NETWORK INTRUSION DETECTION SYSTEM ANLYTICS

USING MEMORY BASED LEARNING APPROACHES

A PROJECT REPORT

Submitted By

KESAVARAMAN M R [211420104131]

JENISH ROJIN S [21142010111]

LINGESH P [21142010147]

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report "**NETWORK INTRUSION DETECTION SYSTEM ANALYTICS USING MEMORY BASED MACHINE LEARNING APPROACHES"** is the bonafide work of **"KESAVARAMAN M R [211420104131], JENISH ROJIN S [211420104111], LINGESH P [211420104147]"** who carried out the project work under my supervision.

Signature of the HOD with date	**Signature of the Supervisor with date**
DR L.JABASHEELA M.E.,Ph.D.,	**Mr.A.N.SASIKUMAR M.E,**
PROFESSOR AND HEAD,	**ASSISTANT PROFESSOR**
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
Panimalar Engineering College,	Panimalar Engineering College,
Chennai – 123	Chennai - 123

Certified that the above candidates were examined in the End Semester Project Viva-Voce Examination held on 26.03.2024

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **KESAVARAMAN M R (211420104131), JENISH ROJIN S (211420104111), LINGESH P (211420104147)** hereby declare that this project report titled "**NETWORK INTRUSION DETECTION SYSTEM ANALYTICS USING MEMORY BASED LEARNING APPROACHES**", under the guidance of **Mr.A.N.SASIKUMAR** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

Name of the student(s)

KESAVARAMAN M R

JENISH ROJIN S

LINGESH P

ACKNOWLEDGEMENT

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We want to express our deep gratitude to our Directors, **Tmt. C. VIJAYARAJESWARI, Dr. C. SAKTHI KUMAR, M.E., Ph.D., and Dr. SARANYASREE SAKTHI KUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

Our gratitude is also extended to our Principal, **Dr. K. MANI, M.E., Ph.D.,** whose facilitation proved pivotal in the successful completion of this project.

We express our heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to **Project Coordinator Dr.G.SENTHIL KUMAR, M.C.A, M.Phil, M.E, Ph.D** and **Project Guide Mr.A.N.SASIKUMAR, M.E** and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

NAME OF THE STUDENT(S)

KESAVARAMAN M R [211420104131]

JENISH ROJIN S [211420104111]

LINGESH P [211420104147]

PROJECT COMPLETION CERTIFICATE

SPIRO PRIME TECH SERVICES

21.03.2024

To Whomsoever It May Concern

This is to certify that **Mr. KESAVARAMAN M R (Reg No: 211420104131), Mr. JENISH ROJIN S (Reg No: 211420104111), Mr. LINGESH P (Reg No: 211420104147),** students of final year **BE** of **"PANIMALAR ENGINEERING COLLEGE"** has completed his major project with great success at our concern, under the Title: **"NETWORK INTRUSION DETECTION SYSTEM ANALYTICS USING MEMORY BASED LEARNING APPROACHES"** from **DECEMBER 2023 to MARCH 2024.**

His project is found to be relevant regarding his stream and he had submitted a copy of the project report to us. During his Project period we found his sincere & hard working & possessing a good behaviour and a moral character.

We wish him grand success in future endeavours.

For SPIRO PRIME TECH SERVICES,

M.SAMPATH KUMAR

MANAGER

SPIRO PRIME TECH SERVICES NO.78, IIIrd Floor, Usman Road, T.Nagar. CHENNAI-600 017

Corporate Office : # 78, 3rd Floor, Usman Road, T.Nagar, Chennai - 600 017.
Mail: hr@spiroit.com Mobile: 9791 044 044 / www.spiroprojects.com.

ABSTRACT

Network Intrusion Detection Systems (NIDS) play a critical role in protecting computer networks from various security threats and attacks. As the complexity and frequency of network attacks continue to evolve, there is a growing need for advanced analytics techniques to enhance the detection and response capabilities of NIDS. This research focuses on the development and utilization of analytics methods for network intrusion detection systems. The goal is to leverage these techniques to improve the accuracy, efficiency, and effectiveness of NIDS in identifying and mitigating security breaches. The research begins by exploring the fundamental concepts of network intrusion detection, including the different types of attacks and the challenges associated with their detection. Various types of NIDS, including signature-based and anomaly-based systems, are discussed, highlighting their strengths and limitations. Overall, this research aims to advance the field of network intrusion detection by leveraging analytics techniques to enhance the capabilities of NIDS. The proposed methods offer the potential to improve the accuracy of attack detection, reduce false positives, enable efficient processing of big data, and facilitate automated incident response. The findings of this research will contribute to the development of more robust and effective network security systems in the face of ever-evolving cyber threats.

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for the harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Although intrusion detection systems monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the intrusion detection systems to recognize what normal traffic on the network looks like as compared to malicious activity.

Intrusion prevention systems also monitor network packets inbound the system to check the malicious activities involved in it and at once send the warning notifications.

1.2 Problem Definition

The aim of this project is to develop an effective Intrusion Detection System (IDS) capable of monitoring network activities and promptly detecting any suspicious or potentially malicious actions. By leveraging this system, it becomes possible to swiftly generate alerts upon the detection of anomalies or intrusions within the network. These alerts serve as early warnings that empower security operations center (SOC) analysts and incident responders

to investigate the issues at hand and initiate the necessary actions for threat mitigation. The ultimate goal is to enhance network security by accurately identifying and categorizing attack types, thus enabling a proactive and efficient response to potential intrusions.

1.3 Objectives

Detected anomalies are typically reported and aggregated through a centralized system like a Security Information and Event Management (SIEM) platform. The primary focus of this project is to harness the power of machine learning to construct a predictive model for intrusion detection. This model aims to potentially replace traditional supervised machine learning classification models by delivering superior accuracy in identifying and categorizing security threats through a comparative analysis of various supervised algorithms. Ultimately, the goal is to enhance network security by employing a predictive model that can swiftly and accurately respond to emerging threats.

1.4 Scope of the project

The project's scope involves building an Intrusion Prevention System (IPS) that actively safeguards network security. This IPS achieves its goal by dropping malicious packets, blocking problematic IP addresses, and alerting security personnel about potential threats. It relies on a signature database for recognizing known attack patterns and can also identify anomalies in network traffic. This holistic approach enhances network security by preventing attacks and swiftly responding to emerging threats.

CHAPTER 2

LITERATURE REVIEW

2.1 A Dependable Hybrid Machine Learning Model for Network Intrusion Detection

Author: Md. Alamin Talukdera , Khondokar Fida Hasanb , Md. Manowarul Islama

Year: 2023

This paper proposed that The Network Intrusion Detection Systems (NIDSs) are pivotal in safeguarding computer networks against evolving cyber threats. In the face of increasingly sophisticated attacks, the demand for NIDSs with heightened accuracy and dependability have grown substantially. This research presents a novel hybrid model that combines machine learning and deep learning techniques to bolster detection rates while preserving dependability.

The methodology encompasses data preprocessing, featuring the application of Synthetic Minority Over-sampling Technique (SMOTE) for addressing

imbalanced datasets, and feature selection through XGBoost. A comprehensive comparison of the hybrid model with diverse machine learning and deep learning algorithms aids in identifying the most efficient approach. Results showcase outstanding performance, with accuracy rates of 99.99% for the KDDCUP'99 dataset and 100% for CIC-MalMem-2022, with no overfitting or Type-1 and Type-2 errors. This research offers a promising avenue for enhancing network intrusion detection and fortifying cybersecurity in modern network environments.

2.2 Analysis of Network Intrusion Detection System with Machine Learning Algorithms

Author: Saddam Hossen, Anirudh Janagam

Year: 2018

This paper proposed that Intrusion Detection Systems (IDS) hold a pivotal role in fortifying networks against unauthorized access and ensuring data confidentiality. This research paper focuses on evaluating the efficacy of a Network Intrusion Detection System (NIDS) employing Deep Reinforcement Learning (DRL) algorithms, specifically the Deep Q Network (DQN) algorithm, renowned for its value-based reinforcement learning capabilities. The objective is to enhance the accuracy of detecting a wide array of network attacks while transcending the reliance on historical data or past experiences. This study meticulously selects a dataset, featuring 85 attributes tailored for precise attack detection, effectively minimizing false alarms. By benchmarking the NIDS-DQN model against alternative approaches, including J48, artificial neural networks, random forests, and support vector machines, the research demonstrates the potential to revolutionize intrusion detection by providing a more accurate and proactive security solution, thus advancing the field of network security.

2.3 Network intrusion detection system: A systematic study of machine learning and deep learning approaches

Author: Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang

Year: 2022

The rapid evolution of internet and communication technologies has led to a substantial expansion in network size and data volume, giving rise to a surge in novel cyberattacks that challenge the accuracy of intrusion detection in network security. Concurrently, the presence of intruders seeking to exploit network vulnerabilities necessitate robust defence mechanisms. Intrusion Detection Systems (IDS) serve as crucial tools in safeguarding networks by scrutinizing network traffic to ensure the confidentiality, integrity, and availability of data. Despite persistent research efforts, IDSs grapple with the twin challenges of enhancing detection precision while mitigating false alarms and identifying emerging intrusion tactics. In recent times, machine learning (ML) and deep learning (DL) has emerged as promising solutions for proficiently detecting network intrusions.

This article elucidates the IDS concept and offers a taxonomy predicated on noteworthy ML and DL techniques applied in the development of network-based IDS (NIDS) systems. It furnishes a comprehensive review of recent NIDS-based research, critically evaluating the strengths and limitations of proposed solutions. Additionally, it delineates contemporary trends and advancements in ML and DL-based NIDS encompassing methodology, evaluation metrics, and dataset selection. By spotlighting prevailing methodological gaps, this work underscores

research challenges and charts the future trajectory for enhancing ML and DL-based NIDS systems.

2.4 An Efficient Hybrid Deep Learning-Based Intrusion Detection System

Author: Vanlalruata Hnamte, Jamal Hussain

Year: 2023

The pervasive migration of real-world processes into the digital realm has led to a profound reliance on interconnected computer systems communicating via the Internet. This shift has, in turn, ushered in a surge in network security vulnerabilities, rendering the task of safeguarding networks against a myriad of cyber threats a formidable challenge for network administrators. While a plethora of network intrusion detection techniques have been devised, they grapple with the formidable task of adapting to the continual emergence of novel vulnerabilities that defy conventional approaches. Recognizing the remarkable prowess of deep learning in various detection and identification tasks, we introduce an intelligent and efficient Network Intrusion Detection System (NIDS) rooted in Deep Learning (DL). This study unveils a DL-based IDS tailored for attack detection, meticulously trained on real-time traffic datasets such as CICIDS2018 and Edge_IIoT. Through rigorous evaluation using multi-class classification, our model attains an astounding accuracy rate of 100% and 99.64% during training and testing with the respective datasets, underscoring its potential as a formidable defence against network intrusions.

CHAPTER 3

THEORITCAL BACKGROUND

3.1 Implementation Environment

3.1.1 Software Requirements

Software Requirements	
Operating System	**Windows 10 or later**
Tool	**Anaconda with Jupyter Notebook**

3.1.2 Hardware Requirements

Hardware Requirements	
Processor	Intel i3
Hard Disk	Minimum 80 GB
RAM	Minimum 2 GB

3.1.3 Software Description

Anaconda is a free and open-source distribution of the Python and R programming languages specifically designed for scientific computing, data science, machine learning applications, and more. It simplifies package management and deployment through its package management system called "Conda." With over 1,400 data-science packages, Anaconda is widely used by over 12 million users on Windows, Linux, and MacOS platforms. The distribution includes the Anaconda Navigator, a Conda-based package and virtual environment manager, streamlining the installation process and eliminating the need to individually install each library. Anaconda supports both Conda and Pip package installations, with the former offering a broader range of features. Additionally, users can create custom packages using the "conda build" command

and share them via Anaconda Cloud, PyPI, or other repositories. While Anaconda2 comes with Python 2.7 and Anaconda3 with Python 3.7 by default, users can create new environments with different Python versions.

Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda.org or in a local Anaconda Repository.

Anaconda. Now, if you are primarily doing data science work, Anaconda is also a great option. Anaconda is created by Continuum Analytics, and it is a Python distribution that comes preinstalled with lots of useful python libraries for data science.

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages and use multiple environments to separate these different versions.

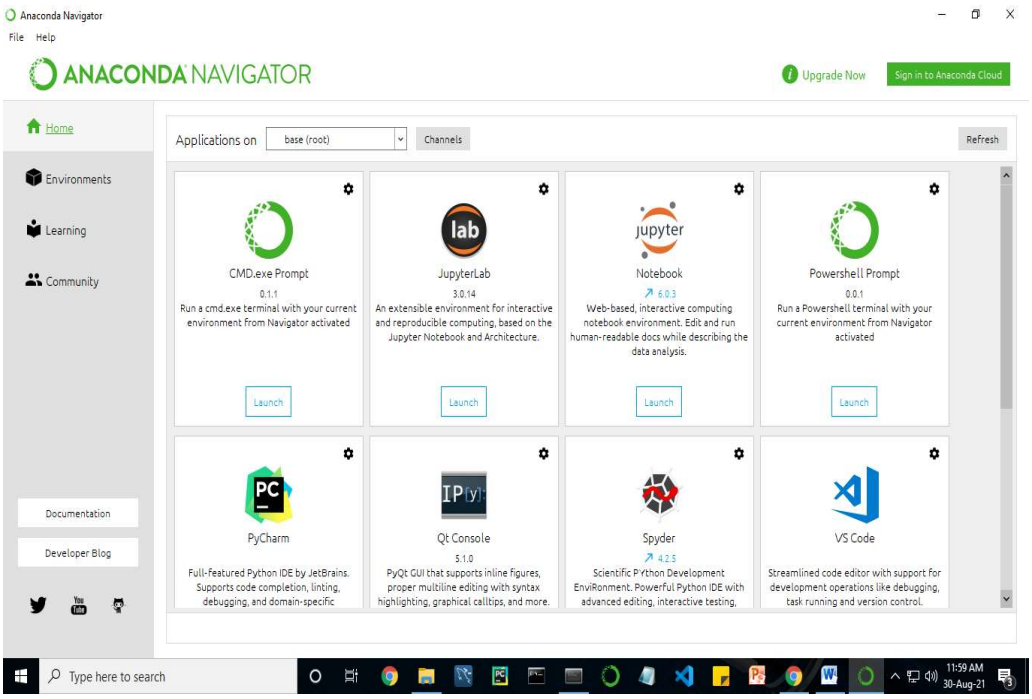
The command-line program conda is both a package manager and an environment manager. This helps data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window.

You can use it to find the packages you want, install them in an environment, run the packages, and update them – all inside Navigator.

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- Spyder
- PyCharm
- VSCode
- Glueviz
- Orange 3 App
- RStudio
- Anaconda Prompt (Windows only)
- Anaconda PowerShell (Windows only)



Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution.

Navigator allows you to launch common Python programs and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository.

Anaconda comes with many built-in packages that you can easily find with conda list on your anaconda prompt. As it has lots of packages (many of which are rarely used), it requires lots of space and time as well. If you have enough space, time and do not want to burden yourself to install small utilities like JSON, YAML, you better go for Anaconda.

Conda

Conda is an open source, cross-platform, language-agnostic package manager and environment management system that installs, runs, and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects. The conda package and environment manager is included in all versions of Anaconda, Miniconda, and Anaconda Repository.

Anaconda is freely available, open source distribution of python and R programming languages which is used for scientific computations. If you are doing any machine learning or deep learning project then this is the best place for you. It consists of many softwares which will help you to build your machine learning project and deep learning project. These softwares have great graphical user interface and these will make your work easy to do. You can also use it to run your python script. These are the software carried by anaconda navigator.

Jupyter Notebook

This website acts as “meta” documentation for the Jupyter ecosystem. It has a collection of resources to navigate the tools and communities in this ecosystem, and to help you get started.

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Perez.

Notebook documents are documents produced by the Jupyter Notebook App, which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc…). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc.) as well as executable documents which can be run to perform data analysis.

Purpose: To support interactive data science and scientific computing across all programming languages.

File Extension: An **IPYNB** file is a notebook document created by Jupyter Notebook, an interactive computational environment that helps scientists manipulate and analyze data using Python.

Jupyter Notebook App

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser.

The Jupyter Notebook App can be executed on a local desktop requiring no internet access (as described in this document) or can be installed on a remote server and accessed through the internet.

In addition to displaying/editing/running notebook documents, the Jupyter Notebook App has a “Dashboard” (Notebook Dashboard), a “control panel” showing local files and allowing to open notebook documents or shutting down their kernels.

3.2 SYSTEM ARCHITECTURE

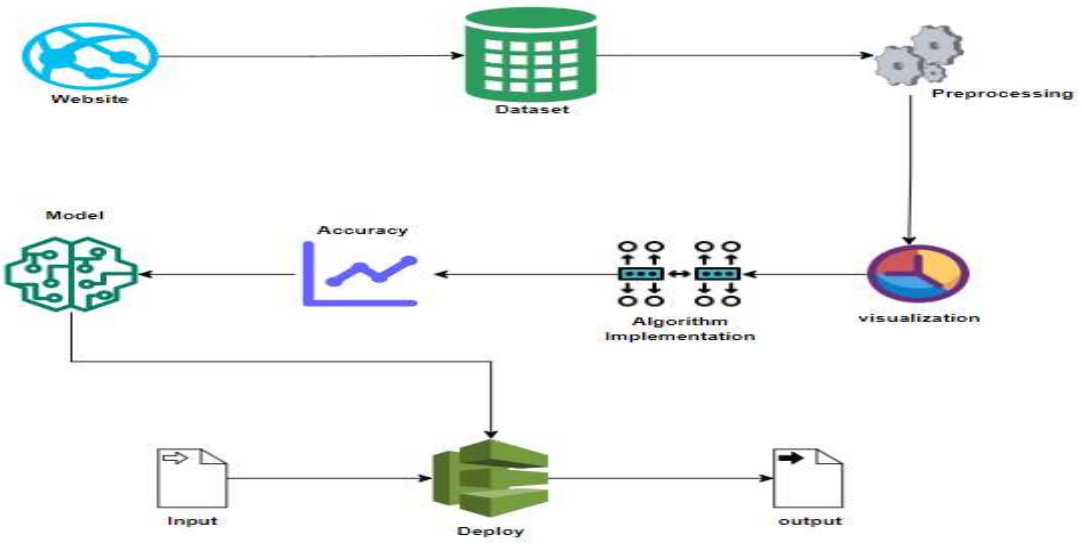


Figure 3.2.1 Architecture diagram for NIDS

3.3 Existing System

Enhancing Network Intrusion Detection Systems (NIDS) with supervised Machine Learning (ML) is tough. ML-NIDS must be trained and evaluated, operations requiring data where benign and malicious samples are clearly labelled. Such labels demand costly expert knowledge, resulting in a lack of real deployments, as well as on papers always relying on the same outdated data. The situation improved recently, as some efforts disclosed their labelled datasets. However, most past works used such datasets just as a ‘yet another’ test bed, overlooking the added potential provided by such availability. Despite many successes, the integration of supervised Machine Learning (ML) methods in Network Intrusion Detection Systems (NIDS) is still at an early stage. This is due to the difficulty in obtaining comprehensive sets of labelled data for training and

evaluating an ML-NIDS. The recent release of labelled datasets for ML-NIDS was appreciated by the research community; however, few works noticed the opportunity that such availability provides to the state-of-the-art.

Disadvantages:

1. They are not predicting the classification of attack types.
2. They are not mentioning the accuracy.
3. They are using machine learning technique only for analysing purpose.

3.4 Proposed Methodology

In the proposed system, we aim to enhance the effectiveness and efficiency of network intrusion detection systems (NIDS) by incorporating machine learning techniques into the analytics process. Machine learning provides the ability to automatically learn patterns and behaviours from network traffic data, enabling more accurate and proactive intrusion detection. The proposed system offers several benefits, including improved detection accuracy, proactive threat identification, and adaptability to evolving intrusion patterns. By leveraging machine learning techniques, the system can effectively analyse large volumes of network traffic data and identify subtle indicators of intrusions that may go unnoticed by traditional rule-based approaches. Overall, the proposed system aims to enhance the capabilities of network intrusion detection systems through the utilization of machine learning algorithms. The proposed system can be designed to continuously update and adapt the machine learning models as new network data becomes available. This allows the system to capture evolving intrusion patterns and improve its detection capabilities over time. By leveraging the power of machine learning, the system can improve the accuracy, efficiency,

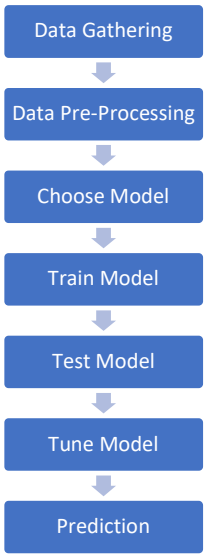
and effectiveness of intrusion detection, ultimately enhancing the security posture of computer networks.

Advantages:

- We build a production level application for deployment purposes.
- We use a machine learning technique for build predictive model.
- We using python language to build a model.
- Improved accuracy and performance level.

3.4.1 Construction of a Predictive Model

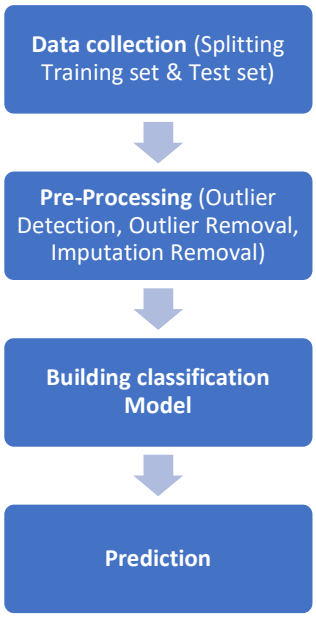
Machine learning relies on data, especially historical and raw data. Raw data, before it can be used, needs to undergo pre-processing. This involves preparing the data and making it suitable for use with specific algorithms and models. The data is then split into training and testing sets. A model is trained using the training data, and its performance is assessed, aiming to make accurate predictions with minimal errors. To improve accuracy, the model is fine-tuned over time, making it more effective and precise in its predictions.



3.4.1 Process of dataflow diagram

3.4.2 Project Workflow

The steps involved in Building the data model is depicted below.



3.4.2 Project Workflow Diagram

3.5 UML Diagrams

3.5.1 Use Case Diagram

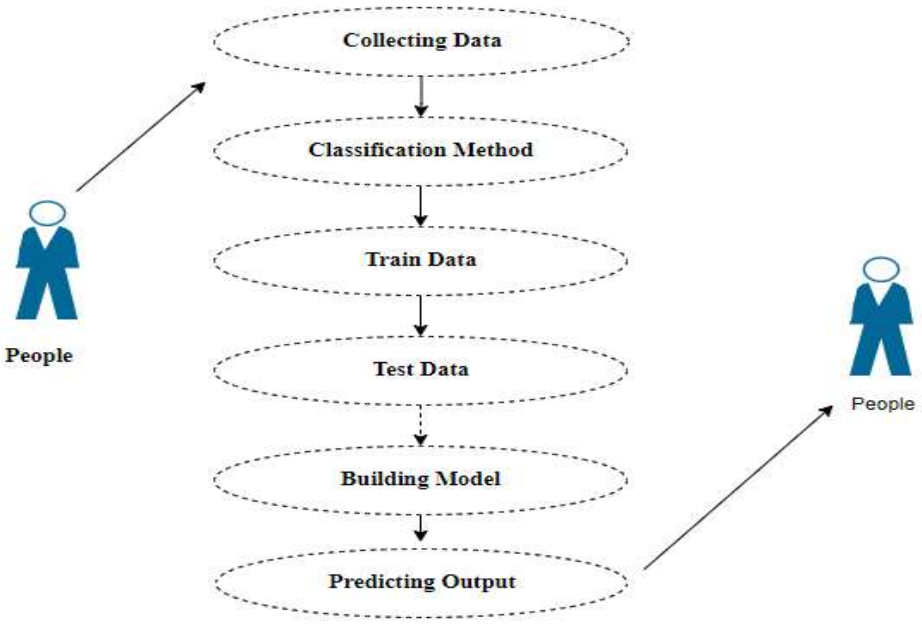


Figure 3.5.1 Use Case diagram

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analysed, the functionalities are captured in use cases. So, it can say that uses cases are nothing but the system functionalities written in an organized manner.

3.5.2 Class Diagram

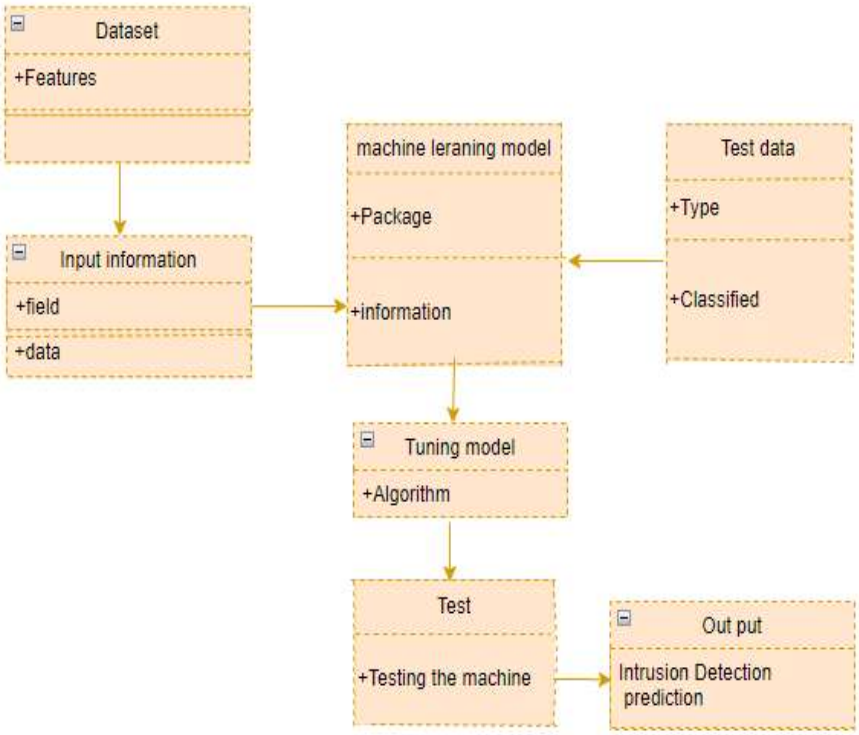


Figure 3.5.2 Class diagram

Class diagram is basically a graphical representation of the static view of the system and represents different aspects of the application. So, a collection of class diagrams represents the whole system. The name of the class diagram should be meaningful to describe the aspect of the system. Each element and their relationships should be identified in advance Responsibility (attributes and methods) of each class should be clearly identified for each class minimum number of properties should be specified and because, unnecessary properties will make the diagram complicated. Use notes whenever required to describe

some aspect of the diagram and at the end of the drawing it should be understandable to the developer/coder. Finally, before making the final version, the diagram should be drawn on plain paper and rework as many times as possible to make it correct.

3.5.3 Activity Diagram

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part. It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams look like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

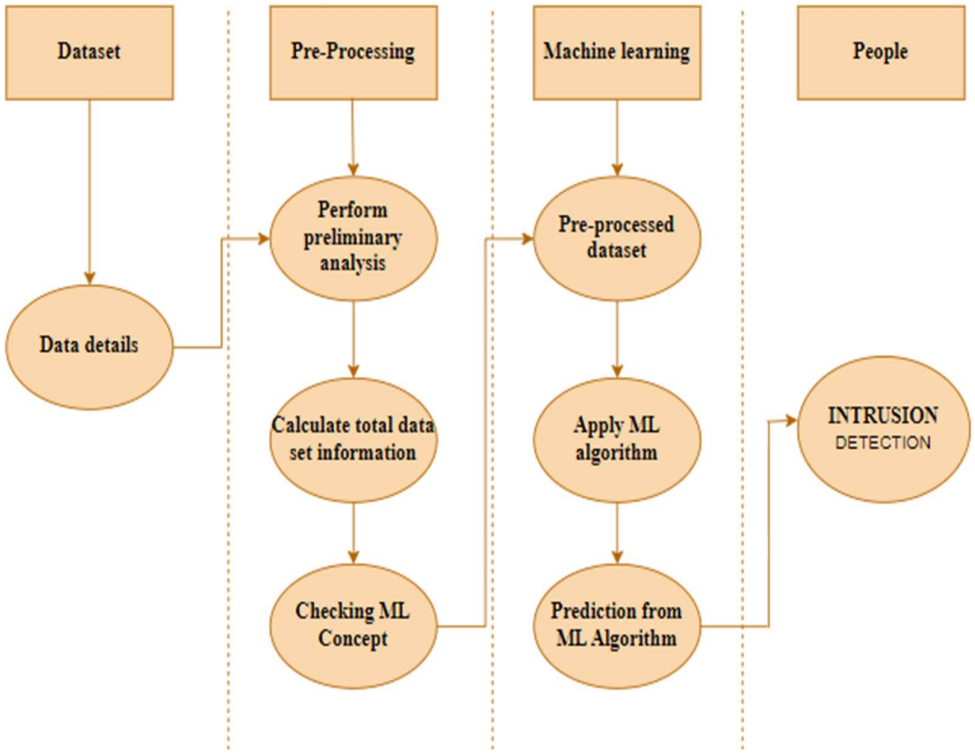


Figure 3.5.3 Activity diagram

3.5.4 Sequence Diagram

Sequence diagrams model the flow of logic within your system in a visual manner, enabling you both to document and validate your logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modelling, which focuses on identifying the behaviour within your system

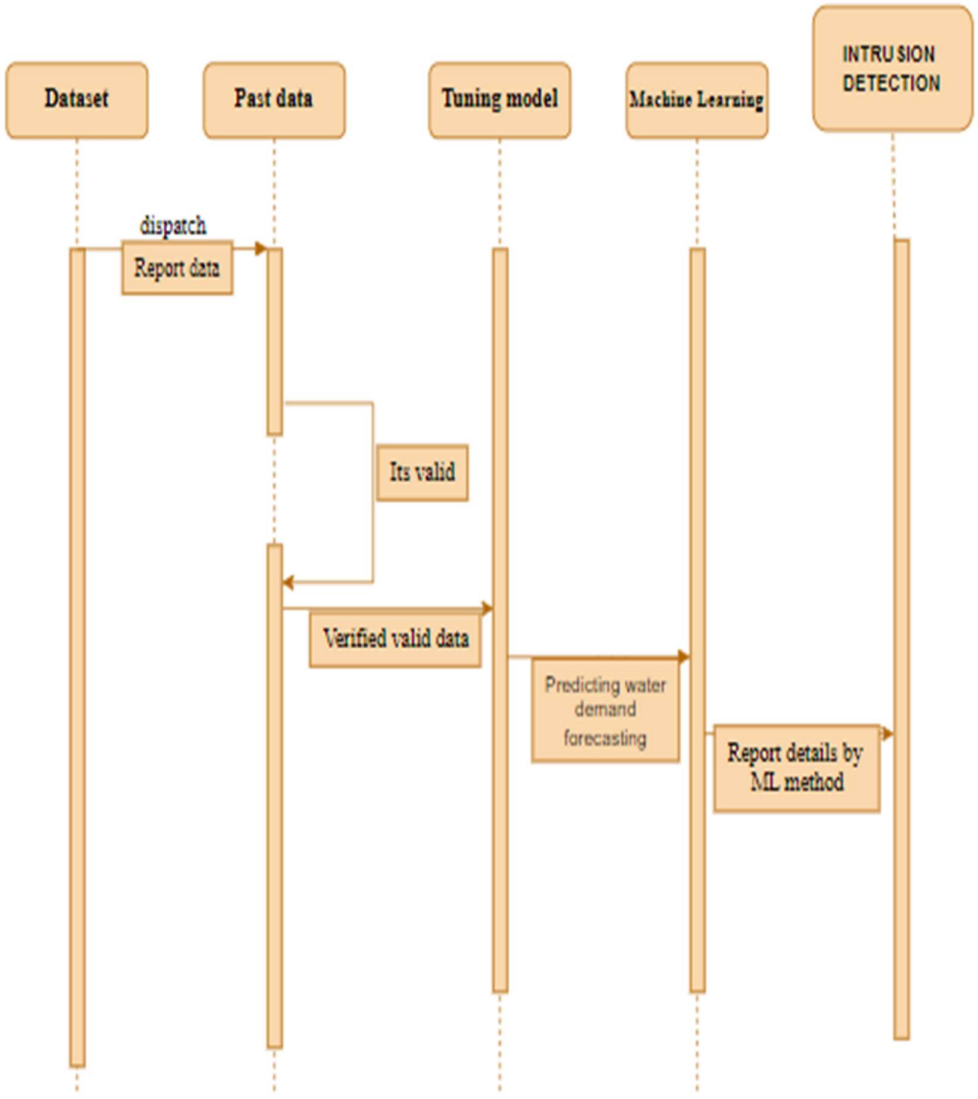


Figure 3.5.4 Sequence diagram

3.5.5 Entity Relationship Diagram (ERD)

An entity relationship diagram (ERD), also known as an entity relationship model, is a graphical representation of an information system that depicts the relationships among people, objects, places, concepts or events within that system. An ERD is a data modelling technique that can help define business processes and be used as the foundation for a relational database. Entity relationship diagrams provide a visual starting point for database design that can also be used to help determine information system requirements throughout an organization

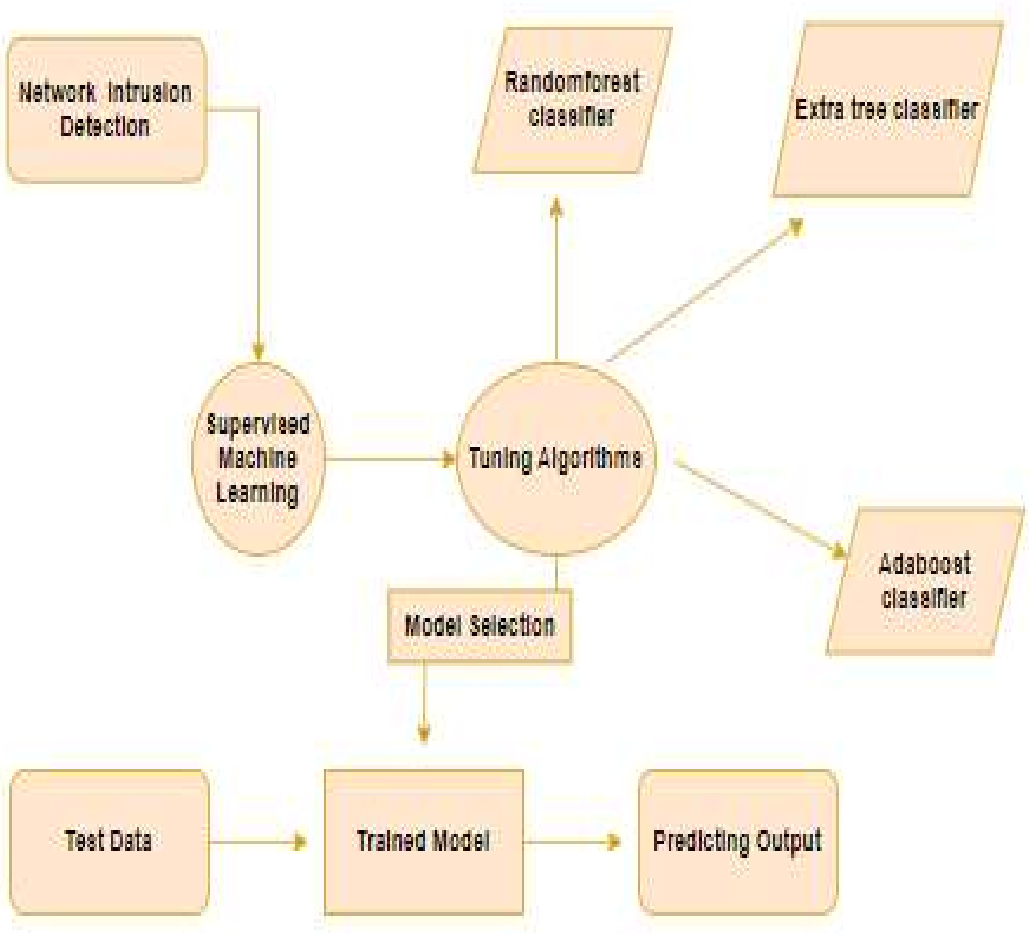


Figure 3.5.5 ERD diagram

CHAPTER 4

SYSTEM IMPLEMENTATION

LIST OF MODULES

- Data Pre-processing
- Data Analysis of Visualization
- Implementing Algorithm 1
- Implementing Algorithm 2
- Implementing Algorithm 3
- Deployment

4.1 DATA PRE-PROCESSING

Validation techniques in machine learning are used to get the error rate of the Machine Learning (ML) model, which can be considered as close to the true error rate of the dataset. If the data volume is large enough to be representative of the population, you may not need the validation techniques. However, in real-world scenarios, to work with samples of data that may not be a true representative of the population of given dataset. To finding the missing value, duplicate value and description of data type whether it is float variable or integer. The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyper parameters.

The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. The validation set is used to evaluate a given model, but this is for frequent evaluation. It as machine learning engineers uses this data to fine-tune the model hyper parameters. Data collection, data analysis, and the process of addressing data content, quality, and structure can add up to a time-consuming to-do list. During the process of data identification, it helps to understand your data and its properties; this knowledge will help you choose which algorithm to use to build your model.

A number of different data cleaning tasks using Python's Pandas library and specifically, it focuses on probably the biggest data cleaning task, missing values and it able to more quickly clean data. It wants to spend less time cleaning data, and more time exploring and modelling.

Some of these sources are just simple random mistakes. Other times, there can be a deeper reason why data is missing. It's important to understand these different types of missing data from a statistics point of view. The type of missing data will influence how to deal with filling in the missing values and to detect missing values, and do some basic imputation and detailed statistical approach for dealing with missing data. Before, joint into code, it's important to understand the sources of missing data. Here are some typical reasons why data is missing:

- User forgot to fill in a field.
- Data was lost while transferring manually from a legacy database.
- There was a programming error.
- Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

Variable identification with Uni-variate, Bi-variate and multi-variate analysis:

- import libraries for access and functional purpose and read the given dataset
- General Properties of Analysing the given dataset
- Display the given dataset in the form of data frame
- show columns
- shape of the data frame
- To describe the data frame
- Checking data type and information about dataset
- Checking for duplicate data
- Checking Missing values of data frame

- Checking unique values of data frame
- Checking count values of data frame
- Rename and drop the given data frame
- To specify the type of values
- To create extra columns

	duration	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins
duration	1.000000	0.093053	0.146755	0.070737	0.034878	-0.001553	-0.009866	0.003830	0.000705	0.009528
service	0.093053	1.000000	0.237246	-0.001626	0.003599	-0.009762	0.084571	0.010969	-0.062840	0.033040
flag	0.146755	0.237246	1.000000	0.001735	0.008683	0.017585	-0.045089	-0.003898	-0.041601	0.000219
src_bytes	0.070737	-0.001626	0.001735	1.000000	0.000204	-0.000109	-0.000693	-0.000059	0.000295	-0.000208
dst_bytes	0.034878	0.003599	0.008683	0.000204	1.000000	-0.000069	-0.000440	0.000248	-0.000344	0.000504
land	-0.001553	-0.009762	0.017585	-0.000109	-0.000069	1.000000	-0.001261	-0.000109	-0.001340	-0.000381
wrong_fragment	-0.009866	0.084571	-0.045089	-0.000693	-0.000440	-0.001261	1.000000	-0.000692	-0.008508	-0.002418
urgent	0.003830	0.010969	-0.003898	-0.000059	0.000248	-0.000109	-0.000692	1.000000	0.000293	0.097507
hot	0.000705	-0.062840	-0.041601	0.000295	-0.000344	-0.001340	-0.008508	0.000293	1.000000	0.003715
num_failed_logins	0.009528	0.033040	0.000219	-0.000208	0.000504	-0.000381	-0.002418	0.097507	0.003715	1.000000
logged_in	-0.064218	-0.135484	-0.353517	-0.003353	-0.002894	-0.011402	-0.072418	0.007299	0.116435	-0.006439
num_compromised	0.042679	0.019757	0.000114	-0.000086	0.001233	-0.000164	-0.001044	0.033329	0.002014	0.019085
root_shell	0.052791	0.033283	0.000531	-0.000272	0.001069	-0.000516	-0.003280	0.075199	0.015379	0.032567
su_attempted	0.087183	0.042855	0.012790	-0.000186	0.001133	-0.000344	-0.002187	0.097710	0.000130	0.073175
num_root	0.045519	0.020098	-0.000117	-0.000093	0.001229	-0.000174	-0.001108	0.032470	0.001510	0.018112
num_file_creations	0.099116	0.040680	-0.001422	-0.000179	0.000089	-0.000369	-0.002343	0.024918	0.028716	0.021774
num_shells	-0.001593	0.001989	-0.009377	-0.000134	-0.000083	-0.000262	-0.001665	-0.000144	0.004723	-0.000503
num_access_files	0.070420	0.030137	-0.015871	-0.000309	0.000339	-0.000581	-0.003689	0.010803	-0.001987	0.000652

```
df.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_di
0	0	icmp	20	2	491	0	0	0	0	0	...	25	0.17	
1	0	udp	45	2	146	0	0	0	0	0	...	1	0.00	
2	0	icmp	50	4	0	0	0	0	0	0	...	26	0.10	
3	0	icmp	25	2	232	8153	0	0	0	0	...	255	1.00	
4	0	icmp	25	2	199	420	0	0	0	0	...	255	1.00	

Figure 4.1.1 Dataset Information after pre-processing

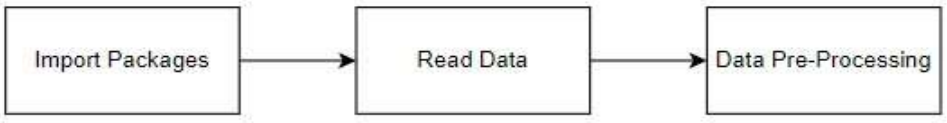


Figure 4.1.2 Data Pre-processing Process

GIVEN INPUT EXPECTED OUTPUT

input: data

output: removing noisy data

4.2 DATA ANALYSIS OF VISUALIZATION

Data visualization is an important skill in applied statistics and machine learning. Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding. This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral and stakeholders than measures of association or significance. Data visualization and exploratory data analysis are whole fields themselves and it will recommend a deeper dive into some the books mentioned at the end.

Sometimes data does not make sense until it can look at in a visual form, such as with charts and plots. Being able to quickly visualize of data samples and others is an important skill both in applied statistics and in applied machine learning. It will discover the many types of plots that you will need to know when visualizing data in Python and how to use them to better understand your own data.

- How to chart time series data with line plots and categorical quantities with bar charts.
- How to summarize data distributions with histograms and box plots.

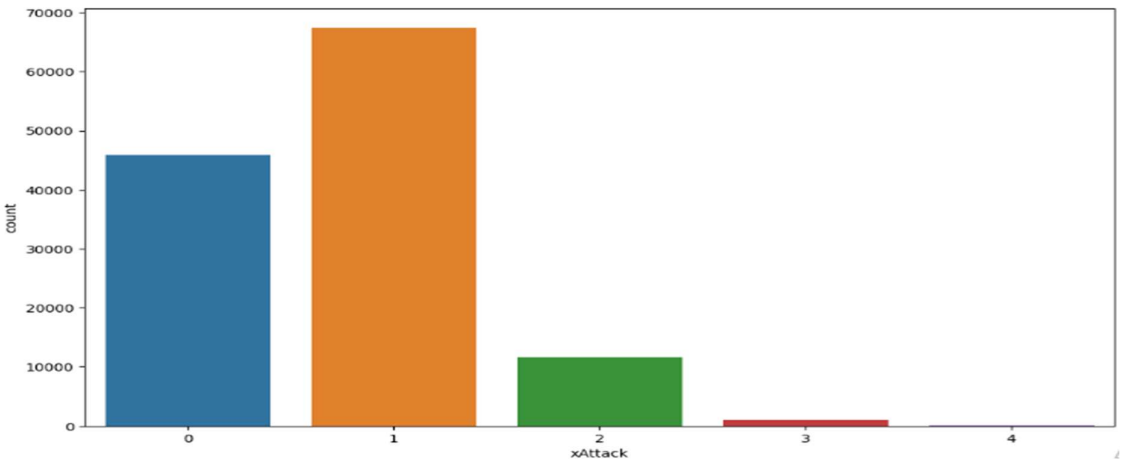


Figure 4.2.1 Histogram on xAttack

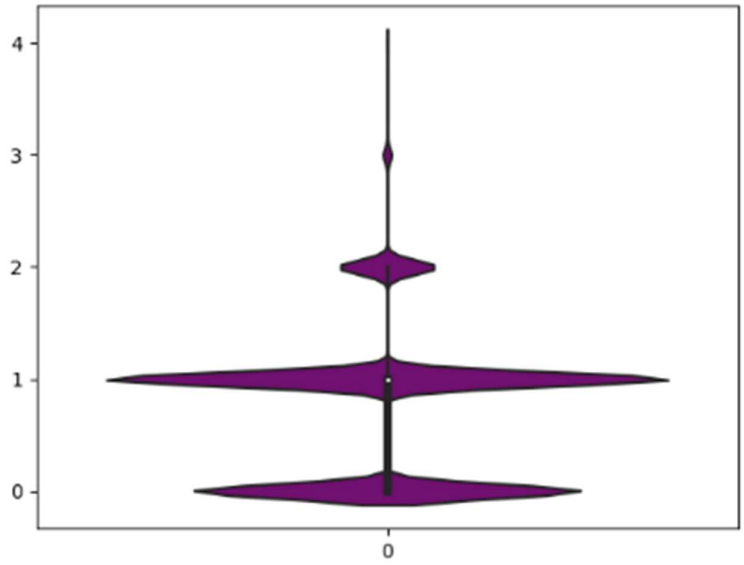


Figure 4.2.2 Visualised Server Rate

Figure 4.2.3 Data Visualisation Process

GIVEN INPUT EXPECTED OUTPUT

input: data

output: visualized data

4.3 ALGORITHM AND TECHNIQUES

It is important to compare the performance of multiple different machine learning algorithms consistently and it will discover to create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. It can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. It needs to be able to use these estimates to choose one or two best models from the suite of models that you have created. When have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two to finalize. A way to do this is to use different visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies.

In the next section you will discover exactly how you can do that in Python with scikit-learn. The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data and it can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

Performance Metrics to calculate:

False Positives (FP): A person who will pay predicted as defaulter. When actual class is no and predicted class is yes. E.g., if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

False Negatives (FN): A person who default predicted as payer. When actual class is yes but predicted class in no. E.g., if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

True Positives (TP): A person who will not pay predicted as defaulter. These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. E.g., if actual class value indicates that this passenger survived and predicted class tells you the same thing.

True Negatives (TN): A person who default predicted as payer. These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g., if actual class says this passenger did not survive and predicted class tells you the same thing.

True Positive Rate (TPR) = TP / (TP + FN)

False Positive rate (FPR) = FP / (FP + TN)

Accuracy: The Proportion of the total number of predictions that is correct otherwise overall how often the model predicts correctly defaulters and non-defaulters.

Accuracy calculation:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure

but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Precision: The proportion of positive predictions that are actually correct.

Precision = TP / (TP + FP)

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labelled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Recall: The proportion of positive observed values correctly predicted. (The proportion of actual defaulters that the model will correctly predict)

Recall = TP / (TP + FN)

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

General Formula:

F- Measure = 2TP / (2TP + FP + FN)

F1-Score Formula:

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

The below 3 different algorithms are compared:

- Extra tree classifier
- Random Forest classifier
- Ada Boost classifier

4.3.1 Adaboost Classifier

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

AdaBoost can be used to boost the performance of any machine learning algorithm. It is best used with weak learners. These are models that achieve accuracy just above random chance on a classification problem. The most suited and therefore most common algorithm used with AdaBoost are decision trees with one level. How does the AdaBoost algorithm work explain?

It works on the principle of learners growing sequentially. Except for the first, each subsequent learner is grown from previously grown learners. In simple words, weak learners are converted into strong ones. The AdaBoost algorithm works on the same principle as boosting with a slight difference.

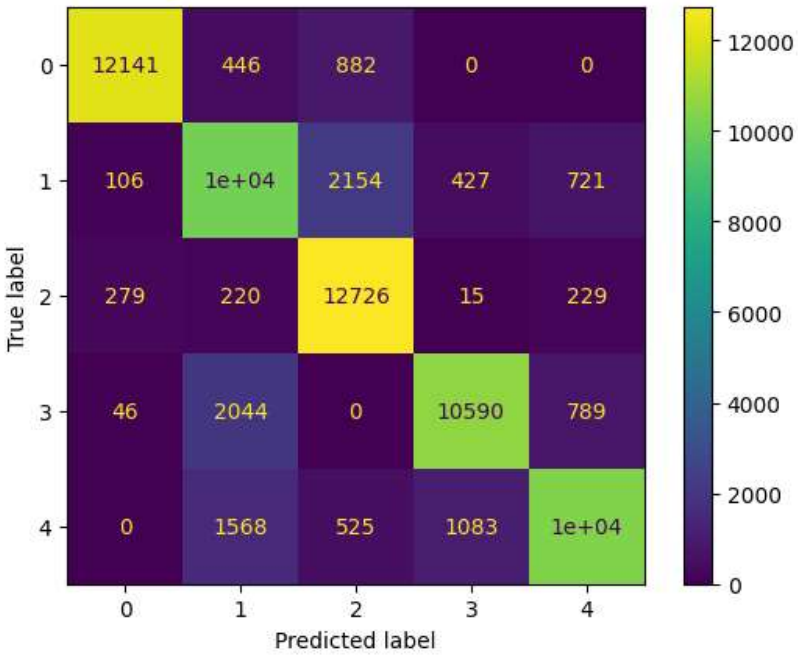


Figure 4.3.1 AdaBoost confusion Matrix

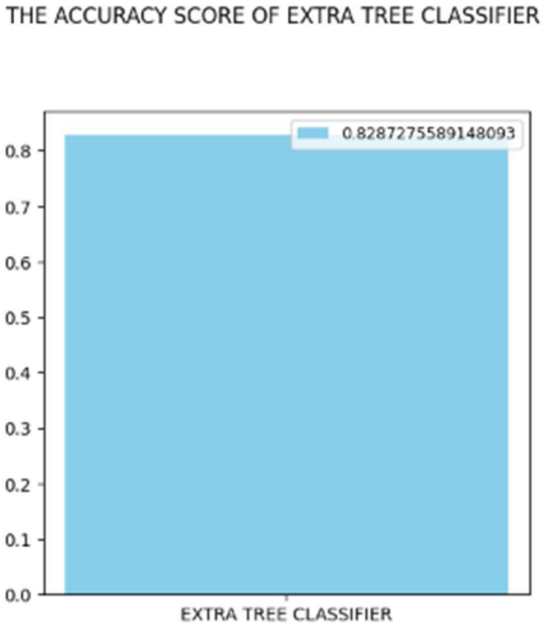


Figure 4.3.2 AdaBoost Accuracy Graph

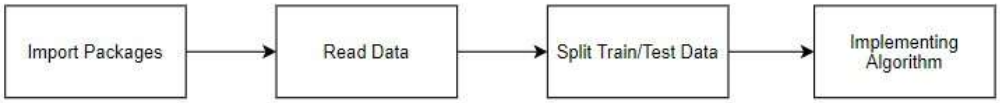


Figure 4.3.3 Algorithm Implementation Process

GIVEN INPUT EXPECTED OUTPUT

input: data

output: getting accuracy

4.3.2 Extra tree Classifier

The Extra Trees Classifier, short for Extremely Randomized Trees Classifier, is a powerful ensemble machine learning algorithm used for both classification and regression tasks. It belongs to the family of decision tree-based methods and shares similarities with Random Forests. However, what sets Extra Trees apart is its extreme randomness during the tree-building process.

When constructing decision trees, Extra Trees randomly selects feature subsets for each node and then chooses the best split among them. This randomness, achieved through a combination of random feature selection and random threshold values, leads to an even greater degree of diversity among the individual trees in the ensemble. This diversity helps reduce overfitting, a common problem in decision tree models.

Extra Trees combines the predictions from multiple such randomized trees to make a final classification decision. During training, it aggregates the votes or averages the predictions of its constituent trees to arrive at the most likely class label for a given input. The algorithm's randomness not only boosts its robustness against noise and outliers but also makes it less computationally intensive than traditional decision trees.

Furthermore, Extra Trees can handle high-dimensional data and is relatively insensitive to hyperparameter tuning, making it an excellent choice for quick and effective classification tasks. However, like any machine learning algorithm, it

requires careful consideration of parameters such as the number of trees in the ensemble and the maximum depth of the trees to ensure optimal performance for a specific problem. In summary, the Extra Trees Classifier is a versatile and robust ensemble method that leverages extreme randomness to mitigate overfitting and deliver accurate classification results across a variety of applications.

```
cm=confusion_matrix(y_test, predicted)
print('THE CONFUSION MATRIX SCORE OF EXTRA TREE CLASSIFIER:\n\n')
print(cm)
print("\n\nDISPLAY CONFUSION MATRIX : \n\n")
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(y_test, predicted, labels=ABC.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=ABC.classes_)
disp.plot()
plt.show()
```

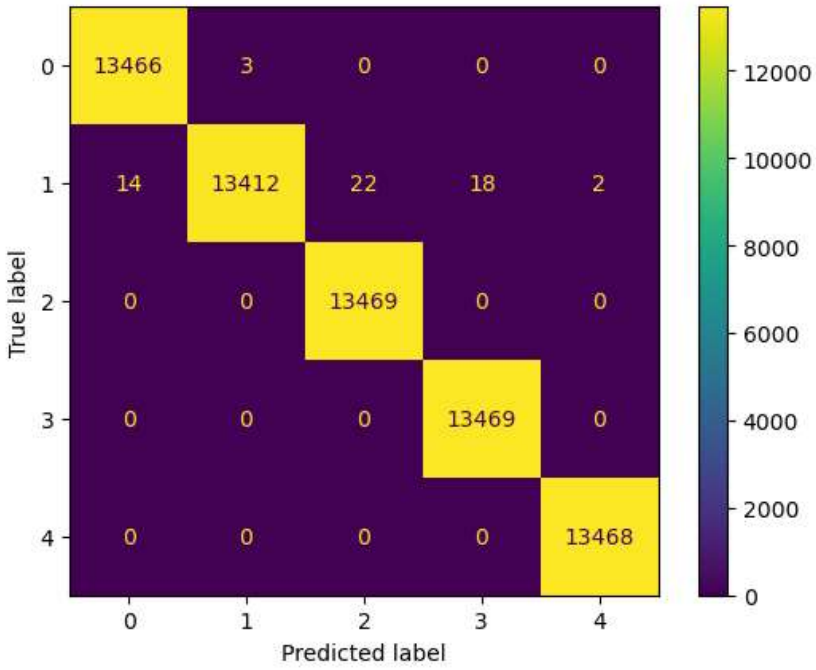


Figure 4.3.4 Extra Tree Classifier Confusion Matrix

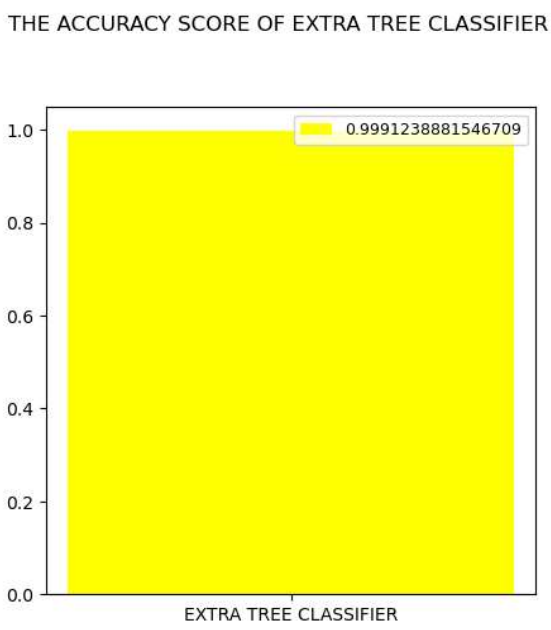


Figure 4.3.5 Extra Tree Classifier Accuracy Graph

GIVEN INPUT EXPECTED OUTPUT

input: data

output: getting accuracy

4.3.3 Random Forest Classifier

Random Forest is an ensemble learning algorithm that leverages the power of decision trees for both classification and regression tasks. Unlike a single decision tree, Random Forest constructs a multitude of trees, each trained on a random subset of the training data using a process called bootstrap sampling. Moreover, at each split in the tree, a random subset of features is considered, fostering diversity among the trees. The final prediction is determined by aggregating the outputs of all individual trees through a majority vote in classification or an average in regression. Notably, Random Forest introduces an out-of-bag (OOB)

error estimation, utilizing data points not used in the training of each tree. The algorithm is robust, resistant to overfitting, and provides insights into feature importance based on the contribution of features to reducing impurity across the ensemble. Widely employed due to its versatility, Random Forest finds applications across various domains, demonstrating efficacy in handling high-dimensional data and capturing intricate relationships within datasets.

```
[[13469     0     0     0     0]
 [    1 13462     2     1     2]
 [    0     0 13469     0     0]
 [    0     0     0 13469     0]
 [    0     0     0     0 13468]]

DISPLAY CONFUSION MATRIX :
```

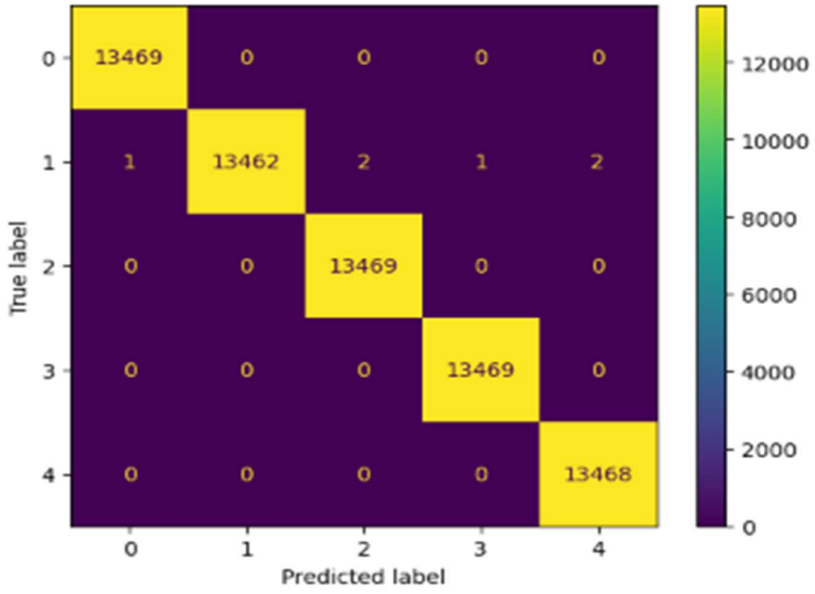


Figure 4.3.6 Random Forest Classifier Confusion Matrix

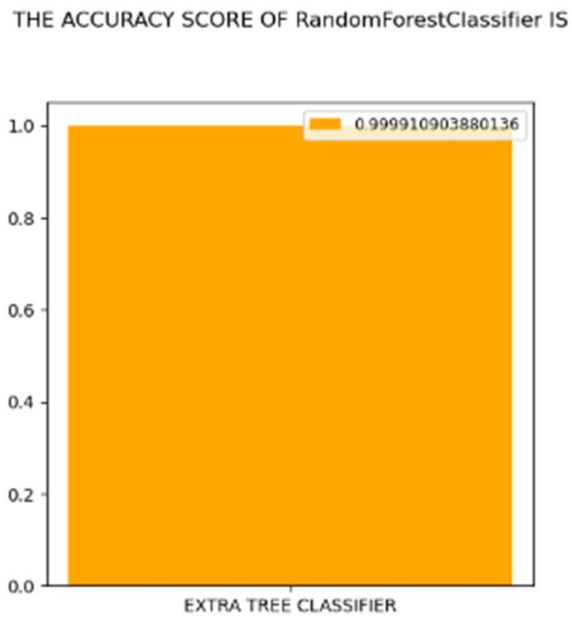


Figure 4.3.7 Random Forest Classifier Accuracy Graph

GIVEN INPUT EXPECTED OUTPUT

input: data

output: getting accuracy

4.4 DEPLOYMENT

Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source. Django was designed to help developers take applications from concept to completion as quickly as possible.

Django takes security seriously and helps developers avoid many common security mistakes. Some of the busiest sites on the web leverage Django's ability to quickly and flexibly scale. With Django, you can take web applications from concept to launch in a matter of hours. Django takes care of much of the hassle of

web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Fully loaded

Django includes dozens of extras you can use to handle common web development tasks. Django takes care of user authentication, content administration, site maps, RSS feeds, and many more tasks — right out of the box.

Reassuringly secure

Django takes security seriously and helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery and clickjacking. Its user authentication system provides a secure way to manage user accounts and passwords.

Incredibly versatile

Companies, organizations and governments have used Django to build all sorts of things — from content management systems to social networks to scientific computing platforms.

1. **Model-View-Template (MVT) Architecture**
    - Django follows a variation of the traditional Model-View-Controller (MVC) architecture known as Model-View-Template (MVT).
    - **Model:** Represents the data structure and business logic, defining how data is stored and retrieved.
    - **View:** Handles user interface and presentation logic, processing user requests and returning appropriate responses.
    - **Template:** Manages the generation of HTML dynamically, separating the presentation layer from the business logic.

2. **Object-Relational Mapping (ORM)**
    - Django features a powerful Object-Relational Mapping (ORM) system that enables developers to interact with databases using Python objects instead of raw SQL queries.

- Models in Django define the data structure, including fields, relationships, and constraints.

3. **URL Routing**
   - Django uses a URL dispatcher to map URLs to views. This is done through a urls.py file, which contains patterns or regular expressions to route requests to the appropriate views.

4. **Views and Controllers**

   - In Django, views are responsible for processing user requests and returning appropriate responses. They contain the business logic of the application.
   - While Django follows the MVT pattern, views often take on the responsibilities of both controllers and views in the traditional MVC pattern.

5. **Templates and Presentation Logic**
   - Templates in Django are responsible for generating dynamic HTML. They use a simple syntax to insert variables, control structures, and template tags.
   - Templates separate the presentation logic from the business logic, promoting a clean and modular design.

6. **Middleware**
   - Django middleware is a way to process requests globally before they reach the view or after the view has processed the request.
   - Middleware components can perform tasks such as authentication, logging, or modifying the response before it is sent to the client.

7. **Forms and User Input Handling**

   - Django provides a robust form handling system for managing user input. Forms can be used to validate and process data submitted by users.

- Forms simplify the process of handling HTML forms and help in maintaining a secure and consistent approach to data validation.

8. **Admin Interface**
   - Django includes a built-in admin interface that allows developers to manage application data through a web-based interface.
   - The admin interface is automatically generated based on the models defined in the application.

9. **Static Files and Media:**
   - Django provides mechanisms for handling static files (e.g., CSS, JavaScript, images) and media files (e.g., user uploads).
   - The `{% static %}` template tag is used to reference static files, while the `MEDIA_ROOT` and `MEDIA_URL` settings manage media files.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 PERFORMANCE ANALYSIS

It is important to compare the performance of multiple different machine learning algorithms consistently and it will discover to create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn. It can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare. Each model will have different performance characteristics. Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. It needs to be able to use these estimates to choose one or two best models from the suite of models that you have created. When have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives. The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two to finalize. A way to do this is to use different visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies.

In the next section you will discover exactly how you can do that in Python with scikit-learn. The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data and it can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

Performance Metrics to calculate:

False Positives (FP): A person who will pay predicted as defaulter. When actual class is no and predicted class is yes. E.g. if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

False Negatives (FN): A person who default predicted as payer. When actual class is yes but predicted class in no. E.g. if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

True Positives (TP): A person who will not pay predicted as defaulter. These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. E.g. if actual class value indicates that this passenger survived and predicted class tells you the same thing.

True Negatives (TN): A person who default predicted as payer. These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g. if actual class says this passenger did not survive and predicted class tells you the same thing.

True Positive Rate(TPR) = TP / (TP + FN)

False Positive rate(FPR) = FP / (FP + TN)

Accuracy: The Proportion of the total number of predictions that is correct otherwise overall how often the model predicts correctly defaulters and non-defaulters.

Accuracy calculation:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same.

Precision: The proportion of positive predictions that are actually correct.

Precision = TP / (TP + FP)

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labelled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Recall: The proportion of positive observed values correctly predicted. (The proportion of actual defaulters that the model will correctly predict)

Recall = TP / (TP + FN)

Recall(Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

General Formula:

F- Measure = 2TP / (2TP + FP + FN)

F1-Score Formula:

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

CHAPTER 6
CONCLUSION AND FUTURE WORK

The analytical process started from data cleaning and processing, missing value, exploratory analysis and finally model building and evaluation. The best accuracy on public test set of higher accuracy score algorithm will be find out. The founded one is used in the application which can help to find the type of intrusions. In conclusion, this project has successfully addressed the objective of developing an effective Intrusion Detection System (IDS) utilizing learning-based approaches for monitoring network activities. Through the implementation of this system, the capability to promptly detect suspicious or potentially malicious actions within the network has been significantly enhanced, providing security operations centre (SOC) analysts and incident responders with early alerts for proactive threat mitigation. Moving forward, future work could focus on several avenues for improvement. Firstly, refining the machine learning algorithms used in the IDS to enhance accuracy and reduce false positives will be crucial. Additionally, expanding the scope of the IDS to cover a broader range of network protocols and behaviours could further strengthen its effectiveness. Furthermore, integrating advanced threat intelligence feeds and adaptive learning mechanisms into the system could enhance its capability to detect emerging threats and adapt to evolving attack strategies. Finally, conducting comprehensive evaluations and real-world testing to validate the performance and reliability of the IDS under various scenarios will be essential for its practical deployment and continuous improvement in network security. Overall, by addressing these areas of future work, the IDS can continue to evolve as a vital tool for safeguarding network infrastructures against cyber threats. Future work includes Deploying the project in the cloud. To optimize the work to implement in the IOT system.

REFERENCES

[1] Vanlalruata Hnamte ∗, Jamal Hussain (2023) “DCNNBiLSTM: An Efficient Hybrid Deep Learning-Based Intrusion Detection System”

[2] Md. Alamin Talukdera, Khondokar Fida Hasanb, Md. Manowarul Islama, Md Ashraf Uddina, Arnisha Akhtera, Mohammad Abu Yousufc, Fares Alharbid, Mohammad Ali Moni. (2023) “A Dependable Hybrid Machine Learning Model for Network Intrusion Detection”

[3] Saddam Hossen, Anirudh Janagam (2018) “Analysis of Network Intrusion Detection System with Machine Learning Algorithms (Deep Reinforcement Learning Algorithm)”

[4] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, Farhan Ahmad (2020) “Network intrusion detection system: A systematic study of machine learning and deep learning approaches”

APPENDICES

A.1 SDG GOALS

The United Nations created 17 world development goals called the Sustainable Development Goals (SDG). They were created in 2015 with the aim of "peace and prosperity for people and the planet, now and into the future". The application of machine learning techniques for the prediction of network intrusion detection system aligns with several United Nations Sustainable Development Goals (SDGs).

Goal 9: Industry, Innovation, and Infrastructure - This goal emphasizes the importance of building resilient infrastructure, promoting sustainable industrialization, and fostering innovation. The development of an IDS aligns with this goal by contributing to the creation of secure and resilient network infrastructure through innovative technologies and approaches.

Goal 16: Peace, Justice, and Strong Institutions - This goal aims to promote peaceful and inclusive societies for sustainable development, provide access to justice for all, and build effective, accountable, and inclusive institutions at all levels. Enhancing network security through the implementation of an IDS supports this goal by contributing to the establishment of stronger cybersecurity measures, which are essential for maintaining peace and security in the digital domain and ensuring the protection of individuals' rights and privacy.

Goal 17: Partnerships for the Goals - This goal underscores the importance of global partnerships and collaboration in achieving the SDGs. The development and deployment of an IDS require cooperation among various stakeholders, including governments, private sector entities, academia, and civil society organizations. By fostering partnerships and knowledge-sharing initiatives, the project can contribute to building collective efforts toward enhancing cybersecurity and achieving sustainable development objectives.

A.2 SOURCE CODE

MODULE – 1

Data Preprocessing and Data Cleaning

```
import pandas as pd

import numpy as np

import warnings

warnings.filterwarnings('ignore')

df = pd.read_csv('INTRUSION.csv')

df.head()

df.tail()

##Before removing the null data

df.shape

df.size

df.columns


##After removing the null data

df.isnull()

df.describe()

df.info()

df = df.dropna()

df['xAttack'].unique()

df.drop(["xAttack","protocol_type"],axis=1).corr()

pd.crosstab(df["dst_host_srv_rerror_rate"], df["diff_srv_rate"])

df.groupby(["dst_host_serror_rate","is_guest_login"]).groups

df["xAttack"].value_counts()

pd.Categorical(df["protocol_type"]).describe()
```

```
df.duplicated()
sum(df.duplicated())
df = df.drop_duplicates()
sum(df.duplicated())
```

MODULE -2

Data visualization

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('INTRUSION.csv')
df.head()
df.columns
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
var = ['protocol_type','xAttack']
for i in var:
    df[i] = le.fit_transform(df[i]).astype(int)
## Histogram with xAttack
plt.figure(figsize=(12,7))
sns.countplot(x='xAttack',data=df)
```

```
## Protocol and xAttack Histogram

plt.figure(figsize=(15,5))

plt.subplot(1,2,1)

plt.hist(df['xAttack'],color='red')

plt.subplot(1,2,2)

plt.hist(df['protocol_type'],color='blue')


## Histogram for every column values

df.hist(figsize=(15,55),layout=(15,4), color='gray')

plt.show()


## Histogram with srv_diff_host_rate

df['srv_diff_host_rate'].hist(figsize=(10,5),color='orange')


## Lineplot with Flag

sns.lineplot(df['flag'], color='brown') # scatter, plot, triplot, stackplot

## Violin Plot with xAttack

sns.violinplot(df['xAttack'], color='purple')

## Plot on Density

df['dst_host_diff_srv_rate'].plot(kind='density')

sns.displot(df['flag'], color='coral') # residplot, scatterplot
```

```
## Pie Chart on xAttack

def plot(df, variable):

    dataframe_pie = df[variable].value_counts()

    ax = dataframe_pie.plot.pie(figsize=(9,9), autopct='%1.2f%%', fontsize = 10)

    ax.set_title(variable + ' \n', fontsize = 10)

    return np.round(dataframe_pie/df.shape[0]*100,2)

plot(df, 'xAttack')
```

MODULE -3

Implementing Extra Tree Classification Algorithm

```
# Importing library packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')
# Loading Given Dataset
df = pd.read_csv('INTRUSION.csv')
df.head()
df.columns
df=df.dropna()
df.shape
df.columns
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

var = ['protocol_type','xAttack']

for i in var:
    df[i] = le.fit_transform(df[i]).astype(int)

# Preprocessing, split test and dataset, split response variable
x1 = df.drop(labels='xAttack', axis=1)
# Response Variable
y1 = df.loc[:,'xAttack']
# Splitting for train and test
# import imblearn
from imblearn.over_sampling import RandomOverSampler
from collections import Counter
ros =RandomOverSampler(random_state=42)
x,y=ros.fit_resample(x1,y1)
print("OUR DATASET COUNT         : ", Counter(y1))
print("OVER SAMPLING DATA COUNT  : ", Counter(y))
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
random_state=42, stratify=y)
print("NUMBER OF TRAIN DATASET    : ", len(x_train))
print("NUMBER OF TEST DATASET      : ", len(x_test))
print("TOTAL NUMBER OF DATASET    : ", len(x_train)+len(x_test))
print("NUMBER OF TRAIN DATASET    : ", len(y_train))
print("NUMBER OF TEST DATASET      : ", len(y_test))
```

```
print("TOTAL NUMBER OF DATASET    : ", len(y_train)+len(y_test))

# Implementing Extra Tree Classifier

from sklearn.tree import ExtraTreeClassifier

ETC = ExtraTreeClassifier()

ETC.fit(x_train,y_train)

# Training

predicted = ETC.predict(x_test)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,predicted)

print('THE CONFUSION MATRIX SCORE OF SUPPORT VECTOR
MACHINE:\n\n\n',cm)

# Finding Accuracy

from sklearn.metrics import accuracy_score

a = accuracy_score(y_test,predicted)

print("THE ACCURACY SCORE OF EXTRA TREE CLASSIFIER IS
:",a*100)

# Classification Report

from sklearn.metrics import hamming_loss

hl = hamming_loss(y_test,predicted)

print("THE HAMMING LOSS OF EXTRA TREE CLASSIFIER IS :",hl*100)

from sklearn.metrics import classification_report

P = classification_report(y_test,predicted)

print("THE CLASSIFICATION REPORT OF EXTRA TREE CLASSIFIER IS
:\n\n",P)

# Finding Confusion Matrix

cm=confusion_matrix(y_test, predicted)

print('THE CONFUSION MATRIX SCORE OF EXTRA TREE
CLASSIFIER:\n\n')

print(cm)
```

```
print("\n\nDISPLAY CONFUSION MATRIX : \n\n")
from sklearn.metrics import ConfusionMatrixDisplay


cm = confusion_matrix(y_test, predicted, labels=ETC.classes_)
disp =
ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=ETC.classes_)
disp.plot()
plt.show()
def graph():
   import matplotlib.pyplot as plt
   data=[a]
   alg="EXTRA TREE CLASSIFIER"
   plt.figure(figsize=(5,5))
   b=plt.bar(alg,data,color=("YELLOW"))
   plt.title("THE ACCURACY SCORE OF EXTRA TREE CLASSIFIER
IS\n\n\n")
   plt.legend(b,data,fontsize=9)
graph()
```

MODULE -4

Implementing Adaboost Algorithm

```
# Import Library Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import warnings
warnings.filterwarnings('ignore')
# Load Given DataSet
df = pd.read_csv('INTRUSION.csv')
df.head()
df.columns
df=df.dropna()
df.shape
df.columns
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
var = ['protocol_type','xAttack']
for i in var:
    df[i] = le.fit_transform(df[i]).astype(int)

# Preprocessing, Split test and Dataset, Split Response Variable
x1 = df.drop(labels='xAttack', axis=1)
y1 = df.loc[:,'xAttack']
# Splitting for Train and test
import imblearn
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros =RandomOverSampler(random_state=42)
x,y=ros.fit_resample(x1,y1)
print("OUR DATASET COUNT         : ", Counter(y1))
print("OVER SAMPLING DATA COUNT  : ", Counter(y))
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
random_state=42, stratify=y)
print("NUMBER OF TRAIN DATASET    : ", len(x_train))
print("NUMBER OF TEST DATASET     : ", len(x_test))
print("TOTAL NUMBER OF DATASET    : ", len(x_train)+len(x_test))
print("NUMBER OF TRAIN DATASET    : ", len(y_train))
print("NUMBER OF TEST DATASET     : ", len(y_test))
print("TOTAL NUMBER OF DATASET    : ", len(y_train)+len(y_test))

# Implementing AdaBoost Algorithm
from sklearn.ensemble import AdaBoostClassifier
# Training
ABC = AdaBoostClassifier()
ABC.fit(x_train,y_train)
predicted = ABC.predict(x_test)

# Finding Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,predicted)
print('THE CONFUSION MATRIX SCORE OF SUPPORT VECTOR
MACHINE:\n\n\n',cm)
# Finding Accuracy
from sklearn.metrics import accuracy_score
a = accuracy_score(y_test,predicted)
print("THE ACCURACY SCORE OF EXTRA TREE CLASSIFIER IS
:",a*100)
# Finding Hamming Loss
from sklearn.metrics import hamming_loss
```

```
hl = hamming_loss(y_test,predicted)
print("THE HAMMING LOSS OF EXTRA TREE CLASSIFIER IS :",hl*100)
# Finding  Classification Report
from sklearn.metrics import classification_report
P = classification_report(y_test,predicted)
print("THE CLASSIFICATION REPORT OF EXTRA TREE CLASSIFIER IS :\n\n",P)
cm=confusion_matrix(y_test, predicted)
print('THE CONFUSION MATRIX SCORE OF EXTRA TREE CLASSIFIER:\n\n')
print(cm)
print("\n\nDISPLAY CONFUSION MATRIX : \n\n")
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(y_test, predicted, labels=ABC.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=ABC.classes_)
disp.plot()
plt.show()

def graph():
    import matplotlib.pyplot as plt
    data=[a]
    alg="EXTRA TREE CLASSIFIER"
    plt.figure(figsize=(5,5))
    b=plt.bar(alg,data,color=("SKYBLUE"))
    plt.title("THE ACCURACY SCORE OF EXTRA TREE CLASSIFIER IS\n\n\n")
    plt.legend(b,data,fontsize=9)
```

```
graph()
```

Module -5

Implementing Extra Tree Classifier Algorithm

```
# Import Library Package
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

#Load Given Dataset
df = pd.read_csv('INTRUSION.csv')
df.head(60)
df.columns
df=df.dropna()
df.shape
df.columns
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
var = ['protocol_type','xAttack']
for i in var:
    df[i] = le.fit_transform(df[i]).astype(int)
df.head()
```

```
# Preprocessing, Split Test and Dataset, Split Response Variable
x1 = df.drop(labels='xAttack', axis=1)
y1 = df.loc[:,'xAttack']

#import imblearn
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

ros =RandomOverSampler(random_state=42)
x,y=ros.fit_resample(x1,y1)
print("OUR DATASET COUNT        : ", Counter(y1))
print("OVER SAMPLING DATA COUNT  : ", Counter(y))

# Splitting for Train and Test
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
random_state=42, stratify=y)
print("NUMBER OF TRAIN DATASET    : ", len(x_train))
print("NUMBER OF TEST DATASET     : ", len(x_test))
print("TOTAL NUMBER OF DATASET    : ", len(x_train)+len(x_test))
print("NUMBER OF TRAIN DATASET    : ", len(y_train))
print("NUMBER OF TEST DATASET     : ", len(y_test))
print("TOTAL NUMBER OF DATASET    : ", len(y_train)+len(y_test))

# Training Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
```

```
predicted = rf.predict(x_test)

# Finding Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,predicted)
print('THE CONFUSION MATRIX SCORE OF
RandomForestClassifier:\n\n\n',cm)

# Finding Accuracy Score
from sklearn.metrics import accuracy_score
a = accuracy_score(y_test,predicted)
print("THE ACCURACY SCORE OF RandomForestClassifier IS :",a*100)
 # Finding Hamming Loss
from sklearn.metrics import hamming_loss
hl = hamming_loss(y_test,predicted)
print("THE HAMMING LOSS OF RandomForestClassifier IS :",hl*100)

# Finding the Classification Report
from sklearn.metrics import classification_report
P = classification_report(y_test,predicted)
print("THE CLASSIFICATION REPORT OF RandomForestClassifier IS
:\n\n",P)
cm=confusion_matrix(y_test, predicted)
print('THE CONFUSION MATRIX SCORE OF RandomForestClassifier:\n\n')
print(cm)
print("\n\nDISPLAY CONFUSION MATRIX : \n\n")
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(y_test, predicted, labels=rf.classes_)
```

```
disp =
ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=rf.classes_)
disp.plot()
plt.show()

def graph():
    import matplotlib.pyplot as plt
    data=[a]
    alg="EXTRA TREE CLASSIFIER"
    plt.figure(figsize=(5,5))
    b=plt.bar(alg,data,color=("ORANGE"))
    plt.title("THE ACCURACY SCORE OF RandomForestClassifier IS\n\n\n")
    plt.legend(b,data,fontsize=9)
graph()
```

DEPLOY

Manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'PROJECT.settings')
```

```
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

forms.py

```
from django import forms
from . models import UserPersonalModel
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User


class UserRegisterForm(UserCreationForm):
    class Meta:
```

```
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class UserPersonalForm(forms.ModelForm):
    class Meta:
        model = UserPersonalModel
        fields = '__all__'
```

Models.py

```
from django.db import models
from django.contrib.auth.models import User
class UserPersonalModel(models.Model):
    firstname = models.CharField(max_length=100)
    lastname = models.CharField(max_length=100)
    age = models.IntegerField()
    address = models.TextField(null=True, blank=True)
    phone = models.IntegerField()
    city = models.CharField(max_length=100)
    state = models.CharField(max_length=100)
    country = models.CharField(max_length=100)
```

```
def __str__(self):
    return self.firstname, self.lastname,
self.age,self.address,self.phone,self.city,self.state,self.country
```

Urls.py

```
from django.urls import path
from . import views
urlpatterns =[
    path('', views.Landing_1, name='Landing_1'),
    path('Register_2/', views.Register_2, name='Register_2'),
    path('Login_3/', views.Login_3, name='Login_3'),
    path('Home_4/', views.Home_4, name='Home_4'),
    path('Teamates_5/', views.Teamates_5, name='Teamates_5'),
    path('Domain_Result_6/', views.Domain_Result_6,
name='Domain_Result_6'),
    path('Problem_Statement_7/', views.Problem_Statement_7,
name='Problem_Statement_7'),
    path('Per_Info_8/', views.Per_Info_8, name='Per_Info_8'),
    path('Deploy_9/', views.Deploy_9, name='Deploy_9'),
    path('Per_Database_10/', views.Per_Database_10, name='Per_Database_10'),
    path('Logout/', views.Logout, name='Logout'),
]
```

Views.py

```
from django.shortcuts import render, redirect
from . models import UserPersonalModel
from . forms import UserPersonalForm, UserRegisterForm
from django.contrib.auth import authenticate, login,logout
from django.contrib import messages
import numpy as np
import joblib
def Landing_1(request):
    return render(request, '1_Landing.html')
def Register_2(request):
    form = UserRegisterForm()
    if request.method =='POST':
        form = UserRegisterForm(request.POST)
        print(form)
        if form.is_valid():
            form.save()
            user = form.cleaned_data.get('username')
            messages.success(request, 'Account was successfully created. ' + user)
            return redirect('Login_3')
    context = {'form':form}
    return render(request, '2_Register.html', context)
```

```
def Login_3(request):
    if request.method =='POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('Home_4')
        else:
            messages.info(request, 'Username OR Password incorrect')
    context = {}
    return render(request,'3_Login.html', context)
def Home_4(request):
    return render(request, '4_Home.html')
def Teamates_5(request):
    return render(request,'5_Teamates.html')
def Domain_Result_6(request):
    return render(request,'6_Domain_Result.html')
def Problem_Statement_7(request):
    return render(request,'7_Problem_Statement.html')
```

```
def Per_Info_8(request):
    if request.method == 'POST':
        fieldss = ['firstname','lastname','age','address','phone','city','state','country']
        form = UserPersonalForm(request.POST)
        if form.is_valid():
            print('Saving data in Form')
            form.save()
        return render(request, '4_Home.html', {'form':form})
    else:
        print('Else working')
        form = UserPersonalForm(request.POST)
        return render(request, '8_Per_Info.html', {'form':form})


Model = joblib.load('D:\INTRUSION.pkl')


def Deploy_9(request):
    if request.method == "POST":
        int_features = [x for x in request.POST.values()]
        int_features = int_features[1:]
        print(int_features)
        final_features = [np.array(int_features, dtype=object)]
        print(final_features)
```

```
prediction = Model.predict(final_features)

print(prediction)

output = prediction[0]

print(f'output{output}')

if output == 0:

    A = "THE DENIAL OF SERVICE (DOS) ATTACK MIGHT BE OCCUR IN THIS CONDITION"

    B = "PREVENTION: Preventing a Denial of Service (DoS) attack involves implementing robust network security measures, such as deploying firewalls and intrusion detection/prevention systems. Additionally, regularly updating and patching software vulnerabilities can thwart potential attack vectors. Employing rate limiting and traffic filtering mechanisms helps mitigate the impact of sudden, excessive requests, enhancing overall system resilience against DoS threats."

    return render(request, '9_Deploy.html', {"prediction_text":A, "prediction_text1":B})

elif output == 1:

    A = "THE NONE OF ATTACK MIGHT BE OCCUR IN THIS CONDITION. THIS IS NORMAL"

    B = "PREVENTION: THERE ARE NO NEED PREVENTIONS. "

    return render(request, '9_Deploy.html', {"prediction_text":A, "prediction_text1":B})

elif output == 2:

    A = "THE PROBE ATTACK MIGHT BE OCCUR IN THIS CONDITION"
```

B = "PREVENTION: Preventing probe attacks involves implementing strong network security practices. Regularly monitoring and analyzing network traffic for anomalous patterns can help detect probes early. Additionally, deploying intrusion detection systems and keeping software and systems updated with the latest security patches further fortifies defenses against potential probing activities."

return render(request, '9_Deploy.html', {"prediction_text":A, "prediction_text1":B})

elif output == 3:

A = "THE REMOTE TO LOCAL (R2L) ATTACK MIGHT BE OCCUR IN THIS CONDITION"

B = "PREVENTION: Preventing Remote-to-Local (R2L) attacks requires securing remote access points. Employing strong authentication mechanisms, such as multi-factor authentication, enhances the security of remote connections. Regularly auditing and monitoring access logs for unusual activities helps identify and mitigate potential R2L threats promptly."

return render(request, '9_Deploy.html', {"prediction_text": A, "prediction_text1": B})

elif output == 4:

A = "THE USER-TO-ROOT (U2R) ATTACK MIGHT BE OCCUR IN THIS CONDITION"

B = "PREVENTION: Mitigating User-to-Root (U2R) attacks involves implementing robust access controls. Employ the principle of least privilege to restrict user permissions, minimizing the impact of potential exploits. Regularly update and patch system vulnerabilities to address known security weaknesses, reducing the likelihood of successful U2R attacks. Additionally, continuous

monitoring of user activities and behavior can aid in the early detection of suspicious actions, enhancing overall system security."

```
            return render(request, '9_Deploy.html', {"prediction_text": A, "prediction_text1": B})
    else:
        return render(request, '9_Deploy.html')
def Per_Database_10(request):
    models = UserPersonalModel.objects.all()
    return render(request, '10_Per_Database.html', {'models':models})
def Logout(request):
    logout(request)
    return redirect('Register_2')
```

A.3 SCREENSHOTS

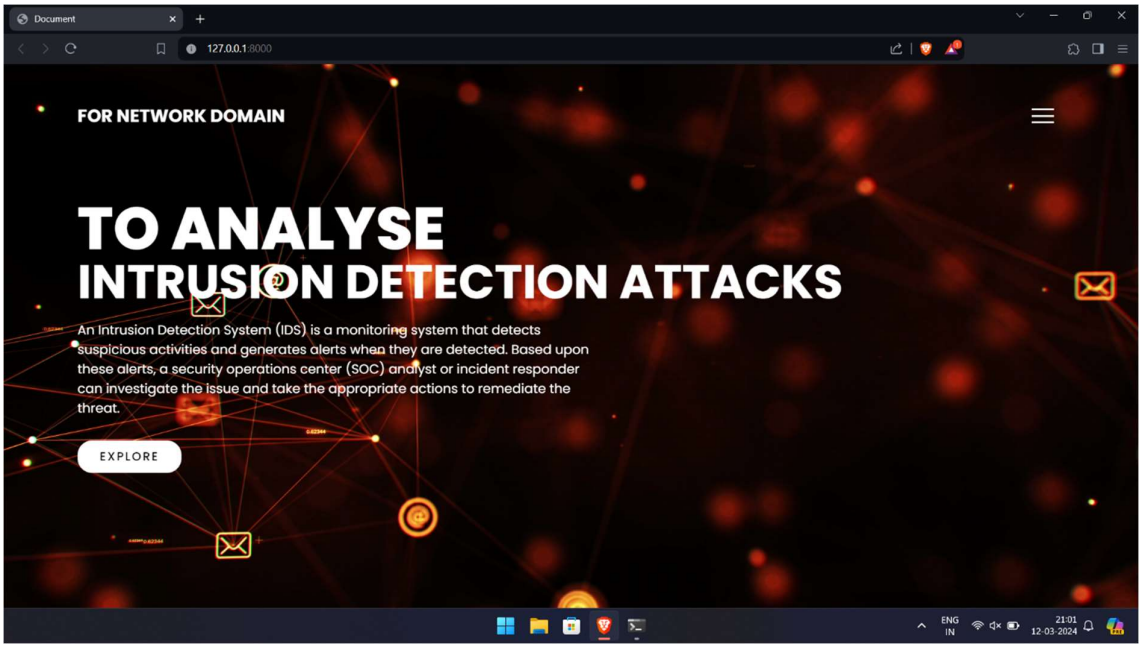


Figure A3.1 Homepage

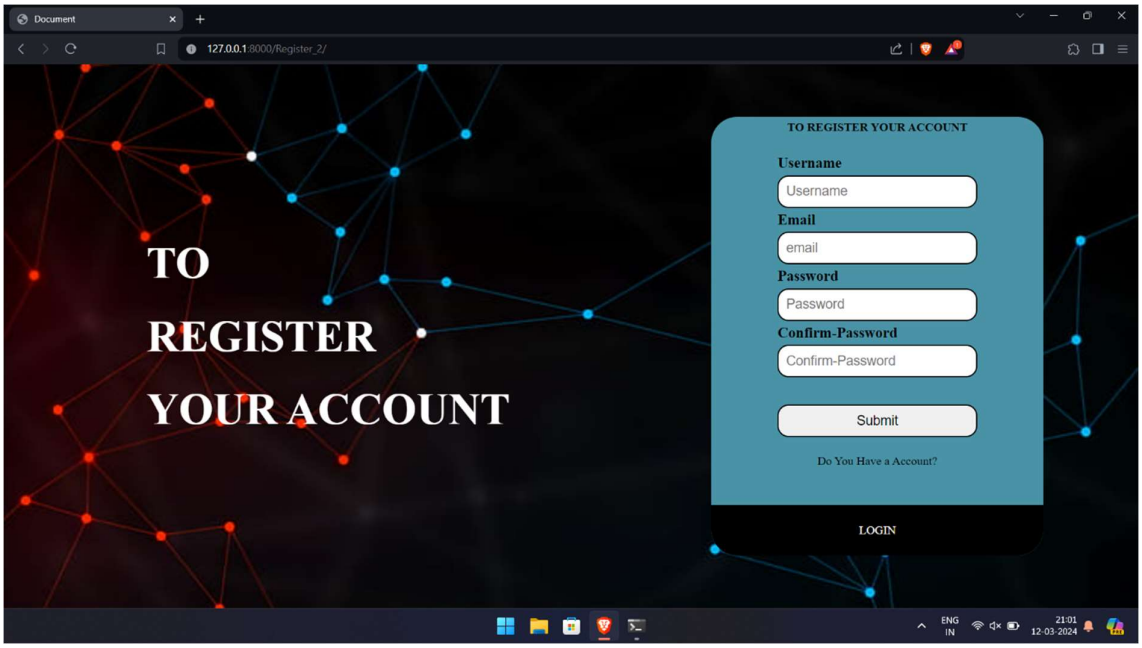


Figure A3.2 User Register Page

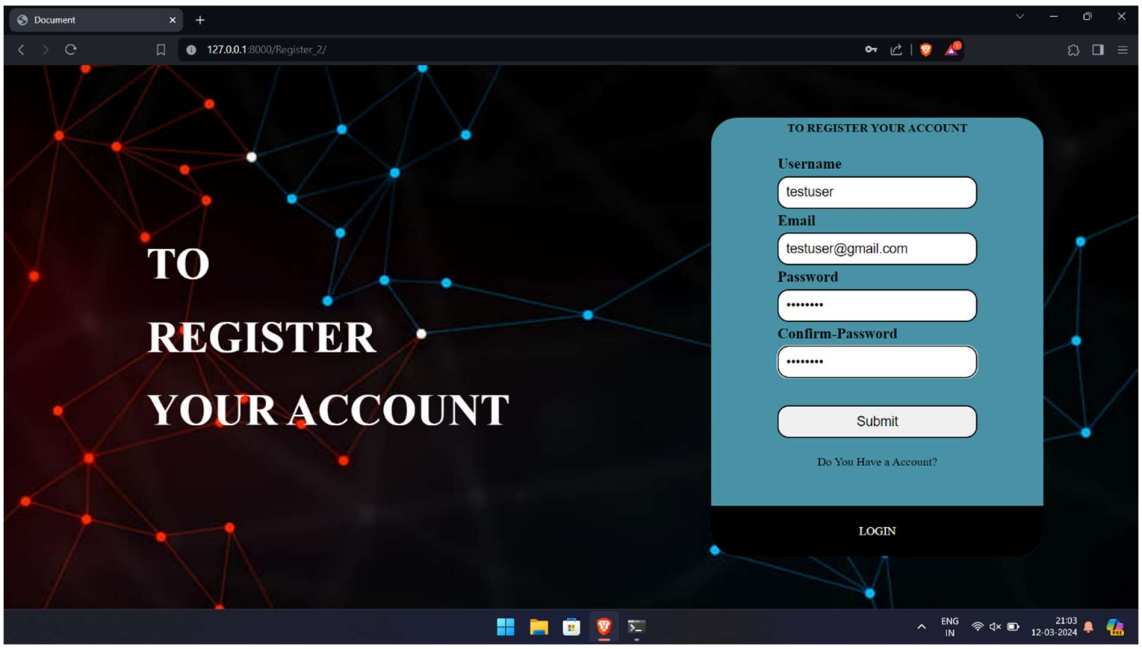


Figure A3.3 User Registration

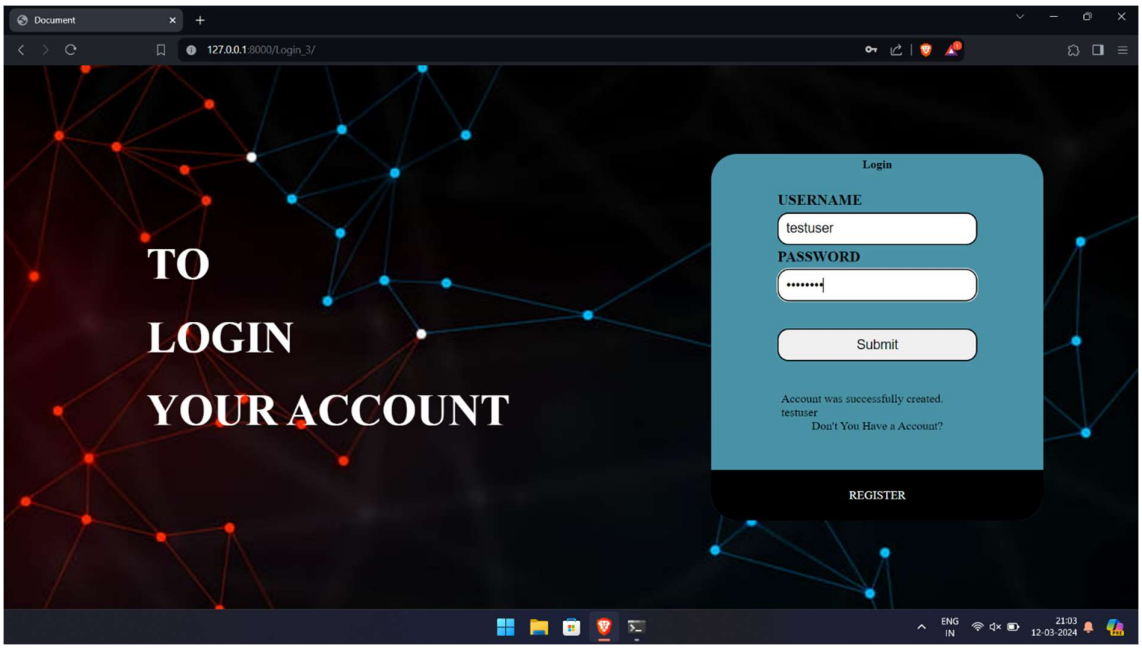


Figure A3.4 Login Page

Figure A3.5 Introduction Page

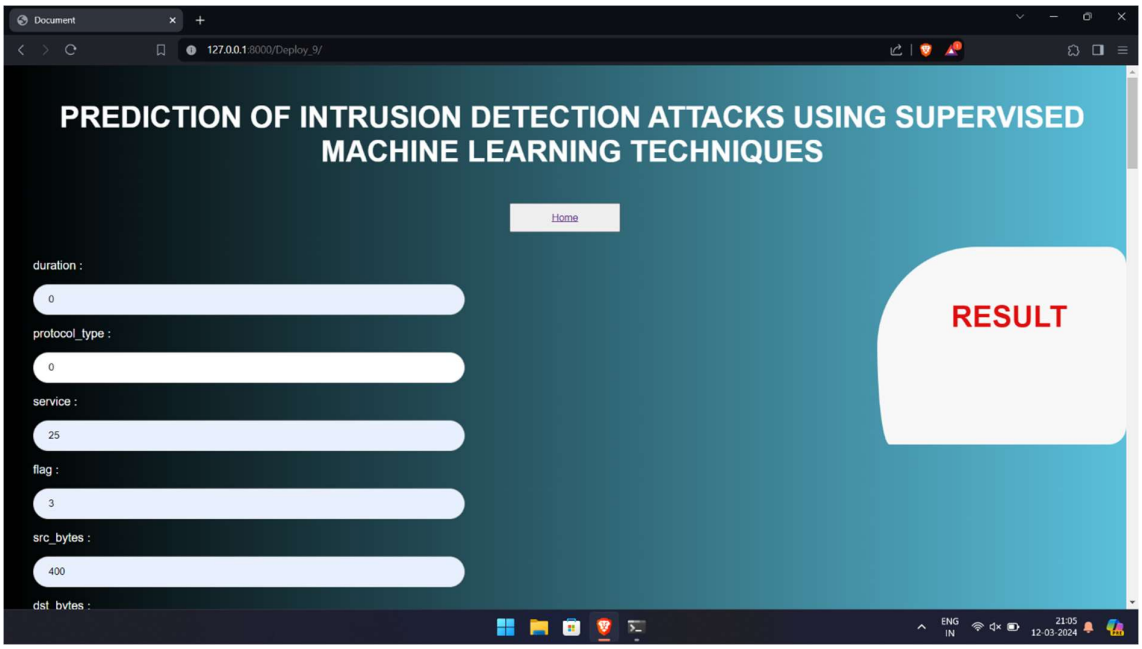


Figure A3.6 Prediction Page 1

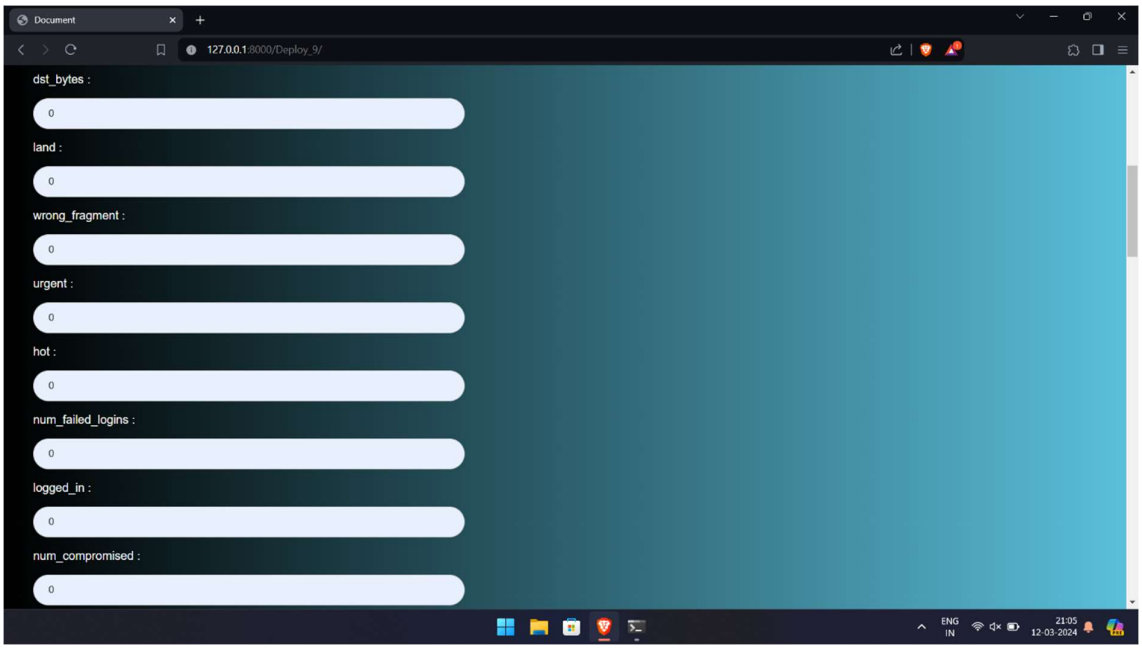


Figure A3.7 Prediction Page 2

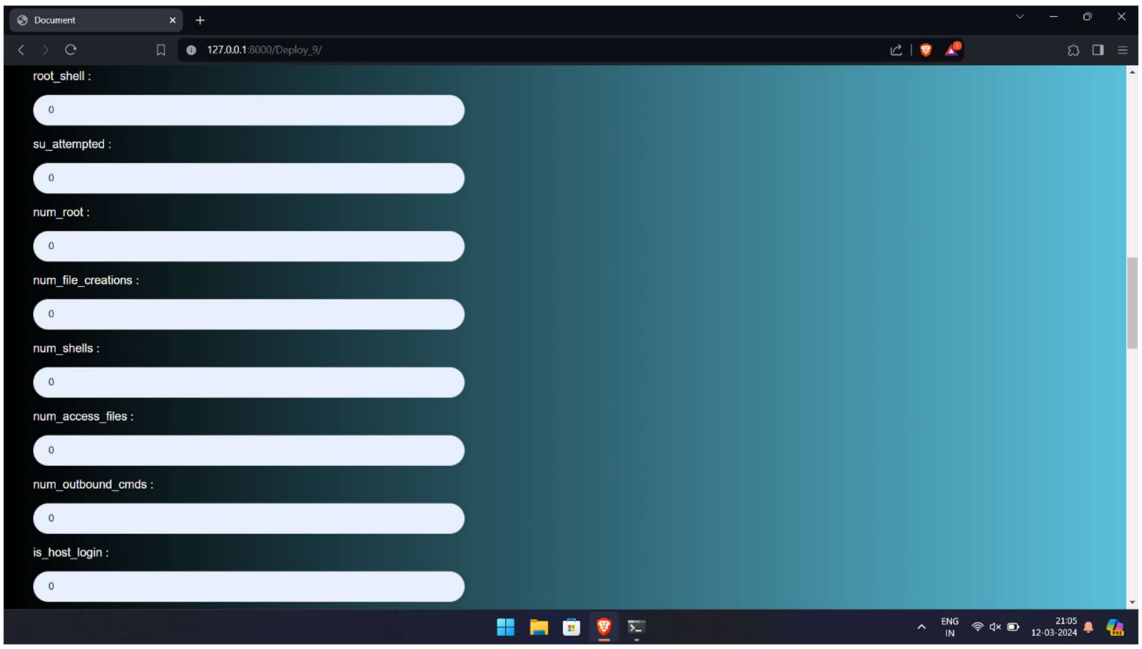


Figure A3.8 Prediction Page 3

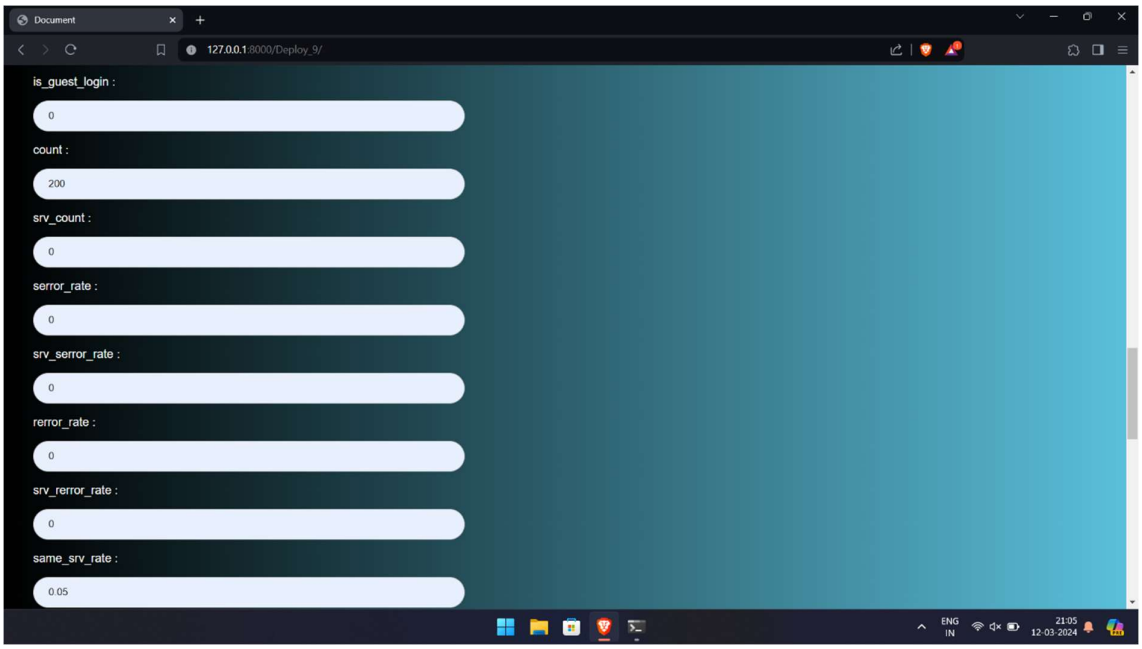


Figure A3.9 Prediction Page 4

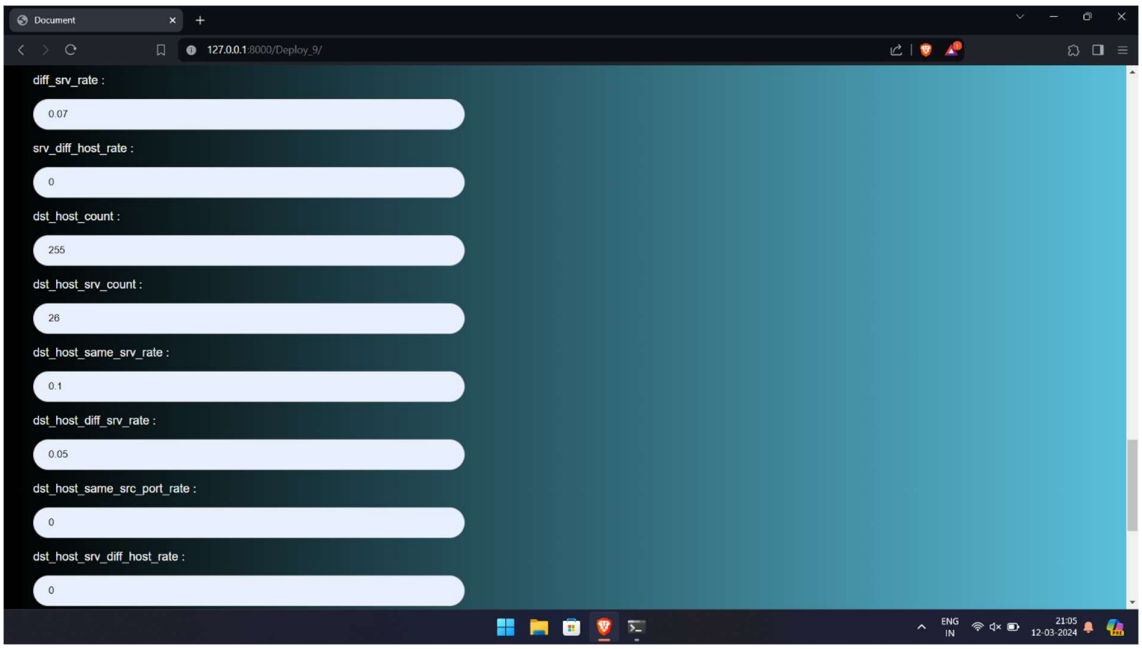


Figure A3.10 Prediction Page 5

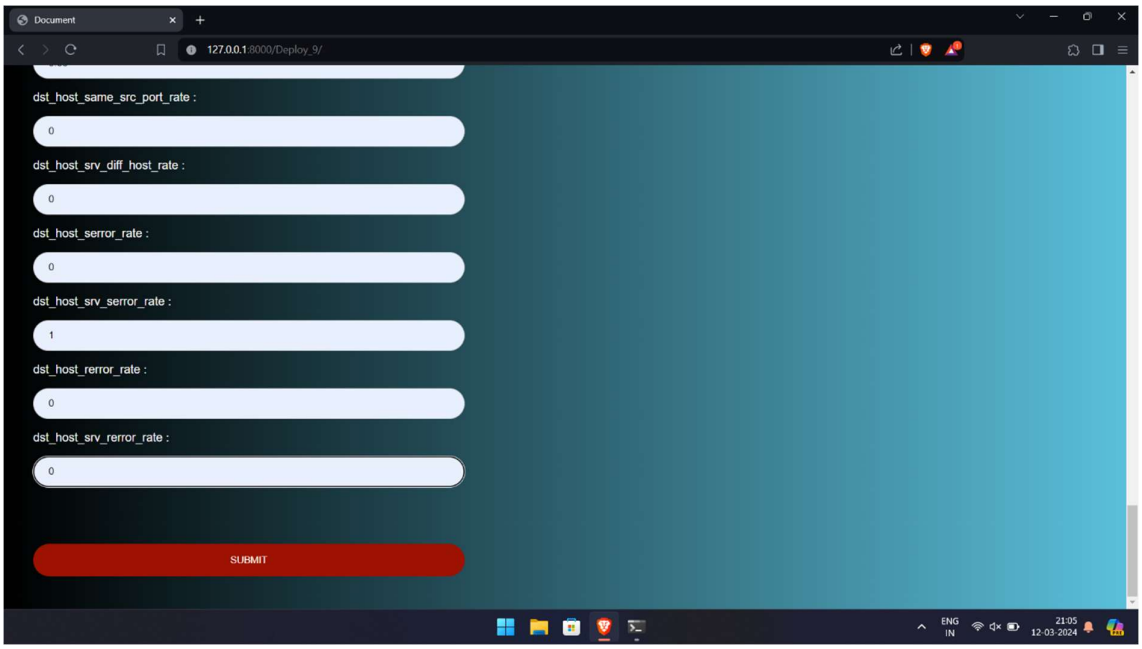


Figure A3.11 Prediction Page 6

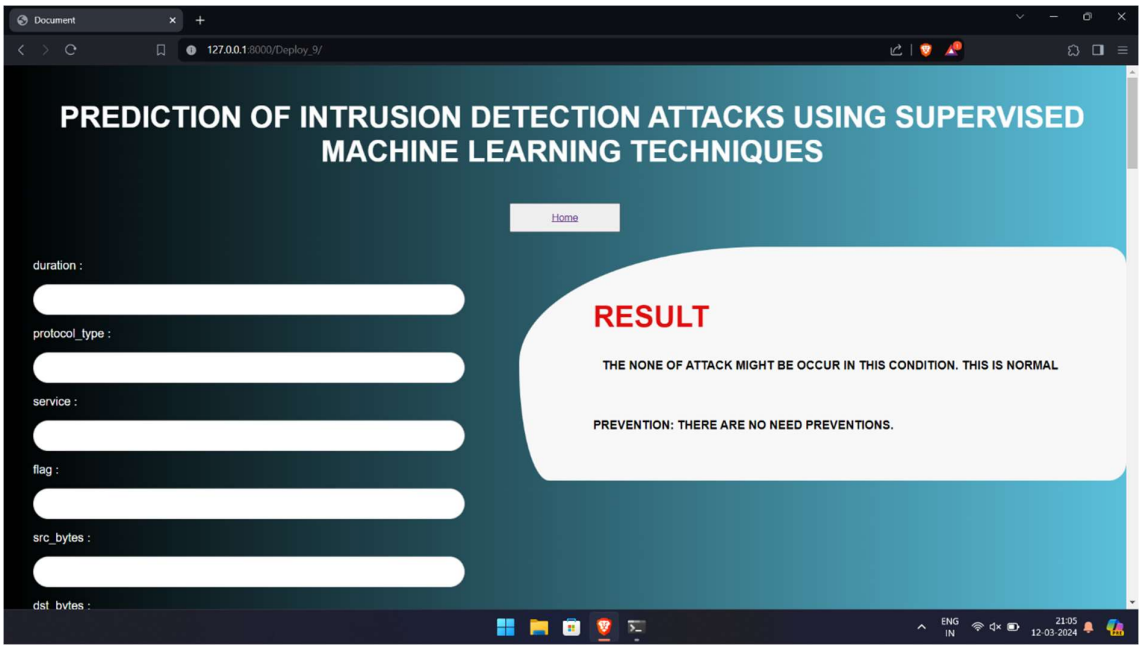


Figure A3.12 Prediction Result – No Attack

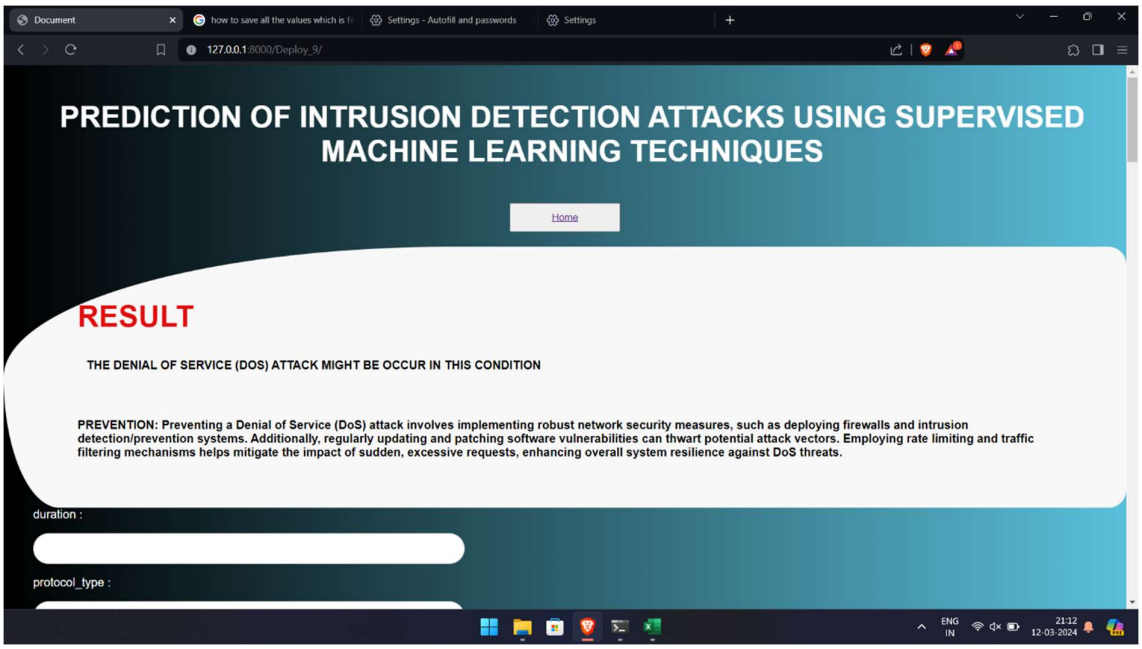


Figure A3.13 Prediction Result - DOS

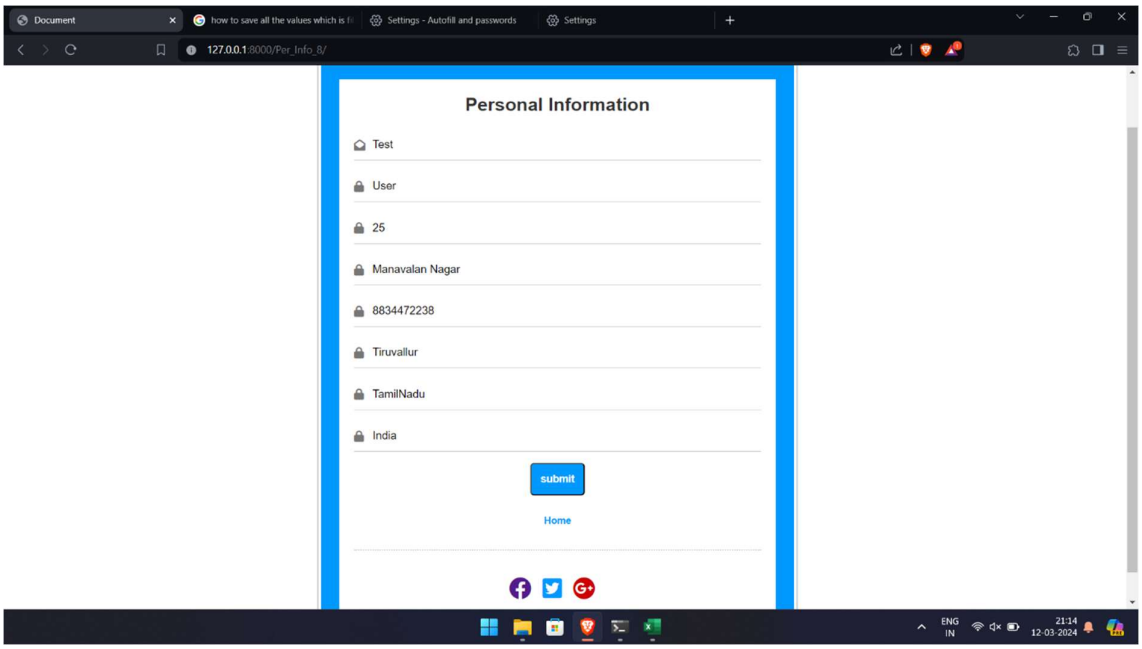


Figure A3.14 Personal Info Form

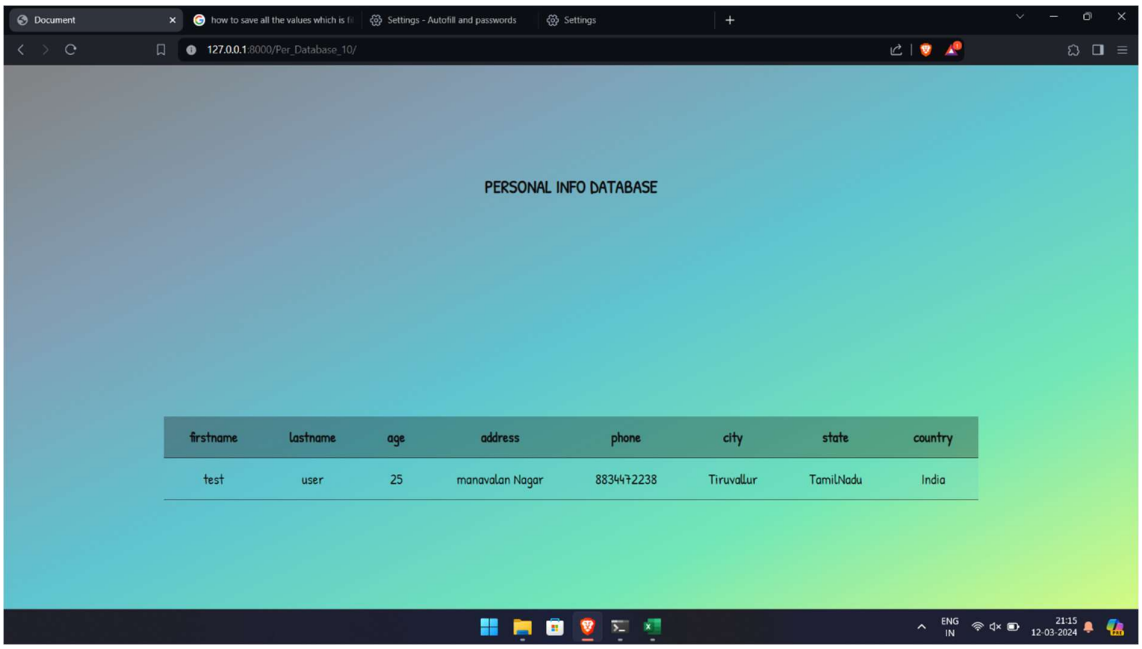


Figure A3.15 Personal Info View

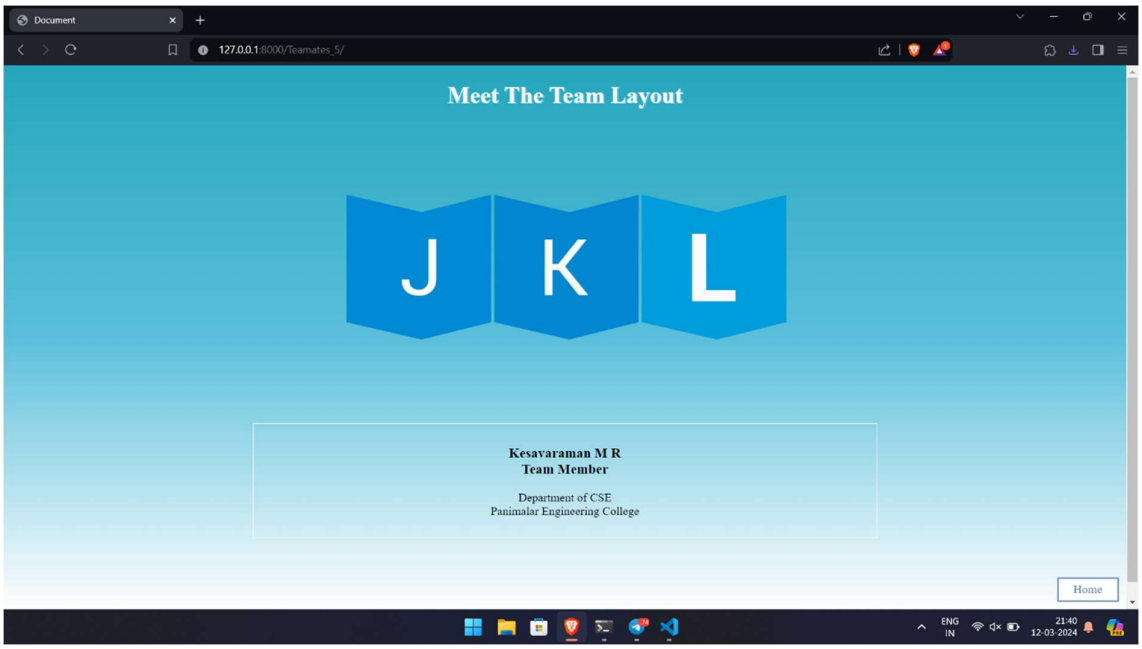


Figure A3.16 Contributors View

A.4 PLAGIRSM REPORT

Network Intrusion System using AI and ML

Mr SasiKumar
Professor
Computer science dept
Panimalar Engineering college
Chennai, India

Lingesh S
Computer Science Dept
Panimalar Engineering College
Chennai,India

Jenish Rojin S
Computer Science dept.
Panimalar Engineering College
Chennai,India

Kesavaraman
Computer Science Dept
Panimalar Engineering College
Chennai, India

Abstract -Network Intrusion Detection Systems (NIDS) are essential tools for safeguarding organizational assets against a wide range of cyber threats in today's hyperconnected digital ecosystem. But the increasing complexity and variety of hostile activity presents significant obstacles for conventional rule-based NIDS, calling for a paradigm change in favor of AI and ML-driven strategies. To strengthen the effectiveness and versatility of NIDS, this study undertakes a thorough investigation into the incorporation of AI and ML approaches.

After a thorough analysis of traditional NIDS constraints, we delve into the complex inner workings of AI and ML algorithms and carefully consider how they fit into the NIDS framework. The ability of supervised learning systems, such as neural networks and support vector machines (SVM), to identify patterns suggestive of harmful activity is investigated. The effectiveness of unsupervised learning techniques like anomaly detection and clustering in spotting new threats is being closely examined. Reinforcement learning methods are also assessed for their capacity to automatically modify the behavior of NIDS in response to environmental input.

We clarify the subtleties involved in integrating AI and ML into NIDS implementations through empirical investigations and case studies. This includes a thorough investigation of feature engineering approaches, model selection tactics, and data pretreatment procedures.

In summary, NIDS may become more flexible, adaptive, and robust guardians against ever evolving cyberthreats by utilizing AI and ML technologies. By means of methodical examination, tactical planning, and cooperative endeavors, we have the potential to usher in a novel phase of cybersecurity, ensuring the protection of vital infrastructure and digital resources from hostile breaches.

Keywords: Network Intrusion Detection Systems (NIDS), Artificial Intelligence (AI), Machine Learning (ML), Supervised Learning, Unsupervised Learning, Reinforcement Learning, Cybersecurity, Threat Detection, Anomaly Detection, Intrusion Detection, Data Preprocessing, Feature Engineering, Model Selection.

I. INTRODUCTION

First of all, It is critical to safeguard network systems from malicious attacks in the digitally connected world of today. The ever-changing nature of attacks and the growth of complex cyber threats mean that classic rule-based network intrusion detection systems (NIDS) frequently fail to detect and mitigate new security breaches. The incorporation of machine learning (ML) and artificial intelligence (AI) methods into network intrusion detection systems (NIDS) has surfaced as a viable remedy for this problem.

By allowing NIDS to automatically learn from and react to changing threats in real time, AI and ML algorithms have the potential to significantly improve their capabilities. Through the use of enormous volumes of network traffic data, these algorithms are able to identify minute patterns that point to malicious activity, increasing the accuracy of detection and decreasing false positives.

Labeled datasets can be used to train supervised learning algorithms, such Support Vector Machines (SVM), Random Forests, and Neural Networks, which categorize network traffic as benign or malicious based on predetermined criteria. By identifying abnormalities from typical network behavior, unsupervised learning techniques like clustering and anomaly detection allow NIDS to uncover risks that were previously undetected. Furthermore, by using reinforcement learning approaches, NIDS are able to dynamically modify their detection algorithms in response to environmental feedback.

In addition to improving threat detection capabilities, the integration of AI and ML into NIDS allows for proactive threat prevention and response. AI-driven NIDS are able to keep ahead of developing threats by continuously studying network traffic patterns and adapting to new attack vectors.

Packet collection, traffic analysis, signature matching, anomaly detection, warning generation, and response/mitigation are all part of an NIDS's operational concept. Network intrusion detection systems (NIDS) come in different flavors, such as host-based (HIDS) and network-based (NIDS) systems, along with inline and passive implementations. Performance effect, false positives and negatives, encryption restrictions, scalability, and integration with other security tools and platforms are among the issues and factors to be taken into account when installing NIDS. In general, by continuously monitoring and analyzing network traffic for possible security breaches, a network intrusion detection system (NIDS) is crucial for safeguarding network assets and data against cyber threats.

.

I. RELATED WORKS

Network intrusion detection system (NIDS) research spans a number of domains with the goal of enhancing these systems' capacity to detect, respond to, and withstand evolving cyberthreats. Here are a few thorough illustrations of related network intrusion detection system works:

Methodologies Based on Machine Learning:

In order to increase the detection accuracy of NIDS, a lot of research focuses on using machine learning techniques such ensemble methods, random forests, deep learning, and support vector machines. In order to determine which network traffic properties are most pertinent for efficient intrusion detection, research investigates feature selection techniques. In order to improve detection performance and resilience against evasion attacks, novel techniques mix different machine learning algorithms.

Finding Anomalies

Within NIDS, anomaly detection is still a major research topic. The goal of research is to create anomaly detection algorithms that are more advanced and can discriminate between malicious and benign network activity. In order to enable NIDS to identify risks that were previously unidentified, research examines the application of unsupervised, semi-supervised, and reinforcement learning techniques for anomaly detection.

Statistical modeling, clustering, and time-series analysis are among the techniques used to find anomalies in network activity.

1. Network intrusion detection and prevention system research spans a number of domains with the goal of defending computer networks against malevolent activity and illegal access. In order to identify suspicious network activity, machine learning techniques—which apply algorithms like decision trees, support vector machines, and neural networks—play a key role in this sector. By using sophisticated neural network topologies like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for pattern identification in network traffic data, deep learning techniques significantly improve intrusion detection capabilities. While signature-based detection relies on established patterns of known assaults for recognition, anomaly detection techniques concentrate on finding anomalies from typical network behavior.
2. Hybrid approaches reduce false positives and increase accuracy by combining different detection techniques. Because adversarial attacks present a challenge to intrusion detection systems, researchers have been forced to create strong defenses against strategies of evasion and poisoning. In order to identify potential threats, network traffic analysis entails analyzing the characteristics of malicious and normal traffic. Rapid action against intrusions is made possible by real-time detection and response mechanisms, and scalability concerns guarantee that detection systems can effectively manage high network traffic volumes. Specialized security measures are also being developed to safeguard connected industrial systems and the growing number of Internet of Things devices from cyber threats.

II. EXISTING SYSTEM

Network intrusion detection and prevention systems (NIDS/NIPS) that are currently in use represent a wide range of technologies designed to protect computer networks from an ever-more-advanced array of cyber threats. Prominent commercial products that are notable for their multidimensional approach to intrusion detection and prevention include Cisco Firepower, Palo Alto Networks NGFW, and Check Point IPS. These systems use a combination of machine learning algorithms and behavioral analysis to find abnormalities suggestive of possible threats, as well as signature-based detection, which identifies established patterns of malicious activity. Notably, in order to meet the intricate security requirements of contemporary businesses, these commercial systems offer a wide range of features like granular traffic control, real-time threat mitigation, and centralized management capabilities.

On the other hand, users can customize and have freedom using open-source alternatives like Zeek, Suricata, and Snort, which allows them to customize their intrusion detection systems to fit certain network situations. These solutions frequently examine network data for indications of malicious activity or unauthorized access using rule-based detection engines and protocol analysis. Furthermore, the cooperative character of open-source development guarantees continuous advancements in detection capabilities and stimulates community-driven innovation.

Simultaneously, with the rise of cloud computing, cloud-based intrusion detection and prevention services have also become more prevalent. Examples of these services are AWS GuardDuty, Microsoft Azure Security Center, and Google Cloud IDS. By offering automatic response mechanisms, continuous monitoring, and threat intelligence integration, these services expand the use of intrusion detection to cloud systems. Cloud-based solutions, which provide scalability and flexibility to enterprises adopting cloud technologies, may quickly identify and mitigate security threats across distributed network topologies by utilizing machine learning algorithms and sophisticated analytics. Additionally, companies with particular network infrastructures or security needs can receive customized solutions from intrusion detection and prevention systems that are made to order. With this integration, Snort will be able to respond more quickly to changing threats, increasing detection accuracy and decreasing false positives. Furthermore, numerous NIDS implementations have made use of deep learning techniques including recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

These tailored solutions can efficiently handle particular security concerns and improve the overall resilience of network defenses by combining unique detection algorithms, custom rule sets, and integration with pre-existing security frameworks. Network intrusion detection and prevention systems are essentially defined by variety, innovation, and a never-ending quest of excellence in thwarting changing cyberthreats.

Network intrusion detection and prevention systems (NIDS/NIPS) now in use offer a broad range of options, each with special features and capacities to handle the various and always changing cyberthreat environment. Comprehensive intrusion detection and prevention capabilities are integrated into network security systems by leading industry players such as Cisco, Palo Alto Networks, and Check Point through their commercial solutions. Advanced techniques like anomaly detection and signature-based detection, which compare known patterns of malicious behavior to network traffic and identify deviations from normal behavior that can point to an intrusion, are commonly used by these systems. Furthermore, these for-profit products frequently use machine learning algorithms to improve detection precision and automate threat response procedures.

Moreover, companies have the option to create bespoke intrusion detection and prevention systems that are suited to their particular network infrastructures and security requirements. Custom rule sets, proprietary detection algorithms, and interface with current security platforms and tools are a few examples of these customized solutions. Organizations can create intrusion detection and prevention systems that solve particular security issues and improve the overall resilience of their network defenses by utilizing open-source components, libraries, and frameworks. To summarize, the current landscape of network intrusion detection and prevention systems includes a wide variety of open-source, cloud-based, commercial, and custom-built options. Based on their deployment preferences, financial limits, and security requirements, organizations can select from a range of solutions.

III. PROPOSED SYSTEM

The suggested network intrusion detection and prevention system (NIDS/NIPS) includes a thorough method for defending computer networks from a variety of online dangers. The system seeks to detect and mitigate possible security breaches in real-time using a variety of intrusion detection engines, including signature-based and anomaly-based detection, in conjunction with advanced network traffic analysis, which includes packet inspection and feature extraction. Strong reporting and alerting systems are included to quickly inform security staff of any dangers that are identified, and automated response systems facilitate quick mitigation measures to lessen the effect of security incidents. Integrating with external threat intelligence feeds improves detection capabilities by giving access to the most recent data on attack vectors and new threats. Essentially, the system monitors and analyzes data packets moving over the network using complex network traffic analysis methods. Examining packet headers and payloads, seeing trends suggestive of possible dangers, and extracting pertinent information for additional study are all part of this research. Through real-time network traffic analysis, the system can identify questionable behaviors including malware dissemination, denial-of-service assaults, and port scanning. To improve detection capabilities, the system incorporates many intrusion detection engines in addition to network traffic analysis 8hese engines identify and mitigate different kinds of cyber threats using a variety of techniques, including anomaly-based detection, signature-based detection, and behavioral analysis.

Scalable architectures and parallel processing methods are used in the system's construction to manage massive data volumes effectively while preserving high detection accuracy and responsiveness. In addition, the system is performance-optimized to reduce latency and guarantee prompt detection and handling of security events. The suggested system has a centralized administration console and an intuitive user interface to make it easier to use and maintain. Access to dashboards, visualization tools, and configurable reports that offer insights into threat data analysis, network security posture, and security event monitoring is available to security workers. Security teams may set up security policies, keep an eye on network activities, and effectively handle security issues with the help of this unified management panel.

The suggested system incorporates essential aspects for compliance and regulatory support, guaranteeing that firms can exhibit adherence to industry norms and regulations. The system has built-in controls, audit trails, and logging features to uphold security regulations and guarantee the integrity and privacy of data. The solution helps businesses to safeguard confidential information, reduce risk and expense, and keep stakeholders and consumers confident by making compliance with laws like GDPR, HIPAA, PCI DSS, and others easier. In conclusion, the suggested network intrusion detection and prevention system provides a thorough and flexible response to network security issues. Through the integration of sophisticated detection methods, automated reaction systems, threat intelligence integration, scalability, performance enhancement, intuitive user interface, and compliance support functionalities, the system enables organizations to proactively counter cyberattacks, protect their vital assets, and uphold the confidentiality and integrity of their data.

By providing access to the latest information on attack vectors and emerging threats, integrating with external threat intelligence feeds enhances detection capabilities. While configuration, monitoring, and administration are made simpler by a user-friendly interface and centralized management panel, scalability and performance improvements ensure that the system can efficiently handle the increasing volume and complexity of network traffic. Thanks to compliance and regulatory support capabilities, which ensure adherence to industry standards and laws, organizations can demonstrate compliance with data protection and privacy requirements. All things considered, the recommended system offers an all-encompassing and adaptable method of network security, empowering businesses to actively defend against evolving cyberthreats and protect their essential assets.

The suggested network intrusion detection and prevention system (NIDS/NIPS) includes a thorough method for defending computer networks from a variety of online dangers. The system seeks to detect and mitigate possible security breaches in real-time using a variety of intrusion

detection engines, including signature-based and anomaly-based detection, in conjunction with advanced network traffic analysis, which includes packet inspection and feature extraction. Strong reporting and alerting systems are included to quickly notify security staff of any dangers found, and automatic reaction systems facilitate quick mitigation measures to lessen the effect of security incidents.

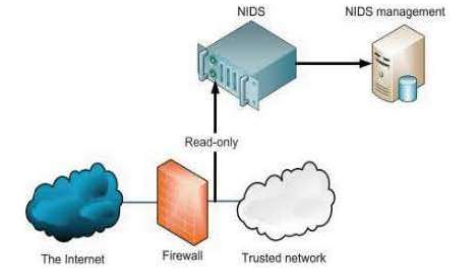
IV. SYSTEM ARCHITECTURE

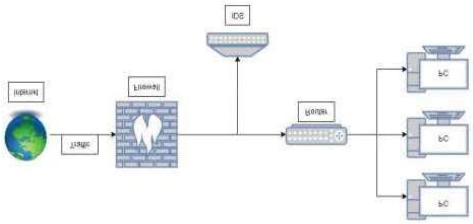
In order to efficiently detect and respond to possible security breaches, the design of the network intrusion detection system (NIDS) employing AI and ML consists of multiple interconnected components. Initially, the system gathers unprocessed network traffic information from switches, routers, and network taps, among other sources. Preprocessing procedures are then applied to this data, which is frequently in the form of packets or logs. Cleaning up the data, dealing with missing values, and formatting it so that it can be analyzed are all part of preprocessing.

After preprocessing, the system performs feature engineering, an important procedure that involves extracting pertinent characteristics from the data. Packet headers, payload properties, timestamps, and other network-specific information are examples of these features. More sophisticated feature engineering methods can be used to improve the system's capacity to identify patterns suggestive of possible intrusions. The system goes on to train machine learning models using the prepared data. These models are trained using historical data that includes examples of different kinds of intrusions as well as typical network behavior. Depending on the task's complexity, supervised learning algorithms like Random Forests or Support Vector Machines, unsupervised techniques like clustering or autoencoders, and deep learning strategies like convolutional neural networks (CNNs) or recurrent neural networks (RNNs) may be used.

Machine learning models are used to monitor network traffic in real time after they have been trained. The models scan the data as it moves through the network for anomalies or indications of questionable behavior. The system sends out alerts or notifications when it finds possible intrusions, which prompts security staff to look into the matter more and take the necessary action. The feedback loop is an essential component of the design that makes sure the ML models continue to work over time. This entails regularly updating and improving the models in light of fresh information and new dangers. Retraining the models on a regular basis and incorporating security analyst feedback aid in maintaining high detection accuracy while adapting to changing attack techniques. Mitigation and incident response are not possible without integration with current security operations workflows and technologies.

Performance and scalability are important factors to take into account, particularly in large-scale corporate settings. In order to effectively manage growing amounts of network traffic, the architecture should take processing power, latency, and throughput requirements into account. Finally, strong security measures are put in place to guard the NIDS infrastructure and guarantee the availability, confidentiality, and integrity of critical data. This covers steps like encryption, access limits, and routine security audits to find and fix such weaknesses. In accordance with laws and privacy standards, sensitive information handling practices should also use privacy-preserving measures.

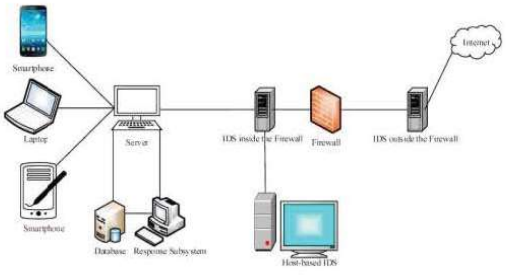




V. METHODOLOGY

1.Data Collection and Preparation:
Collect unprocessed network traffic information from switches, routers, network taps, and other sources. The gathered data should be preprocessed by filtering, cleaning, and formatting it appropriately for analysis. This could entail standardizing data formats, resolving missing numbers, and eliminating noise. To make model training and evaluation easier, divide the preprocessed data into training, validation, and testing sets. Make sure that the data gathering procedure abides by security guidelines and privacy laws, protecting the integrity and confidentiality of sensitive data.

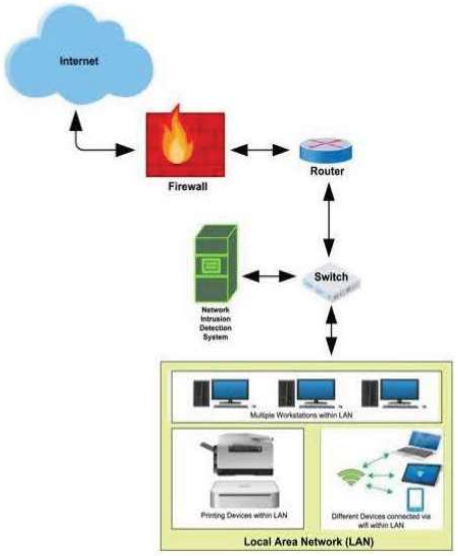
2. Feature Engineering:
From the preprocessed network traffic data, extract pertinent features to illustrate various facets of network behavior and attributes. Think about more sophisticated features that come from domain expertise or feature selection strategies, in addition to more fundamental features like packet headers, source/destination IP addresses, port numbers, and protocol kinds. Use methods like frequency analysis, statistical analysis, and time-series analysis to find meaningful features that record patterns suggestive of abnormalities or invasions. Using domain knowledge and trial and error, validate the chosen features to make sure they can distinguish between legitimate and malicious network activity.



3. Model Selection and Training:
Select the best machine learning algorithms depending on the type of problem, the properties of the data, and the amount of computing power needed. Supervised learning techniques like Random Forests and Support Vector Machines, as well as deep learning architectures like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are often employed algorithms for non-image detection systems (NIDS). Utilizing the labeled training data, optimize the chosen models to learn typical network behavior patterns and identify deviations that point to intrusions or abnormalities. In order to reduce false positives and increase detection accuracy, experiment with various feature combinations, model topologies, and hyperparameters. Make that the trained models fulfill predetermined performance targets and perform effectively when applied to new data by evaluating them using performance metrics including accuracy, precision, recall, F1-score, and ROC curves.

4. Deployment and continuous Improvement:
Integrate the trained models into the current network architecture and security operations procedures to monitor network traffic in real-time. Establish systems that, in the event that the deployed models identify potentially intrusive or suspicious activity, will generate alerts or notifications, allowing for timely incident reaction and mitigation. Create a feedback loop to gather information on false positives, identified intrusions, and model performance indicators. Then, use this information to iteratively enhance the models. Maintain constant observation over the implemented NIDS, making necessary updates to the models and feature set to accommodate new attack methods, network behavior shifts, and growing threats.

By employing this methodology, businesses may create AI and ML-based network intrusion detection systems that are efficient at identifying and thwarting a variety of real-time cyberthreats.



VI. CONCLUSION

In conclusion, there are major improvements in detecting and reducing cybersecurity threats when artificial intelligence (AI) and machine learning (ML) approaches are used with network intrusion detection systems (NIDS). Through the use of advanced algorithms to examine network traffic data, entities can improve their capacity to identify unusual activity and possible breaches in real time. By utilizing AI and ML, network intrusion detection systems (NIDS) can adjust dynamically to changing cyberthreats, continuously learning from fresh data, and gradually increasing detection accuracy. NIDS is capable of limiting false positives while identifying patterns suggestive of harmful activity through efficient feature engineering, model selection, and training. Furthermore, by enabling quick incident response and delivering timely notifications, the use of NIDS with AI and ML capabilities improves an organization's overall cybersecurity posture.

Recognizing that adversarial assaults, data privacy problems, and model interpretability issues are real threats to AI and ML-based NIDS is crucial. It will take continued investigation, cooperation, and financial support for strong cybersecurity plans to overcome these obstacles. All things considered, the incorporation of AI and ML technologies into network intrusion detection systems is a major advancement in cybersecurity defenses, allowing enterprises to successfully minimize possible risks and proactively guard against a variety of cyberthreats. In summary, a major development in cybersecurity is the incorporation of machine learning (ML) and artificial intelligence (AI) techniques into network intrusion detection systems (NIDS). Through the quick analysis of massive amounts of network traffic data and the identification of intricate patterns suggestive of malicious behavior, these systems provide improved threat detection capabilities. AI and ML-based NIDS can grow to identify new threats and advanced attack methods via ongoing learning and adaptation, increasing the overall resilience of cybersecurity defenses. Because of their real-time monitoring capabilities, businesses can react to security problems quickly, lessening the effect of breaches and decreasing the possibility of data loss or system compromise.

Furthermore, operational efficiency is improved by the smooth integration of AI- and ML-based NIDS with current security operations workflows, freeing up security personnel to concentrate on strategic projects and threat hunting. Notwithstanding, it is imperative to tackle obstacles like hostile cyberattacks, data privacy issues, and interpretability problems using models to guarantee the efficiency and dependability of these defense mechanisms against dynamic cyberthreats. By means of continuous investigation, cooperation, and allocation of resources towards resilient cybersecurity tactics, establishments can fully leverage the capabilities of AI and ML-driven NIDS to reinforce their resilience against cyber assaults. Furthermore, operational efficiency is improved by the smooth integration of ML- and AI-based NIDS with current security operations procedures. These systems free up security professionals to concentrate on strategic duties like vulnerability management and threat hunting by automating repetitive work and optimizing incident response procedures. By optimizing resources, cybersecurity operations may operate with maximum effectiveness and with less strain on security staff.

VII. FUTURE WORK

Future research projects utilizing artificial intelligence (AI) and machine learning (ML) in the context of network intrusion detection systems (NIDS) show great potential for enhancing cybersecurity capabilities. The development of strategies to improve model explainability and interpretability, which will promote trust and understanding among security analysts, and strengthening NIDS against adversarial assaults through improved robustness measures are important areas for investigation. Furthermore, it will be essential to incorporate privacy-preserving strategies into NIDS design in order to guarantee regulatory compliance and protect sensitive data. Threat mitigation procedures can be streamlined by automating incident response and security orchestration within NIDS, with an emphasis on integrating AI-driven decision-making capabilities. Approaches for multi-modal data fusion provide possibilities for thorough threat detection and correlation by integrating a variety of data sources, including system logs, network traffic, and threat intelligence. Moreover, adaptive strategies and ongoing learning mechanisms will allow NIDS to dynamically change and adapt to new threats. It will be critical to investigate scalable machine learning algorithms and distributed computing frameworks in order to address issues with efficiency and scalability before deploying in large-scale network environments. In order to evaluate the efficacy and operational feasibility of AI and ML-based NIDS and to promote cooperation between academics and industry stakeholders in order to advance the development of resilient cybersecurity, real-world deployment and assessment activities will be crucial.

Approaches for multi-modal data fusion provide possibilities for thorough threat detection and correlation by integrating a variety of data sources, including system logs, network traffic, and threat intelligence. Moreover, adaptive strategies and ongoing learning mechanisms will allow NIDS to dynamically change and adapt to new threats. It will be critical to investigate scalable machine learning algorithms and distributed computing frameworks in order to address issues with efficiency and scalability before deploying in large-scale network environments. In order to evaluate the efficacy and operational viability of AI- and ML-based NIDS, real-world deployment and evaluation projects will be crucial. This will encourage industry stakeholders and researchers to work together to advance the development of robust cybersecurity solutions that can protect against ever-evolving cyber threats.

VIII. REFERENCES

[1] Karasaridis, A., & Manolopoulos, Y. (2011). A review of network intrusion detection systems using machine learning. arXiv preprint arXiv:1106.4642.

[2] Lee, W., Stolfo, S. J., & Mok, K. W. (1999). A data mining framework for building intrusion detection models. In Proceedings of the 1999 IEEE symposium on security and privacy (pp. 120-132). IEEE.

[3] Zhang, J., Zulkernine, M., & Haque, A. (2005). A framework for constructing features and models for intrusion detection systems. ACM Transactions on Information and System Security (TISSEC), 8(3), 296-331.

[4] Scarfone, K., & Mell, P. (2007). Guide to intrusion detection and prevention systems (IDPS). National Institute of Standards and Technology, Special Publication, 800(94).

[5] Lippmann, R., Fried, D. J., Graf, I., Haines, J. W., Kendall, K., McClung, D., ... & Wilson, C. (2000). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In DARPA Information Survivability Conference and Exposition, 2000. Proceedings. (pp. 12-26). IEEE.

[6] García-Teodoro, P., Díaz-Verdejo, J. E., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & security, 28(1-2), 18-28.

[7] Axelsson, S. (2000). The base-rate fallacy and the difficulty of intrusion detection. ACM Transactions on Information and System Security (TISSEC), 3(3), 186-205.

[8] Moustafa, N., & Slay, J. (2016). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). Military Communications and Information Systems Conference (MilCIS), 1-6.

[9] Woodbridge, J., & Lloyd, D. (2019). An evaluation of machine learning techniques for intrusion detection on time series network data. Applied Sciences, 9(12), 2491.

[10] García, S., & Herrera, F. (2009). Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. Evolutionary computation, 17(3), 275-306.

These references cover various aspects of NIDS, including machine learning techniques, evaluation methodologies, datasets, and challenges in intrusion detection.

RE-2022-220676-plag-report

ORIGINALITY REPORT

7%	5%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.researchgate.net Internet Source	<1%
2	Submitted to The University of Memphis Student Paper	<1%
3	id.123dok.com Internet Source	<1%
4	www.giac.org Internet Source	<1%
5	Submitted to Taylor's Education Group Student Paper	<1%
6	www.mdpi.com Internet Source	<1%
7	www.sosyalarastirmalar.com Internet Source	<1%
8	Submitted to UI, Springfield Student Paper	<1%
9	Submitted to University of Cincinnati Student Paper	<1%

10	link.springer.com Internet Source	<1%
11	www.scribd.com Internet Source	<1%
12	finance.dailyherald.com Internet Source	<1%
13	www.cs.ru.ac.za Internet Source	<1%
14	thecannabisindustry.org Internet Source	<1%
15	www.dataversity.net Internet Source	<1%
16	eprints.bournemouth.ac.uk Internet Source	<1%
17	fastercapital.com Internet Source	<1%
18	ieeexplore.ieee.org Internet Source	<1%
19	ntnuopen.ntnu.no Internet Source	<1%
20	www.ijraset.com Internet Source	<1%
21	www.researchsquare.com Internet Source	<1%

22	www.verdict.co.uk Internet Source	<1%
23	Khater, Belal Sudqi Abed Saleh. "A Lightweight Host-Based Intrusion Detection System Using N-Gram and Perceptron Model for Internet of Things", University of Malaya (Malaysia), 2023 Publication	<1%
24	Abedzadeh, Najmeh. "Implementing a New Algorithm to Balance and Classify the Imbalanced Intrusion Detection System Datasets", The Catholic University of America, 2023 Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		