VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI - 590018

Python Project Report (21CSE145)
On

DDos Detection Using Machine Learning Algorithms

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by

1BG21CS001 Abhiram B S
1BG21CS017 B S Sumanth
1BG21CS044 Kushal R

Internal Guide
Smt. Pallavi C V
Assistant Professor, Dept. of CSE
BNMIT, Bengaluru

Vidyayāmruthamashnuthe

B.N.M. Institute of Technology

An Autonomous Institution under VTU
Approved by AICTE, Accredited as grade A Institution by NAAC. All eligible branches – CSE, ECE, EEE, ISE & Mech. Engg. are Accredited by NBA for academic years 2018-19 to 2024-25 & valid upto 30.06.2025
URL: www.bnmit.org

Department of Computer Science and Engineering
2022 - 2023

B.N.M. Institute of Technology

An Autonomous Institution under VTU

Approved by AICTE, Accredited as grade A Institution by NAAC. All eligible branches – CSE, ECE, EEE, ISE & Mech. Engg. are Accredited by NBA for academic years 2018-19 to 2024-25 & valid upto 30.06.2025

URL: www.bnmit.org

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the Python project entitled **DDos Detection Using Machine Learning Algorithms** carried out by Mr. **ABHIRAM B S (1BG21CS001)**, Mr. **B S SUMANTH (1BG21CS017)**, Mr. **KUSHAL R (1BG21CS044)**, are bonafide students of IV Semester, BNM Institute of Technology in partial fulfillment for the award of Bachelor of Engineering in COMPUTER SCIENCE AND ENGINEERING of Visvesvaraya Technological University, Belagavi during the year 2022-23. It is certified that all corrections / suggestions indicated for Internal Assessment have been incorporated in the project report deposited in the departmental library. The Python project report (21CSE145) has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

Smt. Pallavi C V
Assistant Professor
Department of CSE
BNMIT, Bengaluru

Dr. Chayadevi M L
Professor & HOD
Department of CSE
BNMIT, Bengaluru

	Name	**Signature with Date**
Examiner 1:		
Examiner 2:		

ACKNOWLEDGEMENT

We would like to place on record our sincere thanks and gratitude to the concerned people, whose suggestions and words of encouragement has been valuable.

We express our heartfelt gratitude to the management of **BNM Institute of Technology**, for givingus the opportunity to pursue Degree of Computer Science and Engineering and helping us to shape our career. We take this opportunity to thank **Shri. Narayan Rao R. Maanay**, Secretary, **Prof. T. J. Rama Murthy**, Director, **Dr. S. Y. Kulkarni**, Additional Director, **Prof. Eishwar N Maanay**, Dean and **Dr. Krishnamurthy G. N.**, Principal for their support and encouragement to pursue this project. We would like to thank **Dr. Chayadevi M L**, Professor and Head, Dept. of Computer Science and Engineering, for her support and encouragement.

We would like to thank our guide **Smt. Pallavi C V**, Assistant Professor, Dept. of Computer Science and Engineering, who has been the source of inspiration throughout our project work and has provided us with useful information at every stage of our project.

Finally, we are thankful to all the teaching and non-teaching staff of Department of Computer Science and Engineering for their help in the successful completion of our project. Last but not the least we would like to extend our sincere gratitude to our parents and all our friends who were a constant source of inspiration.

1BG21CS001 Abhiram B S
1BG21CS017 B S Sumanth
1BG21CS044 Kushal R

ABSTRACT

In today's digital landscape, where connectivity and reliance on online services are paramount, the threat posed by cyberattacks has reached unprecedented levels. Among these threats, Distributed Denial of Service (DDoS) attacks stand out as a significant concern for organizations and individuals alike. These attacks can disrupt the availability of critical online services, causing substantial financial losses and reputational damage. To combat this evolving threat, the integration of machine learning algorithms for DDoS attack detection has emerged as a promising avenue.

At its core, a Denial of Service (DoS) attack involves overwhelming a target system, such as a web server or network, with an excessive volume of requests, rendering the system inaccessible to legitimate users. This form of attack exploits the finite resources of the targeted system, causing temporary unavailability. In contrast, a Distributed Denial of Service (DDoS) attack takes this malevolent strategy to a new level. By orchestrating a network of compromised devices, the attacker magnifies their impact, creating a surge of malicious traffic that can overpower even robust infrastructure.

In the realm of network security, the detection and mitigation of Distributed Denial of Service (DDoS) attacks represent a pivotal challenge. Conventional approaches relying on rule-based and signature-based methods are proving inadequate in the face of evolving attack strategies. Their rigidity hinders adaptation to emergent attack patterns, resulting in elevated rates of false positives and a deficiency in real-time responsiveness. The requisite manual intervention exacerbates the issue, culminating in delayed detection and response times.

The project "DDoS Detection using Machine Learning" focuses on developing an advanced system to effectively detect and classify Distributed Denial of Service (DDoS) attacks within network traffic. The project employs various machine learning algorithms including Logistic Regression, Support Vector Machine (SVM), Random Forest, and Gradient Boosting, and evaluates their performance against a dataset of network traffic features. The dataset is extensively analyzed and visualized to understand its characteristics. Each model is trained, optimized, and rigorously tested to determine its accuracy, precision, recall, and F1-score. The outcomes highlight the superiority of the Random Forest model with a perfect accuracy of 100%, while SVM and Gradient Boosting also demonstrate robust performance. The study emphasizes the significance of comprehensive evaluation metrics in selecting the optimal model for DDoS detection in real-world network security scenarios.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER – I
INTRODUCTION

CHAPTER-I

Introduction

In today's digital landscape, where connectivity and reliance on online services are paramount, the threat posed by cyberattacks has reached unprecedented levels. Among these threats, Distributed Denial of Service (DDoS) attacks stand out as a significant concern for organizations and individuals alike. These attacks can disrupt the availability of critical online services, causing substantial financial losses and reputational damage. To combat this evolving threat, the integration of machine learning algorithms for DDoS attack detection has emerged as a promising avenue.

- **Understanding DoS and DDoS Attacks:**

  At its core, a Denial of Service (DoS) attack involves overwhelming a target system, such as a web server or network, with an excessive volume of requests, rendering the system inaccessible to legitimate users. This form of attack exploits the finite resources of the targeted system, causing temporary unavailability. In contrast, a Distributed Denial of Service (DDoS) attack takes this malevolent strategy to a new level. By orchestrating a network of compromised devices, the attacker magnifies their impact, creating a surge of malicious traffic that can overpower even robust infrastructure.

- **Major Threats and Losses Due to DDoS Attacks:**

  DDoS attacks are not merely technical nuisances; they have far-reaching implications across various sectors:

- **Service Disruption:** DDoS attacks have the potential to disrupt online services, causing inconvenience to users and affecting business operations. This disruption could range from intermittent slowdowns to complete unavailability.
- **Financial Impacts:** Beyond operational disturbances, DDoS attacks lead to direct financial losses due to reduced revenue during downtime and the resources required for mitigation efforts.
- **Reputation Erosion:** Prolonged service unavailability can inflict severe damage to an organization's reputation. In an era of instant communication, negative experiences spread quickly, eroding customer trust and loyalty.

- **Cybercrime Diversions:** DDoS attacks are often used as smokescreens to distract security teams from other malicious activities, such as data breaches or malware injections.

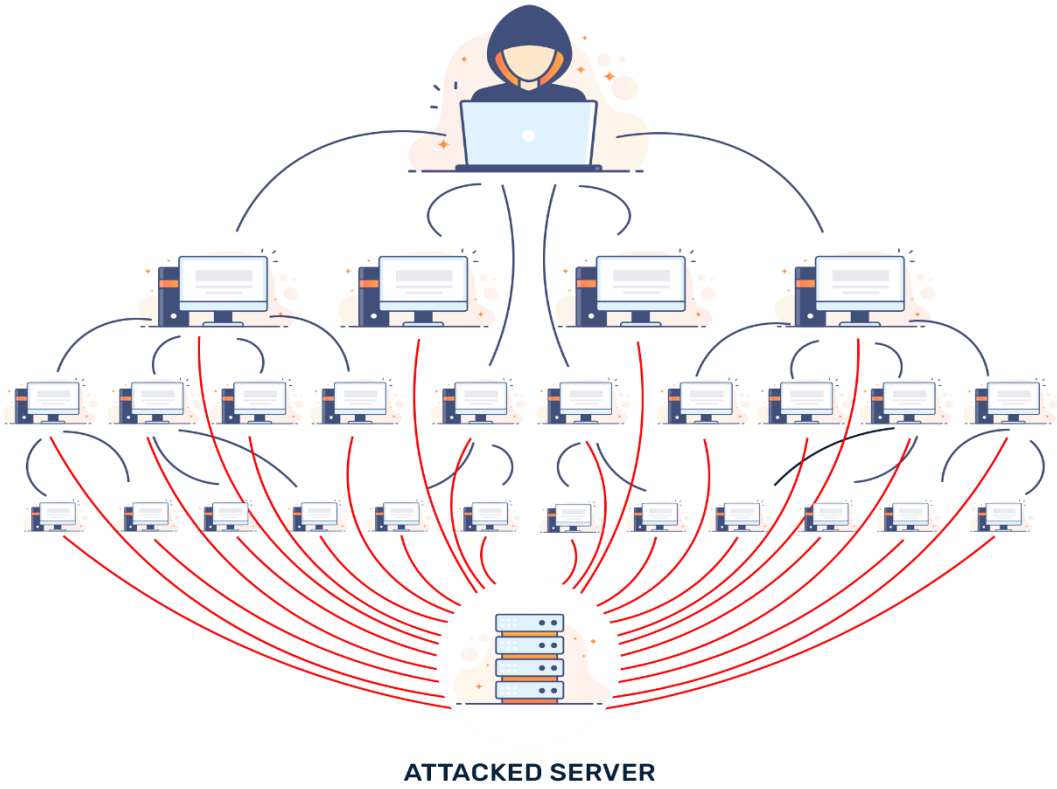


Figure 1.1: DDoS Attack

- **Cost Estimation of Loss Due to DDoS Attacks**

DDoS attacks inflict substantial financial losses on organizations across industries, ranging from small businesses to large enterprises. The magnitude of these losses is influenced by various factors, including the duration and intensity of the attack, the type of services affected, and the overall business impact. While precise estimations can be challenging due to the diverse nature of attacks and the wide range of businesses, a general overview of the potential costs can be provided:

- **Lost Revenue:** DDoS attacks can result in hours or even days of service disruption, causing potential customers to abandon transactions or switch to competitors. This leads to direct revenue losses that can range from hundreds to millions of dollars, depending on the size of the business and its online operations.
- **Operational Costs:** Organizations often need to allocate resources for immediate mitigation and recovery efforts during and after a DDoS attack. These costs include expenses related to hiring cybersecurity experts, deploying additional infrastructure, and investing in advanced security solutions.

- **Downtime Impact:** The longer a DDoS attack persists, the greater the impact on productivity and overall business operations. Extended downtime can affect employee productivity, delay critical projects, and disrupt supply chain processes.
- **Reputational Damage:** A tarnished reputation can have lasting effects on customer trust and loyalty. The negative publicity resulting from a successful DDoS attack can lead to customer attrition, increased customer acquisition costs, and diminished brand value.
- **Mitigation Services:** Many organizations turn to third-party DDoS mitigation services to help protect their infrastructure during an attack. These services come at a cost, and the fees can vary depending on the level of protection required.
- **Investment in Preventive Measures:** Organizations often invest in upgrading their cybersecurity infrastructure to prevent future attacks. This includes implementing advanced firewall solutions, intrusion detection systems, and other security measures.

Figure 1.2: DDoS Attack Cost Estimation of Loss

The total cost of losses due to a DDoS attack can vary widely, potentially ranging from thousands to millions of dollars for larger businesses. Smaller businesses might also experience significant financial strain, as they may lack the resources to recover quickly. Additionally, the intangible costs related to reputational damage and customer trust erosion can have long-lasting effects that are challenging to quantify accurately.

It is important to note that the costs outlined above are estimates and can vary based on factors such as the industry, geographical location, and the organization's overall preparedness for handling cyber threats. Organizations should proactively invest in robust cybersecurity measures, including DDoS attack detection and mitigation strategies, to minimize potential losses and protect their digital assets.

1.1 Statement of the Problem

In the realm of network security, the detection and mitigation of Distributed Denial of Service (DDoS) attacks represent a pivotal challenge. Conventional approaches relying on rule-based and signature-based methods are proving inadequate in the face of evolving attack strategies. Their rigidity hinders adaptation to emergent attack patterns, resulting in elevated rates of false positives and a deficiency in real-time responsiveness. The requisite manual intervention exacerbates the issue, culminating in delayed detection and response times.

Addressing these shortcomings is imperative, ushering in the demand for a more robust and adaptive DDoS detection system. The objective is to harness the power of machine learning algorithms to cultivate an environment capable of effectively identifying and mitigating DDoS attacks. By exploiting the inherent capabilities of machine learning, this system aims to surpass the limitations of its predecessors, ensuring swift and accurate detection, while pre-emptively adapting to the dynamic nature of DDoS attack methodologies.

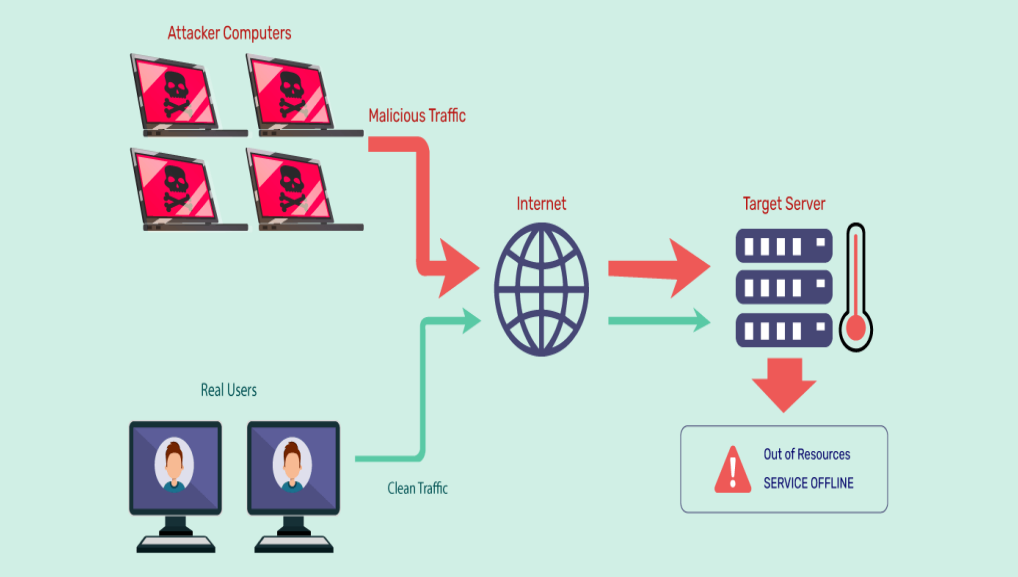


Figure 1.1.1: DDoS Attack on the servers

1.2 Objectives of the project

- **Data Collection and Pre-processing:** Source a comprehensive and representative dataset containing instances of normal network traffic as well as various types of DDoS attacks. Pre-process and cleanse the data to ensure consistency and reliability for subsequent analysis.

- **Algorithm Selection and Implementation:** Delve into the landscape of machine learning algorithms and choose those that are well-suited for binary classification tasks. For this project, we explore the applicability of Logistic Regression, Support Vector Machine (SVM), Random Forest, and Gradient Boosting techniques.

- **Model Training and Evaluation:** Employ the chosen algorithms to train predictive models using the pre-processed data. Evaluate the models' performance using a suite of evaluation metrics, including accuracy, precision, recall, and F1-score.

- **Comparative Analysis:** Perform an in-depth comparative analysis of the implemented algorithms' effectiveness in detecting DDoS attacks. Understand the trade-offs between different algorithms in terms of accuracy, efficiency, and adaptability to varying attack scenarios.

- **Reduced False Positives:** Design algorithms that minimize false positive rates by discerning between legitimate traffic fluctuations and genuine DDoS attacks. Achieving a higher level of accuracy will alleviate the unnecessary strain on resources caused by false alarms.

CHAPTER – II
LITERATURE SURVEY

CHAPTER-II

Literature Survey

A literature survey is required in building a project that is DDoS Detection Using Machine Learning Algorithms because it helps to identify the existing solutions or similar projects that have been carried out in the past. By studying the experiences of others and keeping up-to-date with the latest trends, developers can make informed decisions about the design, development, and testing of their project. A thorough and systematic literature survey is an essential part of the project development process that can help ensure the project is developed efficiently and effectively, relevant and competitive.

Literature Survey 1:

- In the era of the Internet of Things (IoT), where interconnected devices span across numerous aspects of daily life, cybersecurity stands as a paramount concern. With the vulnerabilities inherent in a diverse array of IoT devices, safeguarding against security threats has become increasingly challenging. Among these threats, Distributed Denial of Service (DDoS) attacks pose a significant menace, targeting critical services that operate online. The ubiquity of online services, including finance, communication, e-commerce, and more, renders these services vulnerable to DDoS attacks, which can cause widespread disruption.
- The paper's primary focus lies in utilizing a Deep Learning model called Long Short-Term Memory (LSTM) for the classification of DDoS attacks. The dataset used for experimentation, known as CICDDoS2019, contains a wide variety of DDoS attack scenarios, enabling the model to learn from diverse attack patterns. LSTM's ability to capture time domain correlations in network traffic data sets it apart as a suitable choice.
- Through the experiments, it was observed that the LSTM model outperformed traditional Machine Learning methods such as K-Nearest Neighbors (KNN) and Artificial Neural Networks (ANN) with an accuracy of approximately 98.6% in classifying DDoS attacks. The model's feature selection and extraction capabilities, combined with its deep architecture, contributed to its superior performance.
- The research's contributions include the proposal of a high-accuracy Deep Learning model for DDoS attack detection, its successful implementation on the CICDDoS2019 dataset, and the demonstration of the LSTM model's potential for incorporation into intrusion detection systems. The findings emphasize the importance of embracing advanced AI techniques, such as Deep Learning, to combat evolving cyber threats effectively.(1)

Literature Survey 2:

- In this research paper, the authors focus on addressing the security challenges of Software-Defined Networks (SDN), specifically Distributed Denial of Service (DDoS) attacks. SDN is a revolutionary network paradigm that offers enhanced control and management of network infrastructure. However, its centralized architecture makes it vulnerable to DDoS attacks, particularly at the control layer. To counter these threats, the authors propose using machine-learning techniques for fast and accurate DDoS attack detection.
- The SDN controller is crucial in this context, acting as the network's operating system and responsible for executing network applications and maintaining services. DDoS attacks aim to overwhelm resources and disrupt network functionality. Due to the unique characteristics of SDN, traditional DDoS detection methods are not directly applicable. The authors propose utilizing machine-learning methods that use historical data to distinguish between normal and attack traffic. They emphasize the importance of selecting relevant features from the dataset to train machine-learning models effectively.
- The paper evaluates different feature selection methods and machine learning classifiers for DDoS detection in SDN. The selected features influence the accuracy of the classification models and the performance of the SDN controller. Various feature selection techniques, such as filter, wrapper, and embedded methods, are employed to identify the optimal subset of features. The authors compare the performance of classifiers like Random Forest (RF), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Naïve Bayes (NB), and Decision Trees (DT) on these selected features.
- The experimental results show that the Random Forest (RF) classifier achieves the highest accuracy of 99.97% using features selected by the Recursive Feature Elimination (RFE) method. This combination of machine learning techniques and feature selection demonstrates effective DDoS attack detection in SDN controllers. The authors also discuss the importance of mitigating the resource consumption of SDN controllers during large-scale network traffic scenarios.
- In conclusion, this research highlights the significance of utilizing machine learning and proper feature selection techniques to detect DDoS attacks in SDN environments. It emphasizes the importance of accurately selecting relevant features to improve the efficiency of detection models, with the Random Forest classifier performing particularly well in this context.(2)

Literature Survey 3:

- This research paper focuses on the detection of Distributed Denial-of-Service (DDoS) attacks, which are increasingly common and harmful in the field of network security. The paper presents a DDoS attack detection method based on machine learning, aiming to improve accuracy in identifying these attacks. It acknowledges the challenges posed by the diversity of DDoS attack modes and varying attack traffic sizes, leading to a lack of highly accurate detection methods.
- The proposed method consists of two main stages: feature extraction and model detection. In the feature extraction phase, attack traffic characteristics are identified by comparing data packets using predefined rules. The paper outlines specific differences in attributes between normal and attack traffic for TCP, UDP, and ICMP protocols. These differences are converted into features that can be used for machine learning.
- For the model detection phase, the extracted features are utilized as input for machine learning. The random forest algorithm is employed to train the attack detection model. Random forests are explained as a powerful ensemble learning method that uses decision trees for classification tasks. The paper discusses the construction process of random forests and highlights their advantages, such as improved prediction accuracy and ease of parallelization.
- To evaluate the proposed method, experiments were conducted using both the random forest model and the Support Vector Machine (SVM) algorithm. The research employed false positive rate, detection rate, and total detection rate as evaluation metrics. The results demonstrated that the random forest-based detection model outperformed the SVM approach in terms of accuracy for TCP, UDP, and ICMP flood attacks. The accuracy of the proposed model increased as the proportion of normal traffic to attack traffic grew.
- In conclusion, the paper introduces a novel approach for DDoS attack detection using machine learning, particularly the random forest algorithm. By extracting attack traffic features and training the model, the method shows promising results in accurately identifying various types of DDoS attacks. The proposed model exhibits better performance compared to the SVM algorithm across different attack scenarios.(3)

Literature Survey 4:

- The paper focuses on addressing the significant threat posed by Distributed Denial of Service (DDoS) attacks in the realm of cybersecurity. DDoS attacks are a form of cyberwarfare wherein malicious actors overwhelm a server's resources, causing network congestion and impeding genuine users' access to services. This is achieved by flooding the targeted system with an excessive amount of traffic, rendering it unable to process legitimate requests effectively. The consequences of such attacks can be severe, affecting the availability and functionality of critical online services.
- To combat this threat, the paper proposes a mathematical model that aims to understand and characterize DDoS attacks better. Additionally, the study leverages machine learning algorithms, specifically Logistic Regression and Naive Bayes, to detect and differentiate between attack scenarios and normal network traffic. The CAIDA 2007 Dataset is utilized for experimental analysis, allowing the algorithms to be trained, tested, and validated.
- The mathematical model proposed in the paper attempts to establish a relationship between the inter-arrival time of incoming requests and the throughput of the system. Throughput analysis is a key element in identifying potential DDoS attacks. The paper explains that the inter-arrival time of requests and throughput are inversely related, as attacks tend to increase inter-arrival time and subsequently decrease throughput due to the congestion caused by malicious traffic.
- To improve the analysis of these attack scenarios, machine learning algorithms are applied to the CAIDA 2007 Dataset. Logistic Regression is utilized for prediction purposes, with a focus on identifying DDoS attacks accurately. Naive Bayes, based on the Bayes theorem, is used for its ability to assume independent features of variables, leading to enhanced accuracy. The algorithms' performances are assessed, and the results are analyzed and compared using the Weka data mining platform.
- In conclusion, the paper contributes to the field of cybersecurity by proposing a mathematical model and machine learning techniques to detect DDoS attacks effectively. The study demonstrates the advantages of machine learning over traditional methods and provides insights into the identification and prevention of these attacks. The results suggest that machine learning algorithms, especially Logistic Regression, offer a robust approach to identifying and mitigating the threats posed by DDoS attacks.(4)

Literature Survey 5:

- The paper revolves around addressing the issue of Distributed Denial of Service (DDoS) attacks, a common network threat where malicious users overwhelm websites or servers with excessive information to disrupt services, causing inconvenience to legitimate users. Unlike Denial of Service (DoS) attacks that come from a single source, DDoS attacks involve multiple sources or a "Botnet" that remotely controls devices for malicious purposes. The paper aims to detect DDoS attacks using supervised machine learning algorithms and identifies the best model for accurate detection.
- The study employs a set of eight supervised machine learning algorithms to identify DDoS attacks effectively. These algorithms are assessed based on parameters like accuracy, precision, recall, and false alarm rate. The research utilizes the CIC IDS 2017 dataset for training and testing, focusing on detecting attacks at an early stage. K-Fold cross-validation is applied during preprocessing, and the algorithms are trained and tested to identify the most effective model for detecting DDoS attacks.
- The primary types of DDoS attacks targeted are traffic-based, application-based, and volume-based attacks, which can severely impact network performance and application accessibility. Traditional methods like filtering and threshold-based techniques are insufficient for identifying unknown attacks, making machine learning a valuable tool in detecting both known and unknown attacks.
- The paper details the use of several machine learning algorithms, including Logistic Regression, Random Forest Classifier, Support Vector Machine (SVM), Decision Tree, Gaussian Naive Bayes, and K-Nearest Neighbors. These algorithms are compared based on their performance metrics using the CIC IDS 2017 dataset. Random Forest Classifier is identified as the most effective model due to its high accuracy, precision, recall, and low false alarm rate. The paper concludes that Random Forest is a robust choice for detecting DDoS attacks, offering accurate results by mitigating overfitting through its ensemble approach.
- In summary, the paper underscores the threat of DDoS attacks and the importance of effective detection methods. It demonstrates the efficacy of supervised machine learning algorithms in identifying DDoS attacks and highlights Random Forest Classifier as the optimal model based on its superior accuracy and low false alarm rate. The study's findings contribute to enhancing cybersecurity measures against DDoS attacks by utilizing advanced machine learning techniques for early detection and prevention.(5)

CHAPTER – III
SYSTEM REQUIREMENT SPECIFICATION

CHAPTER-III

System Requirement Specification

This chapter presents the software and hardware requirements for the whole project. System requirements are essential in building a project. The system requirements help in identifying the resources required to execute the application. These requirements include hardware, software and other dependencies. By identifying the necessary resources, developers can ensure that the application runs smoothly, without any performance issues. System requirements also help in ensuring that the application is compatible with the intended hardware and software environment. This is important because different environments have different capabilities and limitations. By specifying the system requirements, developers can ensure that the application is optimized for the target environment and can operate efficiently

3.1 Software Requirement

3.1.1 Jupyter Notebook

In the context of our project focused on DDoS Detection using machine learning techniques, Jupyter Notebook emerges as a versatile tool that can significantly enhance various aspects of the project. Here's how Jupyter Notebook's functionalities align with the requirements of the DDoS Detection project:

- **Exploratory Data Analysis (EDA):** Jupiter Notebook's interactive environment is ideal for conducting exploratory data analysis. We can use Pandas for data manipulation, statistical analysis, and data cleaning. Through Jupyter's flexibility, we can iteratively explore datasets, identify patterns, and preprocess data for feeding into machine learning models.
- **Feature Engineering and Model Selection:** In machine learning, feature engineering plays a crucial role in model performance. Jupyter Notebook allows us to experiment with different features, transform variables, and assess their impact on model accuracy. We can also try out various machine learning algorithms, tuning hyperparameters to select the best-performing model.
- **Code Transparency and Documentation:** DDoS detection involves complex algorithms and models. Jupyter Notebook facilitates clear documentation by allowing us to intersperse code cells with explanations, visualizations, and mathematical equations. This documentation is valuable for both team members and stakeholders to understand the rationale behind the chosen approaches.

- **Interactive Visualization:** Jupyter Notebook enables us to create interactive visualizations using libraries like Matplotlib and Seaborn. These visualizations aid in comprehending data distributions, model outputs, and trends, enhancing our ability to make informed decisions about model adjustments.
- **Model Evaluation and Testing:** Jupyter Notebook supports iterative model testing and evaluation. We can split data into training and testing sets, deploy machine learning models, and assess their performance metrics, such as accuracy, precision, recall, and F1-score. This iterative approach helps us fine-tune models for optimal results.
- **Collaborative Development:** For collaborative projects, Jupyter Notebook's version control integration (such as Git) ensures seamless collaboration. Multiple team members can work on the same notebook simultaneously, contributing their expertise to different parts of the project.
- **Documentation of Experimentation:** As we experiment with different algorithms, parameters, and techniques, Jupyter Notebook can document each experiment's results and insights. This documentation aids in understanding the project's progression, decisions, and potential next steps.
- **Rapid Prototyping:** DDoS detection requires rapid prototyping of machine learning models. Jupyter Notebook's interactive nature allows us to quickly experiment with different code blocks, instantly observing results and making necessary adjustments.

In conclusion, Jupyter Notebook serves as an invaluable asset in the "DDoS Detection Using Machine Learning" project. From data exploration and preprocessing to model selection, evaluation, and collaboration, Jupyter Notebook's capabilities enhance the efficiency, transparency, and effectiveness of the project's development process. It empowers us to delve deep into data, experiment with algorithms, and make informed decisions to build a robust DDoS detection solution.

Figure 3.1.1: Jupyter

3.1.2 Kali Linux, Golden Eye and Wireshark

- **Kali Linux**
- Kali Linux, a powerful open-source operating system, has earned a prominent reputation as a premier choice for cybersecurity professionals, hackers, and ethical hackers alike. Developed and maintained by Offensive Security, Kali Linux is specifically designed for penetration testing, digital forensics, and ethical hacking tasks.
- One of Kali Linux's defining features is its extensive toolkit of pre-installed security and penetration testing tools. This arsenal includes tools for network analysis, vulnerability assessment, password cracking, and much more. The user-friendly interface, coupled with regular updates and a supportive community, makes it an accessible and efficient platform for both beginners and experts.
- Kali Linux is based on Debian and inherits its stability and software repository, allowing users to customize and expand their toolsets easily. It supports various platforms, including ARM devices and cloud environments, making it versatile for various applications.
- Additionally, Kali Linux emphasizes ethical hacking and responsible use of its tools, providing resources and documentation to educate users on ethical practices. This commitment to ethical hacking distinguishes it from malicious cyber activities.
- Overall, Kali Linux stands as an indispensable resource in the cybersecurity realm, equipping professionals with the tools needed to identify vulnerabilities and secure systems effectively. Its continuous development and dedication to ethical hacking principles solidify its status as a vital tool in the battle against cyber threats.

Figure 3.1.2.1: Kali Linux

- **Golden Eye**

In the realm of cybersecurity and network defense, Distributed Denial of Service (DDoS) attacks stand as a potent threat. To bolster preparedness against such attacks, security professionals and organizations turn to testing tools that simulate DDoS scenarios. One such tool that has garnered attention is "GoldenEye."

GoldenEye is a DDoS testing tool designed to assess the resilience of a network or system against DDoS attacks. It operates by simulating various attack patterns to evaluate how well the target infrastructure can withstand a real-world assault. As the name suggests, the tool provides insights into the vulnerabilities and weak points in the system that could be exploited by malicious actors.

Key Features and Capabilities:

- **Attack Simulation**: GoldenEye allows users to simulate DDoS attacks on their own networks or systems. By emulating attack traffic, including various types of application layer and network layer attacks, the tool helps to identify potential vulnerabilities and points of failure.
- **Customizable Attack Profiles:** Users can tailor the attack profiles based on their requirements, such as the intensity and duration of the attack. This customization enables organizations to mimic different attack scenarios that could target their systems.
- **Attack Metrics and Reporting:** The tool provides detailed metrics and reporting on the effects of the simulated attacks. This information helps security professionals gauge the impact on system performance, network availability, and resource utilization during an attack.
- **Load Testing:** Beyond DDoS simulations, GoldenEye can be used for load testing. It assists in determining the maximum capacity a network or system can handle without collapsing under high traffic loads.

In essence, GoldenEye serves as a powerful tool in the arsenal of cybersecurity experts, aiding them in assessing their network's resilience against DDoS attacks. By providing simulated attack scenarios, customizable parameters, and insightful reporting, GoldenEye contributes to enhancing an organization's readiness to face and mitigate the impact of real-world DDoS threats.

Figure 3.1.2.2: Golden Eye

- **Wireshark**
- Wireshark is a widely-used open-source network protocol analyzer that allows users to capture and inspect data traveling over a network. It's a powerful tool for network administrators, security professionals, and developers, enabling them to troubleshoot network issues, monitor network activity, and analyze network traffic comprehensively.
- With a user-friendly interface, Wireshark supports a variety of network protocols, making it versatile for analyzing data in both local area networks (LANs) and wide area networks (WANs). Users can capture packets in real-time or load pre-recorded capture files for analysis.
- Wireshark provides detailed information about each packet in a network communication, including source and destination IP addresses, ports, protocol types, and payload data. This level of granularity helps diagnose network problems, identify security threats, and optimize network performance.
- One of Wireshark's key features is its filtering and search capabilities, which allow users to focus on specific network traffic of interest. Additionally, it can decrypt encrypted traffic if you possess the necessary keys. Wireshark is a valuable tool for ensuring network security, troubleshooting connectivity issues, and gaining insights into network performance.

Figure 3.1.2.3: Wireshark

3.1.3 Python and Libraries

- **Python**
- Python is a versatile and popular high-level programming language known for its readability and simplicity. Created by Guido van Rossum and first released in 1991, Python has since become one of the most widely used programming languages in the world.
- One of Python's key strengths is its ease of learning and use. It’s clear and concise syntax, which emphasizes indentation for code structure, makes it accessible to both

beginners and experienced developers. This readability fosters collaboration and reduces the likelihood of errors.

- Python's extensive standard library provides a wide range of pre-built modules and functions, simplifying tasks such as file handling, network communication, and data manipulation Python has found applications in web development (Django, Flask), data analysis (NumPy, pandas), artificial intelligence (TensorFlow, PyTorch), automation (scripting), and more. Its cross-platform compatibility allows developers to write code once and run it on different operating systems.

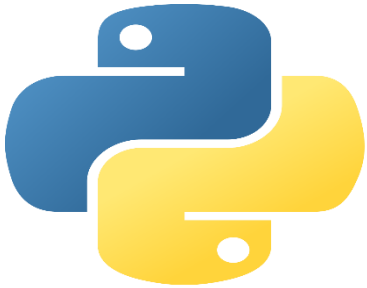


Figure 3.1.3.1: Python

- **Pandas**

- Pandas is a powerful open-source data manipulation and analysis library for the Python programming language. It provides essential data structures, primarily the DataFrame and Series, designed to simplify data handling and analysis. With its intuitive and flexible functionality, Pandas is widely used in data science, data analysis, and machine learning tasks.
- The DataFrame, at the core of Pandas, resembles a spreadsheet or SQL table, allowing users to organize data into rows and columns effortlessly. Users can load data from various sources, such as CSV files, Excel spreadsheets, SQL databases, and more, and perform diverse operations, including data cleaning, filtering, aggregation, and transformation.
- Overall, Pandas simplifies data manipulation, enabling users to efficiently analyze and visualize data, making it a cornerstone tool in the toolkit of data scientists, analysts, and researchers.

Figure 3.1.3.2: Pandas

- **NumPy**

- NumPy, short for "Numerical Python," is a fundamental and indispensable library in the Python ecosystem, primarily designed for performing numerical computations and data manipulation. It serves as the foundation for many other scientific and data analysis libraries like SciPy, Pandas, and Matplotlib.
- At its core, NumPy provides a powerful and efficient multi-dimensional array object called "ndarray." These arrays are the building blocks for creating and working with data in a structured manner. NumPy offers a wide range of mathematical functions and operations for array manipulation, including element-wise arithmetic, linear algebra, statistical analysis, and more. It also supports advanced indexing and slicing techniques, making it easy to extract and manipulate specific parts of arrays.
- In summary, NumPy is the backbone of numerical computing in Python, providing a robust framework for working with large datasets and performing complex mathematical operations efficiently. Its versatility and performance make it a cornerstone of the scientific Python ecosystem.

Figure 3.1.3.3: NumPy

- **MatplotLib**

- Matplotlib is a versatile and widely-used Python library for creating high-quality, customizable, and publication-ready 2D visualizations and graphs. Developed by John D. Hunter in 2003, it has become a fundamental tool in the data science and scientific computing communities for data visualization and exploration.
- Matplotlib offers a flexible and intuitive interface for producing a wide range of static and interactive plots. It supports various plot types, including line plots, scatter plots, bar plots, histograms, heatmaps, and more. Users have fine-grained control over every aspect of their plots, from colors and line styles to labels and legends, ensuring that visualizations meet specific requirements.
- In summary, Matplotlib is an essential tool for data visualization in Python, enabling users to create publication-quality plots and explore data effectively, making it a cornerstone of data analysis and scientific research.

Figure 3.1.3.4: MatplotLib

- **Seaborn**
- Seaborn is a powerful Python data visualization library built on top of Matplotlib. It is specifically designed to create aesthetically pleasing and informative statistical graphics. With Seaborn, you can easily generate a wide range of data visualizations for exploring and understanding complex datasets.
- Seaborn's primary strength lies in its simplicity and high-level interface. It provides a plethora of functions and color palettes that make it effortless to create stunning visualizations. Common plots like scatter plots, bar plots, line plots, histograms, and heatmaps are just a few lines of code away.
- Overall, Seaborn is a valuable tool for anyone looking to produce beautiful, informative, and publication-ready data visualizations with minimal effort. Whether you're a beginner or an experienced data scientist, Seaborn can greatly enhance your data exploration and presentation capabilities.

Figure 3.1.3.5: Seaborn

- **Scikit-Learn**

Scikit-learn is a versatile and accessible machine learning library designed to facilitate the development of machine learning models. It boasts a rich set of tools and algorithms for tasks such as classification, regression, clustering, dimensionality reduction, and more. With a strong emphasis on ease of use and a consistent API, scikit-learn allows both beginners and experts to build robust machine learning models with minimal effort.

Key features of scikit-learn include:

- **Wide Range of Algorithms:** It offers a diverse array of machine learning algorithms, including support vector machines, decision trees, random forests, k-nearest neighbors, and many more.

- **Data Preprocessing:** Scikit-learn simplifies data preprocessing tasks, such as feature scaling, encoding categorical variables, and handling missing data.
- **Model Evaluation:** The library provides tools for model evaluation, including metrics for measuring accuracy, precision, recall, and more. It also supports cross-validation techniques for assessing model performance.
- **Hyper parameter Tuning:** Scikit-learn makes hyperparameter tuning accessible through methods like GridSearchCV and RandomizedSearchCV.
- **Integration:** It integrates seamlessly with other popular Python libraries like NumPy, pandas, and Matplotlib, enabling efficient data manipulation and visualization alongside machine learning.

Figure 3.1.3.6: Scikit-Learn

3.2 Hardware Requirement

The hardware requirements for running the DDoS Detection using Machine Learning Algorithms are as follows:

- Processor: Intel Core i3/i7 / AMD Ryzen 5/7 equivalent
- RAM: Minimum 4 GB (8 GB recommended for smoother performance)
- Storage: At least 100- 250MB of free disk space
- Internet Connectivity: Required for DDoS testing in Kali linux

CHAPTER – IV

METHODOLOGY AND IMPLEMENTATION

CHAPTER-IV

Methodology and Implementation

Implementing a DDoS detection system using machine learning algorithms involves a systematic approach to develop a robust and accurate model. The following methodology outlines the key steps and considerations for building a DDoS detection system:

4.1 Methodology

1. Environment Setup:
    - Install Python: Ensure that Python 3.8 or above is installed on the system.
    - Install Libraries: Install the necessary libraries such as pandas, numpy, matplotlib, seaborn, scikitlearn using pip.
    - Jupyter Notebook Setup: Setup Jupyter notebook for the analysis and implementation of ML classification models for the DDoS Detection using Machine Learning algorithms

2. Problem Understanding and Data Collection:
    - Define the scope and objectives of the DDoS detection system.
    - Identify the types of DDoS attacks to be detected, such as network layer attacks, application layer attacks, and volumetric attacks.
    - Collect a comprehensive dataset containing both normal network traffic and DDoS attack traffic. This dataset serves as the foundation for training and testing the machine learning models.

3. Data Preprocessing:
    - Clean and preprocess the dataset by removing duplicate entries, handling missing values, and addressing outliers.
    - Perform feature engineering to extract relevant features from the raw data. Features can include packet attributes, network traffic patterns, and IP addresses.
    - Normalize or scale the features to ensure consistent input for the machine learning algorithms.

4. Feature Selection:
    - Analyze the importance of each feature in relation to the target variable (normal/attack).
    - Select the most relevant features to improve the efficiency and effectiveness of the machine learning models.

5. Model Selection:
    - Choose appropriate machine learning algorithms for DDoS detection. Common choices include Random Forest, Support Vector Machine (SVM), Logistic Regression, and Gradient Boost algorithm.
    - Consider the characteristics of the dataset, such as its size, dimensionality, and balance between normal and attack instances, when selecting models.

6. Model Training:
    - Split the preprocessed dataset into training and validation sets to train and evaluate the models.
    - Use techniques like K-fold cross-validation to avoid overfitting and ensure the generalization of the models.
    - Train the selected machine learning algorithms on the training data, tuning hyper parameters as needed.
    -

7. Model Evaluation:
    - Evaluate the trained models using various metrics such as accuracy, precision, recall, F1-score, and ROC curves.
    - Compare the performance of different algorithms to identify the best-performing model for DDoS detection.

8. Selecting the Best Model:
    - Choose the model with the highest accuracy or the best combination of evaluation metrics. Consider the trade-offs between different metrics based on the project's requirements. For instance, a DDoS detection system might prioritize high recall to minimize false negatives.

4.2 Execution Flowchart

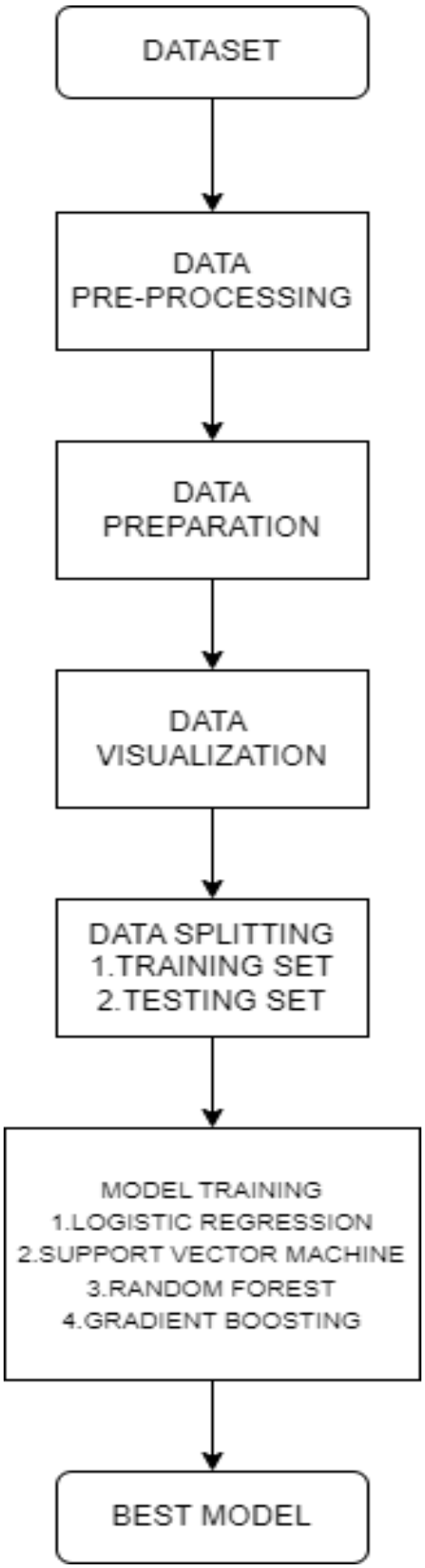


Figure 4.2: Flowchart of the Project

CHAPTER – V
TESTING AND VALIDATION

CHAPTER-V

Testing and Validation

The testing and validation phase of the DDoS Detection Using Machine Learning Algorithms project ensured the reliability, accuracy, and performance of the developed system. Rigorous testing procedures were employed to assess the functionality, effectiveness, and robustness of the DDoS detection models. This section outlines the comprehensive testing stages carried out to validate the accuracy and efficiency of the implemented DDoS detection system.

5.1 Performing DDoS and its testing

A Distributed Denial of Service (DDoS) attack is a malicious attempt to disrupt the normal functioning of a target website or online service by overwhelming it with a massive volume of traffic from multiple sources. Kali Linux is a popular penetration testing and ethical hacking platform that provides various tools for assessing and testing the security of networks and systems. One such tool is GoldenEye, a Python-based application used to launch DDoS attacks. Wireshark, on the other hand, is a widely used network protocol analyzer that captures and inspects network traffic in real-time. Here is a description of how to perform a DDoS attack using GoldenEye and capture the data using Wireshark:

- Performing the DDoS Attack using GoldenEye:
    - **Setup Kali Linux:** Ensure that you have Kali Linux installed on your system or a virtual machine.
    - **Install GoldenEye:** Open a terminal and use the package manager to install GoldenEye
    - **Choose a Target:** Identify the target website you want to perform the DDoS attack on. This could be a site you own and have permission to test, or you can use a public testing site with permission.
    - **Launch the Attack:** Run GoldenEye from the terminal with the target website's URL as an argument. You can also specify other options like the number of threads and duration of the attack. GoldenEye will start sending a flood of requests to the target server, overwhelming its resources and potentially causing it to become unresponsive.

❖ Capturing Data in Wireshark:

- **Start Wireshark:** Open Wireshark by typing wireshark in the terminal.
- **Choose Capture Interface:** Select the network interface through which the DDoS attack traffic will be sent and received. This could be your Ethernet interface or Wi-Fi adapter.
- **Start Capturing:** Click the "Start" button in Wireshark to begin capturing network traffic on the selected interface.
- **Analyze Captured Data:** As the DDoS attack is ongoing, Wireshark will capture and display the network packets in real-time. You can analyze the captured data to see the flood of requests being sent to the target server and the responses it generates.
- **Save Captured Data:** Once you have captured enough data to analyze, you can stop the capture and save the captured packets to a file for further examination.

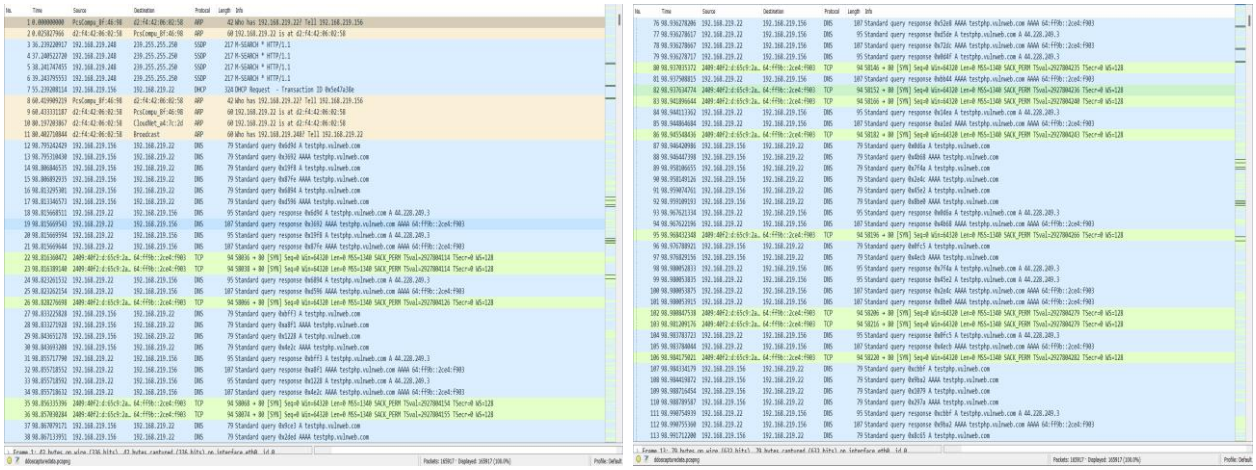


Figure 5.1.1: Capturing Of Dataset in Wireshark

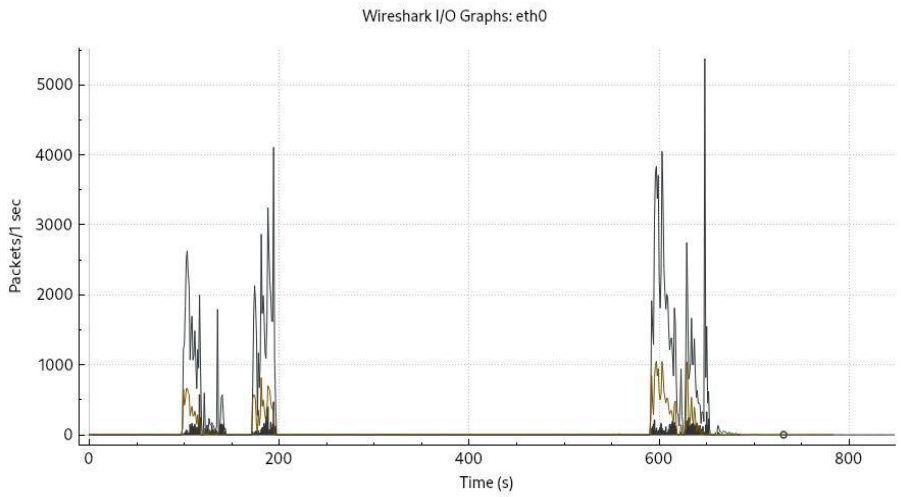


Figure 5.1.2: Analysis of Traffic on the website Due to DDoS attack

5.2 Analysing and applying ML algorithms on the Dataset

- **Initial Analysis on the dataset and its Description:**
    - **dt:** This column represents the date and time of the recorded network data. It provides a timestamp for each network event or packet capture.
    - **switch:** The "switch" column denotes the network switch involved in the data transmission. It indicates the specific network hardware component handling the network traffic.
    - **pktcount:** This column signifies the count of packets exchanged during the network activity. It tracks the number of individual packets sent or received.
    - **bytecount:** The "bytecount" column quantifies the total number of bytes transmitted during the network communication. It measures the data volume in bytes.
    - **dur:** "dur" represents the duration of the network event in seconds. It indicates the time span over which the network activity occurred.
    - **dur_nsec:** This column provides additional granularity to the duration, representing the duration in nanoseconds. It offers more precise timing information.
    - **tot_dur:** "tot_dur" stands for total duration and represents the cumulative duration of multiple network events or flows. It aggregates the durations for related activities.
    - **flows:** The "flows" column indicates the number of distinct network flows during the recorded event. A flow refers to a sequence of packets exchanged between specific source and destination IP addresses.
    - **packetins:** This column records the count of packets inserted into the network during the event. It tracks the number of packets added to the network.
    - **pktperflow:** "pktperflow" calculates the average number of packets per network flow. It provides insights into the packet distribution across different flows.
    - **byteperflow:** Similar to "pktperflow," "byteperflow" calculates the average volume of bytes per network flow. It helps analyze the data volume distribution.
    - **pktrate:** The "pktrate" column represents the rate of packet transmission per second. It measures the pace at which packets are exchanged.
    - **Pairflow:** "Pairflow" denotes the paired flow count, which refers to the number of bidirectional flows between source and destination pairs.

- **port_no:** This column indicates the port number associated with the network traffic. Ports are used to categorize different types of network communication.
- **tx_bytes:** "tx_bytes" represents the count of transmitted bytes during the network event. It measures the amount of data sent from the source.
- **rx_bytes:** The "rx_bytes" column indicates the count of received bytes during the network event. It measures the amount of data received by the destination.
- **tx_kbps:** This column calculates the transmission rate in kilobits per second (kbps). It quantifies the data transfer speed from the source.
- rx_kbps: "rx_kbps" calculates the reception rate in kilobits per second (kbps). It quantifies the data transfer speed received by the destination.
- **tot_kbps:** The "tot_kbps" column signifies the total data transfer rate in kilobits per second (kbps). It represents the combined transmission and reception speeds.
- **label:** The "label" column indicates the classification label assigned to the network event. It distinguishes between different types of network behaviors, such as normal or malicious traffic.

- ## Data Visualization, Data Splitting and Modal Evaluation:
  - **Unit Testing:** Unit testing was the initial phase of testing, focusing on the individual components and algorithms of the DDoS detection system. Each machine learning algorithm, including Logistic Regression, Support Vector Machine, Random Forest, and any additional models, underwent thorough testing to identify any isolated issues or inaccuracies. This approach allowed for the identification and resolution of specific algorithmic errors.
  - **Integration Testing:** Following unit testing, integration testing was performed to assess the interaction and collaboration between different machine learning models. The goal was to ensure that the models worked cohesively to collectively detect and classify DDoS attacks. For instance, the interaction between ensemble models like Random Forest and individual models like Logistic Regression was tested to verify the system's overall efficiency in detecting attacks.

- **Accuracy and Metrics Validation:** Validation of detection accuracy and metrics was a crucial aspect of the testing phase. Metrics such as accuracy, precision, recall, and false positive rates were measured to gauge the effectiveness of the detection system.
- **Reporting and Visualization Validation:** The reporting and visualization components of the system were validated to ensure that generated reports, graphs, and visualizations accurately represented the detected DDoS attacks. This validation was important for security personnel to make informed decisions based on the detected threats.

By conducting these comprehensive testing stages, the DDoS detection system's functionality, performance, and accuracy were systematically verified. This phase ensured that the system was well-prepared to effectively detect and respond to DDoS attacks, contributing to a more secure network environment.

CHAPTER – VI
RESULTS AND DISCUSSIONS

CHAPTER-VI

Results and Discussion

➢ Initial Analysis on the dataset:

- The first step is to import all the required libraries

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing

from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import GridSearchCV
import time

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn import metrics
```

Figure 6.1: Importing Libraries

- Moving forward, the subsequent task involves loading the dataset and conducting an analysis encompassing its size, shape, information, and descriptive attributes. This step aims to gain insights into the dataset's overall structure and content.

```
data = pd.read_csv( r"C:\Users\91809\Desktop\Python project 4thsem\Project folder\dataset_sdn.csv")
```

```
data
```

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_by
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	3
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	3
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	1
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
104340	5262	3	10.0.0.5	10.0.0.7	79	7742	81	842000000	8.184200e+10	5	...	0	0	ICMP	1	15209	12
104341	5262	3	10.0.0.5	10.0.0.7	79	7742	81	842000000	8.184200e+10	5	...	0	0	ICMP	3	15099	14
104342	5262	3	10.0.0.11	10.0.0.5	31	3038	31	805000000	3.180500e+10	5	...	1	0	ICMP	2	3409	3
104343	5262	3	10.0.0.11	10.0.0.5	31	3038	31	805000000	3.180500e+10	5	...	1	0	ICMP	1	15209	12
104344	5262	3	10.0.0.11	10.0.0.5	31	3038	31	805000000	3.180500e+10	5	...	1	0	ICMP	3	15099	14

104345 rows × 23 columns

```
data.shape
```

```
(104345, 23)
```

Figure 6.2: Uploading the Dataset

- The command "sns.pairplot(data)" is utilized to create a pair plot using the Seaborn library. This plot provides a comprehensive visual representation of the relationships between pairs of variables within the dataset. Each pair of attributes is displayed as a scatter plot, showcasing potential correlations and patterns. By examining this pair plot, we can swiftly identify any evident trends, clusters, or associations among the variables, aiding in initial insights into the dataset's underlying patterns and characteristics.

```
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x20ec8cf95e0>
```

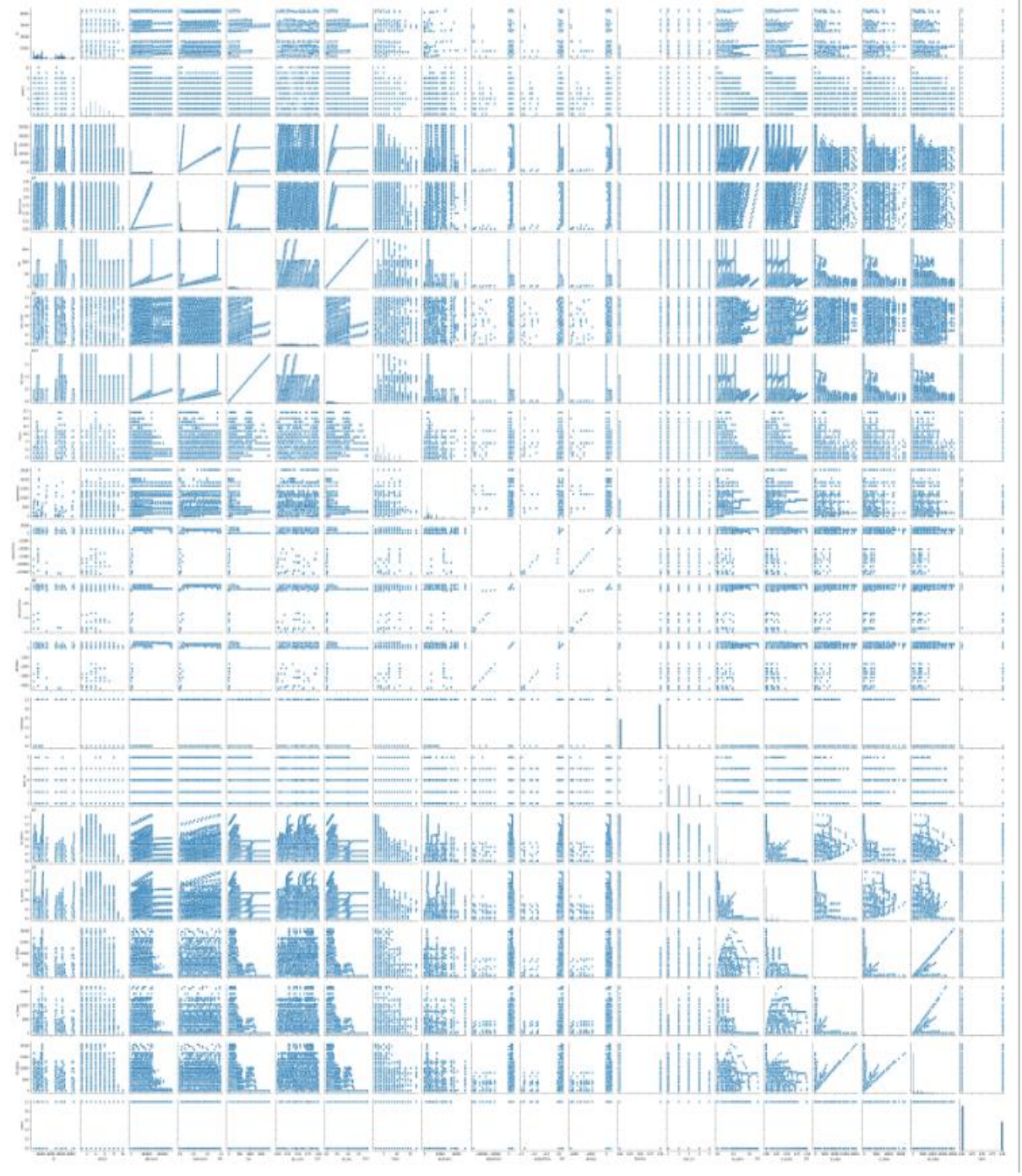


Figure 6.3: Pairplot of the dataset

- The below code snippet employs the Seaborn library to visualize the distribution of labels within the dataset. By utilizing a bar plot, the code illustrates the frequency of each label category. The x-axis represents the different label categories, while the y-axis signifies the corresponding counts. This visualization offers a clear understanding of the distribution of labels, enabling us to assess the balance or skewness of the dataset's target classes. The plot's title and axis labels further enhance its interpretability.

```
label_counts = data['label'].value_counts()
sns.barplot(x=label_counts.index, y=label_counts.values)
plt.xlabel('Label')
plt.ylabel('Count')
plt.title('Label Distribution')
plt.show()
```

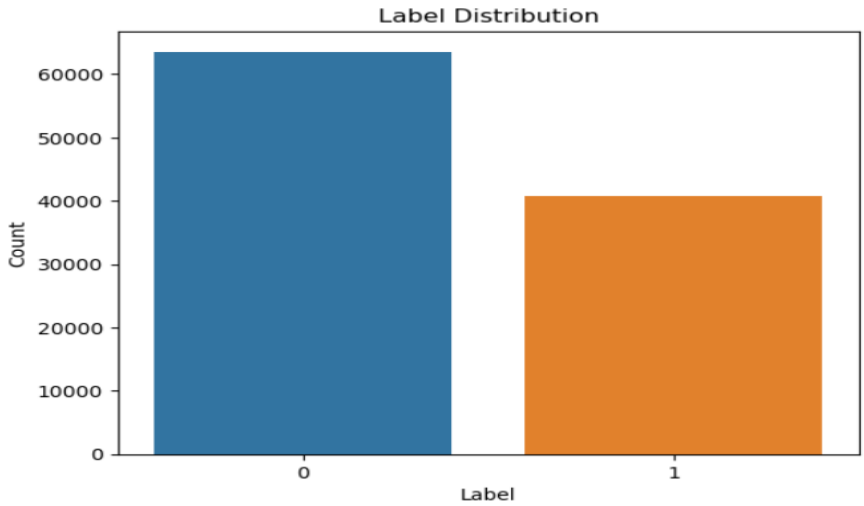


Figure 6.4: Bar Graph of the Label Count attribute

- The below code snippet utilizes a pie chart created using the Matplotlib library to visually represent the distribution of malicious and benign requests within the dataset. The chart is divided into two segments, each corresponding to "Malicious" and "Benign" labels, respectively. The percentage of occurrences for each label is displayed within the segments using autopct formatting. This visualization offers an intuitive view of the proportion of malicious and benign requests in the dataset, enhancing our understanding of the data's composition. The title and legend provide additional context for interpretation.

```
labels = ["Maliciuous",'Benign']
sizes = [dict(data.label.value_counts())[0], dict(data.label.value_counts())[1]]
plt.figure(figsize = (13,8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.legend(["Maliciuous", "Benign"])
plt.title('The percentage of Benign and Maliciuos Requests in dataset')
plt.show()
```

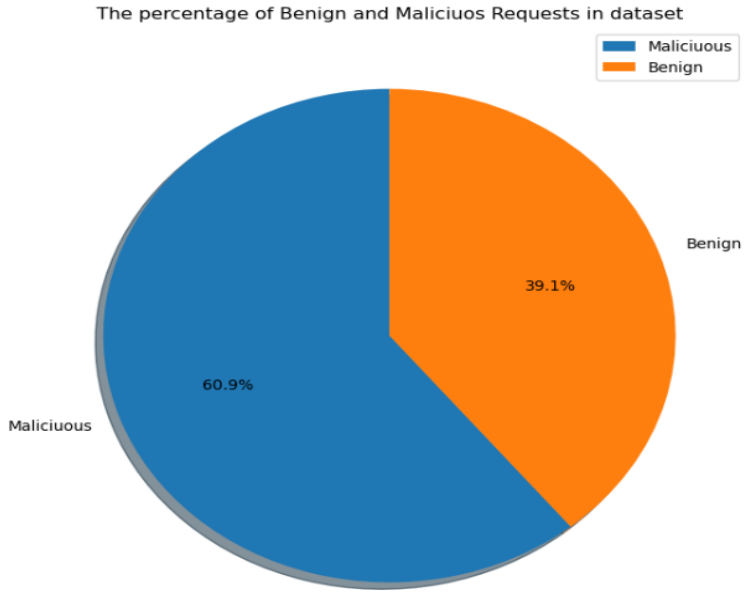
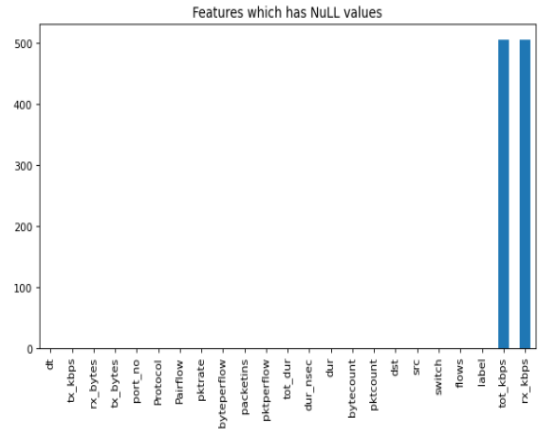


Figure 6.5: Analysis of the Benign and Malicious requests in the dataset

- The below code segment generates a bar plot using Matplotlib to visualize the presence of null values within different features of the dataset. The x-axis represents the features with null values, while the y-axis displays the corresponding count of null values. This visualization allows for a quick assessment of the completeness of data across various features. By sorting the features based on the count of null values, it becomes evident which attributes have missing data and require further attention during preprocessing and analysis stages. The title provides a clear context for the purpose of the visualization.

```
# vizualisation of Null valued features
figure(figsize=(9, 5), dpi=80)
data[data.columns[data.isna().sum() >= 0]].isna().sum().sort_values().plot.bar()
plt.title("Features which has NuLL values")
```

```
Text(0.5, 1.0, 'Features which has NuLL values')
```




```
data.isnull().sum()
```

```
dt              0
switch          0
src             0
dst             0
pktcount        0
bytecount       0
dur             0
dur_nsec        0
tot_dur         0
flows           0
packetins       0
pktperflow      0
byteperflow     0
pktrate         0
Pairflow        0
Protocol        0
port_no         0
tx_bytes        0
rx_bytes        0
tx_kbps         0
rx_kbps       506
tot_kbps      506
label           0
dtype: int64
```

Figure 6.6: Analysis of null values in the dataset

- The below code segment performs data type-based selection of columns in the dataset. It separates columns with numeric data types (integers and floating-point) and object data types (typically representing categorical or textual data). The printed output displays the names of numeric and object columns separately, providing insight into the composition of features. Additionally, it presents the counts of numeric and object features, offering a quantitative overview of the data structure. This code snippet is essential for understanding the data's composition and deciding on appropriate preprocessing and analysis strategies based on feature types.

```
numeric_df = data.select_dtypes(include=['int64', 'float64'])
object_df = data.select_dtypes(include=['object'])
numeric_cols = numeric_df.columns
object_cols = object_df.columns
print('Numeric Columns: ')
print(numeric_cols, '\n')
print('Object Columns: ')
print(object_cols, '\n')
print('Number of Numeric Features: ', len(numeric_cols))
print('Number of Object Features: ', len(object_cols))
```

```
Numeric Columns:
Index(['dt', 'switch', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur',
       'flows', 'packetins', 'pktperflow', 'byteperflow', 'pktrate',
       'Pairflow', 'port_no', 'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps',
       'tot_kbps', 'label'],
      dtype='object')

Object Columns:
Index(['src', 'dst', 'Protocol'], dtype='object')

Number of Numeric Features:  20
Number of Object Features:  3
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           104345 non-null  int64
 1   switch       104345 non-null  int64
 2   src          104345 non-null  object
 3   dst          104345 non-null  object
 4   pktcount     104345 non-null  int64
 5   bytecount    104345 non-null  int64
 6   dur          104345 non-null  int64
 7   dur_nsec     104345 non-null  int64
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64
 10  packetins    104345 non-null  int64
 11  pktperflow   104345 non-null  int64
 12  byteperflow  104345 non-null  int64
 13  pktrate      104345 non-null  int64
 14  Pairflow     104345 non-null  int64
 15  Protocol     104345 non-null  object
 16  port_no      104345 non-null  int64
 17  tx_bytes     104345 non-null  int64
 18  rx_bytes     104345 non-null  int64
 19  tx_kbps      104345 non-null  int64
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

```
numeric_df
```

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflow	pktrate	Pairflow	port_no	tx_bytes	n
0	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428310	451	0	3	143928631	
1	11605	1	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	14424046	451	0	4	3842	
2	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	451	0	1	3795	
3	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	451	0	2	3688	
4	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	451	0	3	3413	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
104340	5262	3	79	7742	81	842000000	8.184200e+10	5	10	29	2842	0	0	1	15209	
104341	5262	3	79	7742	81	842000000	8.184200e+10	5	10	29	2842	0	0	3	15099	
104342	5262	3	31	3038	31	805000000	3.180500e+10	5	10	30	2940	1	0	2	3409	
104343	5262	3	31	3038	31	805000000	3.180500e+10	5	10	30	2940	1	0	1	15209	
104344	5262	3	31	3038	31	805000000	3.180500e+10	5	10	30	2940	1	0	3	15099	

104345 rows × 20 columns

```
object_df
```

	src	dst	Protocol
0	10.0.0.1	10.0.0.8	UDP
1	10.0.0.1	10.0.0.8	UDP
2	10.0.0.2	10.0.0.8	UDP
3	10.0.0.2	10.0.0.8	UDP
4	10.0.0.2	10.0.0.8	UDP
...	...	...	...
104340	10.0.0.5	10.0.0.7	ICMP
104341	10.0.0.5	10.0.0.7	ICMP
104342	10.0.0.11	10.0.0.5	ICMP
104343	10.0.0.11	10.0.0.5	ICMP
104344	10.0.0.11	10.0.0.5	ICMP

104345 rows × 3 columns

Figure 6.7: Analysis of numerical and object type data in the dataset in brief

- The below code snippet generates a horizontal bar plot to visualize the distribution of requests originating from different IP addresses. The length of each bar represents the frequency of requests sent from a specific IP address. Alongside each bar, the corresponding frequency count is displayed in red. This visualization offers insights into the frequency distribution of requests based on sender IP addresses. The x-axis indicates the number of requests, and the y-axis displays the IP addresses of the senders. This visualization aids in identifying patterns and potential anomalies in request origins.

```
figure(figsize=(12, 7), dpi=80)
plt.barh(list(dict(data.src.value_counts()).keys()), dict(data.src.value_counts()).values(), color='lawngreen')

for idx, val in enumerate(dict(data.src.value_counts()).values()):
    plt.text(x = val, y = idx-0.2, s = str(val), color='r', size = 13)

plt.xlabel('Number of Requests')
plt.ylabel('IP addres of sender')
plt.title('Number of all reqests')
```

```
Text(0.5, 1.0, 'Number of all reqests')
```

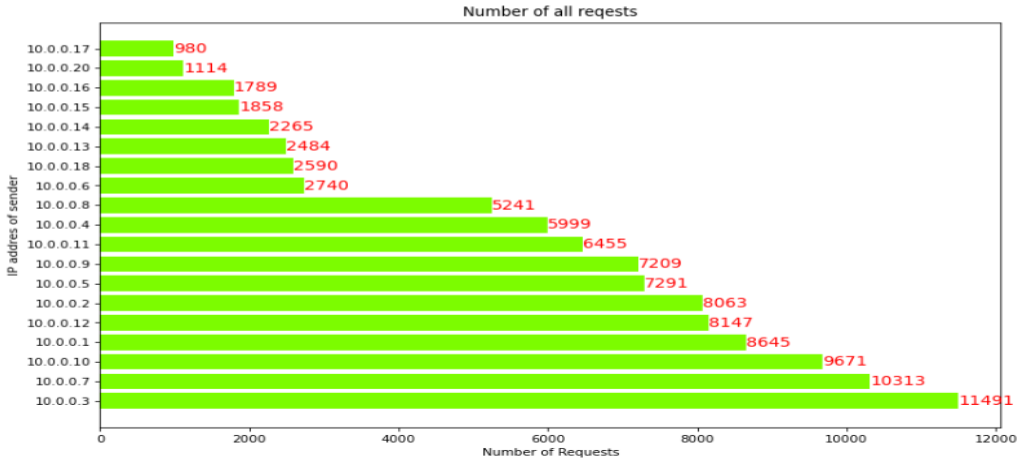


Figure 6.8: Number of All Requests

- The below code snippet creates a horizontal bar plot specifically focused on visualizing the distribution of attack requests based on their source IP addresses. The length of each blue bar corresponds to the frequency of attack requests originating from a particular IP address. Similar to the previous visualization, the frequency count is displayed in red text alongside each bar. This visualization helps to discern patterns and characteristics of attack sources, enabling a more targeted understanding of attack origins within the dataset. The x-axis represents the number of attack requests, while the y-axis displays the corresponding IP addresses of the senders.

```
figure(figsize=(12, 7), dpi=80)
plt.barh(list(dict(data[data.label == 1].src.value_counts()).keys()), dict(data[data.label == 1].src.value_counts()).values(), c

for idx, val in enumerate(dict(data[data.label == 1].src.value_counts()).values()):
    plt.text(x = val, y = idx-0.2, s = str(val), color='r', size = 13)

plt.xlabel('Number of Requests')
plt.ylabel('IP addres of sender')
plt.title('Number of Attack requests')
```

```
Text(0.5, 1.0, 'Number of Attack requests')
```

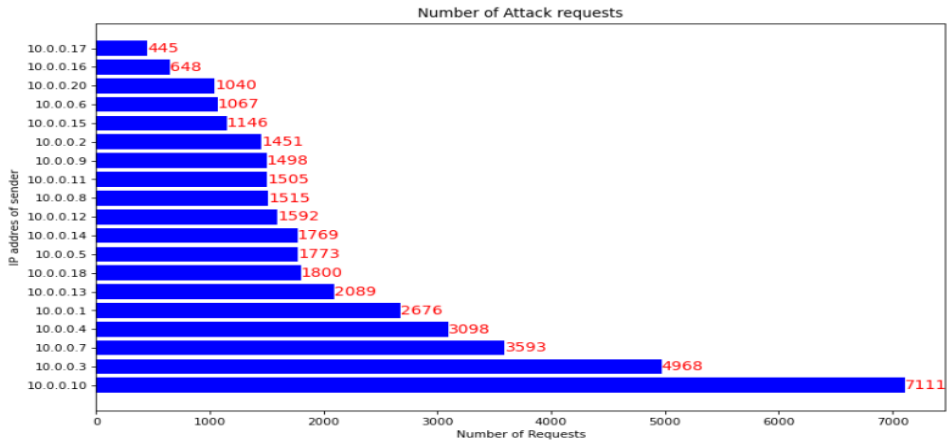


Figure 6.9: Number of attacked request

- The below code generates a horizontal bar plot that displays the distribution of requests originating from different IP addresses. The plot includes two sets of bars: green bars represent all requests, while blue bars represent malicious requests. Each bar's length corresponds to the frequency of requests from the respective IP address. The red and white text on the bars indicates the frequency count, with red indicating all requests and white indicating malicious ones. This visualization provides insights into the distribution of requests from various IP addresses, allowing for comparison between overall and malicious traffic. The x-axis represents the number of requests, and the y-axis displays the corresponding IP addresses of the senders.

```
figure(figsize=(12, 7), dpi=80)
plt.barh(list(dict(data.src.value_counts()).keys()), dict(data.src.value_counts()).values(), color='lawngreen')
plt.barh(list(dict(data[data.label == 1].src.value_counts()).keys()), dict(data[data.label == 1].src.value_counts()).values(), c

for idx, val in enumerate(dict(data.src.value_counts()).values()):
    plt.text(x = val, y = idx-0.2, s = str(val), color='r', size = 13)

for idx, val in enumerate(dict(data[data.label == 1].src.value_counts()).values()):
    plt.text(x = val, y = idx-0.2, s = str(val), color='w', size = 13)


plt.xlabel('Number of Requests')
plt.ylabel('IP addres of sender')
plt.legend(['All','malicious'])
plt.title('Number of requests from different IP adress')
```

```
Text(0.5, 1.0, 'Number of requests from different IP adress')
```

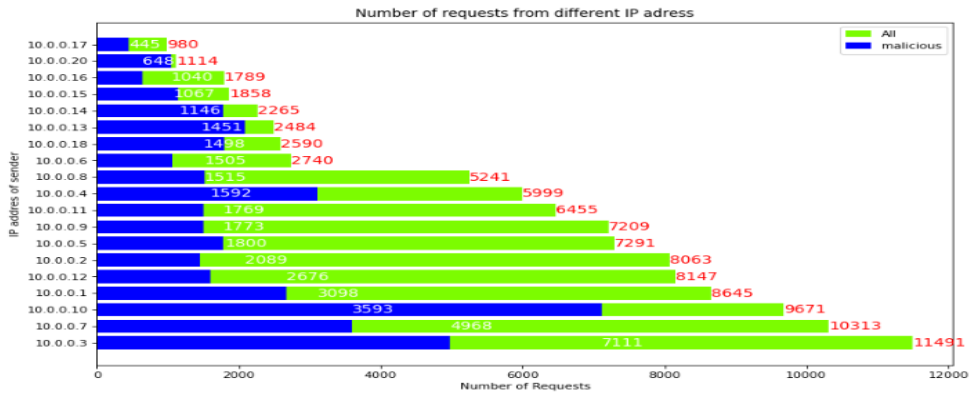


Figure 6.10: Number of requests from different IP Address and malicious

- The below code generates a bar plot that illustrates the distribution of requests based on different protocols. The plot consists of two sets of bars: red bars represent all requests, while blue bars represent malicious requests. Each bar's height corresponds to the count of requests associated with a specific protocol. The text annotations on top of each bar indicate the respective counts for both overall and malicious requests. The x-axis represents different protocols, and the y-axis displays the count of requests. This visualization provides insights into how different protocols are involved in both overall and malicious traffic patterns within the dataset.

```
figure(figsize=(10, 6), dpi=80)
plt.bar(list(dict(data.Protocol.value_counts()).keys()), dict(data.Protocol.value_counts()).values(), color='r')
plt.bar(list(dict(data[data.label == 1].Protocol.value_counts()).keys()), dict(data[data.label == 1].Protocol.value_counts()).va

plt.text(x = 0 - 0.15, y = 41321 + 200, s = str(41321), color='black', size=17)
plt.text(x = 1 - 0.15, y = 33588 + 200, s = str(33588), color='black', size=17)
plt.text(x = 2 - 0.15, y = 29436 + 200, s = str(29436), color='black', size=17)

plt.text(x = 0 - 0.15, y = 9419 + 200, s = str(9419), color='w', size=17)
plt.text(x = 1 - 0.15, y = 17499 + 200, s = str(17499), color='w', size=17)
plt.text(x = 2 - 0.15, y = 13866 + 200, s = str(13866), color='w', size=17)

plt.xlabel('Protocol')
plt.ylabel('Count')
plt.legend(['All', 'malicious'])
plt.title('The number of requests from different protocols')
```

```
Text(0.5, 1.0, 'The number of requests from different protocols')
```

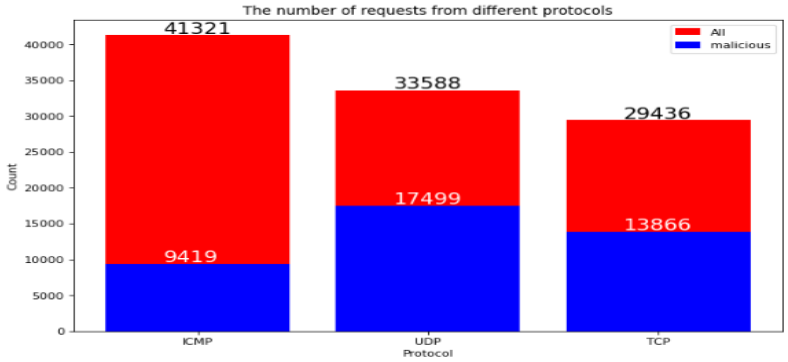


Figure 6.11: Number of Requests from different protocols

- The below code generates a histogram plot depicting the distribution of request durations within the dataset. The x-axis represents the duration values, divided into bins for better visualization, while the y-axis indicates the frequency or count of occurrences for each bin. The plot provides insights into the distribution of request durations, allowing us to observe patterns and concentrations of durations across the dataset.

```
figure(figsize=(8, 4), dpi=80)
plt.hist(df.dur, bins=20, color='b')
plt.title('Duration')
plt.show()
```

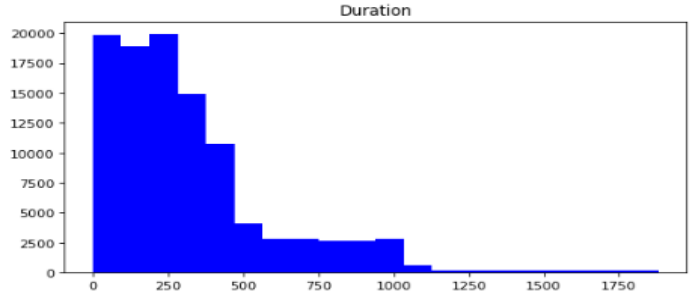


Figure 6.12: Duration of the attack

- The code generates a histogram plot illustrating the distribution of transmitted bytes (TX_BYTES) within the dataset. The x-axis represents different ranges of transmitted bytes, divided into bins for better visualization, while the y-axis shows the frequency or count of occurrences falling within each bin. This plot offers insights into the distribution of transmitted bytes across the dataset, enabling the observation of patterns and concentrations of transmitted byte values. It aids in understanding the spread of transmitted bytes and potential variations in network traffic.

```
figure(figsize=(8, 4), dpi=80)
plt.hist(df.tx_bytes, bins=20, color='r')
plt.title('TX_BYTES - Transmitted Bytes')
plt.show()
```

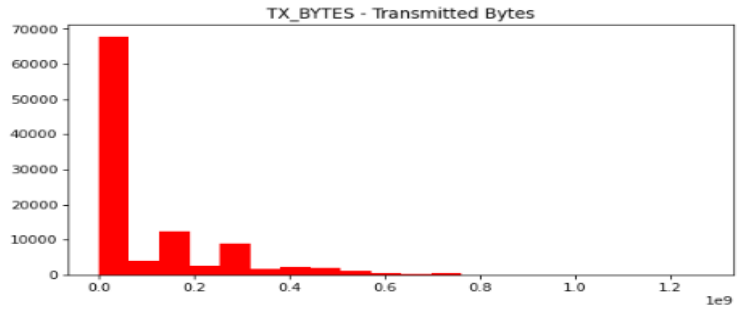


Figure 6.13: Transmitted Bytes

- The provided code generates a histogram plot depicting the distribution of transmitted kilobits per second (TX_KBPS) in the dataset. The x-axis displays different ranges of TX_KBPS values, which are divided into bins for clearer visualization. The y-axis represents the frequency or count of occurrences falling within each bin. This histogram provides insights into the distribution pattern of transmitted data rates in kilobits per second, offering an overview of the varying levels of network traffic speeds. It aids in understanding the prevalence of different TX_KBPS values across the dataset.

```
figure(figsize=(8, 4), dpi=80)
plt.hist(df.tx_kbps, bins=10, color='g')
plt.title('TX_KBPC')
plt.show()
```

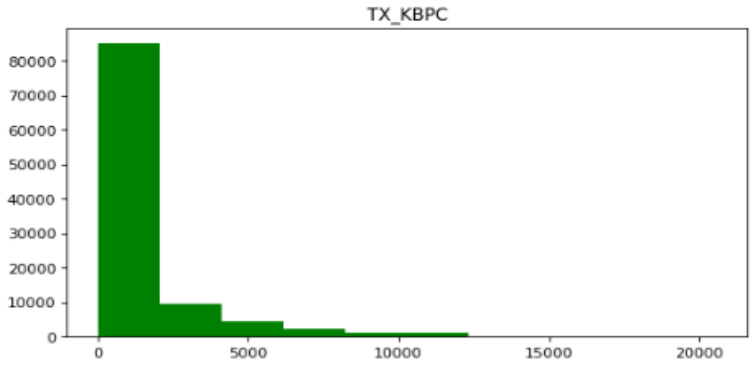


Figure 6.14: Distribution of the transmitted data

- The provided code generates a histogram plot displaying the distribution of the "SWITCH" attribute in the dataset. The x-axis represents different ranges of SWITCH values, organized into bins for visual clarity. The y-axis illustrates the frequency or count of occurrences falling within each bin. This histogram offers insights into the distribution pattern of the "SWITCH" attribute's values, aiding in understanding the prevalence of different SWITCH values across the dataset. It provides a visual summary of how frequently each value of the "SWITCH" attribute occurs and helps identify any trends or patterns in the data related to the network switches involved.

```
plt.hist(df.switch, bins=20, color='r')
plt.title('SWITCH')
plt.xlabel('SWITCH')
plt.show()
```

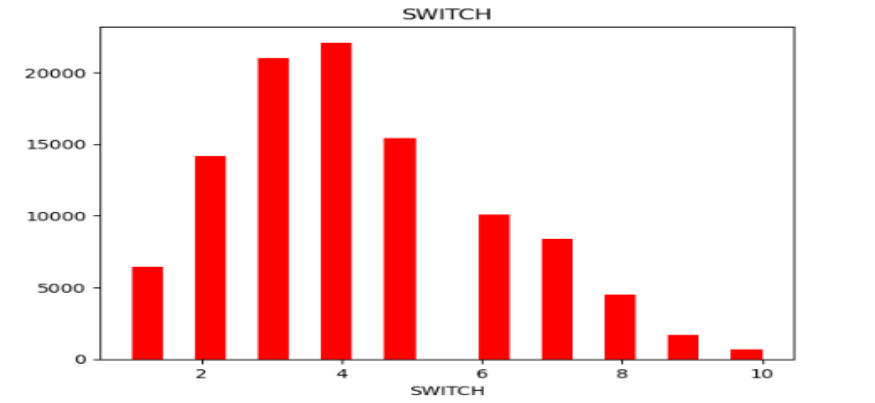


Figure 6.15: SWITCH values

- The below code creates a heatmap using the Seaborn library. The heatmap displays the absolute correlation coefficients between pairs of features. Each cell in the heatmap represents the correlation between two features, with colors indicating the strength of correlation. Higher absolute values are typically represented with darker colors. The heatmap provides an overview of the relationships between different features in the dataset. Correlation values close to 1 or -1 suggest strong positive or negative relationships between the features, respectively. On the other hand, correlation values close to 0 suggest a weak or no linear relationship. This visualization aids in identifying potential multicollinearity among features, which can impact the effectiveness of machine learning algorithms that assume independence between features.

```
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(abs(X.corr()), annot=True)
```

```
C:\Users\91809\AppData\Local\Temp\ipykernel_12156\255498303.py:2: FutureWarning: The default value of numeric_only in DataFram
e.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_
only to silence this warning.
  sns.heatmap(abs(X.corr()), annot=True)
<Axes: >
```

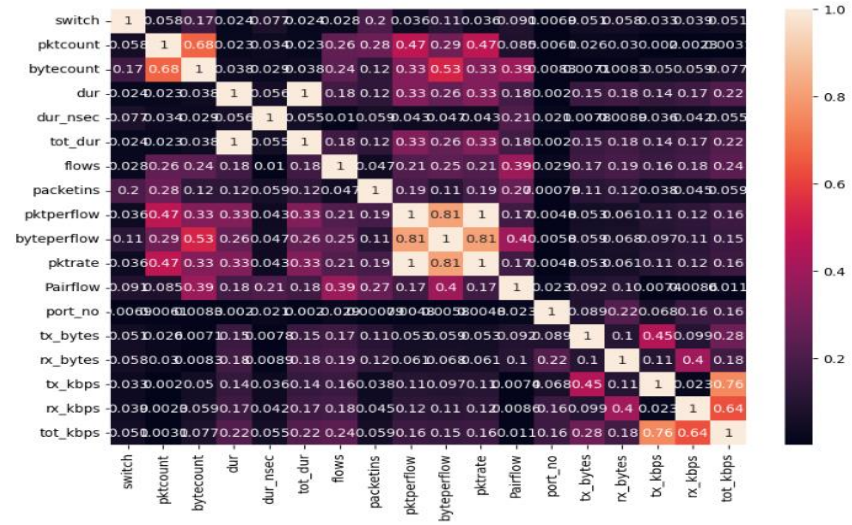


Figure 6.16: Heatmap of the dataset

➢ Model Preparation and Evaluation:

The provided code defines a Python class named Model that encapsulates the process of training and evaluating machine learning models for the DDoS detection system. The class includes methods to perform Logistic Regression, Support Vector Machine (SVM), Random Forest, and Gradient Boosting classification using the scikit-learn library. These methods involve training each respective model on the preprocessed data and evaluating their accuracy using the test set.

The plot_accuracy method visualizes the accuracy of different models using a bar chart, where each bar corresponds to a model and its associated accuracy score. The run_models method sequentially executes the defined model methods and generates accuracy scores for each model. It then calls plot_accuracy to visualize the accuracy results for comparison.

The purpose of this class is to streamline the process of testing multiple machine learning models on the DDoS detection dataset and assessing their performance. The visualizations provide insights into how different models perform on the given dataset, aiding in the selection of the most effective model for the task.

```python
class Model:
    global y

    def __init__(self, data):
        self.data = data
        X = preprocessing.StandardScaler().fit(self.data).transform(self.data)
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, random_state=42, test_size=0.3)

    def LogisticRegression(self):
        solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

        start_time = time.time()
        results_lr = []
        accuracy_list = []
        for solver in solvers:
            LR = LogisticRegression(C=0.03, solver=solver).fit(self.X_train, self.y_train)
            predicted_lr = LR.predict(self.X_test)
            accuracy_lr = accuracy_score(self.y_test, predicted_lr)

            results_lr.append({'solver': solver, 'accuracy': f"{round(accuracy_lr * 100, 2)}%",
                               'Coefficients': {'W': LR.coef_, 'b': LR.intercept_}})

            accuracy_list.append(accuracy_lr)

        solver_name = solvers[accuracy_list.index(max(accuracy_list))]
        LR = LogisticRegression(C=0.03, solver=solver_name).fit(self.X_train, self.y_train)
        predicted_lr = LR.predict(self.X_test)
        accuracy_lr = accuracy_score(self.y_test, predicted_lr)
        print(f"Accuracy of LR model is: {round(accuracy_lr * 100, 2)}%", '\n')
        print("############################################################")
        print('Best solver is:', solver_name)
        print("############################################################")
        print(classification_report(predicted_lr, self.y_test), '\n')
        print("############################################################")
        print("--- %s seconds --- time for LogisticRegression" % (time.time() - start_time))
```

```python
    def RandomForest(self):
        start_time = time.time()

        param_grid = {
    'n_estimators': [100, 200, 500],
    'min_samples_split': [2, 5, 10],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 3, 4, 5, 6]  # Include 'None' for unlimited depth
}


        rf_base = RandomForestClassifier(criterion='gini',
                                         oob_score=True,
                                         random_state=1,
                                         n_jobs=-1)

        rf_search = GridSearchCV(rf_base, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
        rf_search.fit(self.X_train, self.y_train)

        best_rf = rf_search.best_estimator_
        predicted_rf = best_rf.predict(self.X_test)
        accuracy_rf = accuracy_score(self.y_test, predicted_rf)

        print(f"Accuracy of RF model is: {round(accuracy_rf * 100, 2)}%", '\n')
        print("############################################################")
        print(classification_report(predicted_rf, self.y_test))
        print("############################################################")

        print("--- %s seconds ---" % (time.time() - start_time))
```

```python
def SupportVectorMachine(self):
    start_time = time.time()
    accuracy_list = []
    result_svm = []
    kernels = ['linear', 'poly', 'rbf', 'sigmoid']

    for kernel in kernels:
        SVM = svm.SVC(kernel=kernel).fit(self.X_train, self.y_train)
        predicted_svm = SVM.predict(self.X_test)
        accuracy_svm = accuracy_score(self.y_test, predicted_svm)
        result_svm.append({"kernel": kernel, "accuracy": f"{round(accuracy_svm * 100, 2)}%"})
        print("Accuracy: %.2f%%" % round((accuracy_svm * 100.0), 2))
        print('############################################################')
        accuracy_list.append(accuracy_svm)

    kernel_name = kernels[accuracy_list.index(max(accuracy_list))]
    SVM = svm.SVC(kernel=kernel_name).fit(self.X_train, self.y_train)
    predicted_svm = SVM.predict(self.X_test)
    accuracy_svm = accuracy_score(self.y_test, predicted_svm)
    print(f"Accuracy of SVM model {round(accuracy_svm, 2) * 100}%", '\n')
    print("############################################################")
    print('Best kernel is:', kernel_name)
    print("############################################################")
    print(classification_report(predicted_svm, self.y_test))
    print("############################################################")
    print("--- %s seconds ---" % (time.time() - start_time))
```

```python
def GradientBoost(self):
    start_time = time.time()
    gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=1)
    gbc.fit(self.X_train, self.y_train)

    predicted_gbc = gbc.predict(self.X_test)
    accuracy_gbc = accuracy_score(self.y_test, predicted_gbc)
    print(f"Accuracy of Gradient Boosting model is: {round(accuracy_gbc * 100, 2)}%", '\n')
    print("############################################################")
    print(classification_report(predicted_gbc, self.y_test))
    print("############################################################")

    print("--- %s seconds ---" % (time.time() - start_time))

def plot_accuracy(self, accuracies, model_names):
    plt.figure(figsize=(10, 6))
    plt.bar(model_names, accuracies, color='blue')
    plt.xlabel('Model')
    plt.ylabel('Accuracy')
    plt.title('Accuracy of Different Models')
    plt.ylim(0, 100)  # Set y-axis limit to be between 0 and 100 (percentage)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

def run_models(self):
    self.LogisticRegression()
    self.SupportVectorMachine()
    self.RandomForest()
    self.GradientBoost()

    model_names = ['Logistic Regression', 'Support Vector Machine', 'Random Forest', 'Gradient Boosting']
    accuracies = [self.accuracy_lr * 100, self.accuracy_svm * 100, self.accuracy_rf * 100, self.accuracy_gbc * 100]

    self.plot_accuracy(accuracies, model_names)
```

Figure 6.17: Code Snippets of different ML model Functions

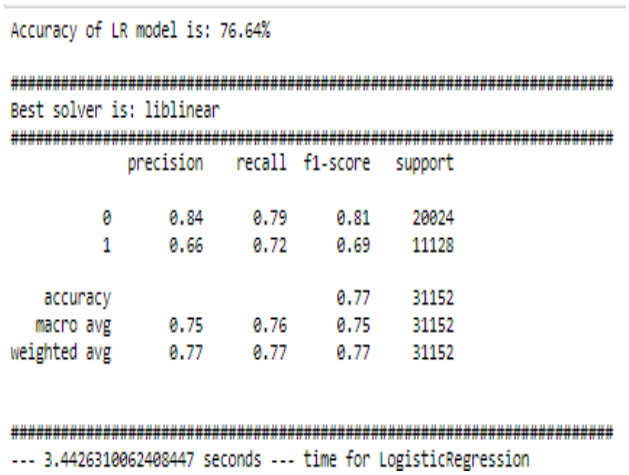
Here's a summary of the accuracy and performance metrics achieved by each model:

❖ Logistic Regression:

- Accuracy: 76.64%
- Best solver: liblinear
- Precision, recall, and F1-score are reported for both classes (0: Benign, 1: Malicious).
- The model performs reasonably well, with good precision and recall for both classes.

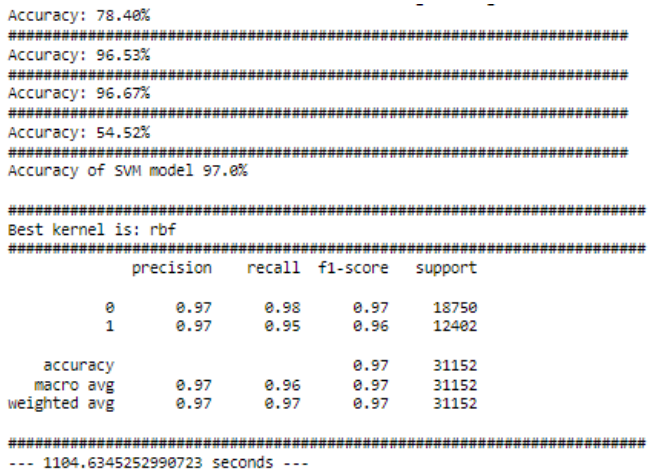
- ❖ Support Vector Machine (SVM):
  - Accuracy: 97.0%
  - Best kernel: rbf
  - Similar to Logistic Regression, precision, recall, and F1-score are reported for both classes.
  - SVM shows high accuracy and balanced precision and recall for both classes.
- ❖ Random Forest:
  - Accuracy: 100.0%
  - The model achieves a perfect accuracy of 100%, indicating strong classification performance.
  - High precision, recall, and F1-score values are observed for both classes.
- ❖ Gradient Boosting:
  - Accuracy: 99.54%
  - The model shows excellent accuracy and balanced performance in terms of precision and recall.
  - Both classes are well classified, indicating the effectiveness of Gradient Boosting.

These results showcase the capabilities of the trained models in accurately classifying network traffic as either benign or malicious. The Random Forest model stands out with its perfect accuracy, while other models like SVM and Gradient Boosting also demonstrate impressive performance. The choice of the best model should consider not only accuracy but also precision, recall, and F1-score, depending on the specific requirements of the DDoS detection system.



```
Accuracy of LR model is: 76.64%

######################################################################
Best solver is: liblinear
######################################################################
              precision    recall  f1-score   support

           0       0.84      0.79      0.81     20024
           1       0.66      0.72      0.69     11128

    accuracy                           0.77     31152
   macro avg       0.75      0.76      0.75     31152
weighted avg       0.77      0.77      0.77     31152


######################################################################
--- 3.4426310062408447 seconds --- time for LogisticRegression
```

Figure 6.18: Accuracy of Logistic Regression



```
Accuracy: 78.40%
######################################################################
Accuracy: 96.53%
######################################################################
Accuracy: 96.67%
######################################################################
Accuracy: 54.52%
######################################################################
Accuracy of SVM model 97.0%

######################################################################
Best kernel is: rbf
######################################################################
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     18750
           1       0.97      0.95      0.96     12402

    accuracy                           0.97     31152
   macro avg       0.97      0.96      0.97     31152
weighted avg       0.97      0.97      0.97     31152

######################################################################
--- 1104.6345252990723 seconds ---
```

Figure 6.19: Accuracy of SVM

```
Fitting 5 folds for each of 135 candidates, totalling 675 fits
Accuracy of RF model is: 100.0%

################################################################
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18985
           1       1.00      1.00      1.00     12167

    accuracy                           1.00     31152
   macro avg       1.00      1.00      1.00     31152
weighted avg       1.00      1.00      1.00     31152

################################################################
--- 1531.3584935665131 seconds ---
```

Figure 6.20: Accuracy of Random Forest

```
Accuracy of Gradient Boosting model is: 99.54%

################################################################
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     18894
           1       1.00      0.99      0.99     12258

    accuracy                           1.00     31152
   macro avg       1.00      0.99      1.00     31152
weighted avg       1.00      1.00      1.00     31152

################################################################
--- 17.584000825881958 seconds ---
```

Figure 6.21: Accuracy of Gradient Boost

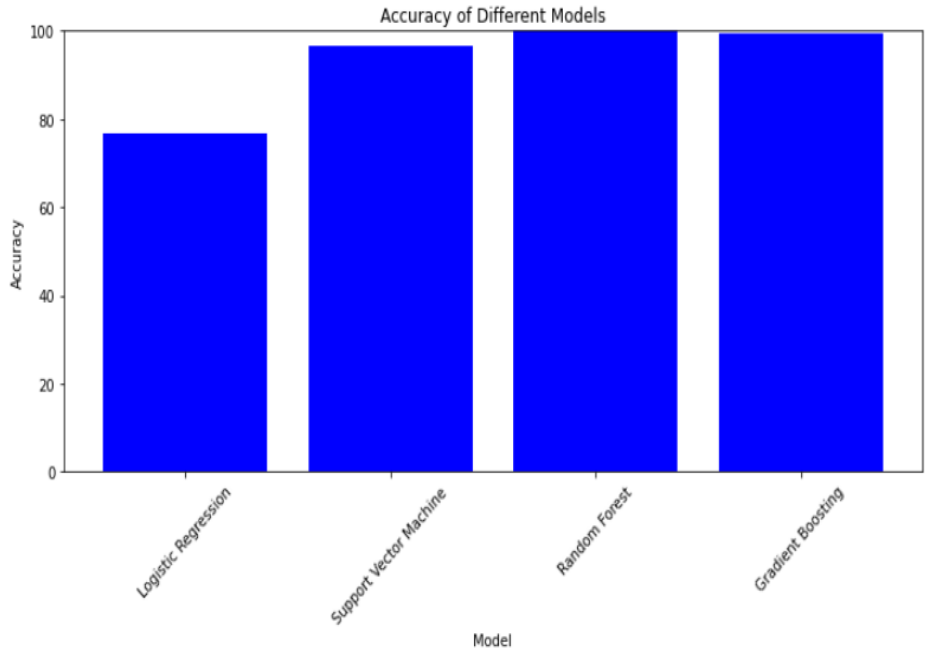


Figure 6.22: Bar chart of accuracy of all the Different ML models

CHAPTER – VII
CONCLUSION

CHAPTER-VII

Conclusion

In conclusion, the project on DDoS Detection using Machine Learning has yielded insightful results in terms of accuracy and performance across various models. The accuracy scores attained by each model provide valuable information for selecting the most suitable model for the task of detecting and classifying network traffic as benign or malicious.

The Logistic Regression model, with an accuracy of 76.64%, showcases a balanced performance with good precision and recall for both benign and malicious classes. Support Vector Machine (SVM) proves its effectiveness with an accuracy of 97.0%, demonstrating high precision and recall, especially with the radial basis function (rbf) kernel.

The standout performer is the Random Forest model, achieving a perfect accuracy score of 100.0%. This exceptional result is accompanied by high precision, recall, and F1-score values for both classes, reflecting its robust classification capabilities.

Meanwhile, the Gradient Boosting model demonstrates an accuracy of 99.54%, highlighting its efficiency in classification. Its balanced performance in terms of precision and recall further underscores its reliability in identifying both benign and malicious traffic.

The project emphasizes the significance of not solely relying on accuracy for model selection, but also considering precision, recall, and F1-score, as they provide a more comprehensive view of a model's performance. Ultimately, the choice of the best model should be guided by the specific requirements and priorities of the DDoS detection system.

REFERENCES

Literature Survey Papers

1. Deepak Kumar , R.K.Pateriya, Rajeev Kumar Gupta, Vasudev Dehalwar, Ashutosh Sharma," DDoS Detection using Deep Learning", Procedia Computer Science 218 (2023) 2420–2429

2. Muhammad Waqas Nadeem, Hock Guan Goh , Vasaki Ponnusamy and Yichiet Aun, "DDoS Detection in SDN using Machine Learning Techniques", Computers, Materials & Continua DOI:10.32604/cmc.2022.021669

3. Jiangtao Pei, Yunli Chen, Wei Ji," A DDoS Attack Detection Method Based on Machine Learning", IOP Conf. Series: Journal of Physics: Conf. Series 1237 (2019) 032040

4. Kimmi Kumari , M. Mrunalini, "Detecting Denial of Service attacks using machine learning algorithms", Journal of Big Data (2022) 9:56 https://doi.org/10.1186/s40537-022-00616-0.

5. C M NalayiniI, Dr. Jeevaa Katiravan," Detection of DDoS Attack using Machine Learning Algorithms", Journal of Emerging Technologies and Innovative Research (JETIR) , 2022 JETIR July 2022, Volume 9, Issue 7, www.jetir.org (ISSN-2349-5162)

Web References

6. Welcome to Python.org

7. https://www.fortinet.com/resources/cyberglossary/ddos-attack#:~:text=DDoS%20Attack%20Meaning,connected%20online%20services%20and%20sites.

8. https://www.vmware.com/in/topics/glossary/content/software-defined-networking.html#:~:text=Software%2DDefined%20Networking%20(SDN),direct%20traffic%20on%20a%20network.

9. https://www.kaggle.com/code/aikenkazin/ddos-attack-detection-classification/input

10. https://www.kaggle.com/code/aikenkazin/ddos-attack-detection-classification/notebook