Machine Learning for Networking

Project: DDoS Attacks Detection and Characterization

Contributors:

Mohamed Eltoukhi (S346258)
Mohamed Wafic Alahwal (S288440)
Tedi Llagami (S343908)
Klaus Kullolli (S326603)

Academic Year 2024 / 2025

Contents

Introduction

Internet security, a key subset of computer security, focuses on protecting internet-based systems, including browsers, websites, and networks. Its primary goal is to establish protocols and safeguards to defend against threats such as phishing, malware, ransomware, worms, and other malicious activities that exploit the inherent vulnerabilities of the Internet, which serves as a high-risk medium for information exchange.

A denial-of-service (DoS) attack is a form of cyber assault aimed at disrupting the availability of a machine or network resource for its intended users. This is achieved by overloading the target system with excessive requests, effectively preventing legitimate requests from being processed. In contrast, a distributed denial-of-service (DDoS) attack amplifies this strategy by using traffic originating from multiple sources, often from a network of malware-infected devices.

1 Data exploration and pre-processing

Data exploration and pre-processing are crucial initial steps in data analysis, involving summarizing data characteristics, detecting patterns, and cleaning and transforming data for further analysis. These tasks, including handling missing values, removing duplicates, and normalizing data, enhance data quality and optimize machine learning model performance, resulting in more accurate and reliable outcomes.

1.1 Data Cleaning

The *ddos_df dataset* includes traffic data from common DDoS attacks, covering a specific period with **64,239 rows (flows)** and **87 columns (features)**. Each flow represents packet exchanges between source and destination IPs. The dataset is categorized into 12 labels, with 11 representing different DDoS attacks and one for benign flows. After analysis, 12 non-varying features were removed, reducing the dataset to 75 features.

1.2 Label Analysis

After analyzing the dataset, we split it into **malicious** (91.2%) and **benign** (8.8%) categories. The malicious category significantly dominates with **58,581 flows**, compared to **5,658 benign flows**, as Figure **??** illustrates.

Further investigation into attack types shows *ddos_udp_lag*, *ddos_snmp*, and *ddos_ssdp* as the most frequent, each contributing about 9.3%. *ddos_ntp* is the least common at 1.5%, highlighting the diversity in attack frequencies, depicted in Figure 1.

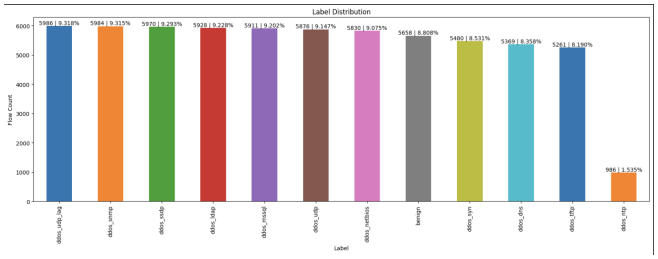


Figure 1: Detailed Label Distribution

This distribution is crucial for guiding machine learning strategies like classification and anomaly detection.

1.3 Traffic Level

A detailed traffic analysis in the dataset focused on protocols and port distribution, using *protocol mapping* to convert numerical codes into protocol names. UDP emerged as the predominant protocol, accounting for 76.1% of traffic, illustrated in Figure 2.

Traffic flow was evaluated over 30-second intervals, identifying peaks indicative of DDoS attacks, which suggest sudden, intense increases in traffic. These findings are depicted in Figure 3.

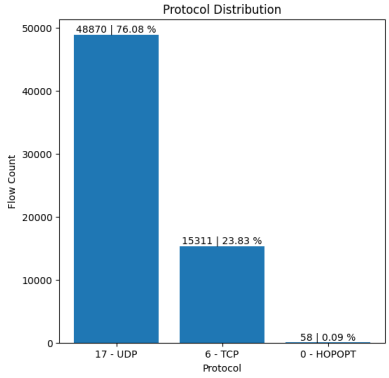


Figure 2: Protocol Distribution

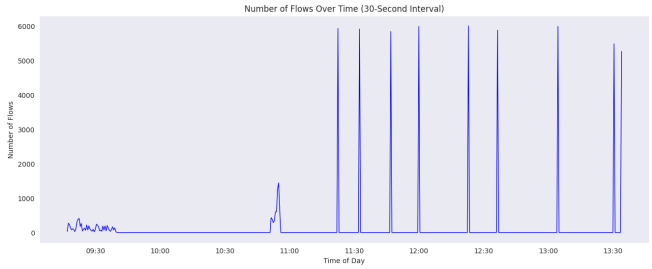


Figure 3: Number of Flows Over Time (30-Second Interval)

Additionally, Figure 4 compares flow counts across labels, revealing that DDoS attacks like *ddos_ldap* and *ddos_udp* show sharp bursts, contrasting with the scattered pattern of *benign* traffic. This analysis aids in understanding traffic distribution and enhancing detection strategies for network threats.

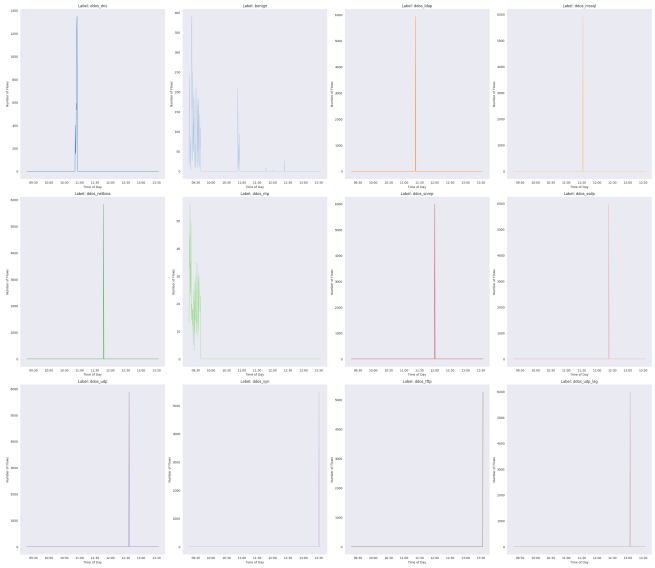


Figure 4: Number of Flows Over Time for Each Label

1.3.1 Port Analysis

An analysis of source ports across different traffic types, including DDoS attacks and benign traffic, was conducted, as shown in Figure 5. The analysis highlights that benign traffic uses a wider range of ports, while DDoS attacks show concentrated port usage:

- UDP-based attacks (e.g *ddos_udp*, *ddos_udp_lag* and *ddos_syn*) use higher source port numbers.

- TCP-based attacks (e.g *ddos_dns*, *ddos_ldap*, *ddos_mssql* and *ddos_netbios*) use lower source port numbers.

This differentiation aids in anomaly detection.

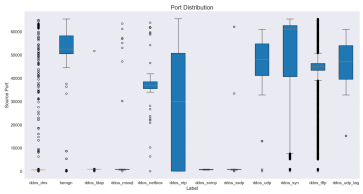


Figure 5: Port Distribution Across Different Labels

The Empirical Cumulative Distribution Function (ECDF) of source ports, shown in Figure 6, reveals:

- A steep increase at lower ports (0–1024).
- A gradual increase at middle ports (10,000–30,000).
- A linear trend at higher ports (above 40,000).

These observations help identify port usage anomalies and align with typical DDoS attack and benign traffic patterns.

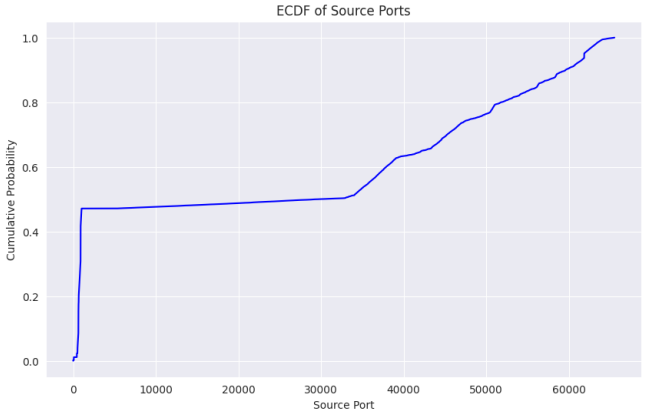


Figure 6: Empirical Cumulative Distribution Function (ECDF) of Source Ports

1.3.2 Destination Port Analysis

An analysis of destination ports was performed to understand traffic distribution across labels, using both boxplot and ECDF graphs, as shown in Figures 7 and 8.

Boxplot Insights:

- Benign traffic shows concentrated destination port usage around 0–1024, indicating typical network activity.
- DDoS attacks like *ddos_ldap* and *ddos_udp* use a broader range of ports, suggesting targeting of ephemeral ports.
- Specific attacks like *ddos_ntp* focus on lower port ranges, while others like *ddos_snmp* and *ddos_syn* show diverse port targeting.

ECDF Insights:

- The ECDF shows a near-linear rise across all ports (0–65,535), indicating a broad distribution.
- This spread suggests that while some attacks target well-known ports, others exploit higher ephemeral ports, blending with benign traffic.

These insights help in identifying DDoS strategies and improving anomaly detection.

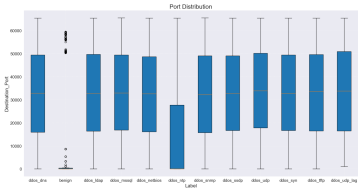


Figure 7: Boxplot of Destination Ports Across Different Labels

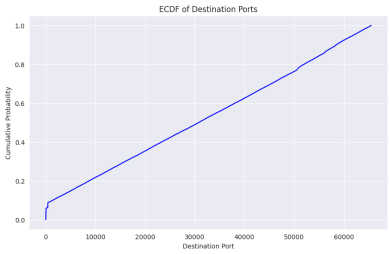


Figure 8: ECDF of Destination Ports

1.4 Correlation Analysis

Network traffic data underwent preprocessing to facilitate correlation analysis. This included converting categorical features to numerical values and standardizing timestamp data into seconds relative to '2018-12-01, 10:51:39.813448'. A standardized DataFrame, `ddos_df`, was then created.

The initial correlation matrix (Figure 9) showed significant interdependence among features.

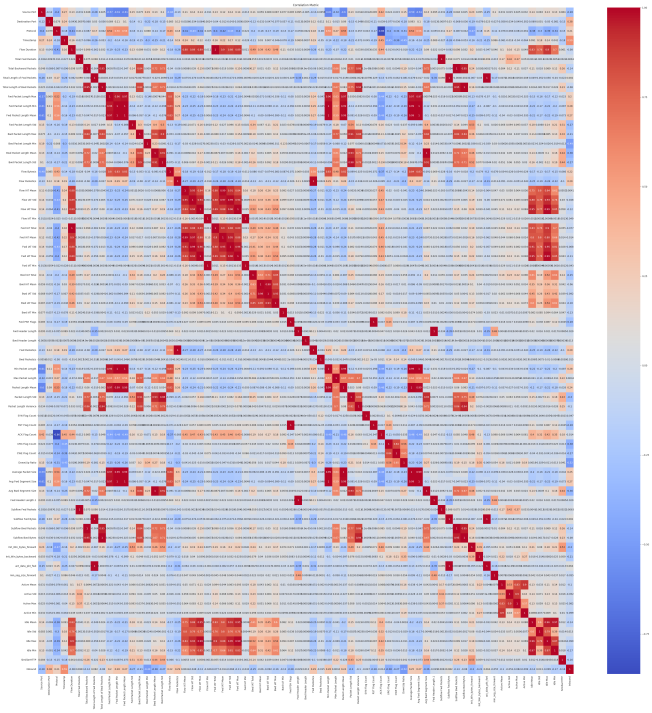


Figure 9: Correlation Matrix Before Feature Reduction

Features with high correlation (above 0.8) were removed, reducing the feature count from 71 to 32 in a refined DataFrame, `new_ddos_df`. This step enhances computational efficiency and retains critical information.

A new correlation matrix was produced for the reduced feature set (Figure 10), better representing the relationships among the remaining features.

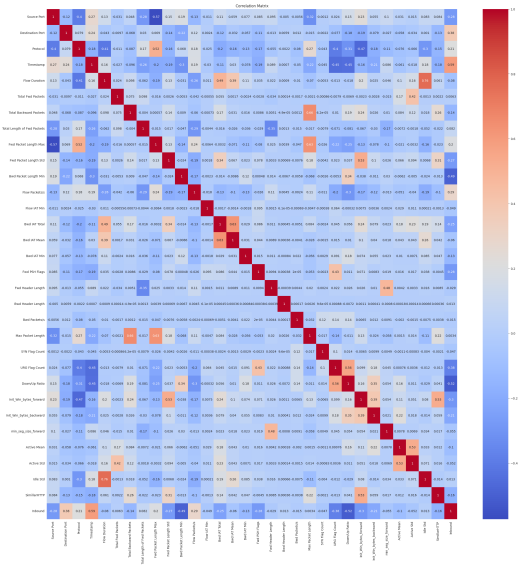


Figure 10: Correlation Matrix After Feature Reduction

1.4.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) was applied to the scaled `new_ddos_df` using Kaiser's rule, which suggests retaining components with eigenvalues over 1. The first 12 principal components captured 68.38% of the variance, effectively reducing dimensionality with minimal information loss, as shown in Figure 11.

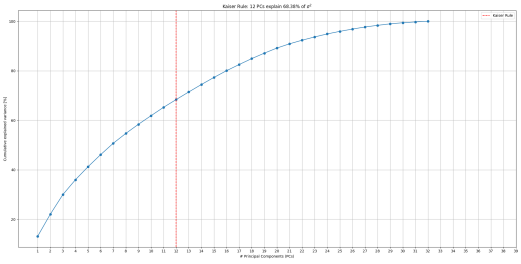


Figure 11: Principal Component Analysis

This dimensionality reduction enhances computational efficiency without sacrificing critical data relationships. The top contributing features for the first two principal components provide insights into their significance and impact, shown in Figures 12 and 13.

Ultimately, the decision to use `new_ddos_df` in further analyses was made to maintain interpretability and transparency, balancing robust analysis with clear, comprehensible results. This approach ensures findings are accessible and meaningful.

```
Top 5 features for PC0:
                        PC0_Loading
Inbound                    0.344371
Protocol                   0.313329
Init_Win_bytes_forward     0.294311
Down/Up Ratio              0.290120
URG Flag Count             0.241704

=====================================
```

Figure 12: Top 5 Features for PC0

```
Top 5 features for PC1:
                        PC1_Loading
Timestamp                  0.453956
Max Packet Length          0.345713
Idle Std                   0.308148
Fwd Packet Length Max      0.305362
Flow Duration              0.303760
```

Figure 13: Top 5 Features for PC1

2 Supervised Learning - Classification

A supervised learning approach is used to classify network flows into attack types, starting with a stratified split of the dataset into training (X) and testing (X_{test}) sets, maintaining attack category proportions. The dataset is divided with 70% allocated to training and 30% to testing. The training data is further split into training (X_{train}) and validation (X_{val}) subsets in a stratified manner to preserve class frequencies.

```python
y = df_label['intLabel']
X, X_test, y, y_test = train_test_split(pca_data, y, stratify=y, train_size=0.7, random_state=15)


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

Figure 14: Code snippet for dataset splitting.

This stratified partitioning ensures that each subset is a representative sample of the overall data, enhancing the model's robustness and minimizing bias to improve generalization to unseen data.

2.1 Supervised Learning Algorithms

Three distinct supervised learning algorithms are employed for the classification task:

- **Random Forest (RF)**: This ensemble method uses multiple decision trees to improve accuracy and stability, reducing overfitting through averaging predictions, ideal for high-dimensional datasets.
- **Support Vector Machine (SVM)**: Effective in high-dimensional spaces, SVM seeks the optimal hyperplane to separate different classes, useful when features outnumber data points.
- **K-Nearest Neighbors (K-NN)**: Classifies based on the majority class among its k nearest neighbors, with performance depending on k and the distance metric.

Default hyperparameter settings are used for straightforward implementation. Model performance is evaluated using metrics such as the confusion matrix, precision, recall, and F1 score, enabling comparison of the algorithms' effectiveness.

2.1.1 Random Forest

After training and prediction, the classification results are presented in the following report 15. The graphs highlight that the `RandomForestClassifier` achieves nearly perfect metrics.

Classification Report – Training Set

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

Classification Report – Validation Set

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

Figure 15: Classification reports for the Random Forest model. (Left: Training Set, Right: Validation Set).

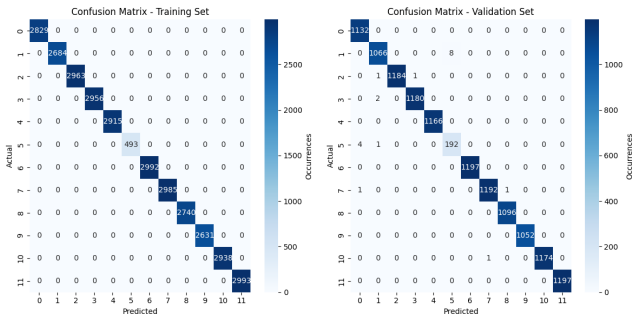


Figure 16: Confusion matrices for the Random Forest model. (Left: Training Set, Right: Validation Set).

The confusion matrices for both the training and validation sets are displayed above in Figure 16. These matrices provide a comprehensive visualization of the model's performance, demonstrating its ability to correctly classify nearly all samples.

2.1.2 Support Vector Machine

After performing the SVM algorithm with default parameters, we get the results shown by the following classification report. Figure 18 The graphs show that the metrics of the SVC are good but not like the RF.

2.1.3 K-Nearest Neighbour

After performing the KNN algorithm with default parameters, the classification results are presented in the following report. The graphs show that the metrics of the `KNeighborsClassifier` are good but not as strong as those observed for the other models.

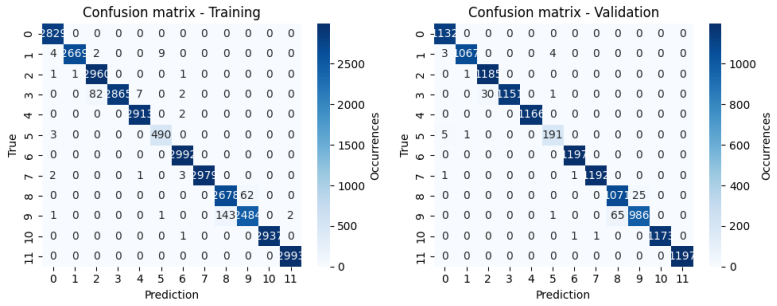


Figure 17: Confusion Matrix for SVM

Training set

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2829
1	1.00	0.99	1.00	2684
2	0.97	1.00	0.99	2963
3	1.00	0.97	0.98	2956
4	1.00	1.00	1.00	2915
5	0.98	0.99	0.99	493
6	1.00	1.00	1.00	2992
7	1.00	1.00	1.00	2985
8	0.95	0.98	0.96	2740
9	0.98	0.94	0.96	2631
10	1.00	1.00	1.00	2938
11	1.00	1.00	1.00	2993
accuracy			0.99	32119
macro avg	0.99	0.99	0.99	32119
weighted avg	0.99	0.99	0.99	32119

Validation set

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

Figure 18: Classification reports for the SVM model. (Left: Training Set, Right: Validation Set).

Classification Report – Training Set

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

Classification Report – Validation Set

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1132
1	0.99	0.98	0.99	1074
2	0.99	0.99	0.99	1186
3	0.99	0.98	0.99	1182
4	1.00	1.00	1.00	1166
5	0.93	0.96	0.94	197
6	1.00	1.00	1.00	1197
7	1.00	1.00	1.00	1194
8	0.99	0.99	0.99	1096
9	0.98	1.00	0.99	1052
10	1.00	1.00	1.00	1175
11	1.00	1.00	1.00	1197
accuracy			0.99	12848
macro avg	0.99	0.99	0.99	12848
weighted avg	0.99	0.99	0.99	12848

Figure 19: Classification reports for the K-Nearest Neighbour model. (Left: Training Set, Right: Validation Set).

The confusion matrices for both the training and validation sets are displayed below. These matrices provide a comprehensive visualization of the model's performance, demonstrating its ability to classify data accurately, though less effectively than other algorithms.

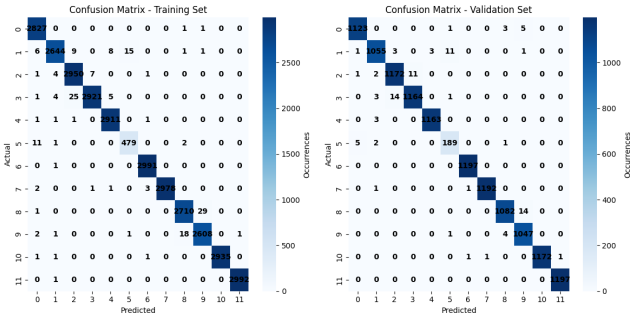


Figure 20: Confusion matrices for the K-Nearest Neighbour model. (Left: Training Set, Right: Validation Set).

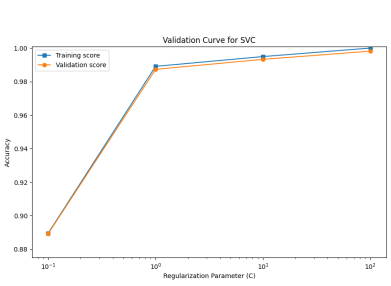
2.2 Hyperparameters Tuning

The analysis of confusion matrices and classification reports shows no underfitting or overfitting for the models. Underfitting is usually marked by low training and validation performance, while overfitting is characterized by high training performance with poor validation results.

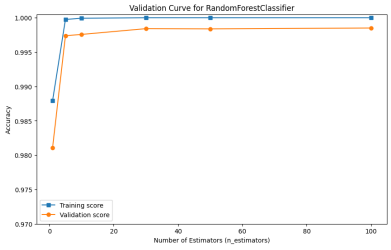
The validation curves for the *Support Vector Classifier (SVC)*, *K-Neighbors Classifier*, and *Random Forest Classifier* support this, showing no signs of underfitting or overfitting:

- **SVC**: Training and validation accuracies are aligned, indicating proper fit.
- **K-Neighbors Classifier**: Accuracies diverge slightly with more neighbors, yet remain high, indicating good generalization.
- **Random Forest Classifier**: Exhibits high and stable training and validation accuracies, suggesting robust generalization.

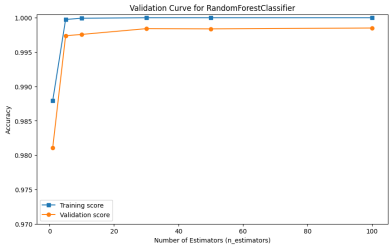
All models demonstrate effective learning from training data and strong generalization to unseen data. The *Random Forest Classifier* stands out as the best performer, with optimal hyperparameter of `10` estimators balancing performance and efficiency.



(a) Validation Curve for SVC



(b) Validation Curve for K-Neighbors



(c) Validation Curve for Random Forest

Figure 21: Validation curves for the three models: SVC, K-Neighbors Classifier, and Random Forest Classifier.

Below we have the tuned parameters and accuracy for each of the algorithms we used.

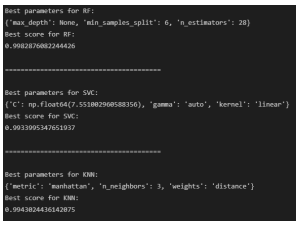


Figure 22: Tuned parameters and accuracy of each algorithm

	TRAIN_TIME	PREDICT_TRAIN_TIME	PREDICT_VAL_TIME	MODEL
0	0.6330	0.0737	0.0352	RANDOM_FOREST
1	0.0229	4.7897	1.8775	K_NEAREST_NEIGHBORS
2	9.9251	1.2429	0.5087	SUPPORT_VECTOR_MACHINE

Figure 23: Time performance

Considering validation curve analysis and time performance, **Random Forest** is the best algorithm for us to use. The following analysis is based on this algorithm.

2.3 False Positive and False Negative Inspection

A detailed evaluation of False Positives and False Negatives was conducted, showing rare misclassifications and exceptional accuracy in identifying true positives. This underscores the model's precision and recall, as evidenced by the low rates of erroneous classifications.

Figure 24: Confusion matrix in terms of labels

The ROC curve further validates the model's high performance, with a trajectory towards the upper-left corner and a high AUC value, indicating effective class differentiation.

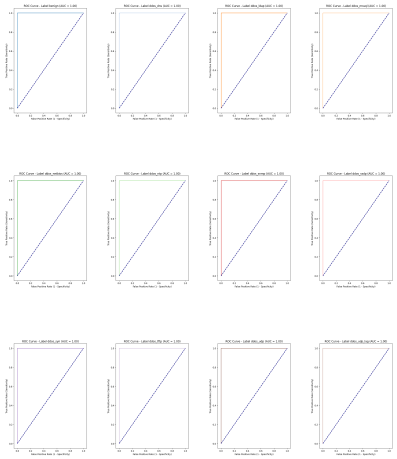


Figure 25: ROC for Random Forest

Feature importance analysis points to Timestamp as a significant factor in misclassifications, suggesting the model's challenge in utilizing temporal information effectively.

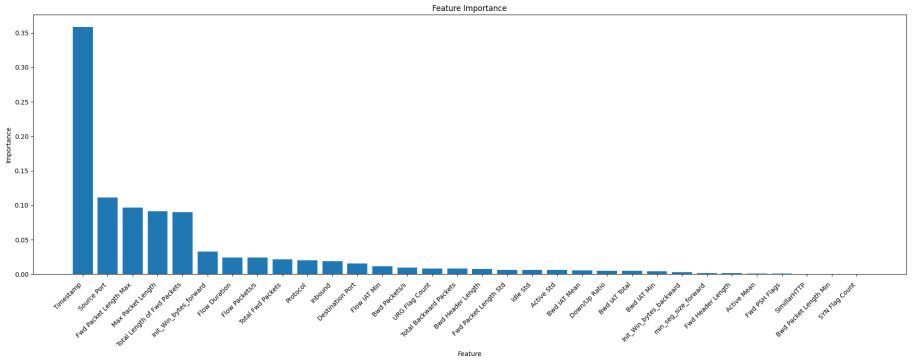
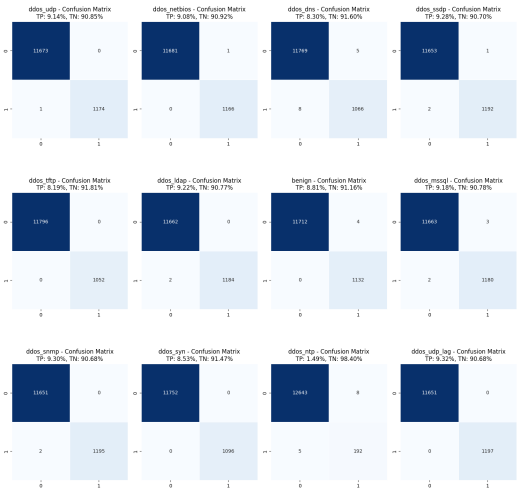


Figure 26: Feature importance analysis related to misclassifications

2.4 Testing Performance

After completing the training phase with the Random Forest model, the testing phase was carried out using the tuned parameters mentioned in Figure 22. The results of this testing phase are presented in the subsequent visuals. These outcomes align with the performance observed during the training phase, further validating the model's effectiveness.



(a) Confusion Matrix

Classification Report – Test Set

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1697
1	1.00	1.00	1.00	1611
2	1.00	1.00	1.00	1779
3	1.00	1.00	1.00	1773
4	1.00	1.00	1.00	1749
5	0.98	1.00	0.99	296
6	1.00	1.00	1.00	1795
7	1.00	1.00	1.00	1791
8	1.00	1.00	1.00	1644
9	1.00	1.00	1.00	1578
10	1.00	1.00	1.00	1763
11	1.00	1.00	1.00	1796
accuracy			1.00	19272
macro avg	1.00	1.00	1.00	19272
weighted avg	1.00	1.00	1.00	19272

(b) Classification Report

Figure 27: Test Performance: (a) Confusion Matrix and (b) Classification Report.

The model's strong performance is linked to the use of temporal features like Timestamp, which may reflect patterns of malicious activity at certain times, enhancing classification accuracy. However, this raises concerns about potential overfitting, where the model might be memorizing timestamp-based patterns rather than learning to generalize.

3 Unsupervised Learning - Clustering

This section explores unsupervised learning techniques for clustering analysis using the dimensionally reduced and scaled **cluster_new_ddos_df**. Initial visualization uses the two most prominent principal components to assess data distribution.

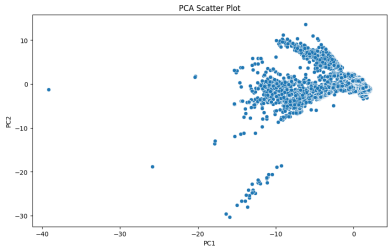


Figure 28: Sample Data Distribution Using Two Principal Components (PCs)

The scatter plot indicates a non-Euclidean structure, suggesting that traditional clustering algorithms like K-means and Gaussian Mixture Models (GMM) may face challenges. Although DBSCAN could be an alternative, its high computational cost limits practicality. Consequently, K-means and GMM will be used despite these challenges.

3.1 K-means

K-means, a hard clustering algorithm, groups m observations into k clusters by assigning each observation to the nearest cluster centroid. Initially, data was partitioned into 12 clusters to match the dataset labels using default hyperparameters. Performance metrics such as the *silhouette score*, *adjusted Rand index*, and *Rand index* were used to evaluate clustering quality.

```
Clustering Error: 1043097.74
Silhouette Score: 0.33
Rand Index: 0.81
Adjusted Rand Index: 0.36
```

Figure 29: K-means with random parameters

The results, showing low silhouette scores and significant errors, indicate a suboptimal initial setup and highlight the need for optimizing cluster numbers and hyperparameters to enhance model performance.

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

Silhouette analysis measures how well an object fits within its cluster compared to other clusters, with scores ranging from -1 to 1, where higher scores indicate better-defined clusters. The elbow method plots the inertia (sum of squared distances) against different cluster counts and identifies the optimal number at the "elbow," where inertia reduction slows noticeably.

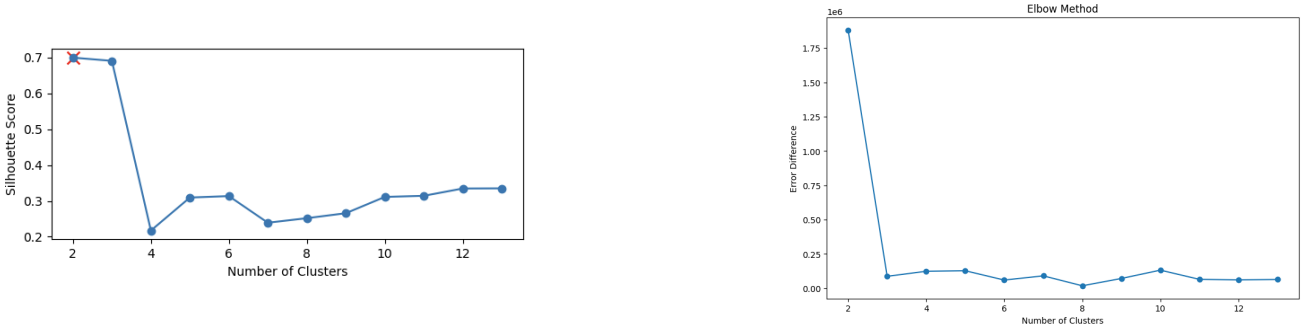


Figure 30: Tuning the number of clusters

Analysis suggests two clusters might be optimal; however, three clusters were chosen to balance clustering accuracy and error, offering lower error with a comparable silhouette score.

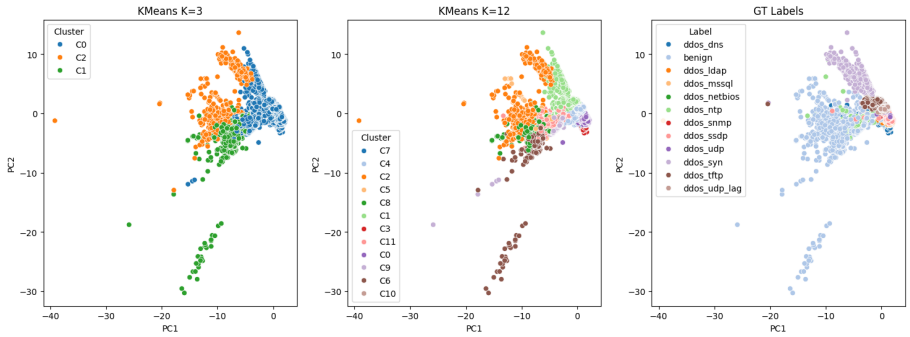


Figure 31: Comparison of data distributions

3.1.2 Hyper-parameter tuning

The K-means algorithm requires the specification of hyperparameters such as *init* and *random_state*. The *init* parameter can be set to either *kmeans++* or *random*, while *random_state* can take different values(till now used parameters are *init* = *kmeans++*, *n_init* = *auto* and *random_state* = *33*). To identify the optimal configuration, a grid search is performed, systematically testing all possible combinations of these values. The configuration that produces the highest silhouette score is selected as the best option. In this analysis, the optimal parameters are those presented in Figure 32.

```
Best Score KMean(k=3):  silhouette                    0.564689
init_method                   k-means++
random_state                         42
adjusted_rand_index            0.024515
normalized_mutual_info_score   0.182724
n_init                               10
Name: 6, dtype: object
Best Tuned params KMean(k=3):  {'init': 'k-means++', 'random_state': np.int64(42), 'n_clusters': 3, 'n_init': np.int64(10)}
```

Figure 32: Tuned parameters for K-means (k=3)

Once the optimal number of clusters and hyperparameter values were identified, the K-means algorithm was executed again using these parameters, yielding the results outlined below. As shown in the Figure 32 the Silhouette score is better then the previously measured in Figure 29.

3.2 Gaussian Mixture Modelling (GMM)

Gaussian Mixture Models (GMM) use a soft clustering approach, assigning a probability to each sample for belonging to various clusters. Unlike K-means, GMM assumes that data points are drawn from Gaussian distributions, allowing it to handle complex, overlapping cluster structures.

```
Log Likelihood Score Average: 132.35584292340573
Silhouette Score: 0.2441912644284167
Rand Index: 0.688142185657336
Adjusted Rand Index: 0.1909311698070487
```

Figure 33: GMM with Random Parameters

Initial tests with GMM using 12 clusters showed a suboptimal silhouette score and significant errors, as seen in Figure 33. This result indicates the need to optimize the number of clusters and adjust hyperparameters to enhance performance.

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

The log-likelihood in GMM quantifies how well the model explains the observed data, aiming for maximization to ensure a better fit. The Adjusted Rand Index (ARI) compares the true class labels with the model's predictions, with values ranging from -1 to 1; a higher ARI indicates a better match.

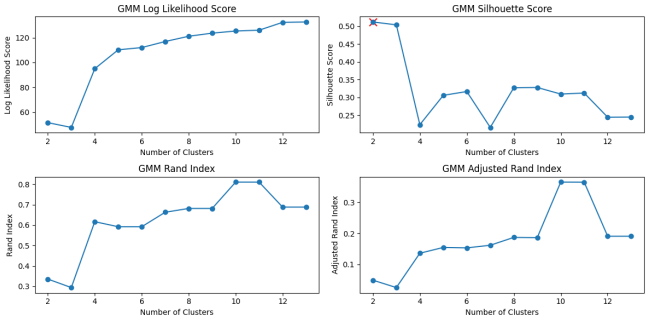


Figure 34: Tuning the number of clusters

Figure 34 shows the silhouette score peaking at 3 clusters, but log-likelihood and ARI suggest a higher cluster count might be better. Opting for 7 clusters balances these metrics, allowing for a more comprehensive performance evaluation.

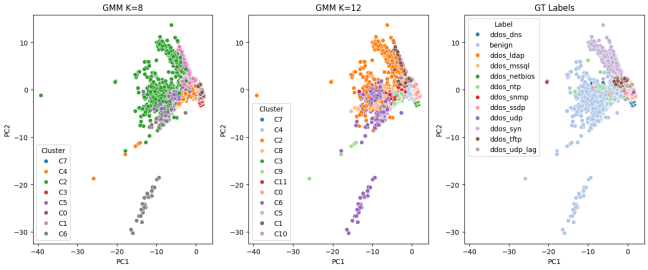


Figure 35: Comparison of data distribution GMM

This balanced approach aims to optimize the fit of the model to the data, considering both cluster cohesion and alignment with true classifications.

3.2.2 Hyper-parameter tuning

Similar to K-means, GMM also requires the specification of hyperparameters such as *init* and *random_state*. These parameters are selected through a grid search, identifying the combination that achieves the highest silhouette score as the optimal configuration. In this analysis, the recommended parameters are those depicted in Figure 36.

```
Best Score GMM(k=8):  likelihood_avg        121.140589
cluster                 8.000000
silhouette              0.327108
rand_index              0.681427
adjusted_rand_index     0.187170
Name: 2, dtype: float64
Tuned GMM Parameters with k=8: {'n_components': 8, 'random_state': 33}
```

Figure 36: Tuned parameters for GMM (k=8)

After identifying the optimal number of clusters and hyperparameter values, the GMM algorithm was executed again using these parameters, yielding the results presented below.

As illustrated in Figure 36, the silhouette score demonstrates an improvement compared to the value previously observed in Figure 33. Finally, the subsequent graphs showcase the contribution of each cluster to the overall silhouette score, highlighting the quality of the algorithm employed.

4 Clusters explainability and analysis

In this section we will use the clusters obtained in Section 3 to find patterns in the data and see how they compare and match up to the ground truth. First of all, we need to choose which clustering algorithm gave us the best results and focus on them for the analysis. So, we need to take a look at the performance metrics for the two algorithms we used, K-means clustering and GMM.

```
KMeans Labels with k=3: [0 1 2]
Rand Index for KMeans with k=3: 0.024514555184463882
Normalized Mutual Information for KMeans with k=3: 0.18272388215370372
Silhouette Score for KMeans with k=3: 0.5646886492039027
```

```
GMM Labels with k=8: [0 1 2 3 4 5 6 7]
Rand Index for GMM with k=8: 0.18716992977134117
Normalized Mutual Information for GMM with k=8: 0.5188208401156332
Silhouette Score for GMM with k=8: 0.327107862612523
```

Figure 37: Performance metrics for K-means and GMM

As we can see GMM with $K = 8$ gives us better RI and NMI metrics, however we will choose K-means clustering with $K = 3$ due to its higher Silhouette Score. The difference in this metric indicates that the K-means algorithm will give us better defined clusters, which we can use to draw important considerations.
Now that we have chosen which algorithm we are going to use for the analysis, let's take a look at the results we obtained from K-means clustering and see how they compare to the Ground Truth.

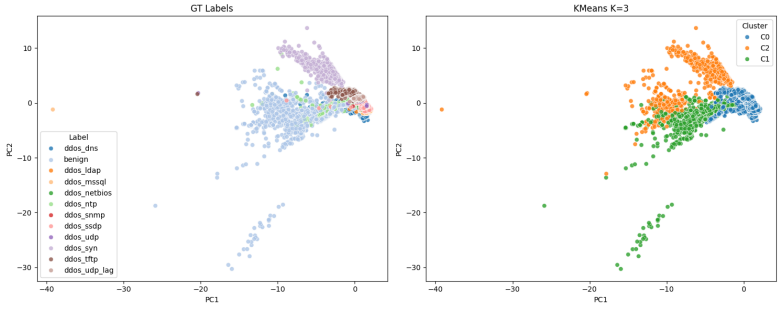


Figure 38: K-Means (k=3) comparison with GT

Looking at the two graphs side by side, we notice that there is quite a bit of mismatch between the clusters we obtained and the Ground Truth. However, we can look at how the different traffic types are spread across the three clusters to see if we can notice any patterns. In order to do this, we can plot the ECDF of each class as in Figure 39

Looking at the graphs, there are some observations we can make. The majority of malicious traffic is located in cluster 0. The only types of malicious traffic which don't have the majority of their data points in this cluster are **ddos_ntp**, which is mostly in cluster 1 and **ddos_syn**, which is evenly spread among cluster 0 and cluster 2. Benign traffic is spread between cluster 0 and cluster 1 and a relatively small amount of data points in cluster 2. There are no clusters which have exclusively one type of traffic, but most of the data points corresponding to malicious

traffic are almost exclusively in cluster 0, which is useful information to us.

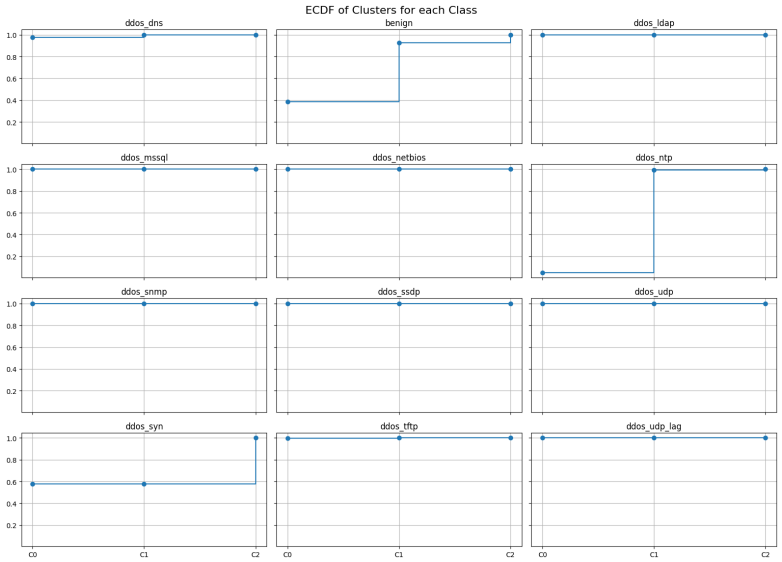


Figure 39: Traffic Types spread in Clusters

It is somewhat concerning that there is a substantial amount of benign traffic in cluster 0, which implies that these flows have similar characteristics to malicious ones. Therefore, it would be useful to perform some feature importance analysis to see which features contributed to the formation of the clusters.

Since we are using K-means clustering, we can utilize Centroid Distance Based Weighting (CDBW) to understand which features contribute to centroid positions and intra-cluster distances the most.

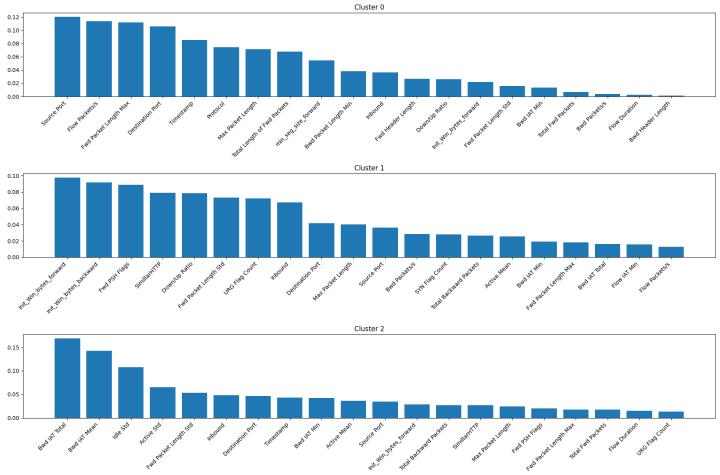


Figure 40: Feature importance for each Cluster

Based on the results from Figure 40, the three clusters are impacted quite differently by certain features.

- Cluster 0 is most affected by *Source/Destination Port*, *Flow_Packets/s* and *Fwd_packet_length_max*. Since most of the malicious flows are in this cluster, these features are crucial to understanding their behavior. It could mean that the DDoS attacks target specific port numbers

and flood them with packets, potentially very large ones, which is useful information when we want to prevent this from happening. However, we need to be careful as a lot of benign flows also exhibit these same patterns. For this reason, we will pay special attention to this cluster to hopefully understand how to distinguish between benign and malicious packets.

- Cluster 1 is most affected by *Init_Win_bytes_forward*, *Init_Win_bytes_backward* and *Find_PSH_Flags*. Cluster 1 mostly consists of benign flows and **ddos_ntp** attacks, which could mean that these attacks try to replicate benign flows window size to not be as easily detected.
- Cluster 2 is most affected by *Bwd_IAT_Total*,*Bwd_IAT_Mean* and *Idle_Std*. Since this cluster is mostly made up of **ddos_syn** attacks, it could mean that they exhibit distinguishable traffic patterns.

Now that we have seen the most impactful features in defining these clusters, we will be diving deeper in cluster 0 specifically, because most of the malicious flows are contained in it. We will start by plotting only the data points in this cluster and apply K-means clustering again to attempt to isolate the different traffic types.

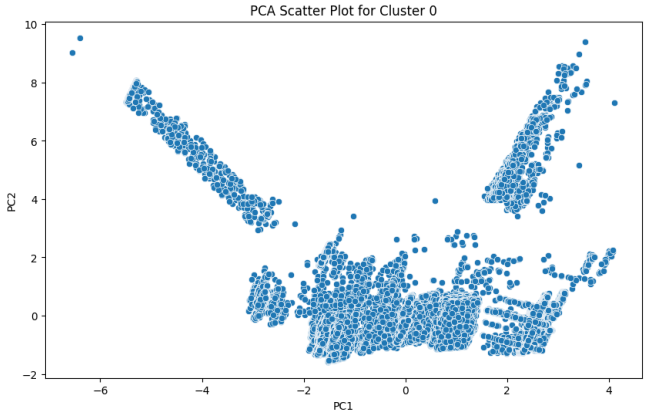


Figure 41: Cluster 0

To optimize our results, we first need to determine the number of clusters K , which we will do in a similar fashion to how we did it in section 3, except this time we will also take into consideration RI and ARI, along with the Silhouette score and the elbow method over a range of values of K.

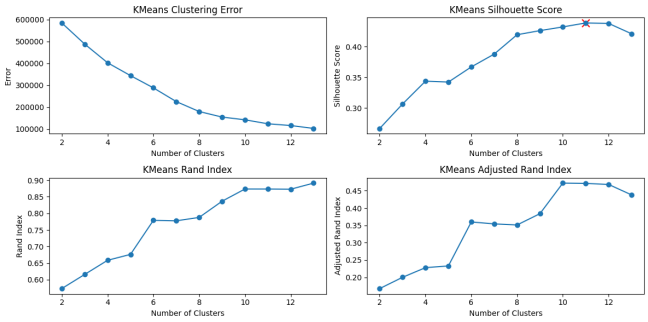


Figure 42: Performance metrics for cluster 0 sub-clustering

As we can see from the graphs in Figure42, $K = 11$ seems to be a very good choice. For this value of K we get the highest Silhouette score and it also performs very well according to the other metrics.

Now, let's take a look at the clusters we generated via the K-means algorithm for cluster 0 and compare them to the Ground Truth.

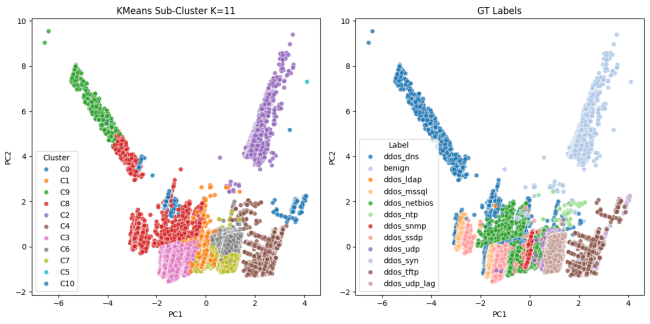


Figure 43: Sub-clusters (k=11) comparison with GT

Although the results don't match perfectly, the sub-clusters we have formed are still useful for detecting potential patterns between the different attack types. In order to detect these patterns more effectively, let's take a look at how the different classes are spread through the sub-clusters.

cluster	0	1	2	3	4	5	6	7	8	9	10
label											
benign	0	50	2116	0	9	1	0	0	1	0	1
ddos_dns	3401	271	0	652	9	0	24	100	245	531	0
ddos_ldap	0	11	0	5414	0	0	0	0	500	0	0
ddos_mssql	0	10	0	5721	0	0	0	5	173	0	0
ddos_netbios	0	3	0	188	0	0	363	4558	717	0	0
ddos_ntp	0	2	2	0	30	0	14	2	0	0	0
ddos_snmp	0	5984	0	0	0	0	0	0	0	0	0
ddos_ssdp	0	10	0	5607	0	0	0	0	349	0	0
ddos_syn	0	0	0	0	3145	0	0	0	0	0	18
ddos_tftp	0	2	0	0	5202	0	0	1	0	0	54
ddos_udp	0	3	0	2	0	0	3383	2487	0	0	0
ddos_udp_lag	0	3	0	0	0	0	3809	2174	0	0	0

Figure 44: GT spread in sub-clusters

One thing we notice immediately is that sub-cluster 2 is almost exclusively composed of benign flows, most of which are in this grouping, so we have effectively managed to separate benign traffic from malicious ones.

With malicious traffic however, it is a bit more difficult to achieve the same result given how many types of DDoS attacks are in the dataset. We can see that attacks such as **ddos_dns** are spread in multiple clusters, making it challenging to determine any specific patterns for this specific type. Still, we can draw some useful conclusions when examining attack types that have a similar spread throughout the sub-clusters, such as **ddos_udp** and **ddos_udp_lag**, which are quite evenly spread in sub-clusters 6 and 7. Since their spread is quite similar, these two types of attack could be easily mistaken for each other, so it is worth examining them further to determine which features are responsible for their similarities. Let's plot only these two attacks to examine them further.

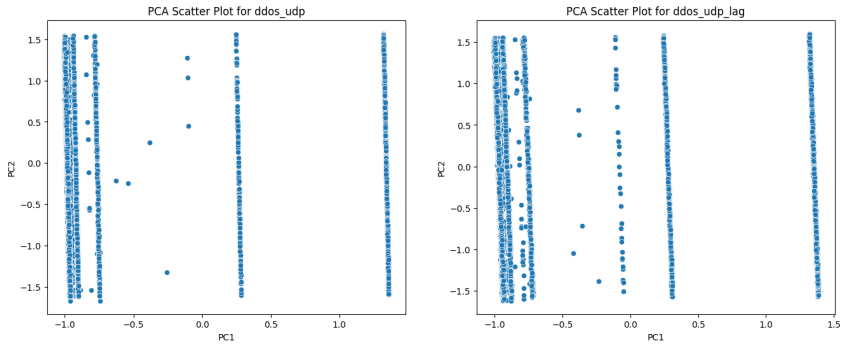
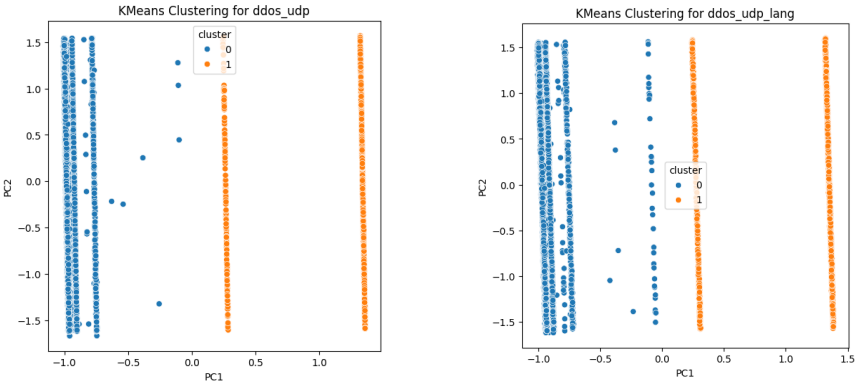


Figure 45: Scatter-plot for **ddos_udp** and **ddos_udp_lag**

As we can clearly see, these two attacks have nearly identical distributions. Thus it would be useful to perform some feature importance analysis in order to determine which features impact their behavior the most. Let's also take a look at how they are spread out in clusters 6 and 7.



DDoS UDP DDoS UDP lag

Figure 46: DDos UDP and DDoS UDP lag spread comparison

Even in the way they are spread out among sub-clusters 6 and 7, which we have named in the legend 0 and 1 respectively, they are practically identical. Finally, we will perform feature importance analysis for the two attack types in the sub-cluster they belong in. As before, we will be using CDBW.

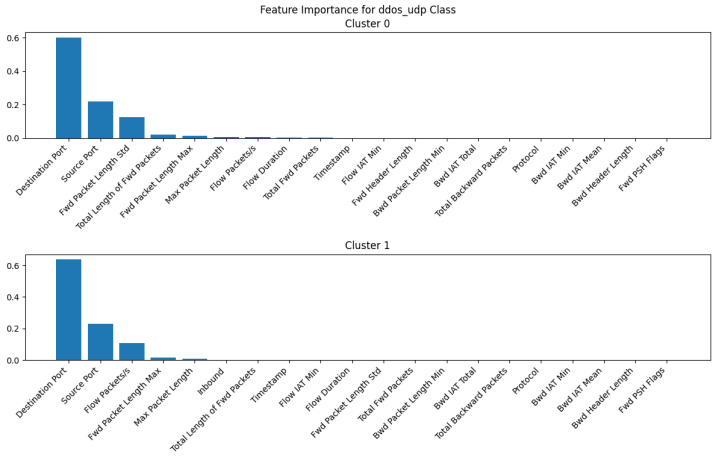


Figure 47: Feature Importance for DDoS UDP

According to our analysis, the Destination and Source Ports are among the most important features in both sub-clusters for **ddos_udp**. This suggests that these two groupings differ in these features, because if they were similar in both groups they would be closer to each other and possibly form only one sub-cluster.

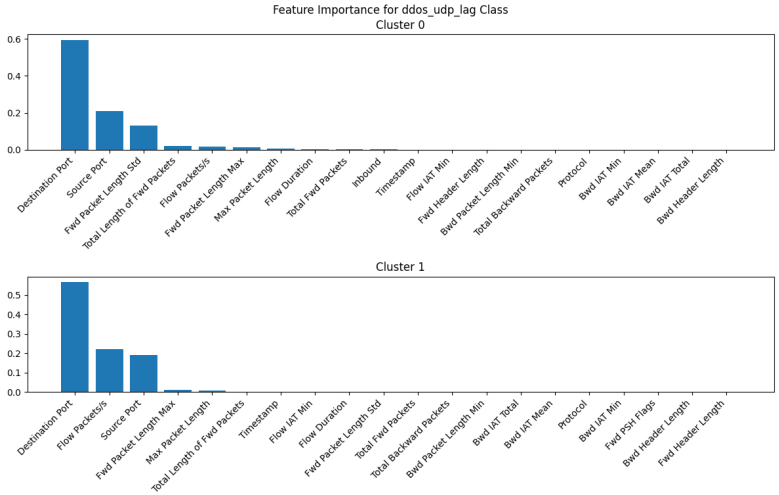


Figure 48: Feature Importance for DDos UDP lag

Even for **ddos_udp_lag**, the Destination and Source Ports are among the most important features in both sub-clusters. This helps us explain the similarities in their spread and also to differentiate between the sub-clusters with the same reasoning as before. It is reasonable that these attacks would exhibit these patterns as they function in a very similar manner to each other.

To conclude this section, although it is very difficult to determine the specific attack type using this method, analyzing the results we got allowed us to notice patterns which could be helpful when attempting to distinguish between benign and malicious traffic, and even between different types of attacks.