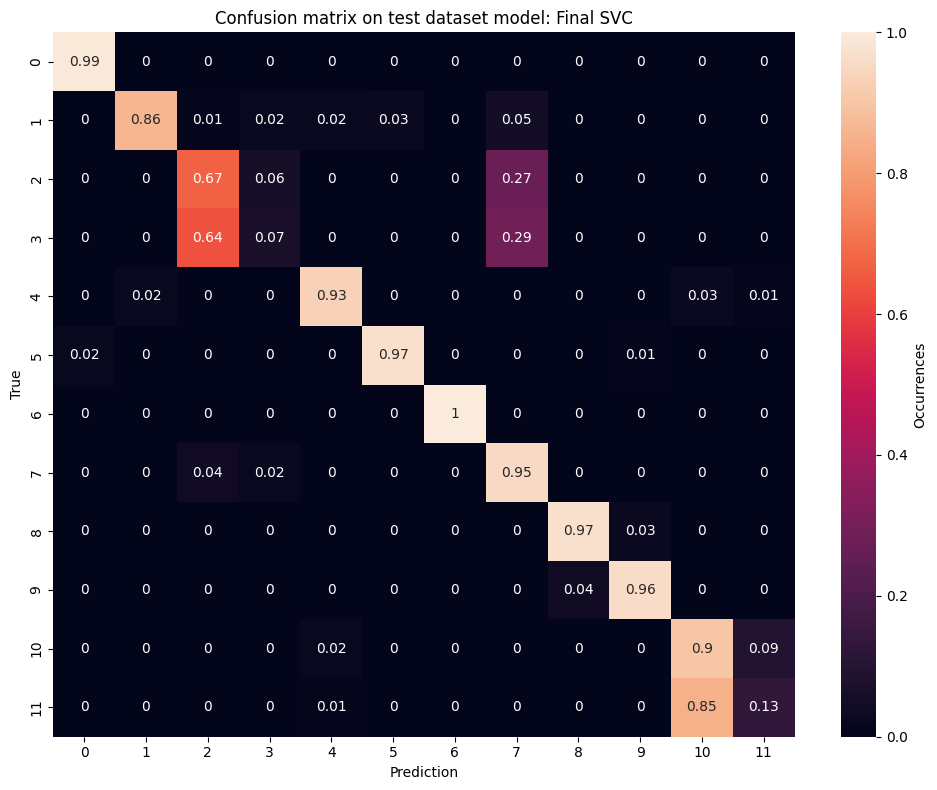
Immagine che contiene testo, schermata, linea, diagramma

Descrizione generata automaticamente

### **Tuned Hyperparameters results**

The models have been initialized with their best hyperparameters and evaluating them on the dataset has produced the following results:

As a result, 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 as display as follows:



The final model (on the left) had meaningful increase in DDoS\_UDP detection 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 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**

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).

### **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 and density estimation. 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 modeling complex data distributions, making them suitable for applications where clusters may have different shapes and sizes. They are also effective for density estimation, enabling the generation of new data points from the learned distribution.

### **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 neighboring 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 neighborhood 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 labeled 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 algorithm 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 to .

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

* 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**

Immagine che contiene testo, linea, Diagramma, diagramma

Descrizione generata automaticamente

As a result, the best number of clusters is 13 with a corresponding silhouette score value of 0.54. 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*

Immagine che contiene testo, linea, Diagramma, schermata

Descrizione generata automaticamente

#### Immagine che contiene linea, Diagramma, diagramma, testo Descrizione generata automaticamente**GMM**

It is reported also the log-likelihood metric, which is a GMM specific one. It 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 best number of clusters, both for silhouette score and log-likelihood, is 17.

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’)

#### **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]

Immagine che contiene schermata, testo, linea, quadrato

Descrizione generata automaticamente

### **Metric evaluation**