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

Politecnico di Torino

1 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.

The following types of DDoS attacks, alongside benign traffic, are discussed:

- **DDoS UDP LAG:** Involves flooding a targeted server with a significant volume of UDP packets.
- **DDoS SNMP:** Utilizes spoofed IP addresses to generate requests, causing a surge of SNMP responses to overwhelm the target.
- **DDoS SSDP:** Exploits Universal Plug and Play (UPnP) protocols to launch attacks.
- **DDoS LDAP:** Sends small, malicious queries to publicly accessible LDAP servers, creating an overload through their responses.
- **DDoS MSSQL:** Leverages vulnerabilities in Microsoft SQL protocols to execute reflection-based attacks.
- **DDoS UDP:** Bombards the victim's server with UDP packets from numerous IP addresses.
- **DDoS NETBIOS:** Renders the target system incapable of communicating with other NetBIOS hosts.
- **DDoS SYN:** Sends an overwhelming number of spoofed SYN requests with forged IP addresses to the target system.
- **DDoS DNS:** Exploits DNS servers by crafting queries that trigger responses containing excessive data.

- **DDoS TFTP:** Abuses TFTP servers by making default file requests, causing the server to transmit large volumes of data even when file names do not match.
- **DDoS NTP:** Overwhelms the target by generating fake NTP requests from numerous IP addresses.

These attack types highlight the diverse techniques cybercriminals use to incapacitate systems and networks, emphasizing the need for robust security measures to mitigate their impact.

2 Data exploration and pre-processing

Data exploration and pre-processing are fundamental phases in the data analysis pipeline. These steps focus on understanding, cleaning, transforming, and organizing raw data to prepare it for subsequent analysis. Data exploration involves summarizing the main characteristics of the data, identifying patterns, spotting anomalies, and visualizing distributions to gain a deeper understanding of its structure. Pre-processing, on the other hand, includes essential tasks such as handling missing values, removing duplicates, encoding categorical variables, scaling numerical features, and normalizing data to ensure it is in the optimal format for analysis.

Performing these tasks is vital for deriving valuable insights, improving data quality, and optimizing the performance of machine learning models. A well-prepared dataset reduces noise, eliminates inconsistencies, and allows models to generalize better, leading to more accurate predictions and reliable outcomes.

2.1 Data Cleaning

The provided dataset, named `ddos_df` in the Python source code, contains traffic traces collected during the execution of common DDoS attacks. The dataset spans a specific time window and includes network traffic data for analysis.

The dataset contains **64,239 rows (objects)** and **87 columns (features)**. Each row represents a flow, which is defined as a sequence of packets exchanged between a source IP and a destination IP. The dataset is classified into **12 labels**, where 11 labels correspond to different types of DDoS attacks, and 1 label represents benign flows that are not part of any attack.

Upon analyzing the dataset, it was observed that there are no null features. However, **12 features** were found to have unique values across all

rows. These features were removed because they do not contribute to the analysis as their standard deviation is equal to 0, indicating no variation. After removing these columns, the shape of the dataset was reduced to **(64,239, 75)**.

2.2 Label Analysis

Following a comprehensive dataset analysis, attention has been directed towards characterizing the behavior of the labels. Upon aggregating the dataset into two categories, **malicious** and **benign** traffic, it becomes evident that the malicious category constitutes a predominant portion, accounting for **91.2%** of the dataset, while benign traffic represents only **8.8%**.

The bar plot in Figure 1 visually confirms this imbalance, showing the significant dominance of malicious traffic flows over benign ones. Specifically, **58,581 flows** belong to the malicious category, while only **5,658 flows** are benign.

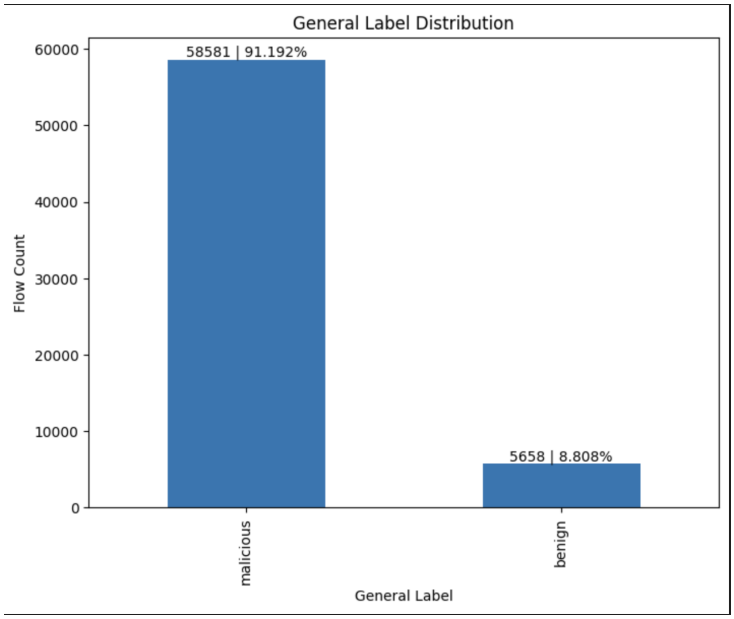

Figure 1: General Label Distribution

Subsequently, a focused examination of attack frequencies reveals the most prevalent types of DDoS attacks. The distribution of attack labels, as shown in Figure 2, provides detailed insights into the counts and proportions of each attack type. Among the attacks, the most frequent are *ddos_udp_lag*, *ddos_snmp*, and *ddos_ssdp*, each contributing approximately **9.3%** of the

dataset. Other attacks, such as *ddos_ldap*, *ddos_mssql*, and *ddos_udp*, also exhibit similar frequencies, ranging from **9.2%** to **9.1%**.

However, some attack types occur far less frequently. Notably, *ddos_ntp* is the least common attack, contributing only **1.5%** of the dataset with a total of **986 flows**. This disparity in attack frequencies is significant and highlights the varying levels of prominence among different DDoS attack types.

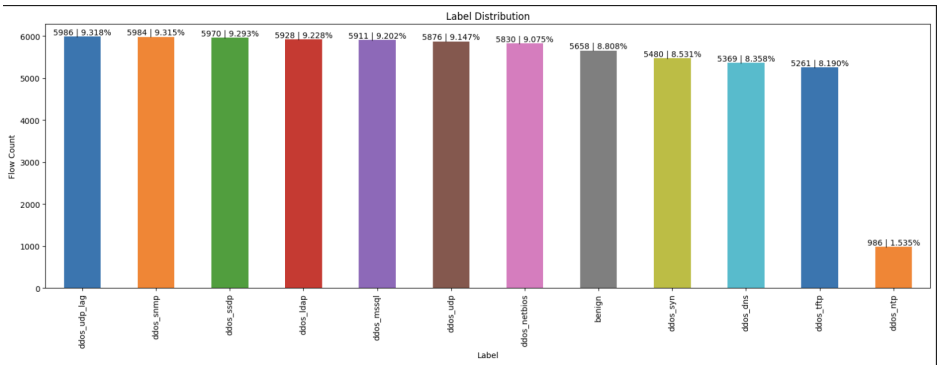

Figure 2: Detailed Label Distribution

Overall, the label analysis demonstrates that while the majority of the dataset consists of malicious flows, the distribution of attack types is diverse. This comprehensive understanding of label behavior provides a solid foundation for subsequent machine learning tasks, such as classification and anomaly detection.

2.3 Traffic Level

To gain a deeper understanding of how traffic is distributed within the dataset, a thorough analysis was conducted. This analysis focused on various aspects such as protocols and the distribution of ports.

To improve interpretability, a *protocol mapping* technique was applied, converting numerical protocol codes into their corresponding protocol names. The resulting distribution of protocols was visually represented, highlighting that UDP is the most frequently used protocol, accounting for 76.1 as we can see in Figure 3

A comprehensive analysis of the traffic flow over time was also performed to identify patterns and anomalies. The time axis was divided into fixed 30-second intervals to examine the number of traffic flows within each slot. The resulting plot revealed multiple peaks, each representing periods of heightened activity. These peaks are significant as they indicate short bursts of

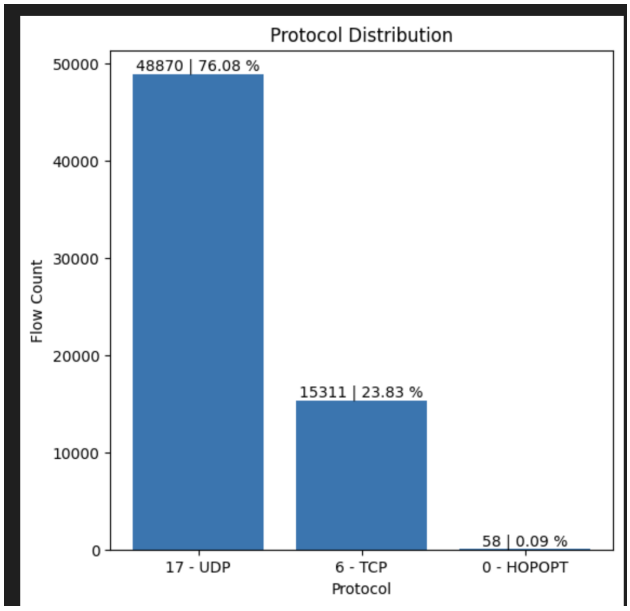

Figure 3: Protocol Distribution

traffic, which can often be attributed to specific network events or malicious activities such as Distributed Denial of Service (DDoS) attacks. DDoS attacks are characterized by a sudden and intense increase in flow volume over short durations, overwhelming network resources and impacting service availability. By analyzing these traffic surges, potential attack patterns can be recognized, enabling better detection and mitigation strategies for such threats. 4 The figure 5 presents the flow count over time for different la-

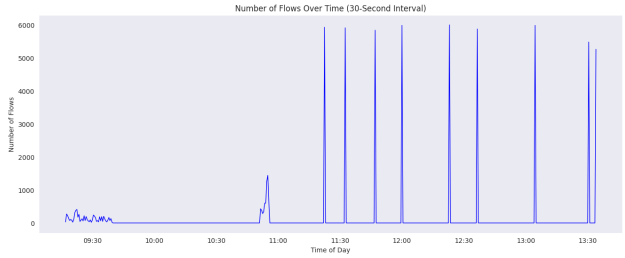

Figure 4: Number of Flows Over Time (30-Second Interval)

bels. Each subplot corresponds to a specific label (e.g., *ddos_dns*, *benign*, *ddos_ldap*, etc.), illustrating how flows are distributed across various categories. By analyzing the peaks and patterns in each label, we can observe that certain DDoS attack types, such as *ddos_ldap*, *ddos_snmp*, and *ddos_udp*, exhibit sharp, short bursts of activity. These bursts align with typical DDoS

attack behavior, where a sudden surge of flows occurs within a short time window.

Conversely, the *benign* traffic shows a more gradual and scattered flow pattern without clear peaks, differentiating it from the structured nature of DDoS-related traffic. This distinction allows for a more detailed understanding of traffic patterns and potential attack detection based on label-specific flow behavior.

Figure 5: Number of Flows Over Time for Each Label

2.3.1 Port Analysis

To provide a deeper insight into traffic behavior, an analysis of the source ports across different attack types was conducted using a boxplot. The figure below 6 illustrates the distribution of source ports for various labels, including 'benign' traffic and multiple DDoS attack types such as 'ddos_dns', 'ddos_udp', 'ddos_syn', and others.

The analysis reveals that benign traffic tends to utilize a wider range of ports, as indicated by the higher median and broader interquartile range. This suggests that benign traffic patterns are more diverse and less predictable.

In contrast, DDoS attacks exhibit more concentrated port usage. Specifically:

- **UDP-based attacks** (e.g., *ddos_udp*, *ddos_udp_lag*, *ddos_syn*) show higher source port numbers, reflecting their nature of overwhelming the target with UDP packets.
- **TCP-based attacks** (e.g., *ddos_dns*, *ddos_ldap*, *ddos_mssql*, *ddos_netbios*) demonstrate lower source port numbers, which is characteristic of attacks relying on TCP SYN floods to overwhelm network connections.

This distinction between benign and DDoS traffic provides valuable insights for anomaly detection, as the concentration of port usage can serve as an indicator of malicious activities.

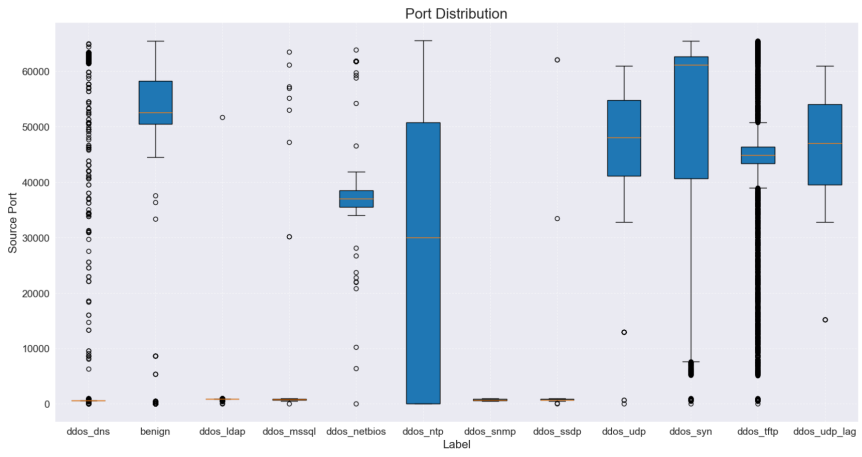

Figure 6: Port Distribution Across Different Labels

The figure 7 below presents the Empirical Cumulative Distribution Function (ECDF) of source ports. The ECDF is a statistical measure that shows

the cumulative probability of observing a particular source port value or smaller within the dataset.

From the graph, we observe the following key insights:

- There is a steep increase at the lower source port range (0–1024), indicating that a significant portion of the traffic uses well-known ports reserved for system processes and standard services.
- The middle range of ports (10,000–30,000) shows a gradual and steady increase, representing less frequent usage of ephemeral ports often allocated dynamically.
- The upper range (above 40,000) displays a more linear increase, suggesting that some traffic, particularly from benign or UDP-based sources, utilizes higher-numbered ephemeral ports.

This ECDF analysis helps us understand the overall distribution of source ports across all traffic types. The concentration at lower port numbers aligns with DDoS attack patterns where reserved ports are often targeted. Conversely, the smoother growth at higher port ranges is consistent with benign traffic behavior or attacks using high-entropy source ports to evade detection.

Such insights are valuable for identifying anomalies in port usage and detecting traffic deviations that may indicate malicious activities.

2.3.2 Destination Port Analysis

The destination ports were analyzed to understand how traffic is distributed across various labels. The boxplot 8 and ECDF 9 graphs below provide complementary insights into the distribution and cumulative probability of destination ports.

Boxplot Insights:

- The destination port usage for benign traffic is highly concentrated around a limited range of ports (0–1024), as seen in the presence of outliers and a narrow interquartile range. This reflects standard network activity where well-known ports are frequently used for communication.
- DDoS attack types, such as *ddos_ldap*, *ddos_udp*, and *ddos_udp_lag*, show a broader range of destination ports with higher median values. This pattern is indicative of attacks targeting ephemeral ports or dynamically allocated ports to overwhelm network services.
- The *ddos_ntp* label stands out with destination ports clustered around the lower ranges, likely targeting specific protocols for exploitation.

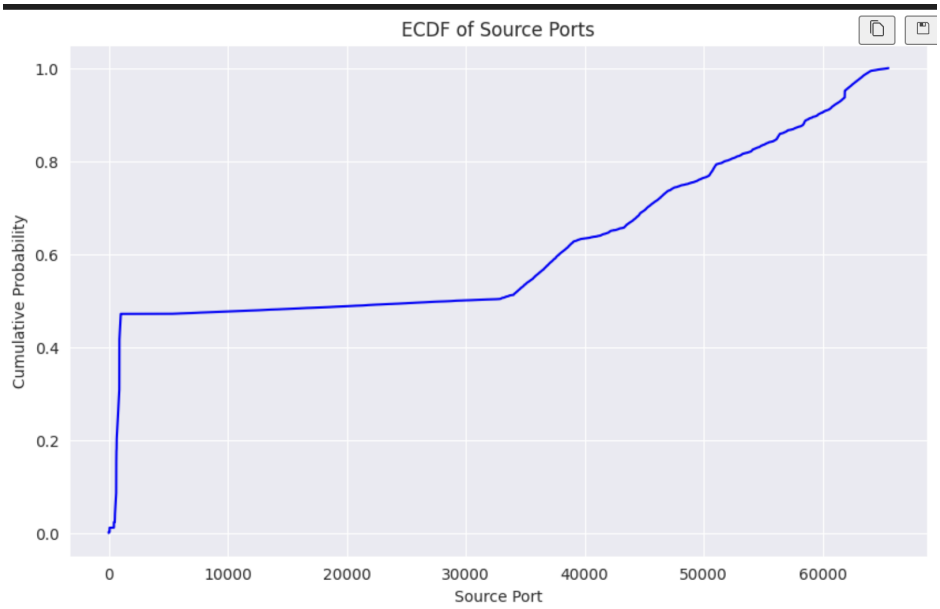

Figure 7: Empirical Cumulative Distribution Function (ECDF) of Source Ports

- Other labels, such as *ddos_snmp* and *ddos_syn*, exhibit a consistent spread across destination ports, highlighting diverse targeting strategies.

ECDF Insights:

- The ECDF graph for destination ports reveals a near-linear increase, indicating that destination ports are more evenly distributed across the full range (0–65,535) compared to source ports.
- The cumulative probability shows a gradual growth, reflecting that a significant portion of traffic is spread over higher port numbers, aligning with both benign and attack-related traffic patterns.
- The smooth progression suggests that while certain attack types focus on well-known ports, others exploit a broader range of ephemeral ports, blending with legitimate network traffic.

These findings demonstrate that DDoS attacks can target both specific low-numbered ports and higher-numbered ephemeral ports, depending on the attack strategy. Understanding this behavior allows for improved anomaly detection and mitigation strategies.

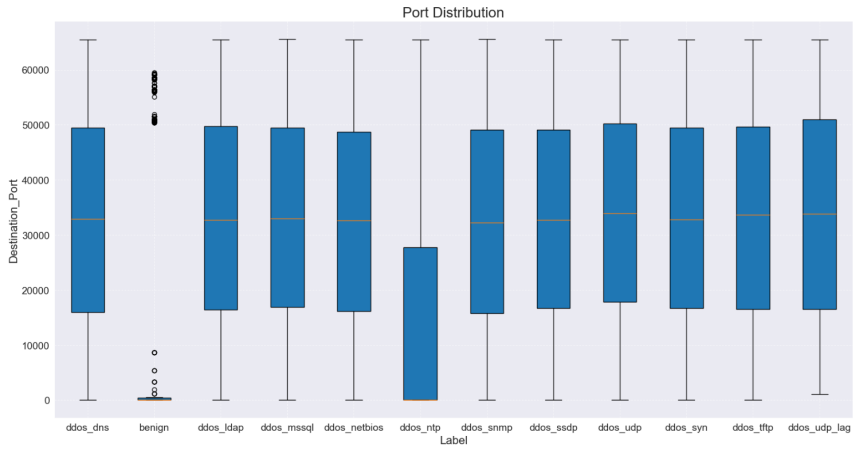

Figure 8: Boxplot of Destination Ports Across Different Labels

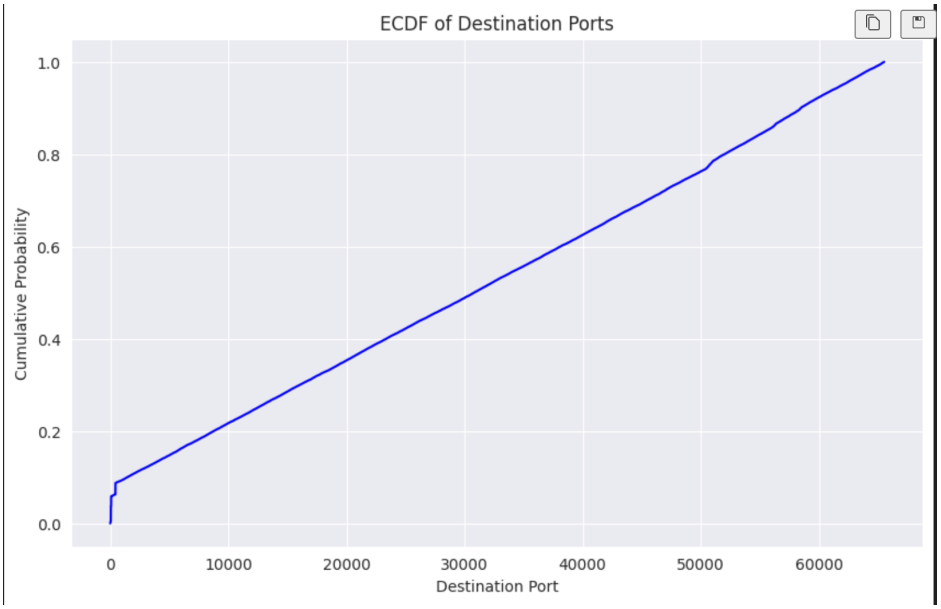

Figure 9: ECDF of Destination Ports

2.4 Correlation Analysis

Following the evaluation of network traffic, the next step involves conducting a correlation analysis on the dataset's features. To facilitate this process, preliminary steps were carried out, including converting categorical features into numerical values. The timestamp data was standardized by transforming it into integer values, representing time in seconds relative to a predefined reference date ('2018-12-01, 10:51:39.813448').

To prepare for the standardization process, the dataset required some adjustments, including handling features with varying measurement units. Afterward, a new standardized DataFrame, referred to as `ddos_df`, was created.

The correlation matrix (Figure 10) reveals that many features exhibit significant interdependence. This observation highlights the relationships and interconnectedness among specific attributes in the dataset.

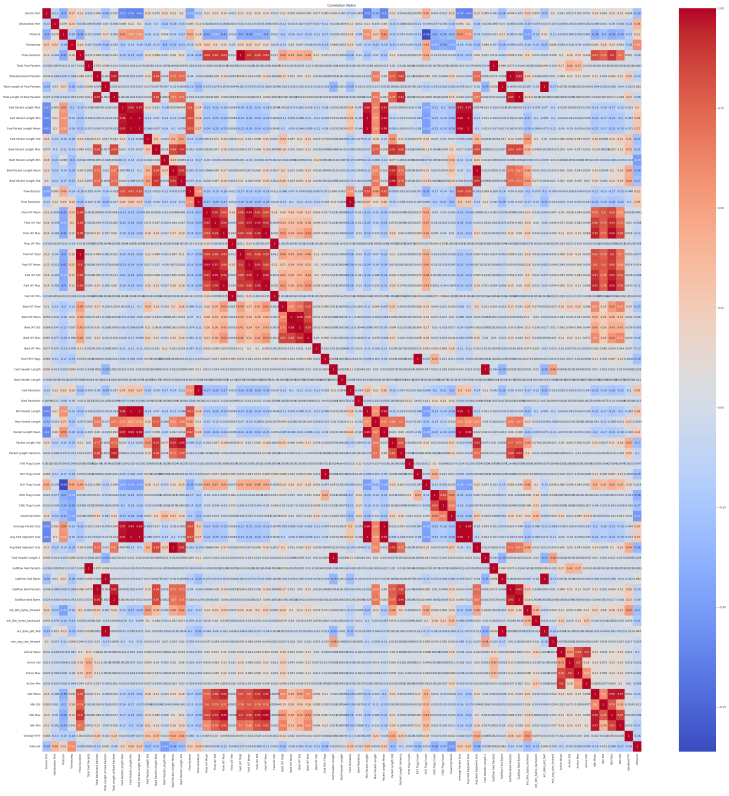

Figure 10: Correlation Matrix Before Feature Reduction

Upon examining the correlation matrix, it was observed that features with a high correlation coefficient (above 0.8) needed to be removed. This

thresholding process resulted in the elimination of 39 features, reducing the total number of features from 71 to 32. A refined DataFrame, referred to as `new_ddos_df`, was generated, containing only the most relevant and distinct attributes. This reduction not only improves computational efficiency but also ensures the retention of meaningful information.

Based on the refined DataFrame, a new correlation matrix was calculated (Figure 11), which reflects the relationships among the remaining features more effectively.

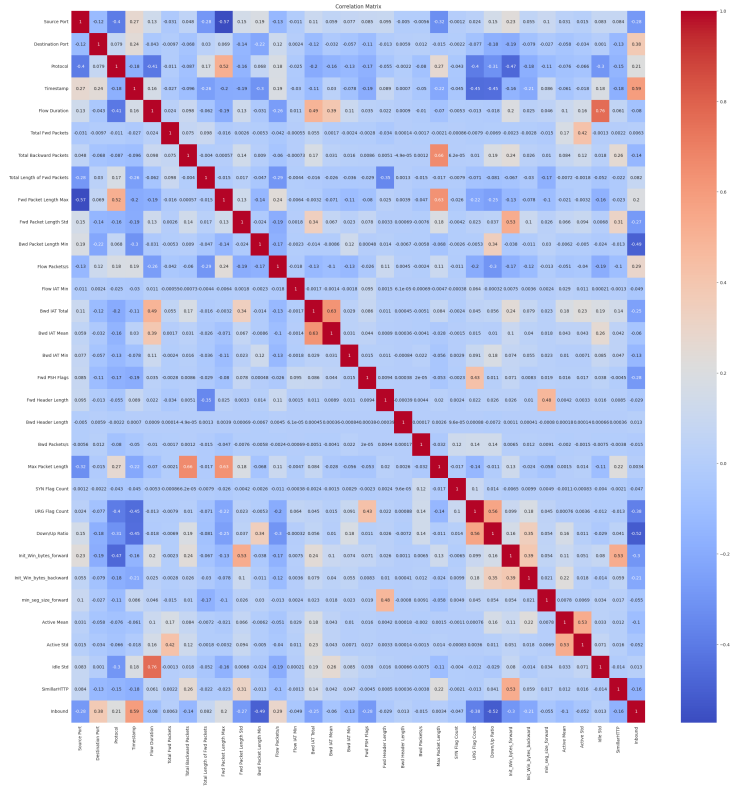

Figure 11: Correlation Matrix After Feature Reduction

2.4.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a critical dimensionality reduction technique utilized to transform high-dimensional datasets into a lower-dimensional space while retaining as much of the original variance as possible. In this analysis, the PCA was applied to the scaled dataset (`new_ddos_df`), and Kaiser's rule was employed to identify the optimal number of principal components to retain. Kaiser's rule recommends retaining components with eigenvalues greater than 1, as they contribute more variance than any individual feature.

The cumulative explained variance curve (Figure 12) provides insights into the variance captured by the principal components. According to the results, the first 12 principal components account for 68.38% of the total variance, indicating significant dimensionality reduction with minimal information loss. The vertical red dashed line, representing Kaiser's rule, confirms that 12 components are sufficient to effectively represent the dataset.

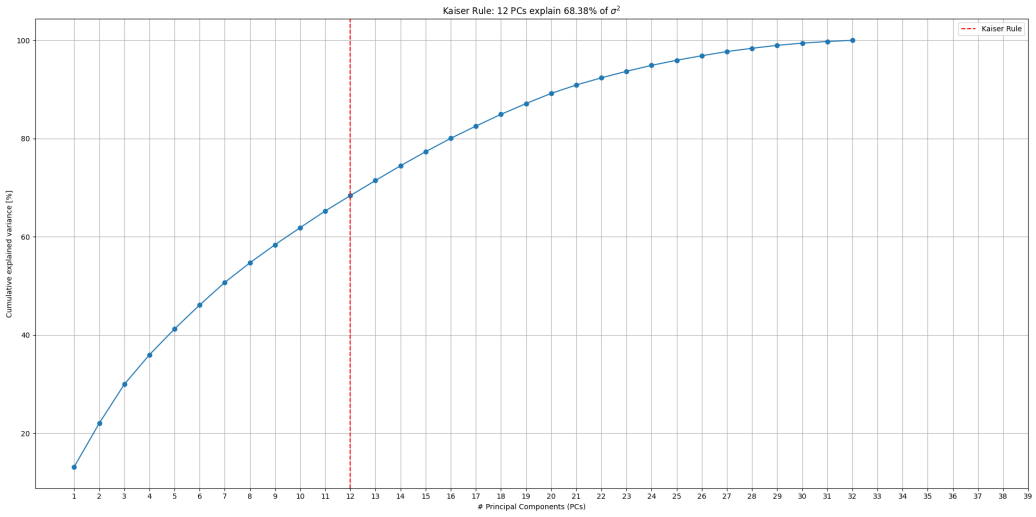

Figure 12: Principal Component Analysis

This analysis demonstrates the effectiveness of PCA in simplifying complex datasets while maintaining a balance between dimensionality and retained variance. By reducing the dataset to 12 principal components, computational efficiency is enhanced without sacrificing the dataset's critical structure and relationships.

Since principal components are linear combinations of the original features, with the first few capturing the most significant variance in the data, it becomes insightful to examine their composition. Highlighting the key

features that contribute to these principal components, along with their corresponding weights, offers valuable perspectives on their relevance and influence.

The tables below illustrate the most impactful features for each principal component, accompanied by their respective weights, providing a deeper understanding of the underlying structure and relationships within the dataset.

```
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

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

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
```

Top 5 Features for PC1

After thorough deliberation, the decision has been taken to utilize the reduced DataFrame (new_ddos_df) in the subsequent analytical processes rather than relying on the Principal Components (PCs). This strategic choice stems from the aspiration to prioritize the interpretability and transparency of the results. While PCs effectively encapsulate critical patterns and variance within the dataset, their abstract nature and complex representation often hinder straightforward visualization and meaningful interpretation. By opting for the reduced DataFrame, the analysis retains a direct connection to the original features, allowing for a more comprehensible and contextually grounded exploration of the dataset. This approach ensures that the insights derived are not only statistically robust but also accessible and actionable for a broader audience.

3 Supervised Learning - Classification

In this section, a supervised learning approach is employed to categorize the various network flows based on their corresponding attack types. The methodology commences with the systematic partitioning of the dataset into distinct subsets for training (X) and testing (X_{test}) purposes. This stratified split is carefully executed to preserve the proportional distribution of attack categories across both subsets, thereby ensuring a representative sampling of the data.

The dataset is divided following a predefined ratio, with 70% of the data allocated to the training set (X) and the remaining 30% reserved for the testing set (X_{test}). To enhance model robustness and prevent overfitting, the

training set (X) undergoes further subdivision into two additional subsets: a training subset (X_{train}) and a validation subset (X_{val}). This second split is also stratified, ensuring that the relative frequencies of attack classes are preserved. The final proportions ensure that the validation set occupies a fraction of the training set while maintaining statistical consistency.

The following code snippet demonstrates the implementation of these steps using Python. The initial split divides the dataset into training and testing sets, followed by the subdivision of the training data into training and validation subsets:

```
X, X_test, y, y_test = train_test_split(new_ddos_df, 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 13: Code snippet for dataset splitting.

This stratified partitioning approach is critical for maintaining the integrity of the data distribution across subsets, ensuring that the model is trained and evaluated on representative samples. By preserving the class proportions throughout the splits, this methodology minimizes the risk of bias and enhances the model's ability to generalize effectively to unseen data.

3.1 Supervised Learning Algorithms

The classification task will be carried out utilizing three distinct supervised learning algorithms, each offering unique characteristics and methodologies for addressing the problem:

- **Random Forest (RF)**: A robust ensemble learning method that constructs multiple decision trees during training and combines their outputs for improved accuracy and stability. It excels in handling high-dimensional datasets and reducing the risk of overfitting by averaging predictions.
- **Support Vector Machine (SVM)**: A powerful algorithm designed to find the optimal hyperplane that separates data points belonging to different classes. SVMs are particularly effective in scenarios involving high-dimensional spaces or when the number of features exceeds the number of data points.

- **K-Nearest Neighbors (K-NN)**: A simple yet effective instance-based learning algorithm that classifies a data point based on the majority class of its k nearest neighbors. The performance of K-NN is influenced by the choice of k and the distance metric used.

For each of these algorithms, the default hyperparameter settings are employed to simplify the implementation process. The effectiveness of the models is quantitatively assessed using a comprehensive set of performance metrics, including the confusion matrix, precision, recall, and F1 score. These metrics provide valuable insights into the accuracy, reliability, and overall effectiveness of the classification process, enabling an informed comparison of the algorithms' strengths and weaknesses.

By leveraging these diverse algorithms, the study aims to evaluate their respective capabilities in accurately classifying network flows and detecting potential attack patterns.

3.1.1 Random Forest

The code snippet below demonstrates the initialization, training, and prediction process:

```
# Initialize the Random Forest Classifier
random_forest_classifier = RandomForestClassifier()

# Train the classifier on the training set
random_forest_classifier.fit(X_train, y_train)

# Predict on the training set
train_predictions_rf = random_forest_classifier.predict(X_train)

# Predict on the validation set
validation_predictions_rf = random_forest_classifier.predict(X_val)
```

Figure 14: Code for Random Forest Classifier implementation.

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

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 correctly classify nearly all samples.

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	0.97	0.98	0.98	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).

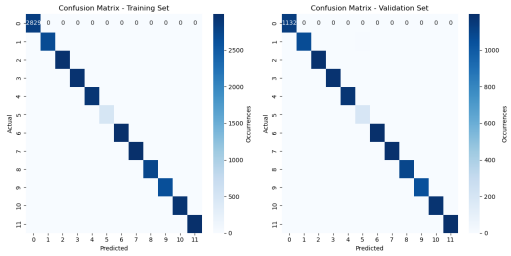

Figure 16: Confusion matrices for the Random Forest model. (Left: Training Set, Right: Validation Set).

3.1.2 K-Nearest Neighbour

The K-Nearest Neighbour (KNN) algorithm is employed for the classification task. The code snippet below demonstrates the initialization, training, and prediction process:

After the fit and predict operations, 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.

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.

```python
# Initialize the K-Nearest Neighbors Classifier
knn_classifier = KNeighborsClassifier()

# Train the classifier on the training set
knn_classifier.fit(X_train, y_train)

# Predict on the training set
train_predictions_knn = knn_classifier.predict(X_train)

# Predict on the validation set
validation_predictions_knn = knn_classifier.predict(X_val)
```

Figure 17: Code for K-Nearest Neighbour implementation.

Classification Report - Training Set

	precision	recall	f1-score	support
0	0.97	0.98	0.97	2829
1	0.98	0.94	0.96	2684
2	0.93	0.96	0.94	2963
3	0.95	0.95	0.95	2956
4	0.99	0.99	0.99	2915
5	0.87	0.85	0.86	493
6	0.99	0.99	0.99	2992
7	0.99	0.98	0.99	2985
8	0.98	0.98	0.98	2740
9	0.97	0.98	0.98	2631
10	0.92	0.91	0.92	2938
11	0.92	0.92	0.92	2993
accuracy			0.96	32119
macro avg	0.96	0.95	0.95	32119
weighted avg	0.96	0.96	0.96	32119

Classification Report - Validation Set

	precision	recall	f1-score	support
0	0.94	0.97	0.96	1132
1	0.96	0.92	0.94	1074
2	0.91	0.94	0.92	1186
3	0.92	0.93	0.93	1182
4	0.98	0.99	0.99	1166
5	0.81	0.73	0.77	197
6	0.99	0.99	0.99	1197
7	0.99	0.97	0.98	1194
8	0.99	0.98	0.98	1096
9	0.98	0.97	0.97	1052
10	0.84	0.82	0.83	1175
11	0.83	0.86	0.84	1197
accuracy			0.94	12848
macro avg	0.93	0.92	0.93	12848
weighted avg	0.94	0.94	0.94	12848

Figure 18: Classification reports for the K-Nearest Neighbour model. (Left: Training Set, Right: Validation Set).

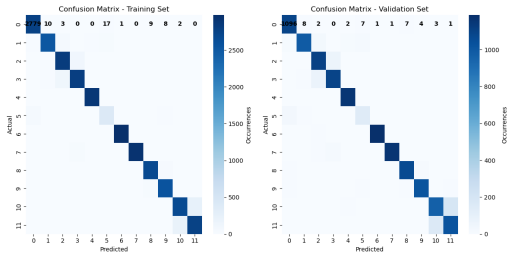

Figure 19: Confusion matrices for the K-Nearest Neighbour model. (Left: Training Set, Right: Validation Set).