A Project report on

DETECTING DDOS ATTACKS USING MACHINE-LEARNING ALGORITHMS

Submitted in partial fulfillment for the award of the degree of

M.Tech (Software Engineering)

by

KAMIREDDY SAI GOUTHAM

19MIS0354

Under the guidance of

Prof. Chadrasegar. T

SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

VIT, Vellore.

MAY, 2024

DETECTING DDOS ATTACKS USING MACHINE-LEARNING ALGORITHMS

Submitted in partial fulfillment for the award of the degree of

M.Tech (Software Engineering)

by

KAMIREDDY SAI GOUTHAM

19MIS0354

Under the guidance of

Prof. Chadrasegar. T

SCHOOL OF COMPUTER SCIENCE ENGINEERING AND INFORMATION SYSTEMS

VIT, Vellore.

MAY, 2024

DECLARATION

I here by declare that the thesis entitled "DETECTING DDOS ATTACKS USING MACHINE-LEARNING ALGORITHMS" submitted by me, for the award of the degree of M.Tech (Software Engineering) to VIT is a record of bonafide work carried out by me under the supervision of CHANDRASEGAR.T.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore
Date: 25-04-2024

Signature of the Candidate

CERTIFICATE

This is to certify that the thesis entitled “DETECTING DDOS ATTACKS USING MACHINE-LEARNING ALGORITHMS” submitted by KAMIREDDY SAI GOUTHAM 19MIS0354, School of Computer Science Engineering And Information Systems, Vellore Institute of Technology, Vellore for the award of the degree M.Tech (Software Engineering) is a record of bonafide work carried out by him under my supervision during the period, 03.01.2024 to 25.04.2024, as per the VIT code of academic and research ethics.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The thesis fulfills the requirements and regulations of the University and in my opinion meets the necessary standards for submission.

Place: Vellore **Signature of the Guide**

Date: 25-04-2024

Internal Examiner **External Examiner**

Head of the Department
Software Engineering

ACKNOWLEDGEMENTS

It is my pleasure to express with deep sense of gratitude to PROF CHANDRASEGAR T, Assistant Professor Sr. Grade 2, School of Computer Science Engineering and Information Systems, Vellore Institute of Technology, for his/her constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavour. My association with him / her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Information Security.

I would like to express my gratitude to DR.G.VISWANATHAN, Chancellor VELLORE INSTITUTE OF TECHNOLOGY, VELLORE, MR. SANKAR VISWANATHAN, DR. SEKAR VISWANATHAN, MR.G V SELVAM, Vice – Presidents VELLORE INSTITUTE OF TECHNOLOGY, VELLORE, Dr. V. S. Kanchana Bhaaskaran, Vice – Chancellor, DR. Partha Sharathi Mallick, Pro-Vice Chancellor and Dr. S. Sumathy, Dean, School of Computer Science Engineering And Information Systems,, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Neelu Khare, HoD/Professor, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Name of the student

KAMIREDDY SAI GOUTHAM

Executive Summary

With the increase in technological advancement, especially the internet, there come various kinds of network attacks. DOS attacks pose one of the most challenging security threats in today's generation of internet. In addition, the Distributed Denial of Service (DDoS) attacks are of a particular concern whose impact can be proportionally severe. The challenging part of this attack is that it comes with little or no advance warning and could easily exhaust away the communication or computation resources of the victim in a short span of time. These attacks are becoming advanced day-by-day and are increasing in number thus making it difficult to detect and counter such attacks. Using the numerous Machine Learning techniques for effective detection of DDoS flooding attack such as K-Nearest Neighbors, SGD, Multi-layer Perceptron, Logistic Regression, Naive Bayes, Support Vector Machine, XGBoost, Decision Tree, Quadratic discriminant and deep learning techniques such as DNN etc. A detailed comparative analysis of the aforementioned algorithms is performed and is evaluated based on the accuracy metrics.

CONTENTS

ACKNOWLEDGEMENT i

EXECUTIVE SUMMARY ii

TABLE OF CONTENTS

LIST OF FIGURES vi

LIST OF TABLES vii

ABBREVIATIONS viii

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE 08

1.2 MOTIVATION 08

1.3 BACKGROUND 09

CHAPTER 2

PROJECT DESCRIPTION AND GOALS 10

CHAPTER 3

TECHNICAL SPECIFICATION

3.1 HARDWARE REQUIREMENTS 11

3.2 SOFTWARE REQUIREMENTS 11

3.3 BUDGET 12

CHAPTER 4

DESIGN APPROACH AND DETAILS

4.1 PROPOSED METHODOLOGY 13

4.2 SYSTEM ARCHITECTURE 15

4.3 MODULE DESCRIPTIONS 16

CHAPTER 5

SCHEDULE, TASKS AND MILESTONES 18

CHAPTER 6

PROJECT DEMONSTRATION 26

6.1 DATA SET 27

6.2 SAMPLE CODE 29

6.3 SAMPLE OUTPUT 45

6.4 TEST PLAN & DATA VERIFICATION 53

CHAPTER 7

COST ANALYSIS / RESULT & DISCUSSION 55

SUMMARY 64

REFERENCES 65

APPENDIX 66

LIST OF FIGURES

Figure No.	Title	Page No.
4.1	PROPOSED METHODOLOGY	14
4.2	SYSTEMARCHITECTURE	15
5.1	SCHEDULE	18

LIST OF TABLES

Table No.	Title	Page No.
3.3	BUDGET	12
5.2	TASKS	19
5.3	LITERATURE SURVEY	20
6.4	DATA VERIFICATION	54

ABBREVIATIONS

DDOS	Distributed Denial of Service
DNN	Deep Neural Network
KNN	K- Nearest Neighbor
SVM	Support Vector Machine
RF	Random Forest
DT	Decision Tree

Chapter -1

INTRODUCTION

1.1 OBJECTIVE

The objective is to address the increasing threat of Distributed Denial of Service (DDoS) attacks on networks, which pose significant challenges due to their ability to exhaust communication and computation resources rapidly with little or no warning. The focus is on developing effective detection methods leveraging machine learning and deep learning techniques to mitigate the impact of such attacks.

1.2 MOTIVATION

The motivation behind the study of DDoS attack detection using machine learning techniques stems from the critical need to enhance the security posture of networked systems and mitigate the impact of cyber threats. Traditional methods of DDoS detection, such as rule-based approaches or signature-based intrusion detection systems, may not be sufficient to combat the evolving nature of DDoS attacks. By training machine learning models on labeled datasets containing both normal and attack traffic, it is possible to develop algorithms capable of recognizing DDoS attacks with high accuracy. The comparative analysis of various machine learning algorithms and deep learning techniques such as Deep Neural Networks (DNN), aims to identify the strengths and weaknesses of each approach in the context of DDoS attack detection. By evaluating these algorithms based on accuracy metrics, researchers can determine which techniques are most effective for mitigating the threat of DDoS attacks and inform the development of robust defense mechanisms.

1.3 BACKGROUND:

With the rapid growth of technology and the increasing reliance on the internet for various tasks, network security has become a critical concern. Among the various network attacks, Distributed Denial of Service (DDoS) attacks pose a significant threat to the availability and stability of online services. In a DDoS attack, multiple compromised systems, often distributed across the globe, are used to flood a target system or network with an overwhelming amount of traffic. This flood of traffic can consume all available resources, making the target system or network inaccessible to legitimate users. Traditional DDoS attacks typically involve flooding a target with high volumes of traffic, exploiting vulnerabilities in network protocols or application layers. However, as technology advances, so do the techniques used in DDoS attacks. Attackers constantly evolve their tactics to bypass traditional defense mechanisms and maximize the impact of their attacks. Machine learning techniques have emerged as promising tools for DDoS detection, leveraging the ability to analyze large volumes of network data and identify patterns indicative of malicious activity.

Chapter-2

PROJECT DESCRIPTION AND GOALS

PROJECT DESCRIPTION:

The project aims to develop and implement a Machine Learning (ML) based solution for detecting and mitigating Distributed Denial of Service (DDoS) attacks on computer networks. DDoS attacks pose a significant threat to network infrastructure by overwhelming systems with fake traffic, leading to service disruption and potential financial losses. With the advancement of technology, these attacks are becoming increasingly sophisticated, making traditional detection methods ineffective.To address this challenge, the project focuses on leveraging ML techniques such as K-Nearest Neighbors, Logistic Regression, and Deep Learning to analyze network traffic patterns and identify anomalous behavior indicative of DDoS attacks in real-time. By training ML models on historical data and continuously updating them with new information, the system aims to achieve high accuracy in detecting DDoS attacks while minimizing false positives.

GOALS:

The primary goal of the project is to identify the most effective ML algorithm for detecting DDoS attacks in real-time. This involves:

- Evaluating the performance of different ML algorithms, including K-Nearest Neighbors, Logistic Regression, and Deep Learning, in accurately identifying DDoS attacks and distinguishing them from legitimate traffic.
- Developing a prototype implementation of the ML-based DDoS detection system that can be deployed in production environments.
- Ultimately, the project aims to contribute to the development of robust and reliable solutions for protecting computer networks from DDoS attacks, ensuring uninterrupted service availability and safeguarding against potential security threats.

Chapter- 3

TECHNICAL SPECIFICATION

3.1 HARDWARE REQUIREMENTS

The hardware requirements may serve as the basis for a contract for the implementation of the system and should therefore be a complete and consistent specification of the whole system. They are used by software engineers as the starting point for the system design. It shows what the system does and not how it should be implemented.

PROCESSOR	:	DUAL CORE 2 DUOS
RAM	:	4GB RAM
MONITOR	:	15” COLOR
HARD DISK	:	250 GB

3.2 SOFTWARE REQUIREMENTS

The software requirements document is the specification of the system. It shouldinclude both a definition and a specification of requirements. It is a set of what the system should do rather than how it should do it. The software requirements provide abasis for creating the software requirements specification. It is useful in estimating cost, planning team activities, performing tasks and tracking the team’s and tracking the team’s progress throughout the development activity.

FRONT END	:	HTML, CSS, PYTHON
BACK END	:	FLASK
IDE	:	JUPTER , VS CODE

3.3BUDGET

Procured Items/Components for the Project work	**Total Cost**
LAPTOP	72000
PRINTER	6000
INTERNET BILL	2500
Total Budget (INR)	80500

Chapter 4

DESIGN APPROACH AND DETAILS

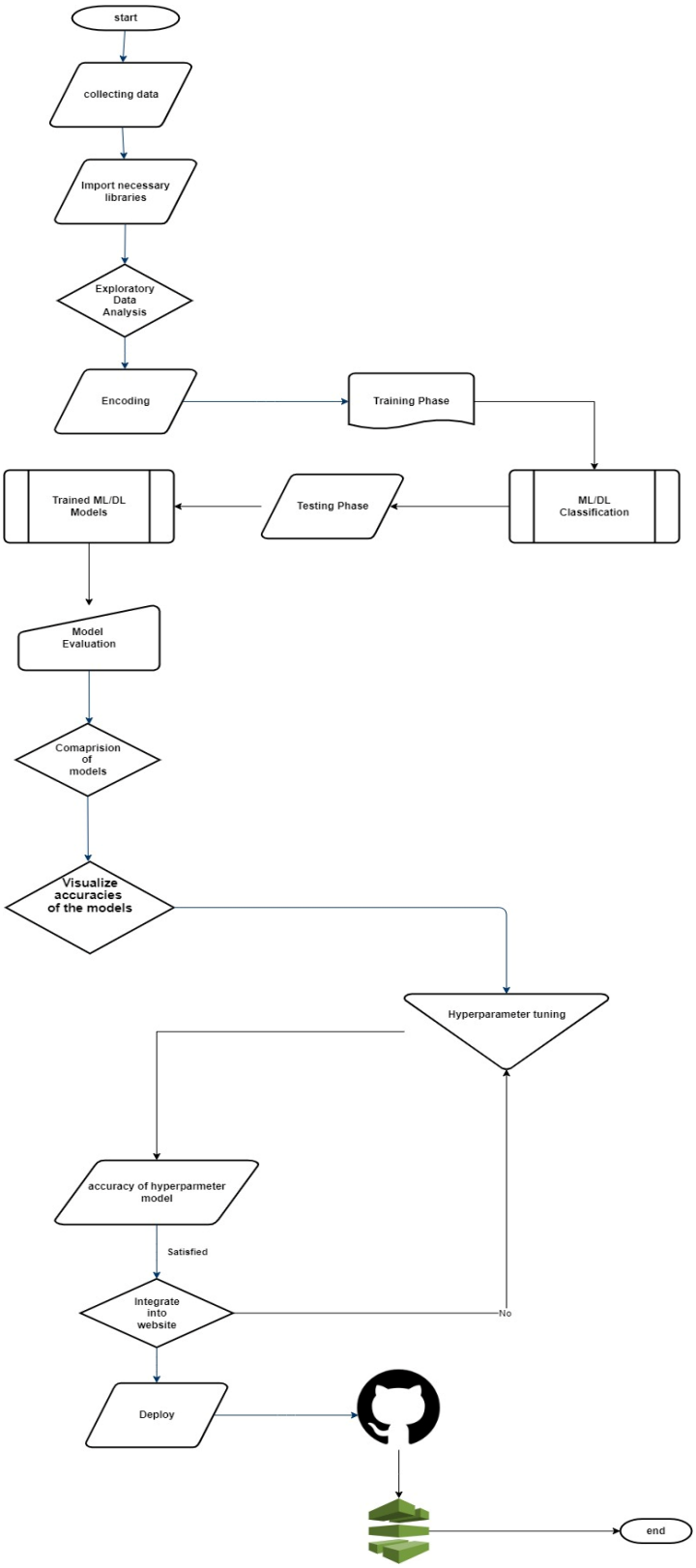
4.1 PROPOSED METHODOLOGY

In this ML-based DDoS prediction project, the first step involves acquiring and understanding the DDoS attack dataset. After obtaining the dataset, preprocessing tasks are carried out, including handling missing values, normalizing numerical features, and encoding categorical features like Source IP, Destination IP, and Port Number. Subsequently, exploratory data analysis (EDA) is conducted to analyze feature distributions, correlations, and identify outliers, facilitating a deeper understanding of the dataset. Moving forward, a selection of baseline classifiers such as DNN, KNN, SVM, Decision Tree, Naive Bayes, and others is made, followed by training each classifier on the prepared dataset. Performance evaluation of these models is then conducted using various metrics like accuracy, precision, recall, and F1-score.

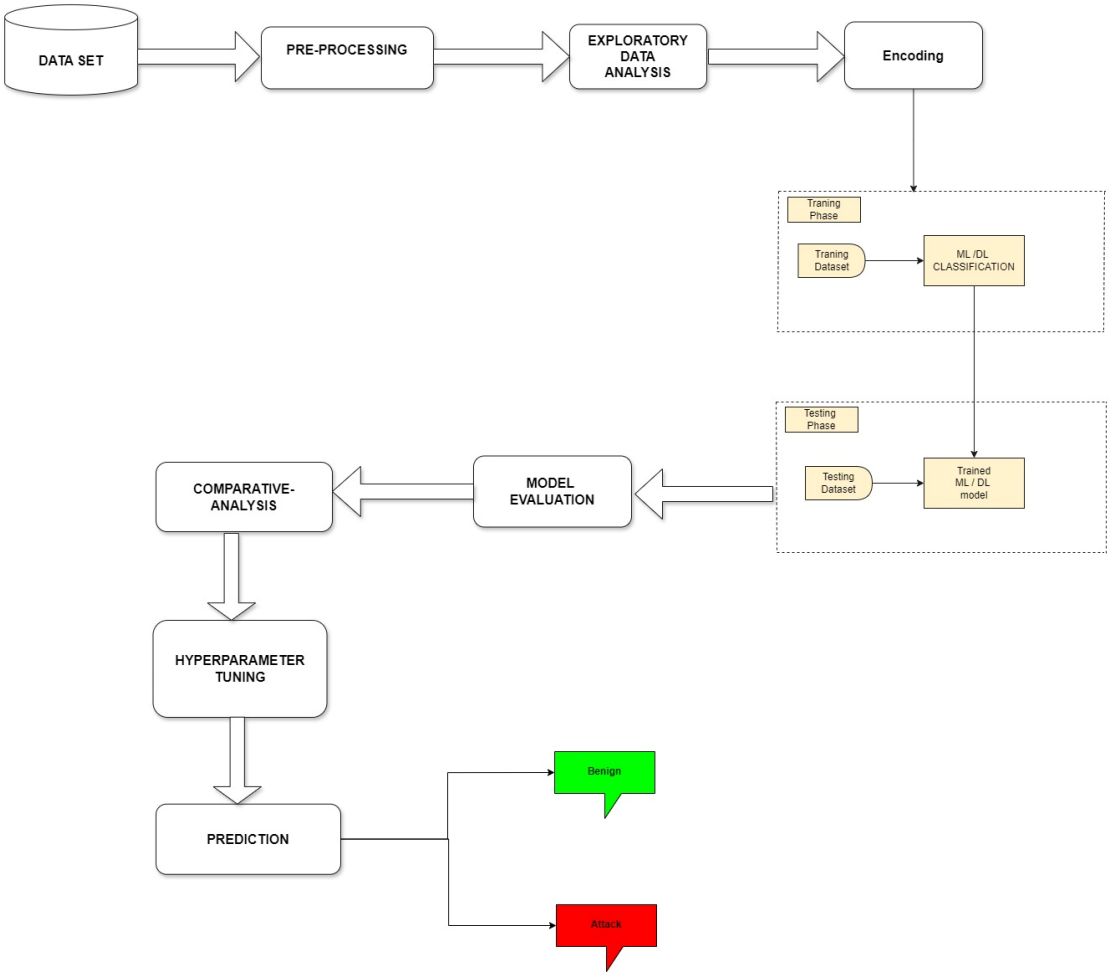
Following model evaluation, hyperparameter tuning techniques like grid search or random search are employed to fine-tune the most promising classifiers, optimizing their performance. Based on the evaluation results, the best-performing classifier is selected, and its performance is assessed using the testing dataset to ensure generalization to unseen data. Subsequently, the trained model is integrated into a practical system for DDoS prediction, incorporating a user interface or API for interaction.

In addition to the ML-based DDoS prediction system, the project involves hosting the system on a cloud platform and implementing Continuous Integration/Continuous Deployment (CI/CD) practices. After finalizing the trained model, the system is deployed on a cloud infrastructure to leverage its scalability, reliability, and accessibility. Whenever changes are made to the codebase or the model, the CI/CD pipeline automatically triggers the necessary steps for testing, building, and deploying the updated system to the cloud environment. This ensures that the prediction system remains up-to-date and operational, with minimal manual intervention required. Additionally, monitoring tools are integrated into the CI/CD pipeline to track system performance and detect any issues that may arise during deployment. This combination of cloud hosting and CI/CD practices ensures that the ML-based DDoS prediction

system is efficiently managed, updated, and maintained, providing reliable protection against DDoS attacks.



4.2 SYSTEM ARCHITECTURE



4.3 MODULE DESCRIPTIONS

Data Collection:

- Collected the data set from Mendeley data which specific SDN data set generated by using mininet emulator and used for traffic classification by machine learning and deep learning algorithms
- The dataset includes features such as Packet_count, byte_count, Switch-id, duration_sec, duration_nsec, Source IP, Destination IP, Port Number, tx_bytes, rx_bytes, dt field, and calculated features like Byte Per Flow, Packet Per Flow, Packet Rate, number of Packet_ins messages, Flow entries of switch, tx_kbps, rx_kbps, and Port Bandwidth.
- The dataset has a binary output feature indicating benign (0) or malicious (1) traffic.

Pre-processing:

- Handle missing values, if any, using appropriate techniques like imputation or deletion.
- Normalize or standardize numerical features to ensure uniform scale.
- Encode categorical features such as Source IP, Destination IP, and Port Number using techniques like one-hot encoding or label encoding.
- Split the dataset into training and testing sets.

Exploratory Data Analysis (EDA):

- Analyze the distribution of features within benign and malicious traffic classes.
- Explore correlations between features and the target variable.
- Visualize important features using plots like histograms, box plots, and scatter plots.
- Identify any outliers and determine whether they should be removed or treated.

Encoding:

- For the training phase, apply one-hot encoding to categorical features if necessary.
- For the testing phase, ensure consistency in encoding with the training phase.

Model Evaluation:

- Train baseline classifiers including DNN, KNN, SVM, Decision Tree, Naive Bayes, Quadratic Discriminant Analysis, SGD, Logistic Regression, and XGBoost using the training dataset.
- Evaluate the performance of each model using appropriate metrics like accuracy, precision, recall, F1-score, and ROC-AUC.
- Use techniques like cross-validation to assess model generalization.

Comparative Analysis:

- Compare the performance of different classifiers based on evaluation metrics.
- Identify the strengths and weaknesses of each model in predicting benign and malicious traffic.

Hyperparameter Tuning:

- Fine-tune hyperparameters for the most promising classifiers using techniques like grid search or random search.
- Optimize hyperparameters to improve model performance.

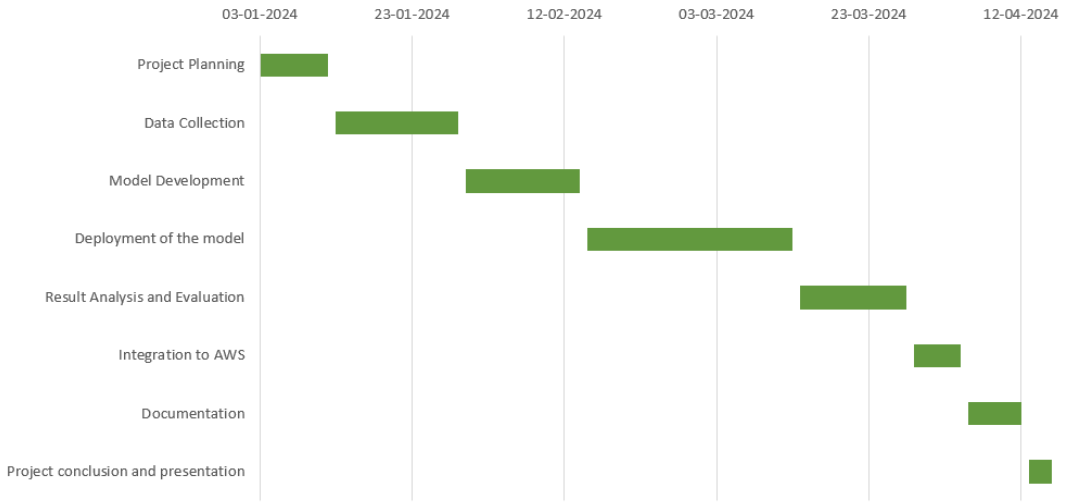
Prediction:

- Use the best-performing classifier to make predictions on unseen data from the testing set.
- Evaluate the final model's performance on the testing set.

Chapter 5

SCHEDULE, TASKS AND MILESTONES

5.1 SCHEDULE



5.2 TASKS

S.NO	Description of the Taks	Guide Remarks
1	• Define objectives • Gather data sources • Set up environment	Research some bunch of papers and maintain a detailed project plan
2	• Data acquisition • Data preprocessing • Data labeling	OK
3	• Model selection • Model training • Model evaluation	Try at least one deep learning model for this project.
4	• Implementation • Testing	OK
5	• Integrating to AWS	Ok
6	• Report preparation	Ok

MILESTONES:

5.3 LITERATURE SURVEY

S.NO	Title	Merits	De-merits
1	Evaluating Machine Learning Algorithms for Detecting DDoS Attacks	• Comprehensive evaluation of machine learning algorithms. • Valuable contribution to DDoS attack detection research.	• Limited discussion on real-world applicability. • Potential bias towards certain algorithms or datasets.
2	Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models	• Integration of feature selection methods enhances model performance. • Contribution to DDoS attack detection in software-defined networks.	• Limited discussion on potential limitations or challenges. • Lack of comparison with alternative approaches. • Potential bias in feature selection or model selection process.
3	Performance Evaluation of Botnet DDoS Attack Detection Using Machine Learning	• Focus on evaluating Botnet DDoS attack detection. • Utilization of machine learning techniques for enhanced detection.	• Potential lack of generalizability to other types of DDoS attacks. • Limited discussion on scalability and efficiency of the proposed method.

4	Detection and Classification of DDoS Flooding Attacks on Software-Defined Networks: A Case Study for The Application of Machine Learning	• Focus on detection and classification of DDoS flooding attacks. • Application of machine learning in a software-defined network environment. • Collaboration among multiple authors potentially brings diverse perspectives.	• Lack of detailed discussion on specific machine learning techniques used. • Insufficient exploration of potential false positive/negative rates. • Possible limited generalizability beyond the specific case study presented.
5	Evaluating ML-based DDoS Detection with Grid Search Hyperparameter Optimization.	• Utilization of grid search hyperparameter optimization enhances model performance. • Contribution to evaluating machine learning-based DDoS detection in the context of network softwarization. • Presentation in a reputable IEEE conference adds credibility to the research	• Potential lack of exploration of alternative hyperparameter optimization methods. • Limited discussion on real-world implementation challenges. • Possible bias towards specific machine learning algorithms or datasets.

6	A Review on DDoS Attack Detection in SDN using ML	• Review format provides comprehensive overview of DDoS attack detection in SDN using ML. • Contribution to understanding current trends and advancements in the field.	• Potential lack of original research or empirical findings. • Limited discussion on practical implementation challenges. • Possible bias towards specific ML techniques or SDN architectures
7	A Hyper-Parameter Model for multiclass Detection of DDoS Attacks	• Introduction of a novel hyperparameter model for multiclass DDoS attack detection. • Contribution to mathematical statistics and engineering applications.	• Potential lack of comparison with existing DDoS detection methods. • Limited discussion on real-world implementation and practical considerations.
8	A Hyper-Parameter Model for multiclass Detection of DDoS Attacks	• Application of deep learning for DDoS attack detection, contributing to the advancement of AI in cybersecurity. • Focus on a contemporary and relevant topic in the field of cybersecurity.	• Potential lack of comparison with other detection methods. • Insufficient exploration of potential limitations or challenges in implementing deep learning for DDoS detection.

			• Limited discussion on the real-world effectiveness or scalability of the proposed approach.
9	Machine learning based DDoS attack detection from source side in cloud	• Focus on DDoS attack detection from the source side in cloud environments, addressing a specific and critical aspect of cybersecurity. • Utilization of machine learning techniques for detection, contributing to the field of AI-driven cybersecurity.	• Insufficient discussion on the real-world applicability and scalability of the proposed method. • Limited examination of potential false positive/negative rates and performance metrics.
10	Machine Learning based DDOS Detection	• Contribution to DDoS detection through the application of machine learning techniques. • Potential exploration of novel approaches or algorithm for detection.	• Insufficient discussion on the practical implementation or real-world effectiveness of the proposed approach. • challenges in deploying machine learning for DDoS detection.

Data Collection and Preprocessing:

- Gather datasets containing network traffic data, including both normal and DDoS attack traffic.
- Preprocess the datasets to clean, normalize, and extract features suitable for ML model training.

Model Selection and Experiment Setup:

- Select ML algorithms for DDoS detection, such as K-Nearest Neighbors, Logistic Regression, and Deep Learning models.
- Set up experiments to evaluate the performance of each algorithm using appropriate metrics.

Model Training and Evaluation:

- Train the selected ML models on the preprocessed datasets.
- Evaluate the performance of the trained models using metrics such as accuracy, precision, recall, and F1-score.

Comparison and Analysis:

- Compare the performance of different ML algorithms in identifying and mitigating DDoS attacks.
- Analyze the strengths and weaknesses of each algorithm in terms of accuracy, speed, scalability, and robustness.

Optimization and Fine-Tuning:

- Fine-tune the hyperparameters of the ML models to optimize their performance.
- Explore techniques for improving the detection accuracy and reducing false positives/negatives.

Integration and Deployment:

- Integrated the ML algorithms into a DDoS detection system.
- Deploy the system in AWS environment to detect DDoS attacks based on the dataset information.

Evaluation and Validation:

- Evaluate the performance of the deployed DDoS detection system under various attack scenarios and traffic conditions.
- Validate the effectiveness of the system in detecting and mitigating DDoS attacks while minimizing disruptions to network services.

Documentation and Reporting:

- Document the entire project process, including methodology, experiments, results, and conclusions.
- Prepared a final report summarizing the findings and recommendations.

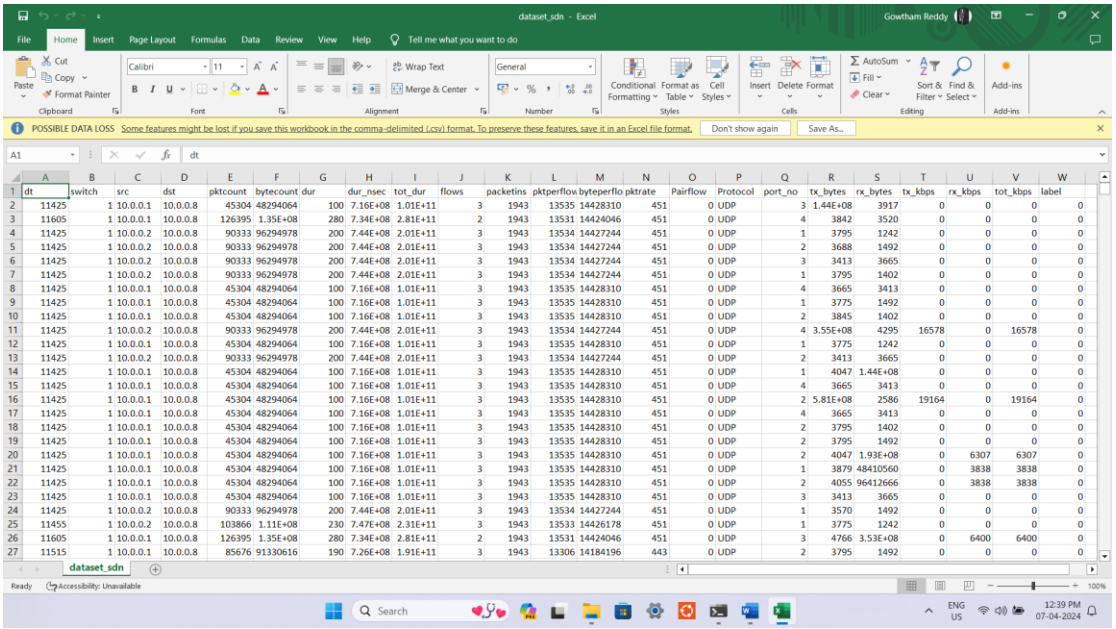
Chapter6

PROJECT DEMONSTRATION

The project focuses on addressing the growing threat of Distributed Denial of Service (DDoS) attacks, which pose significant challenges due to their ability to overwhelm internet resources with little warning. To combat this, various machine learning techniques were employed for effective detection, including K-Nearest Neighbors, SGD, Multi-layer Perceptron, Logistic Regression, Naive Bayes, Support Vector Machine, XGBoost, Decision Tree, Quadratic discriminant, and deep learning techniques like DNN. Using a dataset with traffic type classifications (benign or malicious) ,the dataset originally includes 23 features. The output feature is the last column of the dataset i.e. class label which classifies the traffic type to be benign or malicious. The malicious traffic is labelled as 1 and the benign traffic is labelled as 0. It has 104345 instances. The null values were observed in the rx_kbps and tot_kbps and were hence dropped for model development. The data processing steps were completed, including data preparation/cleaning, One Hot encoding, and normalization. After one hot encoding the dataframe had 103839 instances with 57 features and was fed into the model.A Deep Neural Network was used as the proposed model. The efficacy of our proposed model was observed to be higher than that of the baseline classifiers used. In addition to employing machine learning techniques for detection, the research also involved rigorous evaluation and comparison of these techniques to determine their effectiveness in identifying and mitigating DDoS attacks. This evaluation included assessing metrics such as accuracy, precision, recall, and F1-score to provide a comprehensive understanding of each model's performance.

Moreover, the deployment of the developed model on AWS not only facilitated accessibility but also allowed for scalability and real-time monitoring, enabling timely responses to emerging threats. Additionally, the research may have explored strategies for continuous model improvement and adaptation to evolving attack patterns to ensure long-term effectiveness in combating DDoS attacks.

6.1 DATA SET



dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps	tot_kbps	label
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	3	1.44E+08	3917	0	0	0	0
11605	1	10.0.0.1	10.0.0.8	126395	1.35E+08	280	7.34E+08	2.81E+11	2	1943	13531	14424046	451	0	UDP	4	3842	3520	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	1	3795	1242	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	2	3688	1492	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	3	3413	3665	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	1	3795	1402	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	4	3665	3413	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	3775	1492	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	3845	1402	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	4	3.55E+08	4295	16578	0	16578	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	3775	1242	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	2	3413	3665	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	4047	1.44E+08	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	4	3665	3413	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	5.81E+08	2586	19164	0	19164	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	4	3665	3413	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	3795	1402	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	3795	1492	0	0	0	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	4047	1.93E+08	0	6307	6307	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	1	3879	48410560	0	3838	3838	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	2	4055	96412666	0	3838	3838	0
11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	7.16E+08	1.01E+11	3	1943	13535	14428310	451	0	UDP	3	3413	3665	0	0	0	0
11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	7.44E+08	2.01E+11	3	1943	13534	14427244	451	0	UDP	1	3570	1492	0	0	0	0
11455	1	10.0.0.2	10.0.0.8	103866	1.11E+08	230	7.47E+08	2.31E+11	3	1943	13533	14426178	451	0	UDP	1	3775	1242	0	0	0	0
11605	1	10.0.0.1	10.0.0.8	126395	1.35E+08	280	7.34E+08	2.81E+11	2	1943	13531	14424046	451	0	UDP	3	4766	3.53E+08	0	6400	6400	0
11515	1	10.0.0.1	10.0.0.8	85676	91330616	190	7.26E+08	1.91E+11	3	1943	13306	14184196	443	0	UDP	2	3795	1492	0	0	0	0

The DDoS attack dataset is a SDN specific dataset that is generated by making use of the mininet emulator and is mainly used for the classification of traffic by numerous deep learning and machine learning algorithms. The process involved for the creation of the dataset includes the creation of ten topologies in mininet where the switches were connected to a single Ryu controller. The network simulation runs for the both the benign UDP, ICMP and TCP traffic as well as the collection of malicious traffic for TCP Syn attack, ICMP attack and UDP flood attackThe dataset includes 23 features in total where some of the data is extracted from the switches and others were calculated. Extracted features which are present in the dataset include: -

1. Packet_count – refers to the count of packets
2. byte_count – refers to the count of bytes in the packet
3. Switch-id – ID of the switch
4. duration_sec – packet transmission (in seconds)
5. duration_nsec – packet transmission (in nanoseconds)
6. Source IP – IP address of the source machine
7. Destination IP – IP address of the destination machine
8. Port Number – Port number of the application
9. tx_bytes – number of bytes transferred from the switch port
10. rx_bytes – number of bytes received on the switch port
11. dt field – shows the date and time which has been converted into a number and

the flow is monitored at a monitoring interval of 30 seconds

The calculated features present in the dataset include:

1. Byte Per Flow – byte count during a single flow
2. Packet Per Flow – packet count during a single flow
3. Packet Rate – number of packets transmitted per second and calculated by dividing the packet per flow by monitoring interval
4. number of Packet_ins messages – messages that are generated by the switch and is sent to the controller
5. Flow entries of switch – entries in the flow table of a switch which is used to match and process packets
6. tx_kbps – Speed of packet transmission (in kbps)
7. rx_kbps - Speed of packet reception (in kbps)
8. Port Bandwidth – Sum of tx_kbps and rx_kbps

The output feature is the last column of the dataset i.e. class label which classifies the traffic type to be benign or malicious. The malicious traffic is labelled as 1 and the benign traffic is labelled as 0. The simulation of the network was run for approximately 250 minutes and 1,04,345 instances of data were collected and recorded. Further, the simulation was run for a given interval to collect more instances of data.

6.2 SAMPLE CODE:

```
import joblib

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import missingno as msno

from sklearn import preprocessing

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.layers import Dense

from sklearn import metrics

from sklearn.model_selection import cross_val_score

from sklearn.metrics import classification_report,confusion_matrix

from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

from sklearn.model_selection import StratifiedKFold

from sklearn.metrics import roc_curve

from sklearn import tree

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn.naive_bayes import CategoricalNB

from sklearn.linear_model import LogisticRegression

from sklearn.linear_model import SGDClassifier

from sklearn.ensemble import AdaBoostClassifier

import xgboost as xgb

from sklearn.decomposition import PCA

from mlxtend.plotting import plot_decision_regions

from sklearn.tree import export_graphviz

from six import StringIO

from IPython.display import Image

import graphviz

# %matplotlib inline

df = pd.read_csv('C:/Users/GOWTHAM REDDY/OneDrive/Desktop/DDOS-
Detection/dataset_sdn.csv')

df.head(10)

"""### Data Preprocessing

#### Dataset Dimensions

print("This Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))

"""#### Concise summary of dataset"""

df.info()

"""#### Descriptive statistics of dataset"""

df.describe()

"""#### heatmap of missing values"""

msno.matrix(df)

"""#### Count of null values in each feature"""
```

```
df.isnull().sum()

(df.isnull().sum()/df.isnull().count())*100

"""#### Drop rows with null values"""

df.dropna(inplace=True)

"""#### Info after handling Null Values"""

print(df.isnull().sum())

print("This Dataframe has {} rows and {} columns after removing null
values".format(df.shape[0], df.shape[1]))

"""#### Distribution of Target Class"""

malign = df[df['label'] == 1]

benign = df[df['label'] == 0]

print('Number of DDOS attacks that has occured
:',round((len(malign)/df.shape[0])*100,2),'%')

print('Number of DDOS attacks that has not occured
:',round((len(benign)/df.shape[0])*100,2),'%')

"""### Barplot of Target Class"""

# Let's plot the Label class against the Frequency

labels = ['benign','malign']

classes = pd.value_counts(df['label'], sort = True) / df['label'].count() *100

classes.plot(kind = 'bar')

plt.title("Label class distribution")

plt.xticks(range(2), labels)

plt.xlabel("Label")

plt.ylabel("Frequency %")

"""#### Pairplot of select features"""

sns.pairplot(df,hue="label",vars=['pktcount','flows','bytecount'])

"""#### Columns in the dataset"""
```

```
df.columns

"""#### Unique values in each column"""

print(df.apply(lambda col: col.unique()))

"""#### Numerical Features"""

numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O']

print("The number of numerical features is",len(numerical_features),"and they are :
\n",numerical_features)

categorical_features = [feature for feature in df.columns if df[feature].dtypes == 'O']

print("The number of categorical features is",len(categorical_features),"and they are :
\n",categorical_features)

"""#### Number of Unique values in the numerical features"""

# number of unique values in each numerical variable

df[numerical_features].nunique(axis=0)

"""#### Discrete numerical features"""

#discrete numerical features

discrete_feature = [feature for feature in numerical_features if
df[feature].nunique()<=15 and feature != 'label']

print("The number of discrete features is",len(discrete_feature),"and they are :
\n",discrete_feature)

df[discrete_feature].head(10)

"""#### Continuous features"""

continuous_feature=[feature for feature in numerical_features if feature not in
discrete_feature + ['label']]

print("The number of continuous_feature features is",len(continuous_feature),"and
they are : \n",continuous_feature)

"""### Exploratory Data Analysis

#### Plotting function definition

def countplot_distribution(col):
```

```
    sns.set_theme(style="darkgrid")

    sns.countplot(y=col, data=df).set(title = 'Distribution of ' + col)

def histplot_distribution(col):

    sns.set_theme(style="darkgrid")

    sns.histplot(data=df,x=col, kde=True,color="red").set(title = 'Distribution of ' +
col)

"""#### Visualize the distribution of Categorical features"""

## Lets analyse the categorical values by creating histograms to understand the
distribution

f = plt.figure(figsize=(8,20))

for i in range(len(categorical_features)):

    f.add_subplot(len(categorical_features), 1, i+1)

    countplot_distribution(categorical_features[i])

plt.show()

"""#### Visualize the quartiles of categorical features wrt total duration"""

for i in range(len(categorical_features)):

    g =
sns.catplot(data=df,x="tot_dur",y=categorical_features[i],kind="boxen").set(title =
categorical_features[i])

    g.fig.set_figheight(7)

    g.fig.set_figwidth(15)

"""#### Visualize the distribution of continuous features"""

## Lets analyse the continuous values by creating histograms to understand the
distribution

f = plt.figure(figsize=(20,90))

for i in range(len(continuous_feature)):

    f.add_subplot(len(continuous_feature), 2, i+1)

    histplot_distribution(continuous_feature[i])
```

```
plt.show()

"""#### Visualize the distribution of continuous features wrt packet count, protocol
and type of attack"""

## Relplot of log(variable)

import warnings

warnings.filterwarnings("ignore")

for feature in continuous_feature:

    data=df.copy()

    if 0 in data[feature].unique():

        pass

    else:

        data[feature]=np.log(data[feature])

        data['pktcount']=np.log(data['pktcount'])

        plt.figure(figsize=(20,20))

        sns.relplot(data=data,
x=data[feature],y=data['pktcount'],hue="Protocol",style="Protocol",

                    col="label",kind="scatter").set(title="logarithmic Relplot of feature : " +
feature)

"""#### Visualize the distribution of numerical discrete features"""

for feature in discrete_feature:

    plt.figure(figsize=(8,4))

    cat_num = df[feature].value_counts()

    sns.barplot(x=cat_num.index, y = cat_num).set(title = "Graph for "+feature,
ylabel="Frequency")

    plt.show()

def get_percentage_malign_protocols():

    arr = [x for x, y in zip(df['Protocol'], df['label']) if y == 1]
```

```
    perc_arr = []
    for i in ['UDP','TCP','ICMP']:
        perc_arr.append(arr.count(i)/len(arr) *100)
    return perc_arr
"""#### Distribution of protocols for malign attacks"""
fig1, ax1 = plt.subplots(figsize=[7,7])
ax1.pie(get_percentage_malign_protocols(), explode=(0.1, 0, 0), autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
ax1.legend(['UDP', 'TCP', 'ICMP'],loc="best")
plt.title('Distribution of protocols for malign attacks',fontsize = 14)
plt.show()
"""#### Checking for outliers in Packet count feature"""
fig, ax = plt.subplots(figsize=[10, 10])
sns.boxplot(
    data=df,
    x='pktcount',
    y='Protocol'
)
ax.set_title('Boxplot, Packet count for different protocols')
"""#### Heat map of correlation of features"""
non_numeric_columns = df.select_dtypes(exclude='number').columns
print("Non-numeric columns:", non_numeric_columns)
numeric_df = df.select_dtypes(include='number')
correlation_matrix = numeric_df.corr()
```

```
# Calculate the correlation matrix

correlation_matrix = numeric_df.corr()

# Set up the figure and axis

fig = plt.figure(figsize=(17, 17))

# Create a mask for the upper triangle

mask = np.zeros_like(correlation_matrix, dtype=bool)

mask[np.triu_indices_from(mask)] = True

# Set Seaborn theme

sns.set_theme(style="darkgrid")

# Create the heatmap

ax = sns.heatmap(correlation_matrix, square=True, annot=True, center=0, vmin=-1,
linewidths=0.5, annot_kws={"size": 11}, mask=mask)

# Rotate x-axis labels for better readability

ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')

# Show the plot

plt.show()

"""### Encoding categorical features"""

print("Features which need to be encoded are : \n" ,categorical_features)

df = pd.get_dummies(df, columns=categorical_features,drop_first=True)

print("This Dataframe has {} rows and {} columns after
encoding".format(df.shape[0], df.shape[1]))

#dataframe after encoding

df.head(10)

df.dtypes

"""### Split into Independent and dependent variables"""

#separating input and output attributes
```

```
x = df.drop(['label'], axis=1)

y = df['label']

"""### Normalizing features"""

ms = MinMaxScaler()

x = ms.fit_transform(x)

"""### Train-Test-Split [75-25]"""

X_train, X_test, y_train, y_test = train_test_split(x, y,test_size=0.3)

print(X_train.shape, X_test.shape)

#### Deep Neural Network

Classifier_accuracy = []

"""## Defining the Deep Neural Network"""

from keras.models import Sequential

from keras.layers import Dense

model = Sequential([

    Dense(64, activation='relu', input_shape=(56,)),

    Dense(64, activation='relu'),

    Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam',

              loss='binary_crossentropy',

              metrics=['accuracy'])

"""## Model fitting"""

# Fit the model

history_org = model.fit(

    X_train,
```

```
    y_train,
    batch_size=50,
    epochs=35,
    verbose=2,
    callbacks=None,
    validation_data=(X_test, y_test),
    shuffle=True
)
"""## Plotting Loss v/s Epochs"""
loss = history_org.history['loss']
val_loss = history_org.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g', label = 'Training Loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
plt.title('Loss v/s No. of epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
"""## Plotting Accuracy v/s Epochs"""
loss = history_org.history['accuracy']
val_loss = history_org.history['val_accuracy']
plt.plot(epochs, loss, 'g', label = 'Training accuracy')
plt.plot(epochs, val_loss, 'r', label = 'Validation accuracy')
plt.title('Accuracy Scores v/s Number of Epochs')
```

```
plt.xlabel('No. of Epochs')
plt.ylabel('Accuracy Score')
plt.legend()
plt.show()
"""## Model Evaluation"""
loss, accuracy = model.evaluate(X_test, y_test)
print('Accuracy of Deep neural Network : %.2f' % (accuracy*100))
Classifier_accuracy.append(accuracy*100)
"""#### K-Nearest Neighbor Classifier"""
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
y_pred = knn_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of KNN Classifier : %.2f" % (accuracy*100))
report = classification_report(y_test, y_pred)
print(report)
joblib.dump(knn_clf, 'knn_model.pkl')
"""#### SVM Classifier"""
svc_clf = SVC()
svc_clf.fit(X_train,y_train)
y_pred = svc_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SVM Classifier : %.2f" % (accuracy*100) )
```

```
report = classification_report(y_test, y_pred)
print(report)
joblib.dump(svc_clf, 'svm_model.pkl')
"""#### Decision Tree Classifier"""
dt_clf = DecisionTreeClassifier(max_depth=5)
dt_clf.fit(X_train,y_train)
y_pred = dt_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Decision Tree Classifier : %.2f" % (accuracy*100) )
report = classification_report(y_test, y_pred)
print(report)
joblib.dump(dt_clf, 'decision_tree_model.pkl')
"""#### Stochastic Gradient"""
sgd_clf=SGDClassifier(loss="hinge", penalty="l2")
sgd_clf.fit(X_train,y_train)
y_pred=sgd_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SGD Classifier : %.2f" % (accuracy*100))
report = classification_report(y_test, y_pred)
print(report)
joblib.dump(sgd_clf, 'stocahstic_gradient_model.pkl')
"""#### Logistic Regression"""
lr_clf = LogisticRegression()
```

```
lr_clf.fit(X_train,y_train)

y_pred=lr_clf.predict(X_test)

accuracy = metrics.accuracy_score(y_test, y_pred)

Classifier_accuracy.append(accuracy*100)

print("Accuracy of Logistic Regression Classifier : %.2f" % (accuracy*100))

report = classification_report(y_test, y_pred)

print(report)

joblib.dump(lr_clf, 'logistic_regression_model.pkl')

"""#### XGBoost Classifier"""

xgb_clf=xgb.XGBClassifier(eval_metric =
'error',objective='binary:logistic',max_depth=2, learning_rate=0.1)

xgb_clf.fit(X_train,y_train)

y_pred=xgb_clf.predict(X_test)

accuracy = metrics.accuracy_score(y_test, y_pred)

Classifier_accuracy.append(accuracy*100)

print("Accuracy of XGBoost Classifier : %.2f" % (accuracy*100))

report = classification_report(y_test, y_pred)

print(report)

joblib.dump(xgb_clf, 'xgboost_model.pkl')

# Assuming X_test is a NumPy array

first_row = X_test[100]

first_row_csv = ','.join(map(str, first_row))

print(first_row_csv)

"""### Comparitive analysis of models"""

Classifier_names = ["DNN", "KNN", "RBF_SVM", "Decision Tree","Naive
Bayes","Quadratic","SGD","Logistic Regression","XGBoost"]
```

```
# Assuming Classifier_names and Classifier_accuracy are lists or arrays

# Ensure both lists have the same length

min_length = min(len(Classifier_names), len(Classifier_accuracy))

Classifier_names = Classifier_names[:min_length]

Classifier_accuracy = Classifier_accuracy[:min_length]

df_clf = pd.DataFrame()

df_clf['name'] = Classifier_names

df_clf['Accuracy'] = Classifier_accuracy

df_clf = df_clf.sort_values(by=['Accuracy'], ascending=False)

df_clf.head(10)

sns.set(style="whitegrid",rc={'figure.figsize':(7,7)})

ax = sns.barplot(y="name", x="Accuracy", data=df_clf)

print(f"The best baseline Classifier is {df_clf.name[0]} with an accuracy of
{df_clf.Accuracy[0]}.")
```

App.py

```
from flask import Flask, render_template, request

import joblib

import numpy as np

app = Flask(__name__, template_folder='templates')

# Load the saved models

Deep_Neural_Network_model=joblib.load("dnn_model.pkl")

logistic_reg_model = joblib.load('logistic_regression_model.pkl')

knn_model = joblib.load('knn_model.pkl')

naive_bayes_model = joblib.load('naive_bayes_model.pkl')

svm_model = joblib.load('svm_model.pkl')

decision_tree_model = joblib.load('decision_tree_model.pkl')
```

```
xgboost_model = joblib.load('xgboost_model.pkl')

stocahstic_gradient_model=joblib.load('stocahstic_gradient_model.pkl')

@app.route('/', methods=['GET', 'POST'])

def index():

    predictions = {}

    if request.method == 'POST':

        input_data = request.form['input_data']

        input_data_list = [float(x.strip()) for x in input_data.split(',')]

        # Perform predictions using different models

        predictions['dnn'] = 'Benign' if make_prediction(input_data_list,
Deep_Neural_Network_model) == 0 else 'Malicious'

        predictions['logistic_reg'] = 'Benign' if make_prediction(input_data_list,
logistic_reg_model) == 0 else 'Malicious'

        predictions['knn'] = 'Benign' if make_prediction(input_data_list, knn_model) ==
0 else 'Malicious'

        predictions['naive_bayes'] = 'Benign' if make_prediction(input_data_list,
naive_bayes_model) == 0 else 'Malicious'

        predictions['svm'] = 'Benign' if make_prediction(input_data_list,
decision_tree_model) == 0 else 'Malicious'

        predictions['decision_tree'] = 'Benign' if make_prediction(input_data_list,
stocahstic_gradient_model) == 0 else 'Malicious'

        predictions['xgboost'] = 'Benign' if make_prediction(input_data_list,
xgboost_model) == 0 else 'Malicious'

        predictions['stocahstic'] = 'Benign' if make_prediction(input_data_list,
stocahstic_gradient_model) == 0 else 'Malicious'

        print("Predictions:", predictions)

        return render_template('index.html', predictions=predictions)

    return render_template('index.html')

def make_prediction(input_data, model):
```

```
    try:
        # Perform the prediction using the model
        input_data_reshaped = np.array(input_data).reshape(1, -1)
        prediction = model.predict(input_data_reshaped)
        print("Prediction:", prediction)
        return prediction[0]
    except ValueError:
 # Handle the case where input data cannot be converted to numbers
        print(ValueError)
        return None
if __name__ == '__main__':
    app.run(debug=True)
```

6.3 SAMPLE OUTPUT

Import Essential Modules

In [1]:
```
import joblib
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import CategoricalNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import AdaBoostClassifier
import xgboost as xgb
from sklearn.decomposition import PCA
from mlxtend.plotting import plot_decision_regions
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import graphviz
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('C:/Users/GOWTHAM REDDY/OneDrive/Desktop/DDOS- Detection/dataset_sdn.csv')
df.head(10)
```

Out[2]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx_bytes	rx_bytes	tx_kbps	rx_kbps	tot_kbps	label
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	143928631	3917	0	0.0	0.0	0
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	3842	3520	0	0.0	0.0	0
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1242	0	0.0	0.0	0
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	3688	1492	0	0.0	0.0	0
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	3413	3665	0	0.0	0.0	0
5	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	3795	1402	0	0.0	0.0	0
6	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	4	3665	3413	0	0.0	0.0	0
7	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	1	3775	1492	0	0.0	0.0	0
8	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	2	3845	1402	0	0.0	0.0	0
9	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	4	354583059	4295	16578	0.0	16578.0	0

10 rows × 23 columns

Data Preprocessing

Dataset Dimensions

In [3]:
```
print("This Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))
```

```
This Dataset has 104345 rows and 23 columns
```

Concise summary of dataset

In [4]:
```
df.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
```

Descriptive statistics of dataset

In [5]:
```
df.describe()
```

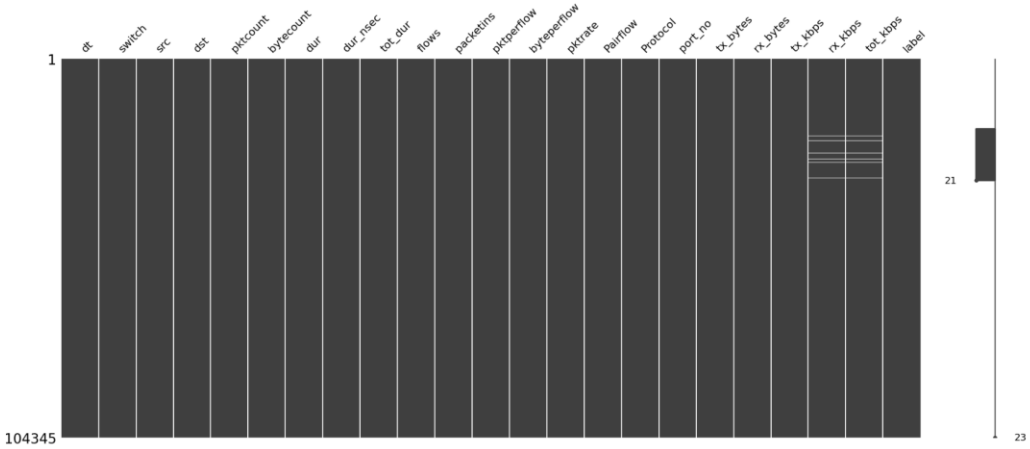
Out[5]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflow	pktrate	Pairflow	po
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	104345.000000	104345.0
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11	5.654234	5200.383468	6381.715291	4.716150e+06	212.210676	0.600987	2.3
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11	2.950036	5257.001450	7404.777808	7.560116e+06	246.855123	0.489698	1.0
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	2.000000	4.000000	-130933.000000	-1.464426e+08	-4365.000000	0.000000	1.0
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11	3.000000	1943.000000	29.000000	2.842000e+03	0.000000	0.000000	1.0
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11	5.000000	3024.000000	8305.000000	5.521680e+05	276.000000	1.000000	2.0
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11	7.000000	7462.000000	10017.000000	9.728112e+06	333.000000	1.000000	3.0
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12	17.000000	25224.000000	19190.000000	1.495387e+07	639.000000	1.000000	5.0

heatmap of missing values

In [6]:
```
msno.matrix(df)
```

Out[6]: <Axes: >



Count of null values in each feature

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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
```

Drop rows with null values

In [9]:
```
df.dropna(inplace=True)
```

Info after handling Null Values

In [10]:
```
print(df.isnull().sum())
print("This Dataframe has {} rows and {} columns after removing null values".format(df.shape[0], df.shape[1]))
```

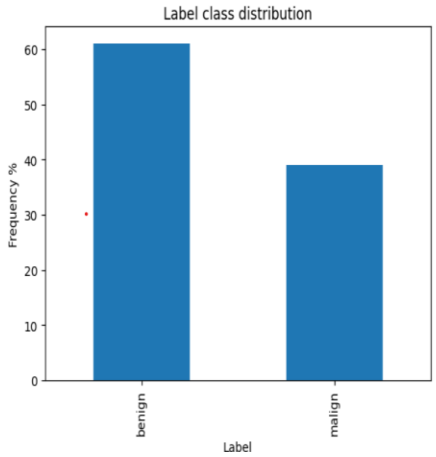
```
dt              0
switch          0
src             0
dst             0
pktcount        0
bytecount       0
dur             0
dur_nsec        0
tot_dur         0
flows           0
packetins       0
pktperflow      0
byteperflow     0
pktrate         0
Pairflow        0
Protocol        0
port_no         0
tx_bytes        0
rx_bytes        0
tx_kbps         0
rx_kbps         0
tot_kbps        0
label           0
dtype: int64
This Dataframe has 103839 rows and 23 columns after removing null values
```

Distribution of Target Class

```
In [11]: malign = df[df['label'] == 1]
         benign = df[df['label'] == 0]

         print('Number of DDOS attacks that has occured :',round((len(malign)/df.shape[0])*100,2),'%')
         print('Number of DDOS attacks that has not occured :',round((len(benign)/df.shape[0])*100,2),'%')

Number of DDOS attacks that has occured : 39.01 %
Number of DDOS attacks that has not occured : 60.99 %
```

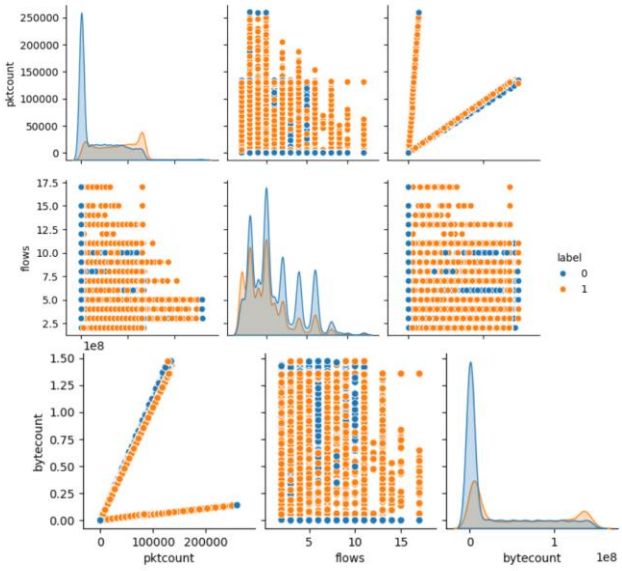
Barplot of Target Class

```
In [12]: # Let's plot the label class against the Frequency
         labels = ['benign','malign']
         classes = pd.value_counts(df['label'], sort = True) / df['label'].count() *100
         classes.plot(kind = 'bar')
         plt.title("Label class distribution")
         plt.xticks(range(2), labels)
         plt.xlabel("Label")
         plt.ylabel("Frequency %")

C:\Users\GOWTHAM REDDY\AppData\Local\Temp\ipykernel_9576\4157118356.py:3: FutureWarning: pandas.value_counts is deprecated and will be removed in a future version. Use pd.Series(obj).value_counts
() instead.
  classes = pd.value_counts(df['label'], sort = True) / df['label'].count() *100

Out[12]: Text(0, 0.5, 'Frequency %')
```



Pairplot of select features

```
In [13]: sns.pairplot(df,hue="label",vars=['pktcount','flows','bytecount'])

Out[13]: <seaborn.axisgrid.PairGrid at 0x1bc06b593f0>
```



Columns in the dataset

In [14]:
```
df.columns
```

Out[14]:
```
Index(['dt', 'switch', 'src', 'dst', 'pktcount', 'bytecount', 'dur',
       'dur_nsec', 'tot_dur', 'flows', 'packetins', 'pktperflow',
       'byteperflow', 'pktrate', 'Pairflow', 'Protocol', 'port_no', 'tx_bytes',
       'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps', 'label'],
      dtype='object')
```

Unique values in each column

In [15]:
```
print(df.apply(lambda col: col.unique()))
```

```
dt            [11425, 11605, 11455, 11515, 9906, 11335, 1157...
switch                          [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
src           [10.0.0.1, 10.0.0.2, 10.0.0.4, 10.0.0.10, 10.0...
dst           [10.0.0.8, 10.0.0.7, 10.0.0.3, 10.0.0.5, 10.0....
pktcount      [45304, 126395, 90333, 103866, 85676, 32914, 4...
bytecount     [48294064, 134737070, 96294978, 110721156, 913...
dur           [100, 280, 200, 230, 190, 73, 10, 250, 80, 260...
dur_nsec      [716000000, 734000000, 744000000, 747000000, 7...
tot_dur       [101000000000.0, 281000000000.0, 201000000000....
flows         [3, 2, 4, 5, 6, 7, 8, 11, 9, 10, 13, 15, 17, 1...
packetins     [1943, 1931, 1790, 1306, 1910, 2242, 2175, 110...
pktperflow    [13535, 13531, 13534, 13533, 13306, 13385, 0, ...
byteperflow   [14428310, 14424046, 14427244, 14426178, 14184...
pktrate       [451, 443, 446, 0, 288, 450, 448, 449, 455, 14...
Pairflow                                                [0, 1]
Protocol                                       [UDP, TCP, ICMP]
port_no                                         [3, 4, 1, 2, 5]
tx_bytes      [143928631, 3842, 3795, 3688, 3413, 3665, 3775...
rx_bytes      [3917, 3520, 1242, 1492, 3665, 1402, 3413, 429...
tx_kbps       [0, 16578, 19164, 12831, 7676, 10271, 2587, 16...
rx_kbps       [0.0, 6307.0, 3838.0, 6400.0, 7676.0, 10271.0,...
tot_kbps      [0.0, 16578.0, 19164.0, 6307.0, 3838.0, 6400.0...
label                                                   [0, 1]
dtype: object
```

Numerical Features

In [16]:
```
numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O']
print("The number of numerical features is",len(numerical_features),"and they are : \n",numerical_features)
```

```
The number of numerical features is 20 and they are :
 ['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']
```

In [17]:
```
categorical_features = [feature for feature in df.columns if df[feature].dtypes == 'O']
print("The number of categorical features is",len(categorical_features),"and they are : \n",categorical_features)
```

```
The number of categorical features is 3 and they are :
 ['src', 'dst', 'Protocol']
```

Number of Unique values in the numerical features

In [18]:
```
# number of unique values in each numerical variable
df[numerical_features].nunique(axis=0)
```

Out[18]:
```
dt              858
switch           10
pktcount       9044
bytecount      9270
dur             840
dur_nsec       1000
tot_dur        4183
flows            15
packetins       168
pktperflow     2092
byteperflow    2793
pktrate         446
Pairflow          2
port_no           5
tx_bytes      12257
rx_bytes      11623
tx_kbps        1800
rx_kbps        1730
tot_kbps       2259
label             2
```

Discrete numerical features

In [19]:
```
#discrete numerical features
discrete_feature = [feature for feature in numerical_features if df[feature].nunique()<=15 and feature != 'label']
print("The number of discrete features is",len(discrete_feature),"and they are : \n",discrete_feature)
```

```
The number of discrete features is 4 and they are :
 ['switch', 'flows', 'Pairflow', 'port_no']
```

In [20]:
```
df[discrete_feature].head(10)
```

Out[20]:

	switch	flows	Pairflow	port_no
0	1	3	0	3
1	1	2	0	4
2	1	3	0	1
3	1	3	0	2
4	1	3	0	3
5	1	3	0	1
6	1	3	0	4
7	1	3	0	1
8	1	3	0	2
9	1	3	0	4

Continuous features

In [21]:
```
continuous_feature=[feature for feature in numerical_features if feature not in discrete_feature + ['label']]
print("The number of continuous_feature features is",len(continuous_feature),"and they are : \n",continuous_feature)
```

```
The number of continuous_feature features is 15 and they are :
 ['dt', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'packetins', 'pktperflow', 'byteperflow', 'pktrate', 'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps']
```

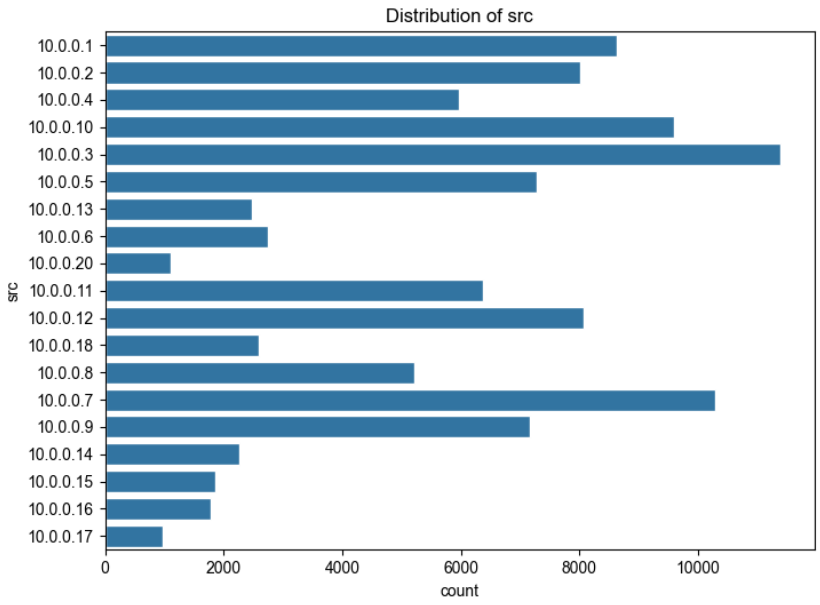
Exploratory Data Analysis

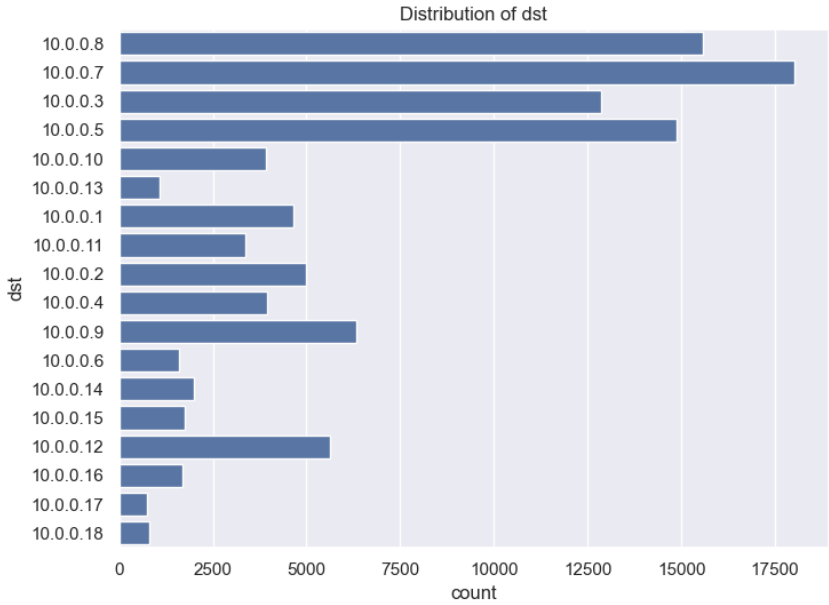
Plotting function definition

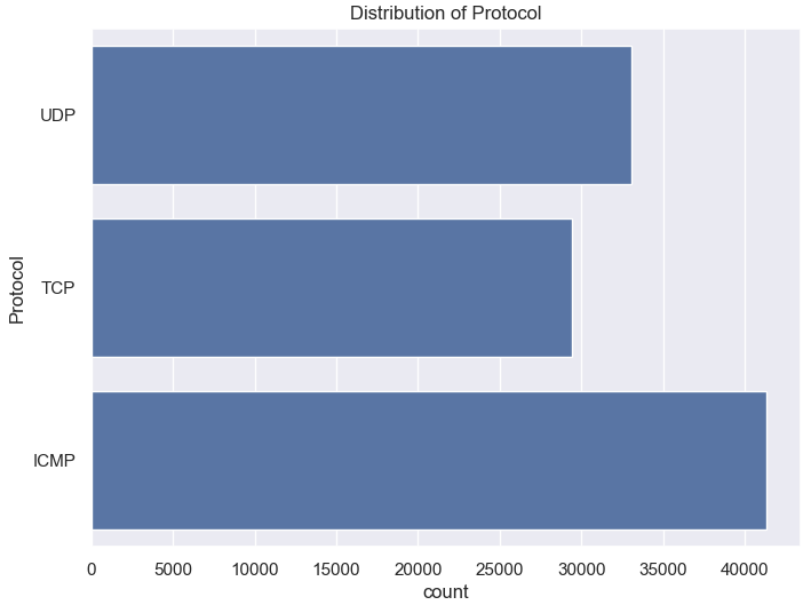
In [22]:
```
def countplot_distribution(col):
    sns.set_theme(style="darkgrid")
    sns.countplot(y=col, data=df).set(title = 'Distribution of ' + col)

def histplot_distribution(col):
    sns.set_theme(style="darkgrid")
    sns.histplot(data=df,x=col, kde=True,color="red").set(title = 'Distribution of ' + col)
```

Visualize the distribution of Categorical features

In [23]:
```
## lets analyse the categorical values by creating histograms to understand the distribution
f = plt.figure(figsize=(9,20))
for i in range(len(categorical_features)):
    f.add_subplot(len(categorical_features), 1, i+1)
    countplot_distribution(categorical_features[i])
plt.show()
```

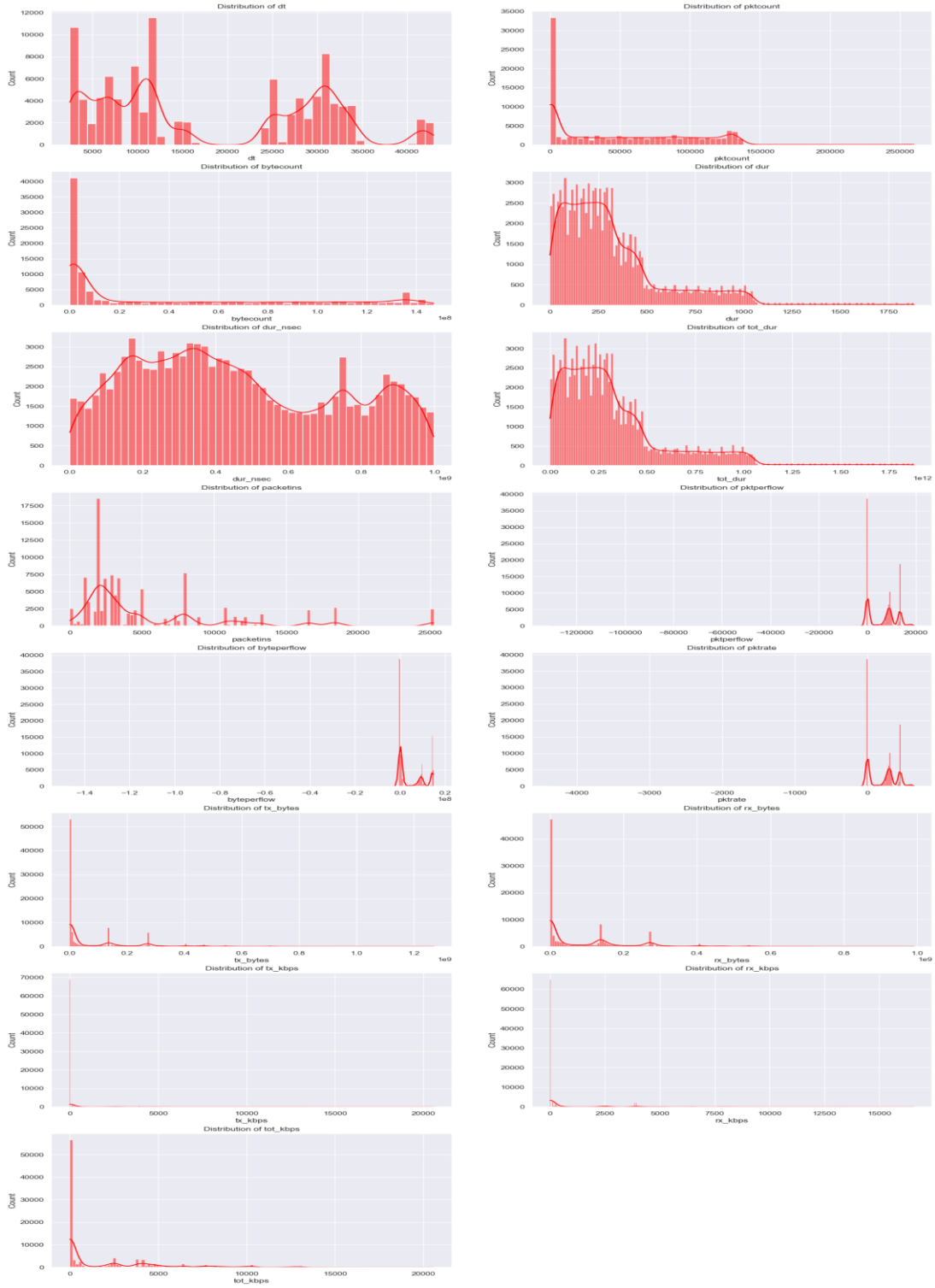

Distribution of src
10.0.0.1
10.0.0.2
10.0.0.4
10.0.0.10
10.0.0.3
10.0.0.5
10.0.0.13
10.0.0.6
10.0.0.20
10.0.0.11
10.0.0.12
10.0.0.18
10.0.0.8
10.0.0.7
10.0.0.9
10.0.0.14
10.0.0.15
10.0.0.16
10.0.0.17
src
0
2000
4000
6000
8000
10000
count


Distribution of dst
10.0.0.8
10.0.0.7
10.0.0.3
10.0.0.5
10.0.0.10
10.0.0.13
10.0.0.1
10.0.0.11
10.0.0.2
10.0.0.4
10.0.0.9
10.0.0.6
10.0.0.14
10.0.0.15
10.0.0.12
10.0.0.16
10.0.0.17
10.0.0.18
dst
0
2500
5000
7500
10000
12500
15000
17500
count


Distribution of Protocol
UDP
TCP
ICMP
Protocol
0
5000
10000
15000
20000
25000
30000
35000
40000
count

Visualize the distribution of continuous features

```
In [25]: ## Lets analyse the continuous values by creating histograms to understand the distribution
f = plt.figure(figsize=(20,90))
for i in range(len(continuous_feature)):
    f.add_subplot(len(continuous_feature), 2, i+1)
    histplot_distribution(continuous_feature[i])
plt.show()
```



Encoding categorical features

In [34]:
```
print("Features which need to be encoded are : \n" ,categorical_features)
```

```
Features which need to be encoded are :
 ['src', 'dst', 'Protocol']
```

In [35]:
```
df = pd.get_dummies(df, columns=categorical_features,drop_first=True)
print("This Dataframe has {} rows and {} columns after encoding".format(df.shape[0], df.shape[1]))
```

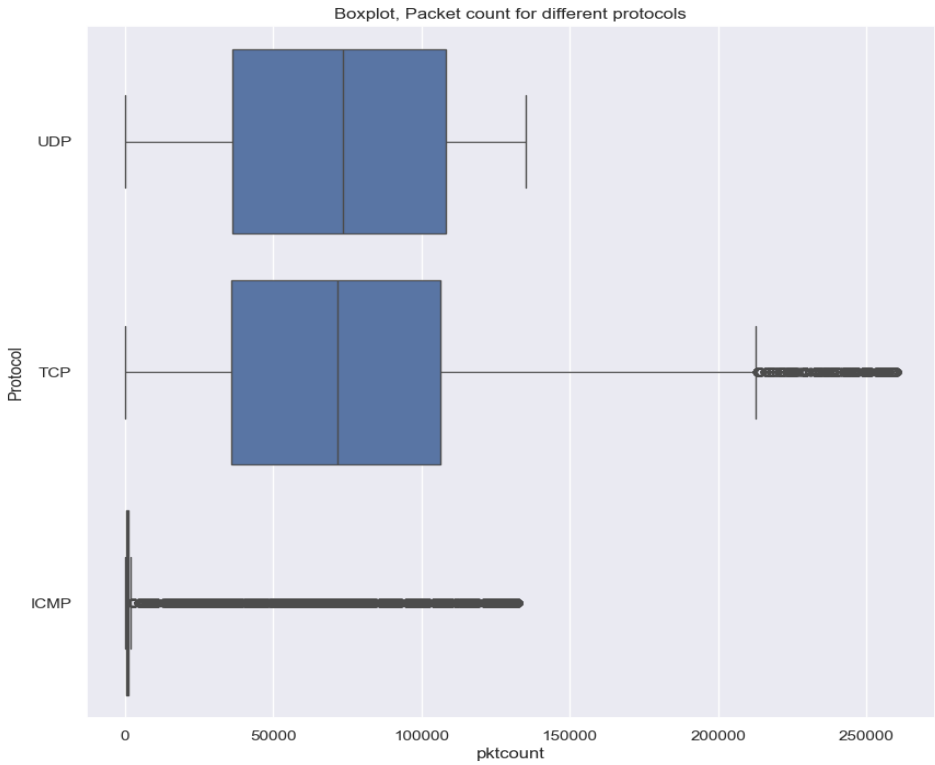
```
This Dataframe has 103839 rows and 57 columns after encoding
```

In [36]:
```
#dataframe after encoding
df.head(10)
```

Out[36]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	...	dst_10.0.0.2	dst_10.0.0.3	dst_10.0.0.4	dst_10.0.0.5	dst_10.0.0.6	dst_10.0.0.7	dst_10.0.0.8	dst_10.0.0.9	Protocol_TCP	Protocol_UDP
0	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
1	11605	1	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	...	False	False	False	False	False	False	True	False	False	True
2	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
3	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
4	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
5	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
6	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
7	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
8	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
9	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True

10 rows x 57 columns



Heat map of correlation of features

In [31]:
```
non_numeric_columns = df.select_dtypes(exclude='number').columns
print("Non-numeric columns:", non_numeric_columns)
```

```
Non-numeric columns: Index(['src', 'dst', 'Protocol'], dtype='object')
```

In [32]:
```
numeric_df = df.select_dtypes(include='number')
correlation_matrix = numeric_df.corr()
```

In [33]:
```
# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Set up the figure and axis
fig = plt.figure(figsize=(17, 17))

# Create a mask for the upper triangle
mask = np.zeros_like(correlation_matrix, dtype=bool)
mask[np.triu_indices_from(mask)] = True

# Set Seaborn theme
sns.set_theme(style="darkgrid")

# Create the heatmap
ax = sns.heatmap(correlation_matrix, square=True, annot=True, center=0, vmin=-1, linewidths=0.5, annot_kws={"size": 11}, mask=mask)

# Rotate x-axis labels for better readability
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, horizontalalignment='right')

# Show the plot
plt.show()
```

Split into Independent and dependent variables

In [38]:
```
#separating input and output attributes
x = df.drop(['label'], axis=1)
y = df['label']
```

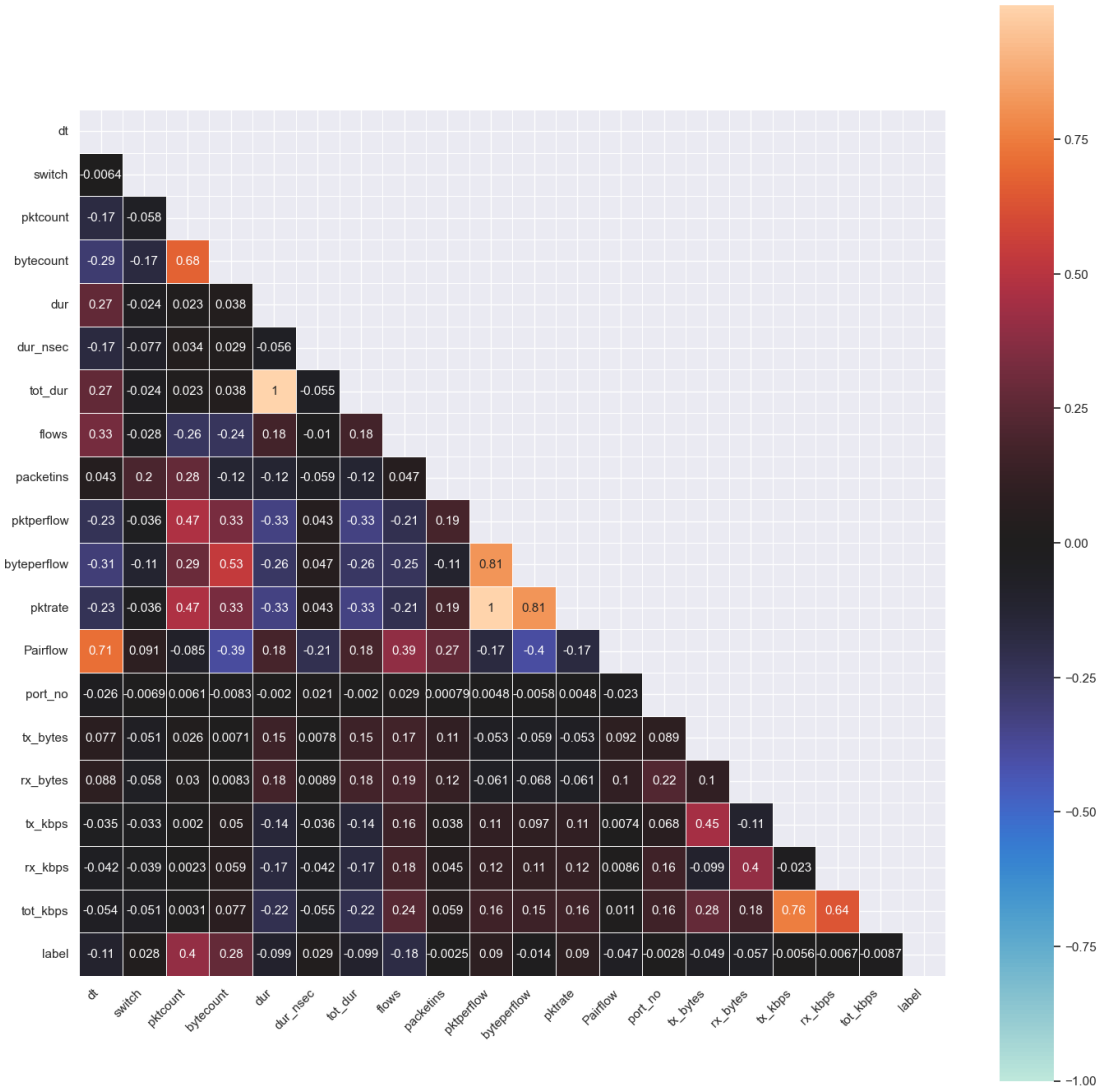
Normalizing features

In [39]:
```
ms = MinMaxScaler()
x = ms.fit_transform(x)
```

Train-Test-Split [75-25]

In [40]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y,test_size=0.3)
print(X_train.shape, X_test.shape)
```

```
(72687, 56) (31152, 56)
```



Encoding categorical features

In [34]:
```
print("Features which need to be encoded are : \n" ,categorical_features)
```

```
Features which need to be encoded are :
 ['src', 'dst', 'Protocol']
```

In [35]:
```
df = pd.get_dummies(df, columns=categorical_features,drop_first=True)
print("This Dataframe has {} rows and {} columns after encoding".format(df.shape[0], df.shape[1]))
```

```
This Dataframe has 103839 rows and 57 columns after encoding
```

In [36]:
```
#dataframe after encoding
df.head(10)
```

Out[36]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	...	dst_10.0.0.2	dst_10.0.0.3	dst_10.0.0.4	dst_10.0.0.5	dst_10.0.0.6	dst_10.0.0.7	dst_10.0.0.8	dst_10.0.0.9	Protocol_TCP	Protocol_UDP
0	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
1	11605	1	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	...	False	False	False	False	False	False	True	False	False	True
2	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
3	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
4	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
5	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True
6	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
7	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
8	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	False	False	False	False	False	False	True	False	False	True
9	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	False	False	False	False	False	False	True	False	False	True

10 rows x 57 columns

Split into Independent and dependent variables

In [38]:
```
#separating input and output attributes
x = df.drop(['label'], axis=1)
y = df['label']
```

Normalizing features

In [39]:
```
ms = MinMaxScaler()
x = ms.fit_transform(x)
```

Train-Test-Split [75-25]

In [40]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y,test_size=0.3)
print(X_train.shape, X_test.shape)
```

```
(72687, 56) (31152, 56)
```

6.4 TEST PLAN

Data Collection:

- Ensure that the dataset includes network traffic data, accurately labeled as benign (0) or malicious (1).
- Verify that the data collection process captures a diverse range of network activities, representing various types of benign and malicious traffic.
- Confirm that the dataset is comprehensive and suitable for training machine learning models for DDoS attack detection.

Data Preprocessing:

- Implement preprocessing techniques such as data cleaning, normalization, and feature extraction to prepare the dataset for model training.
- Verify that preprocessing steps remove irrelevant information, handle missing values appropriately, and encode categorical variables.
- Confirm that the dataset is balanced with respect to the distribution of benign and malicious traffic labels.

ML Algorithms:

- Implement different ML algorithms such as RF,SVM,DT,XG-BOOST and DNN for DDoS attack detection.
- Ensure proper configuration of each ensemble method, including hyperparameter tuning and feature selection.
- Compare the performance of each ensemble technique using evaluation metrics such as accuracy, precision, recall, and F1-score.

Training and Testing:

- Split the dataset into training and testing sets, maintaining the balance of benign and malicious traffic samples in each subset.
- Confirm that the training process effectively learns patterns and features associated with benign and malicious network traffic.
- Evaluate the models' performance using metrics tailored to DDoS attack detection, considering the importance of minimizing false positives and false negatives.

Model -Evaluation:

- Develop a selection process to identify potential DDoS attacks based on model predictions and severity assessment.
- Verify that the selection process accurately identifies instances of malicious traffic while minimizing false alarms.
- Ensure that traffic with a high likelihood of conducting DDoS attacks are flagged for further investigation, while benign actors are excluded from suspicion.

DATA VERIFICATION

Test ID	Test Description	Excepted Outcome
1	Verify that the datasets containing network traffic data are loaded correctly	Yes/No
2	Validate the correctness of the labels assigned to network traffic samples (0 for benign, 1 for malicious)	Yes/No
3	Check for duplicates in the network traffic dataset	Yes/No
4	Verify that the machine learning model generates accurate predictions for the detection of DDoS attacks based on network traffic features	Yes/No
5	Verify the pipeline is successful for building the DDOS application on AWS	Success/Fail
6	If the model detects **no** malicious activity in a network traffic sample	0 (Benign)
7	If the model detects high malicious activity in a network traffic sample	1 (Malicious)

Chapter 7

RESULT & DISCUSSION

Among the trained models DNN has the highest accuracy compared to other models

Defining the Deep Neural Network

In [42]:
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(64, activation='relu', input_shape=(56,)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])


model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

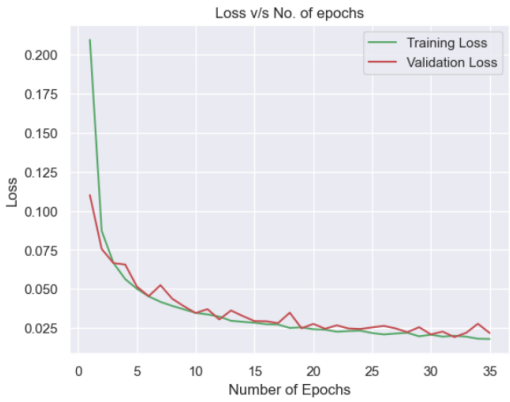
Model fitting

In [43]:
```
# Fit the model
history_org = model.fit(
    X_train,
    y_train,
    batch_size=50,
    epochs=35,
    verbose=2,
    callbacks=None,
    validation_data=(X_test, y_test),
    shuffle=True
)
```

```
Epoch 1/35
1454/1454 - 4s - 3ms/step - accuracy: 0.9105 - loss: 0.2095 - val_accuracy: 0.9607 - val_loss: 0.1100
Epoch 2/35
1454/1454 - 2s - 2ms/step - accuracy: 0.9679 - loss: 0.0872 - val_accuracy: 0.9716 - val_loss: 0.0756
Epoch 3/35
1454/1454 - 3s - 2ms/step - accuracy: 0.9757 - loss: 0.0667 - val_accuracy: 0.9750 - val_loss: 0.0665
Epoch 4/35
1454/1454 - 3s - 2ms/step - accuracy: 0.9791 - loss: 0.0563 - val_accuracy: 0.9724 - val_loss: 0.0656
Epoch 5/35
1454/1454 - 3s - 2ms/step - accuracy: 0.9811 - loss: 0.0501 - val_accuracy: 0.9787 - val_loss: 0.0515
Epoch 6/35
1454/1454 - 2s - 2ms/step - accuracy: 0.9818 - loss: 0.0452 - val_accuracy: 0.9797 - val_loss: 0.0453
```

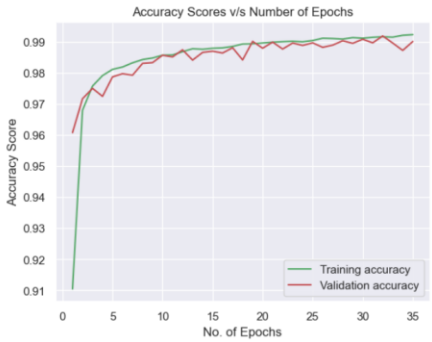
Plotting Loss v/s Epochs

In [44]:
```
loss = history_org.history['loss']
val_loss = history_org.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g', label = 'Training Loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
plt.title('Loss v/s No. of epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



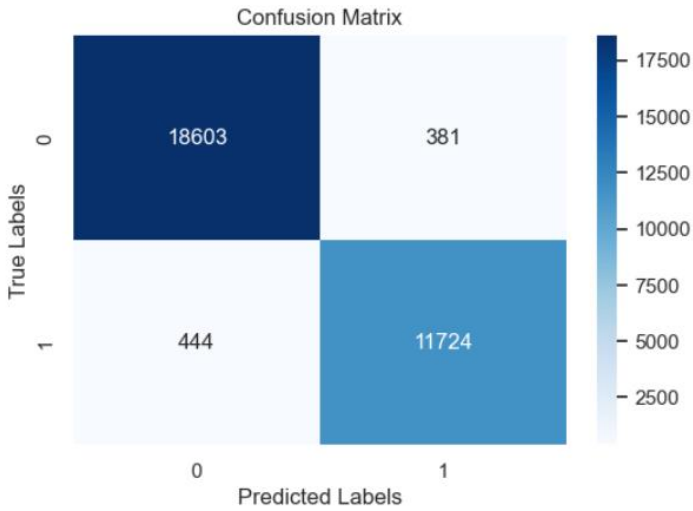
Plotting Accuracy v/s Epochs

```
In [45]: loss = history_org.history['accuracy']
         val_loss = history_org.history['val_accuracy']
         plt.plot(epochs, loss, 'g', label = 'Training accuracy')
         plt.plot(epochs, val_loss, 'r', label = 'Validation accuracy')
         plt.title('Accuracy Scores v/s Number of Epochs')
         plt.xlabel('No. of Epochs')
         plt.ylabel('Accuracy Score')
         plt.legend()
         plt.show()
```



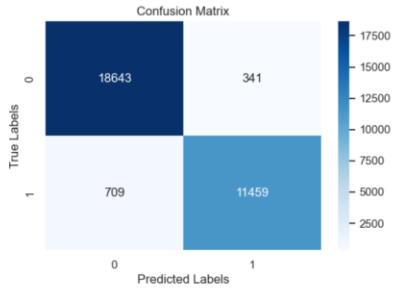
Model Evaluation

```
In [96]: loss, accuracy = model.evaluate(X_test, y_test)
         print('Accuracy of Deep neural Network : %.2f' % (accuracy*100))
         Classifier_accuracy.append(accuracy*100)

         conf_matrix = confusion_matrix(y_test, y_pred)

         # Plot confusion matrix
         plt.figure(figsize=(6, 4))
         sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
         plt.title("Confusion Matrix")
         plt.xlabel("Predicted Labels")
         plt.ylabel("True Labels")
         plt.show()

   1/974 ━━━━━━━━━━━━━━━━━━━━ 37s 38ms/step - accuracy: 1.0000 - loss: 1.8176e-05
 974/974 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9911 - loss: 0.0205
Accuracy of Deep neural Network : 99.00
```



K-Nearest Neighbor Classifier

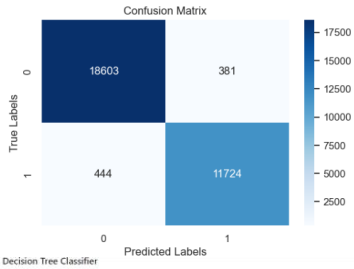
```
In [89]: knn_clf = KNeighborsClassifier()
         knn_clf.fit(X_train, y_train)
         y_pred = knn_clf.predict(X_test)
         accuracy = metrics.accuracy_score(y_test, y_pred)
         Classifier_accuracy.append(accuracy*100)
         print("Accuracy of KNN Classifier : %.2f" % (accuracy*100))
         report = classification_report(y_test, y_pred)
         print(report)

         conf_matrix = confusion_matrix(y_test, y_pred)

         # Plot confusion matrix
         plt.figure(figsize=(6, 4))
         sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
         plt.title("Confusion Matrix")
         plt.xlabel("Predicted Labels")
         plt.ylabel("True Labels")
         plt.show()

Accuracy of KNN Classifier : 96.63
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     18984
           1       0.97      0.94      0.96     12168

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




```
In [48]: joblib.dump(knn_clf, 'knn_model.pkl')
Out[48]: ['knn_model.pkl']
```

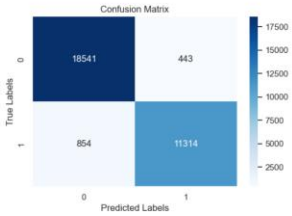
SVM Classifier

```
In [94]: svc_clf = SVC()
         svc_clf.fit(X_train,y_train)
         y_pred = svc_clf.predict(X_test)
         accuracy = metrics.accuracy_score(y_test, y_pred)
         Classifier_accuracy.append(accuracy*100)
         print("Accuracy of SVM Classifier : %.2f" % (accuracy*100) )
         report = classification_report(y_test, y_pred)
         print(report)
         conf_matrix = confusion_matrix(y_test, y_pred)

         # Plot confusion matrix
         plt.figure(figsize=(6, 4))
         sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
         plt.title("Confusion Matrix")
         plt.xlabel("Predicted Labels")
         plt.ylabel("True Labels")
         plt.show()

Accuracy of SVM Classifier : 97.35
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     18984
           1       0.97      0.96      0.97     12168

    accuracy                           0.97     31152
   macro avg       0.97      0.97      0.97     31152
weighted avg       0.97      0.97      0.97     31152
```



Decision Tree Classifier

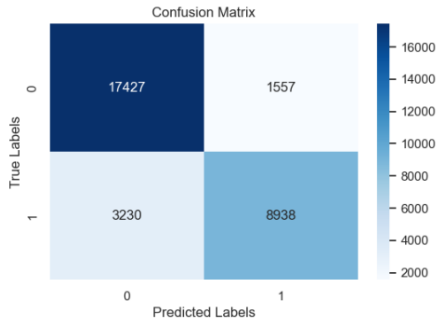
```
In [90]: dt_clf = DecisionTreeClassifier(max_depth=5)
         dt_clf.fit(X_train,y_train)
         y_pred = dt_clf.predict(X_test)
         accuracy = metrics.accuracy_score(y_test, y_pred)
         Classifier_accuracy.append(accuracy*100)
         print("Accuracy of Decision Tree Classifier : %.2f" % (accuracy*100) )
         report = classification_report(y_test, y_pred)
         print(report)

         conf_matrix = confusion_matrix(y_test, y_pred)

         # Plot confusion matrix
         plt.figure(figsize=(6, 4))
         sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
         plt.title("Confusion Matrix")
         plt.xlabel("Predicted Labels")
         plt.ylabel("True Labels")
         plt.show()

Accuracy of Decision Tree Classifier : 95.84
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     18984
           1       0.96      0.93      0.95     12168

    accuracy                           0.96     31152
   macro avg       0.96      0.95      0.96     31152
weighted avg       0.96      0.96      0.96     31152
```



Stochastic Gradient

In [93]:
```
sgd_clf=SGDClassifier(loss="hinge", penalty="l2")
sgd_clf.fit(X_train,y_train)
y_pred=sgd_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SGD Classifier : %.2f" % (accuracy*100))
report = classification_report(y_test, y_pred)
print(report)

conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

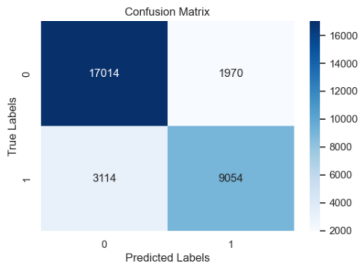
```
Accuracy of SGD Classifier : 84.63
              precision    recall  f1-score   support

           0       0.84      0.92      0.88     18984
           1       0.85      0.73      0.79     12168

    accuracy                           0.85     31152
   macro avg       0.85      0.83      0.83     31152
weighted avg       0.85      0.85      0.84     31152
```



Logistic Regression

In [91]:
```
lr_clf = LogisticRegression()
lr_clf.fit(X_train,y_train)
y_pred=lr_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Logistic Regression Classifier : %.2f" % (accuracy*100))
report = classification_report(y_test, y_pred)
print(report)

conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

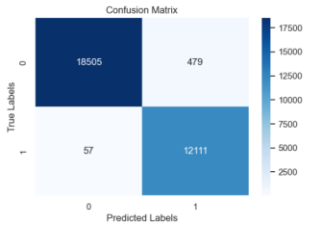
```
Accuracy of Logistic Regression Classifier : 83.68
              precision    recall  f1-score   support

           0       0.85      0.90      0.87     18984
           1       0.82      0.74      0.78     12168

    accuracy                           0.84     31152
   macro avg       0.83      0.82      0.83     31152
weighted avg       0.84      0.84      0.84     31152
```



XGBoost Classifier

In [92]:
```
xgb_clf=xgb.XGBClassifier(eval_metric = 'error',objective='binary:logistic',max_depth=2, learning_rate=0.1)
xgb_clf.fit(X_train,y_train)
y_pred=xgb_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of XGBoost Classifier : %.2f" % (accuracy*100))
report = classification_report(y_test, y_pred)
print(report)

conf_matrix = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

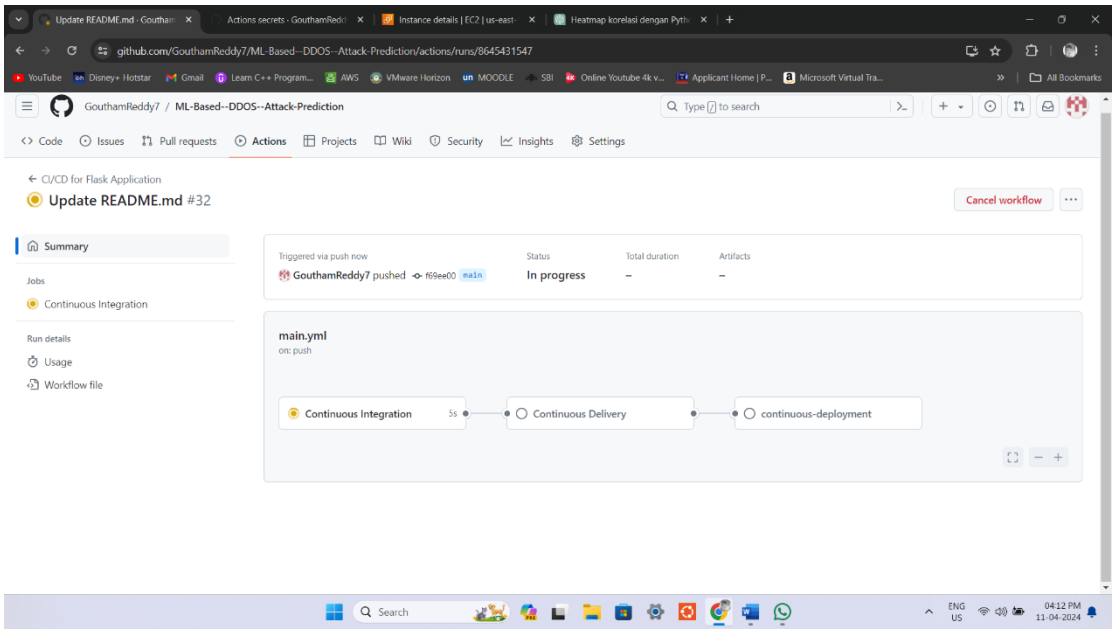
```
Accuracy of XGBoost Classifier : 98.28
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     18984
           1       0.96      1.00      0.98     12168

    accuracy                           0.98     31152
   macro avg       0.98      0.99      0.98     31152
weighted avg       0.98      0.98      0.98     31152
```



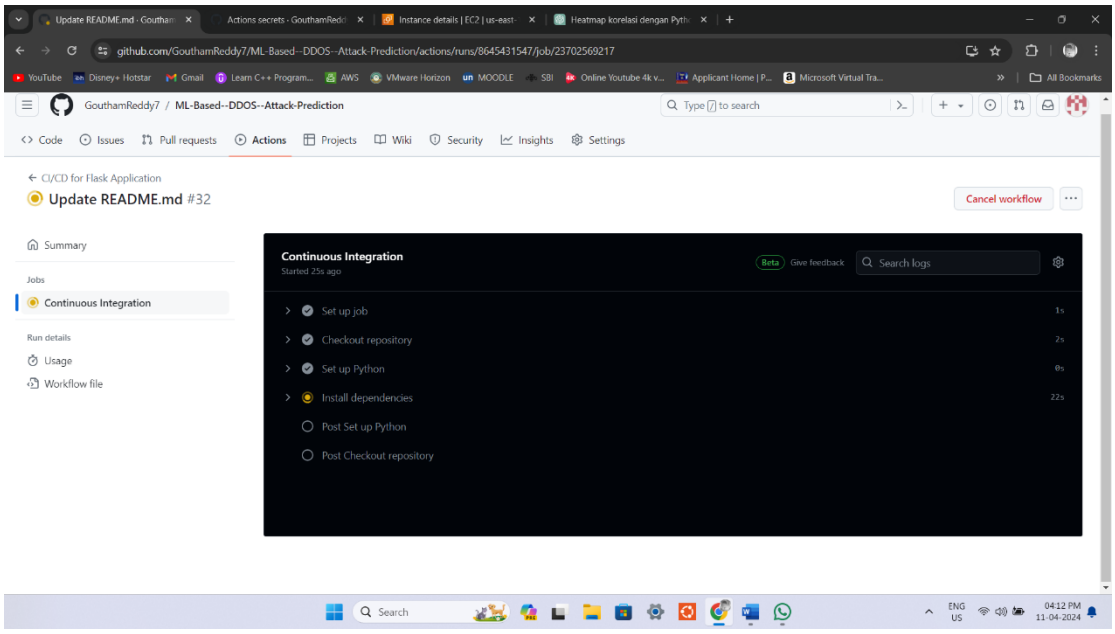
In [59]:
```
# Assuming X_test is a NumPy array
first_row = X_test[100]
first_row_csv = ','.join(map(str, first_row))

print(first_row_csv)
```

```
0.116671199347294,0.3333333333333333,0.275859018630314,0.0282275039037096418,0.128123336649654446,0.791791791791917,0.128723404255319 17,0.466666666666667,0.479857256145915 96,0.931902506611245 4,0.910569380125088 9,0.931854163868905,1.0,0.0,0.333686539049703 63,0.026203598165706,0.411564625850340 15,0.037763165832171 1,0.441982507288629 76,0.0,0.0,0.0
```

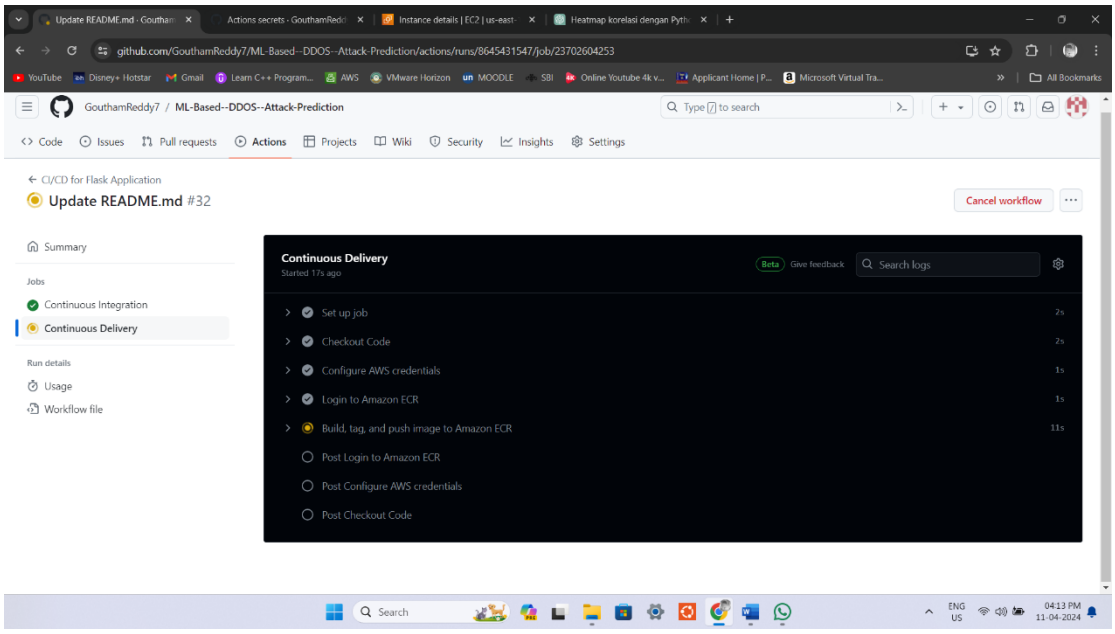
Pipeline:



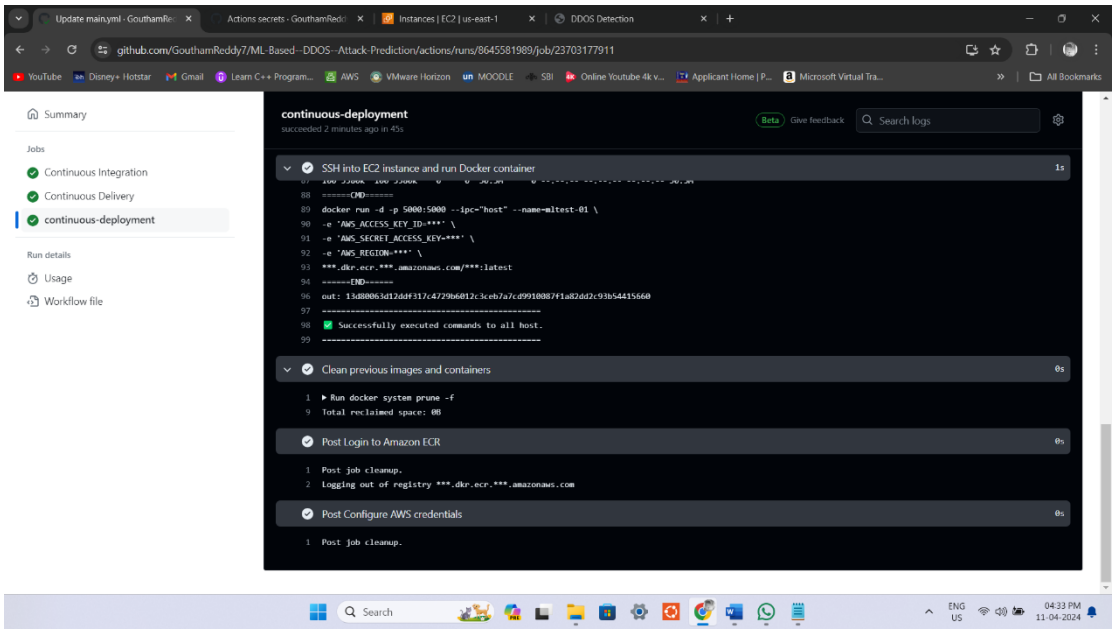
Continuous-Integration:



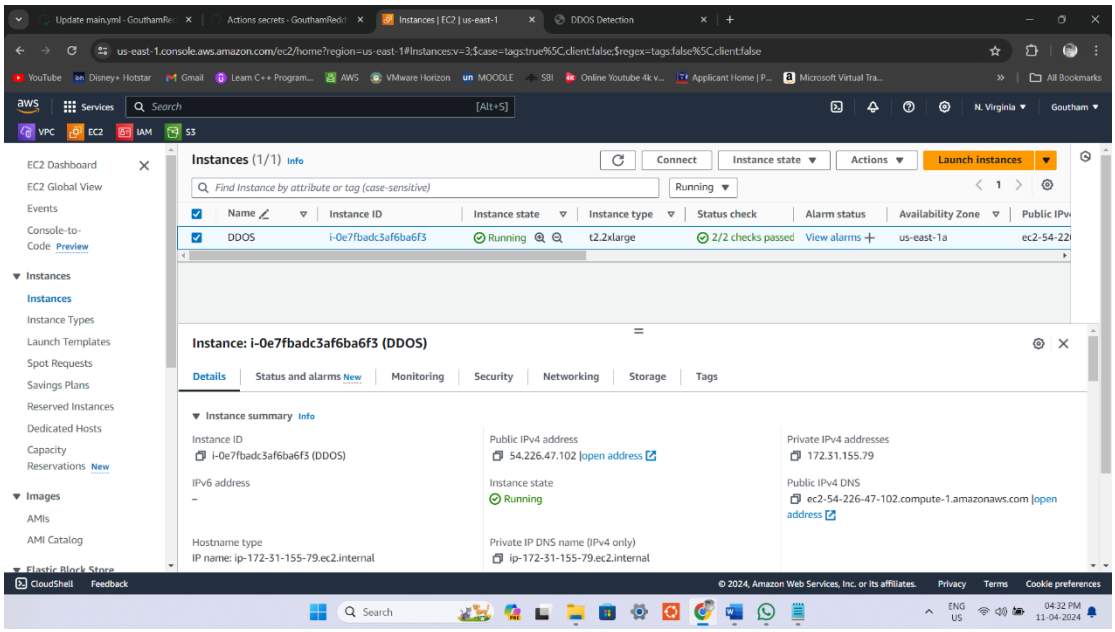
Continuous-Delivery:



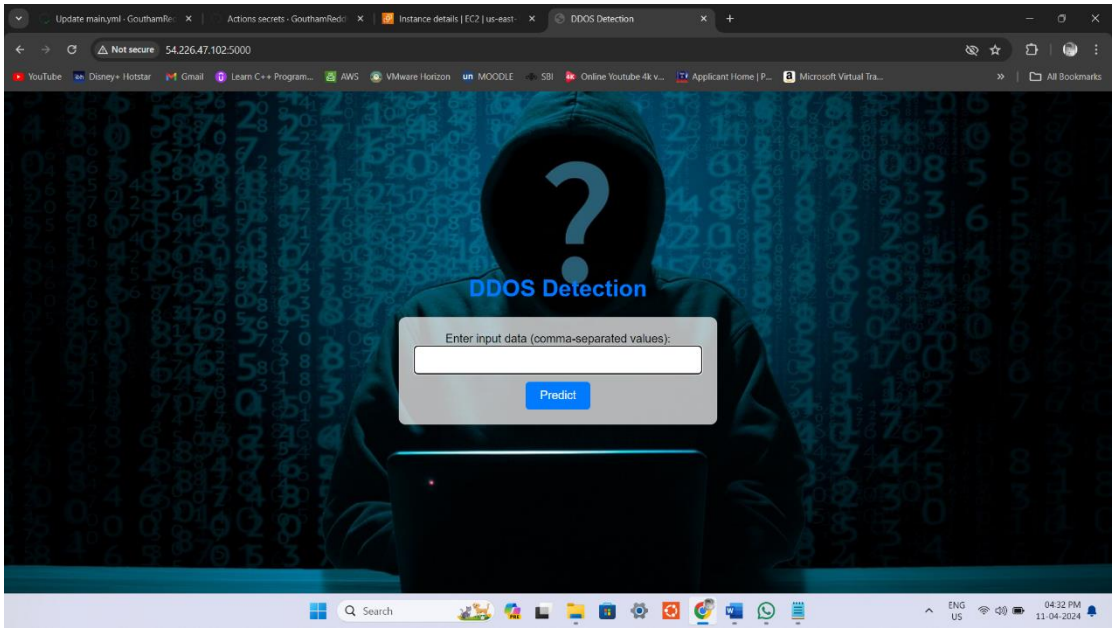
BUILD SUCCESSFUL:

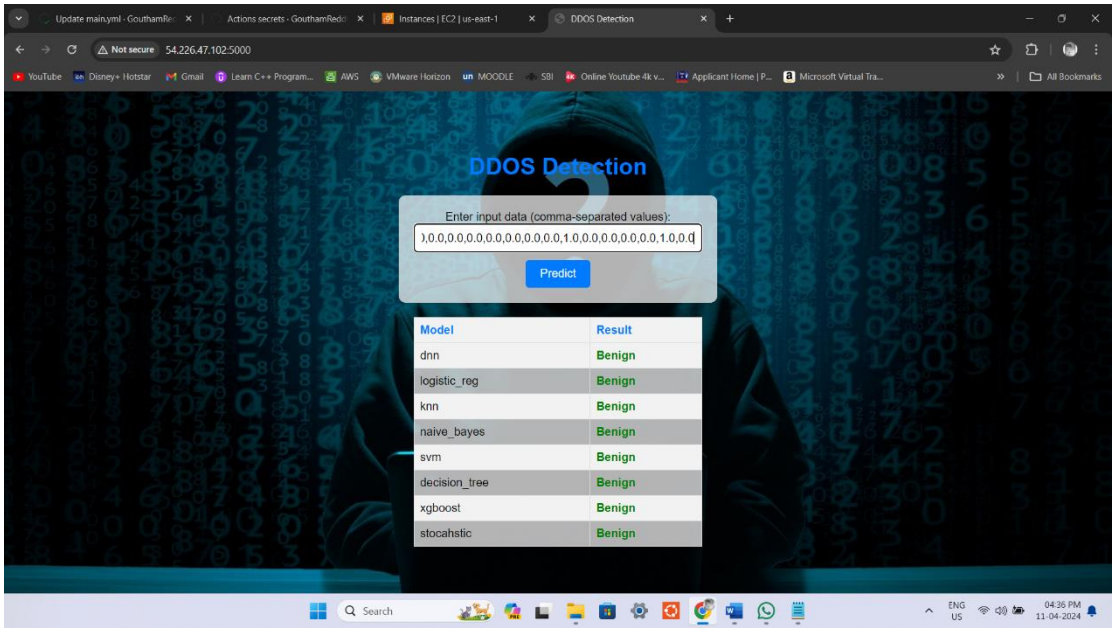


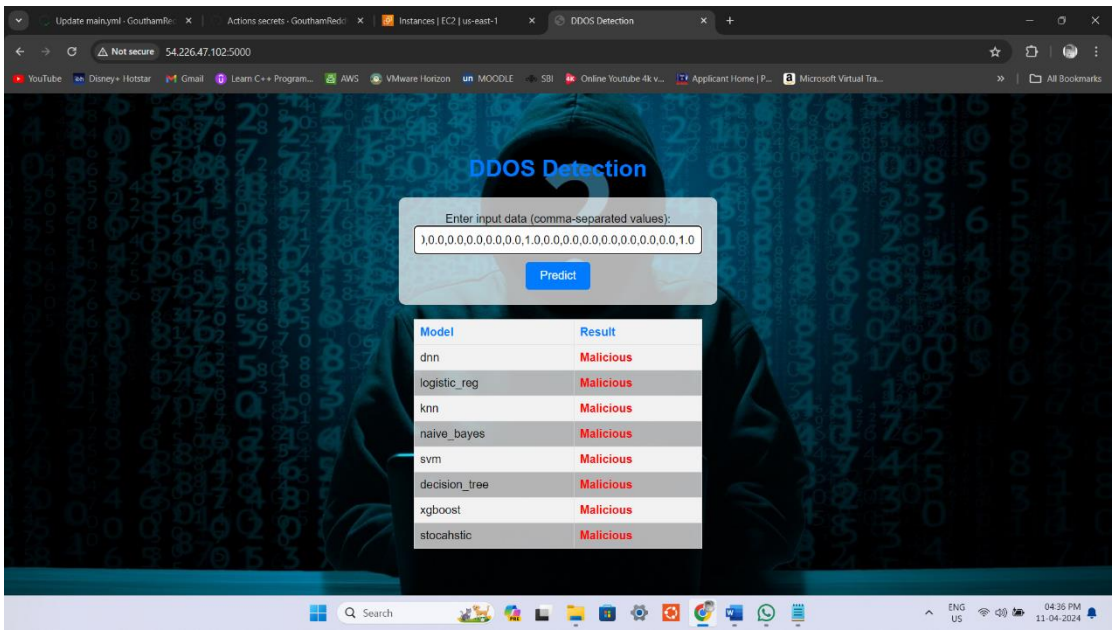
AWS:



Interface:




DDOS Detection
Enter input data (comma-separated values):
),0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0
Predict
Model
Result
dnn
Benign
logistic_reg
Benign
knn
Benign
naive_bayes
Benign
svm
Benign
decision_tree
Benign
xgboost
Benign
stocahstic
Benign


DDOS Detection
Enter input data (comma-separated values):
),0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0
Predict
Model
Result
dnn
Malicious
logistic_reg
Malicious
knn
Malicious
naive_bayes
Malicious
svm
Malicious
decision_tree
Malicious
xgboost
Malicious
stocahstic
Malicious

SUMMARY

DDoS attacks analysis and detection were performed using machine learning method. In this work, a SDN specific dataset is used. The dataset originally includes 23 features. The output feature is the last column of the dataset i.e. class label which classifies the traffic type to be benign or malicious. The malicious traffic is labelled as 1 and the benign traffic is labelled as 0. It has 104345 instances. The null values were observed in the rx_kbps and tot_kbps and were hence dropped for model development. The data processing steps were completed, including data preparation/cleaning, One Hot encoding, and normalization. After one hot encoding the dataframe had 103839 instances with 57 features and was fed into the model. A Deep Neural Network was used as the proposed model. The efficacy of our proposed model was observed to be higher than that of the baseline classifiers used. The accuracy of our proposed model was observed to be 99.38% which is approximately 1.21% higher than the next best model XG-Boost whose accuracy stands at 98.17%

Future Work:

Moving forward, there are several ways to make the DDoS attack detection system better. Firstly, we can tweak the features we're using to train the system to better capture different types of network behavior. Then, we can try different types of neural networks besides the one we're using now, like CNNs or RNNs, to see if they work better. We can also fine-tune the settings of our model to make it work even more accurately. Combining multiple models together, known as ensemble methods, might also improve our system's performance. Additionally, we can create more data for our system to learn from, especially for the cases where the network traffic is harmful. It's also worth looking into more advanced techniques, like Isolation Forests or autoencoders, for spotting unusual patterns in the data. To keep up with new threats, we can make our system learn and adapt in real-time. Finally, we should test our system on new sets of data to make sure it works well in different situations. By exploring these approaches, we can strengthen our DDoS detection system to better defend against cyber-attacks.

REFERENCES

1. Suresh, M., & Anitha, R. (2011). Evaluating machine learning algorithms for detecting DDoS attacks. In Advances in Network Security and Applications: 4th International Conference, CNSA 2011, Chennai, India, July 15-17, 2011 4 (pp. 441-452). Springer Berlin Heidelberg.
2. Polat, H., Polat, O., & Cetin, A. (2020). Detecting DDoS attacks in software-defined networks through feature selection methods and machine learning models. Sustainability, 12(3), 1035.
3. Tuan, T. A., Long, H. V., Son, L. H., Kumar, R., Priyadarshini, I., & Son, N. T. K. (2020). Performance evaluation of Botnet DDoS attack detection using machine learning. Evolutionary Intelligence, 13(2), 283-294.
4. Detection and Classification of DDoS Flooding Attacks on Software-Defined Networks: A Case Study for The Application of Machine Learning User Tuan, T. A., Long, H. V., Son, L. H., Kumar, R., Priyadarshini, I., & Son, N. T. K. (2020). Performance evaluation of Botnet DDoS attack detection
5. Sanchez, O. R., Repetto, M., Carrega, A., & Bolla, R. (2021, June). Evaluating ML-based DDoS detection with grid search hyperparameter optimization.
6. Raj, R., & Kang, S. S. (2022, December). A Review on DDoS attack Detection in SDN using ML. In 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N) (pp. 550-553).
7. Gaur, M. V., & Kumar, R. (2022). HPDDoS: A HyperParameter Model for Detection of Multiclass DDoS Attacks. Mathematical Statistician and Engineering Applications, 71(3s2), 1444-1470.
8. Khempetch, T., & Wuttidittachotti, P. (2021). DDoS attack detection using deep learning. IAES International Journal of Artificial Intelligence, 10(2), 382.
9. He, Z., Zhang, T., & Lee, R. B. (2017, June). Machine learning based DDoS attack detection from source side in cloud. In 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud) (pp. 114-120).
10. Priya, S. S., Sivaram, M., Yuvaraj, D., & Jayanthiladevi, A. (2020, March). Machine learning based DDoS detection. In 2020 international conference on emerging smart computing and informatics (ESCI) (pp. 234-237). IEEE.

APENDICE

Items	Total Cost	In-kind or Match (what you already have)	Requested
Facility Expenses	NA	NA	NA
Utilities	10,000	2,000	8,000
maintenance (cleaning)	3,000	0	3,000
Internet connection	2,500	0	2,500
Total Facility Expenses	15,500	2,000	13,500
Equipment			
laptop	75,000	3,000	72,000
Printer	8,000	2,000	6,000
Other Expenses	5,000	0	5,000
Total Project Costs/Total Request	1,03,000	10,000	93,000