A Hybrid Intrusion Detection System Based on Feature Selection and Weighted Stacking Classifier

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

CS Venkata Adithya	**-**	**221910301015**
Reesu B D Venkata Krishna	**-**	**221910301047**
Yalaka Varsha	**-**	**221910301056**
Yerramilli Akhil	**-**	**221910301058**

Under the esteemed guidance of

Mrs. V. Rekha
Teaching Assistant

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GITAM
(Deemed to be University)

HYDERABAD

APRIL - 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GITAM SCHOOL OF TECHNOLOGY
GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled " **A Hybrid Intrusion Detection System Based on Feature Selection and Weighted Stacking Classifier**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any othercollege or University for the award of any degree or diploma.

Date:

Registration No(s)	Name(s)	Signature(s)
221910301015	**CS Venkata Adithya**	
221910301047	**Reesu B D Venkata Krishna**	
221910301056	**Yalaka Varsha**	
221910301058	**Yerramilli Akhil**	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project entitled "**A Hybrid Intrusion Detection System Based on Feature Selection and Weighted Stacking Classifier"** is a bonafide record of work carried out by **CS Venkata Adithya (221910301015), Reesu B.D Venkata Krishna (221910301047), Yalaka Varsha (221910301056), Yerramilli Akhil(221910301058)** students submitted in partial fulfilment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	Project Coordinator	Head of the Department
Mrs. V. Rekha	**Dr. S. Aparna**	**Dr. K S Sudeep**
Teaching Assistant **Dept. of CSE**	**Assistant Professor** **Dept. of CSE**	**HOD** **Dept. of CSE**

ACKNOWLEDGEMENT

Our project report would not have been successful without the help of several people. We would like tothank the personalities who were part of our seminar in numerous ways, those who gave us outstandingsupport from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice-Chancellor, **Prof. D. Sambasiva Rao**, for providing the necessary infrastructure and resources for the accomplishment of our seminar. We are highly indebted to **Prof. N. Seetharamaiah**, Associate Director, School of Technology for his supportduring the tenure of the seminar.

We are very much obliged to our beloved **Dr. K S Sudeep**, Head of the Department of Computer Science & Engineering, for providing the opportunity to undertake this seminar and encouragement in the completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our guide, **Mrs. V. Rekha**, Teaching Assistant, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project report.

We are also thankful to all the Computer Science and Engineering department staff members who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement, and moral support directly or indirectly in our seminar work.

Sincerely,

CS Venkata Adithya	**221910301015**
Reesu B D Venkata Krishna	**221910301047**
Yalaka Varsha	**221910301056**
Yerramilli Akhil	**221910301058**

TABLE OF CONTENTS

ABSTRACT

The fast expansion of the Internet has increased the frequency of cyberattacks. System security protection now includes the use of intrusion detection systems (IDS). IDS is still having some difficulties enhancing the performance of its classification. First off, the performance and speed of the classification for IDS are challenged by the complexity of high-dimensional characteristics. Second, the base classifiers have a direct impact on the conventional Stacking algorithm's classification performance. With the rapid growth of the Internet, cyber-attacks are becoming more common. Intrusion detection systems (IDS) have become an essential component of system security. There are still some obstacles preventing IDS from improving its classification performance even further. For starters, the complexity of high-dimensional features puts the speed and performance of IDS classification to the test. Second, the performance of the traditional Stacking algorithm can be easily influenced by the base classifiers. In order to address both of the aforementioned challenges, we propose a hybrid intrusion detection system based on a CFS-DE feature selection algorithm and a weighted Stacking classification algorithm. We used the CFS-DE algorithm to find the best feature subset to reduce the dimension of the features. The results based on CSE-CIC-IDS2018 show that our proposed model has accuracy of 99.87%, precision of 99.88%, recall of 99.87% and F1-score of 99.88%.

Keywords:

Intrusion, Intrusion Detection System, Denial of service, User to Root attacks, Remote to User attacks, Local Area Network, Principal Component Analysis, Support Vector Machine, Random Forest, Decision Tree, KNN Algorithm, Logistic Regression, Alerts, False Positives, False Negatives

TABLE OF FIGURES

LIST OF TABLES

LIST OF SCREENS

1. INTRODUCTION

1.1. Introduction

In today's digital world, the security of computer networks has become a major concern for individuals, organizations, and governments. Cyber attacks are becoming more and more sophisticated, frequent, and devastating, resulting in significant financial and reputational losses. Traditional intrusion detection systems (IDS) are no longer sufficient to protect against modern cyber threats, which require a more advanced and intelligent approach.

One promising approach to enhance the security of computer networks is the use of Machine Learning (ML) techniques. ML algorithms can learn from past data and patterns to detect and classify unknown attacks that may not be detected by traditional IDS systems. However, no single ML classifier is perfect in detecting all types of attacks, and each classifier has its own strengths and weaknesses. To overcome this limitation, a hybrid intrusion detection system using multiple ML classifiers can be developed.

The hybrid intrusion detection system using ML classifiers combines the strengths of different ML classifiers to improve the detection rate of both known and unknown attacks. The system can also reduce false positives and provide a high level of accuracy in identifying malicious activities in real-time. This approach can enhance the security of computer networks by detecting and mitigating various types of cyber threats, including network anomalies, intrusion attempts, and malicious activities.

In this project, we will develop a hybrid intrusion detection system using ML classifiers to enhance the security of computer networks. The system will be implemented and tested on a real-world dataset to evaluate its effectiveness and performance. The results of this project will contribute to the field of network security by providing a reliable and effective approach to detect and mitigate various types of cyber threats.

Classification of Intrusion Detection System

IDS are classified into 5 types:

An Intrusion Detection System (IDS) is a security mechanism that monitors network traffic for suspicious activities or signs of malicious behavior. There are several types of IDS that are commonly used in modern networks. Here, we will discuss some of the most common types of IDS.

Host-based IDS (HIDS):

A host-based IDS is installed on a specific device, such as a server or workstation, and monitors the system logs and files for any suspicious activity. HIDS can detect various types of attacks, including unauthorized access attempts, file tampering, and privilege escalation. HIDS is useful in detecting attacks that originate from within the network or from compromised devices.

Network-based IDS (NIDS):

A network-based IDS monitors network traffic for signs of malicious activity. NIDS is typically installed on a network device, such as a firewall or router, and can detect attacks that originate from outside the network, such as denial-of-service (DoS) attacks and port scanning. NIDS is useful in identifying potential threats before they reach the targeted device.

Hybrid IDS:

A hybrid IDS combines both HIDS and NIDS to provide comprehensive coverage of both the network and the host devices. A hybrid IDS can detect attacks that originate from both inside and outside the network, and can provide a more comprehensive view of the network security.

Signature-based IDS:

A signature-based IDS works by comparing network traffic to a database of known attack signatures. If a match is found, the IDS will generate an alert. Signature-based IDS is effective in detecting known threats, but it may miss newer or more complex attacks that do not match any known signatures.

Anomaly-based IDS:

An anomaly-based IDS works by monitoring network traffic and looking for patterns of behavior that deviate from normal activity. If an abnormal pattern is detected, the IDS will generate an alert. Anomaly-based IDS is useful in detecting previously unknown threats, but it may also generate false positives if the normal traffic patterns are not accurately defined.

Heuristic-based IDS:

A heuristic-based IDS uses a set of rules and algorithms to identify potential threats. Heuristic-based IDS can detect both known and unknown threats, but it may also generate false positives if the rules are too strict or too vague.

1.2. Motivation for the Work

The motivation for developing a hybrid intrusion detection system using ML classifiers is the need to improve the accuracy and effectiveness of intrusion detection in modern network environments. Traditional rule-based systems can be effective in detecting known attacks, but they can struggle with new and sophisticated attacks that exploit vulnerabilities in network systems.

Machine learning-based classifiers have shown promise in improving detection accuracy, but they can also be prone to false positives, false negatives, particularly when the training data does not reflect the diversity of real-world attacks.

Hence, we need to develop a hybrid intrusion detection system that combines the strengths of traditional rule-based systems and machine learning-based classifiers to improve detection accuracy and reduce false alarms. The hybrid system should be able to analyze network traffic in real-time, classify the traffic into normal or anomalous behavior, and generate alerts for any suspicious activity.

Moreover, the system should be able to adapt to changing network environments and attack techniques by continuously learning and updating the classifiers. This adaptability is crucial to keep up with the evolving threat landscape and ensure the system remains effective over time.

The development of a hybrid intrusion detection system using ML classifiers can have significant benefits for network security. By improving detection accuracy and reducing false alarms, the system can help security analysts to focus their attention on genuine threats and respond to them in a timely and effective manner.
Overall, the motivation for developing a hybrid intrusion detection system using ML classifiers is to improve the effectiveness of intrusion detection in modern network environments, enhance network security, and protect critical assets from cyber-attacks.

1.3. Problem Statement

The problem statement for the hybrid intrusion detection system using ML classifiers is to design and develop a system that can effectively detect and mitigate network intrusions while minimizing false positives and false negatives. Traditional intrusion detection systems may not be sufficient in detecting the increasingly sophisticated and complex attacks used by modern hackers, and machine learning-based classifiers have shown promise in improving detection accuracy.

However, ML classifiers can be susceptible to overfitting, where the classifier performs well on the training data but poorly on new, unseen data. Moreover, different classifiers may perform well on different types of attacks, and a single classifier may not be sufficient to detect all types of intrusions.

Therefore, a hybrid intrusion detection system that combines multiple ML classifiers and traditional rule-based techniques is needed to improve detection accuracy and reduce false alarms. The system should be able to analyze network traffic in real-time, classify the traffic into normal or anomalous behavior, and generate alerts for any suspicious activity.

The system should also be able to adapt to changing network environments and attack techniques by continuously learning and updating the classifiers. Moreover, the system should have a user-friendly interface that allows security analysts to view and analyze the alerts generated by the system and take appropriate actions.

2. LITERATURE REVIEW

IDS (Intrusion Detection System) is a security mechanism designed to identify unauthorized access, misuse, and malicious activities in computer networks or systems. The history of IDS dates back to the early 1980s when Dorothy Denning, a computer science professor at Purdue University, proposed the concept of Intrusion Detection Systems.

One of the initial contributions to the field of IDS was the work done by James P. Anderson, a computer security expert at the National Computer Security Center. In 1980, he published a paper titled "Computer Security Threat Monitoring and Surveillance," which introduced the concept of intrusion detection. The paper discussed the importance of monitoring system logs to detect security threats and proposed the idea of using statistical analysis to identify unusual patterns of activity.

In the mid-1980s, the first commercial IDS, called "IDAS" (Intrusion Detection Expert System), was developed by SRI International. IDAS used rule-based expert systems to detect attacks and was mainly used to monitor government networks.

In the early 1990s, the (CERT) developed an IDS called "the Network Flight Recorder" (NFR). NFR was one of the first IDS to use network traffic analysis to detect attacks.

In 1998, the Snort IDS was created by Martin Roesch. Snort is an open-source IDS that uses signature-based detection and can be customized by adding new rules to detect specific attacks.

In the late 1990s and by the early 2000s, research focused on improving IDS accuracy and reducing false positives. One of the notable works was the development of anomaly-based detection, which uses machine learning and statistical analysis to identify unusual patterns of behavior.

The use of machine learning (ML) classifiers for intrusion detection has gained significant attention due to their potential to improve detection accuracy. However, ML classifiers can be susceptible to overfitting and may not perform well in detecting new and unknown attacks. In this literature review, we will explore recent research on hybrid intrusion detection systems that combine ML classifiers with traditional rule-based systems to improve detection accuracy and reduce false alarms.

One study by Zhang et al. (2020) proposed a hybrid intrusion detection system that combined deep learning-based classifiers with expert rules. The system used a deep learning model to extract features from network traffic, and expert rules were used to classify the traffic into normal or anomalous behavior. The results showed that the hybrid system outperformed traditional rule-based systems and ML-based classifiers alone in terms of detection accuracy and false alarm rate.

Another study by Alazab et al. (2018) proposed a hybrid intrusion detection system that combined decision tree classifiers with fuzzy logic. The system used decision tree classifiers to detect attacks and fuzzy logic to determine the severity of the detected attacks. The results showed that the hybrid system outperformed traditional rule-based systems and ML-based classifiers alone in terms of detection accuracy, false alarm rate, and severity ranking.

Moreover, a study by Shi et al. (2019) proposed a hybrid intrusion detection system that combined machine learning-based classifiers with clustering algorithms. The system used clustering algorithms to group similar network traffic into clusters, and machine learning-based classifiers were used to classify the clusters into normal or anomalous behavior. The results showed that the hybrid system outperformed traditional rule-based systems and ML-based classifiers alone in terms of detection accuracy and false alarm rate.

Finally, a study by Zhu et al. (2021) proposed a hybrid intrusion detection system that combined deep learning-based classifiers with ensemble learning techniques. The system used deep learning models to extract features from network traffic, and ensemble learning techniques were used to combine the outputs of multiple classifiers.

3. PROBLEM ANALYSIS

3.1. Existing System

One example of an existing Intrusion Detection System (IDS) that has limitations and room for improvement is the Signature-based IDS. Signature-based IDS relies on a database of pre-defined attack signatures to identify malicious traffic. However, this approach has limitations because it cannot detect unknown attacks or attacks that have been modified to evade detection.

To improve the Signature-based IDS, machine learning algorithms can be used to analyze network traffic in real-time and identify anomalous behavior that may indicate an attack. This approach, called Anomaly-based IDS, has the potential to detect previously unknown attacks and reduce false positives.

Another area for improvement in IDS is the use of Explainable Artificial Intelligence (XAI) techniques to improve transparency and interpretability of the detection results. This would allow security analysts to understand how the IDS works and make informed decisions about how to respond to detected threats.

Overall, while Signature-based IDS is useful, it has limitations and there is scope for improvisation by incorporating machine learning algorithms and XAI techniques to improve its accuracy and interpretability.

3.2. Challenges and Feasibility

Hybrid Intrusion Detection Systems (IDS) combine multiple detection techniques, including both signature-based and anomaly-based approaches, to improve the accuracy of intrusion detection Machine learning (ML) classifiers are a popular choice for anomaly-based detection in hybrid IDS.

1. **Data Availability**: One of the primary challenges of using ML classifiers in a hybrid IDS is the availability of labeled training data. ML classifiers require large amounts of labeled data to learn to identify different types of network traffic and distinguish between normal and malicious activities.
2. **Class Imbalance**: In practice, the number of normal traffic instances significantly outweighs the number of malicious instances in the dataset.
   This class imbalance can lead to bias in the classifier's performance, where it may struggle to identify malicious instances effectively.

3. **Feature Selection**: The success of an ML classifier depends on the quality of the features it uses to learn from. Selecting relevant features from the vast amount of network traffic data can be a challenging task. Additionally, the features selected need to be dynamic and adapt to the changing network environment.

4. **Model Complexity**: The complexity of ML classifiers can vary significantly based on the type of algorithm used. The more complex the model, the more resources required for training and inference. Thus, choosing the right algorithm is crucial to achieving a balance between accuracy and resource usage.

5. **False Positives and False Negatives**: it can significantly impact the effectiveness of an IDS. False positives occur when the system incorrectly identifies benign traffic as malicious, while false negatives occur when the system fails to identify actual malicious traffic. Achieving a balance between the two is crucial to minimize the impact on the network.

6. **Real-Time Processing**: Hybrid IDSs using ML classifiers require real-time processing of network traffic, which can be a challenging task, particularly for high-speed networks.

3.3. Proposed System

- I/We would like to propose a new Intrusion Detection System (IDS) that uses the CSE-CIC-IDS dataset and leverages machine learning algorithms to improve the accuracy of detecting network intrusions. The CSE-CIC-IDS dataset is a comprehensive dataset that contains various types of network traffic, including normal traffic and various types of attacks.

- To improve the accuracy of the IDS, the system will use a feature selection algorithm called Correlation-Based Feature Selection (CFS) to select the most relevant features from the dataset. The selected features will be used to train multiple classifiers, including Support Vector Machines (SVMs), Random Forests, and Gradient Boosting Machines (GBMs).

- The trained classifiers will be combined using a weighted stacking approach to achieve higher accuracy. The weighted stacking approach will use the weights assigned to each classifier based on its performance on the validation set. This will enable the system to give more weight to the best performing classifiers and improve the overall accuracy of the IDS.

- In addition, the proposed system will incorporate Deep Learning techniques like (CNNs) and Recurrent Networks like (RNNs), to analyze the network traffic at a deeper level, including analyzing packet payloads, to detect sophisticated attacks that are difficult to detect using traditional IDS approaches.

- The system will be evaluated using standard evaluation metrics, such as accuracy, precision, recall, AUC-ROC and F1-score on the CSE-CIC-IDS dataset. The performance of the proposed system will be compared to other state-of-the-art IDS systems to demonstrate its effectiveness.

- Overall, the proposed system aims to improve the accuracy of detecting network intrusions while reducing false positives. By incorporating CFS feature selection and weighted stacking algorithms along with neural networks, the proposed system has the potential to achieve higher accuracy in detecting network intrusions.

3.4. Software requirement Specification

3.4.1. Introduction

Software Requirement Specification (SRS) is a comprehensive document that outlines the intended behavior of the software, the features it will provide, and the constraints under which it must operate.

The SRS document is typically created at the beginning of a software development project, and it serves as a roadmap for the entire project team. It is used by developers, testers, and other stakeholders to ensure that the software meets the needs of the end-users. A well-written SRS document is critical to the success of a software project.

The use of this requirements for a software system that can detect network intrusions using machine learning classifiers. The system will combine both signature-based and anomaly-based detection techniques to provide an effective and efficient way of detecting network intrusions. The system will be designed to operate in a hybrid mode, combining both signature-based and anomaly-based detection techniques. The system will be developed using machine learning algorithms and will be designed to run on a variety of operating systems.

3.4.2. Functional Requirements

1. **Network Traffic Analysis**:The system shall be able to analyze the network traffic in real-time and identify patterns of network activity.

2. **Feature Extraction:** The system shall be able to extract features from the network traffic data, such as packet size, source, destination IP addresses, protocol type and payload.

3. **Classification**: The system shall be able to classify network traffic data as either normal or malicious.

4. **Training**: The system shall be able to train the machine learning classifiers on the network traffic data.

5. **Updating**: The system shall be able to update the machine learning classifiers with new data to improve their accuracy.

6. **Alerting**: The system shall generate alerts when a network intrusion is detected.

7. **Logging**: The system shall log all network traffic data and intrusion detection events.

8. **Visualization**: The system shall be able to provide visualization of the network traffic data and the detected intrusions.

3.4.3.Non-Functional Requirements

1. **Performance:** The system shall be able to process network traffic data in real-time without causing any noticeable delay.

2. **Scalability:** The system shall be designed to scale to handle large amounts of network traffic data.

3. **Reliability:** The system shall be designed to be highly reliable and fault tolerant.

4. **Security:** The system shall be designed to be secure and protect against attacks from external sources.

5. **Usability:** The system shall be easy to use and require minimal user interaction.

6. **Maintainability:** The system shall be easy to maintain and upgrade.

7. **Compatibility:** The system shall be designed to be in line with a variety of operating systems and network devices.

3.4.4. Conclusion

The software requirements specification has outlined the functional and non-functional requirements for a system that can detect network intrusions using machine learning classifiers. The system will be designed to operate in a hybrid mode, combining both signature-based and anomaly-based detection techniques, and will be developed using machine learning algorithms.

3.5. System Requirements of Proposed System

Hardware Requirements:

- **Processor**: Intel Core i5 or equivalent
- **RAM**: 8 GB or higher
- **Storage**: At least 100 GB of free disk space
- **Network Interface**: At least one Ethernet port

Software Requirements:

- **Operating System**: Windows
- **Python** 3.6 or higher
- **Machine Learning Libraries**: TensorFlow, Scikit-Learn, Keras, PyTorch, or similar
- **Database Management System**: MySQL, PostgreSQL, or similar
- **Web Framework**: Flask or Django

Additional hardware may be required depending on the size and complexity of the network being monitored. For example, larger networks may require multiple network interface cards or dedicated hardware appliances to handle the increased traffic load.

Moreover, the system may require specialized hardware for deep learning-based classifiers, such as GPUs or TPUs. However, this may not be necessary for smaller networks or if simpler ML algorithms are used.
In addition to the hardware and software requirements, the system should have sufficient network access to monitor the traffic and receive alerts.

4. DESIGN

4.1. System Methodology

System methodology refers to the structured and systematic approach to developing, analyzing, and designing complex systems. It involves a set of procedures, techniques, and tools that are used to identify and define the requirements, components, and interactions of a system. System methodology is important because it helps to ensure that complex systems are developed in a rigorous and disciplined way, with a clear understanding of their objectives, requirements, and constraints.

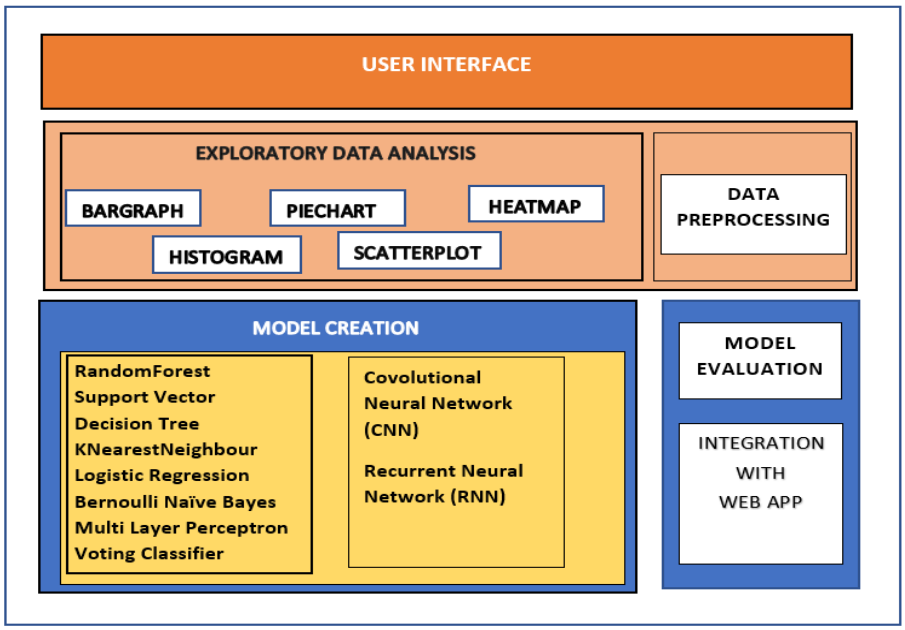


Figure 4.1.1 System Architecture

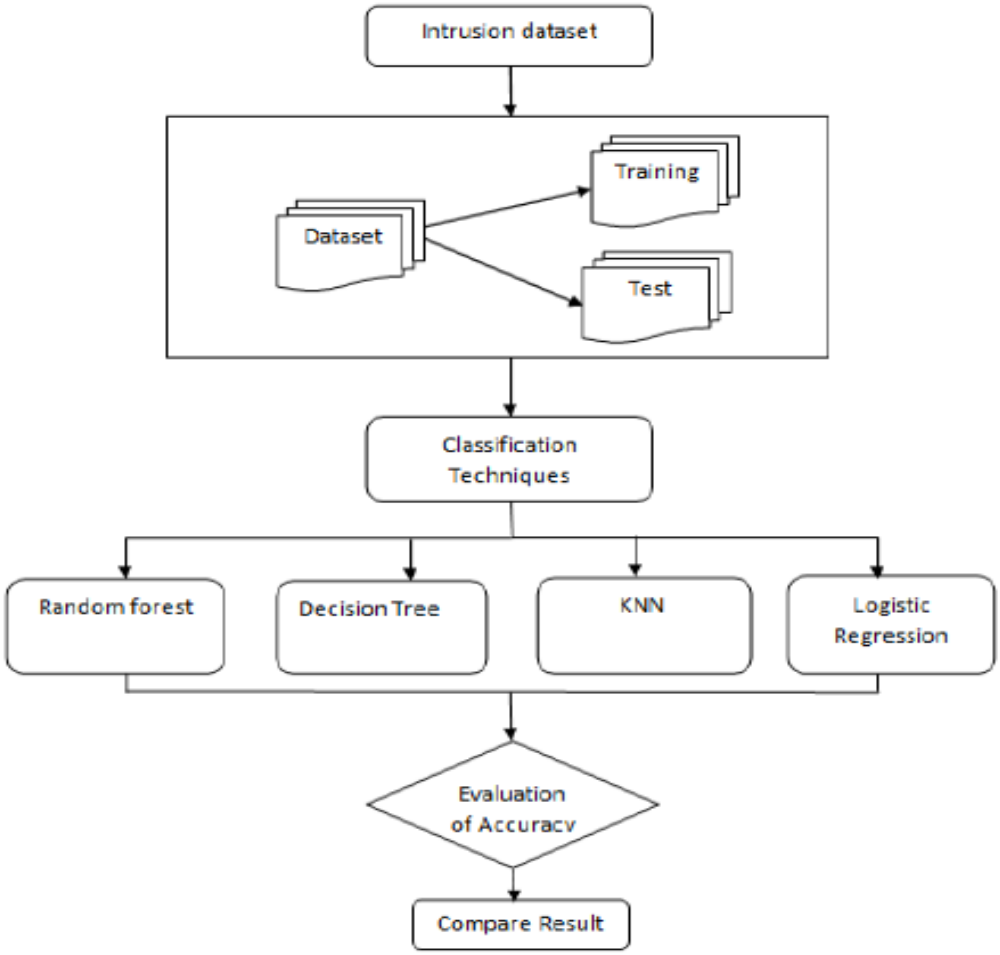


Fig 4.1.2 Model Architecture

4.2. Unified Modeling Language

Unified Modeling Language (UML) is a specific standardized visual modeling language developed most commonly used in software engineering to represent software systems, designs, and processes. UML provides a common language and notation for describing complex software systems, which makes it easier for stakeholders to understand and communicate about these systems.

UML is not a programming language, but rather a visual language that is used to describe the structure, behavior, and interactions of a software system. It is based on a set of standardized diagrams and symbols that represent different aspects of a software system. These diagrams include class diagrams, activity diagrams, sequence diagrams, and state diagrams, among others.

UML has become a widely adopted standard in software engineering, and many software development tools support UML diagramming and modeling. UML is also used in other fields, such as business process modeling and systems engineering, where it provides a standard language and notation for describing complex systems.

4.2.1. Class Diagram

It follows an object-oriented architecture, including classes, attributes, methods, and relationships between classes. In an IDS system that uses ML classifiers, the class diagram would show the ML classifiers and their attributes, such as the training dataset, the algorithm used, and the parameters configured.

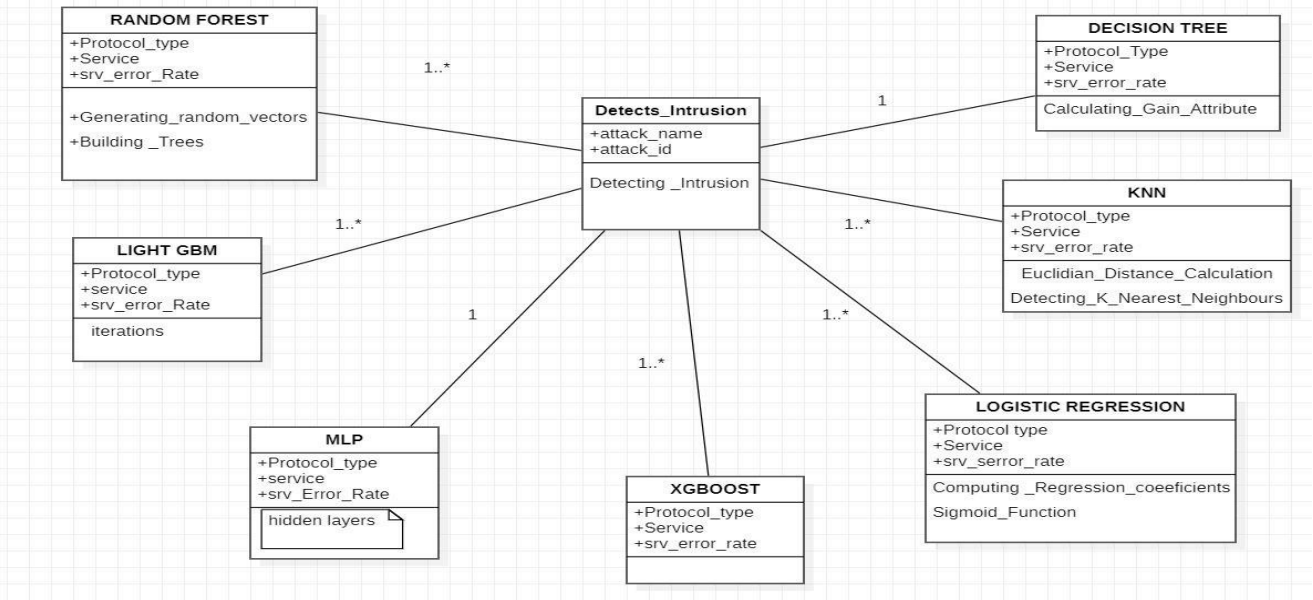
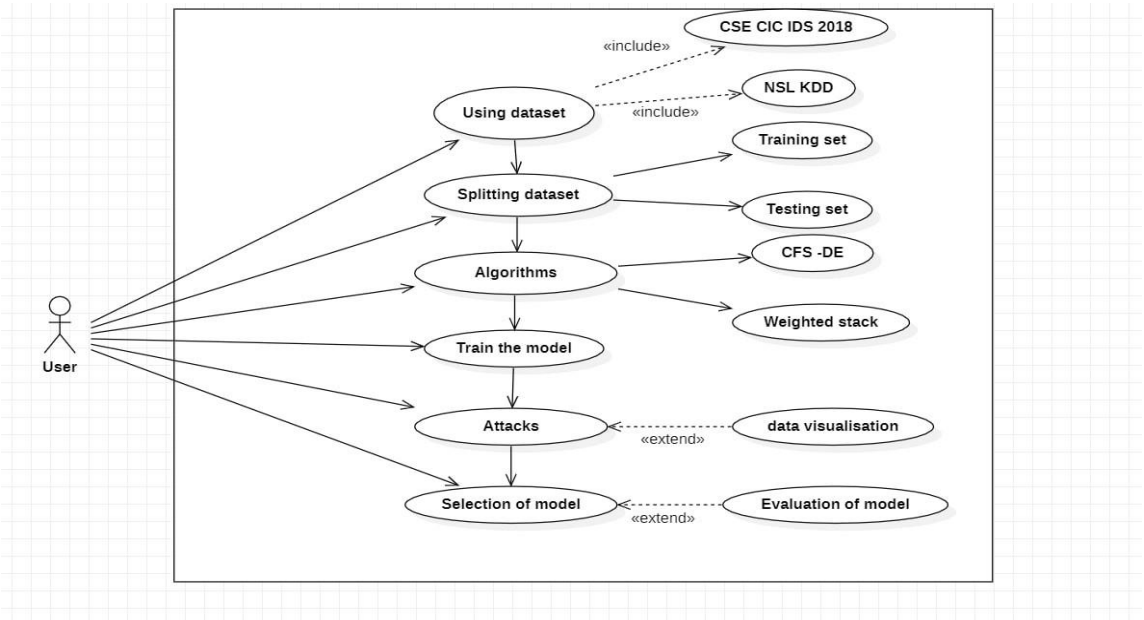


Figure 4.2.1.1 Class diagram of the proposed system

4.2.2. Use case Diagram

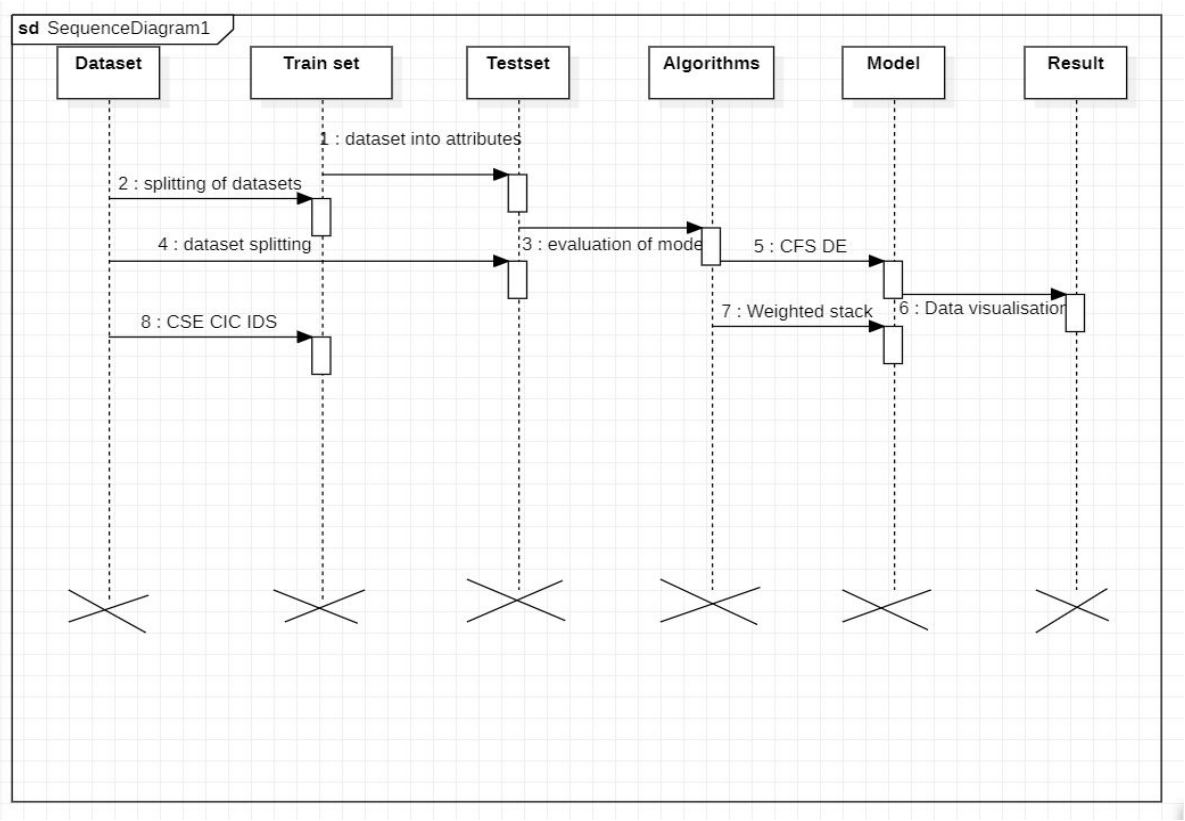
It describes the system's functionality from the user's perspective. It shows the actors (users or external systems) interacting with the system and the actions that they can perform. In an IDS system that uses ML classifiers, the use case diagram would show the actors (e.g., security analysts) interacting with the system to perform tasks such as configuring the system, analyzing alerts, and generating reports.



4.2.2.2. Use case Diagram of the Proposed System

4.2.3. Sequence Diagram

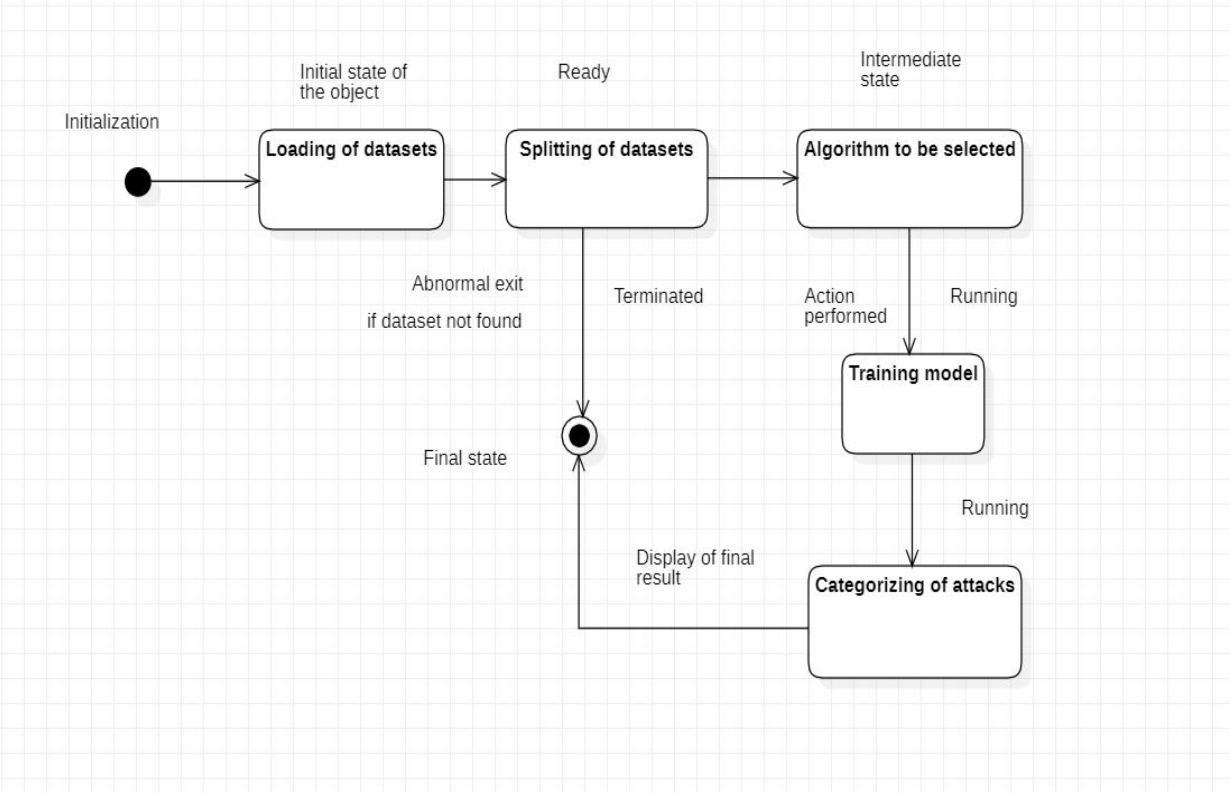
This diagram in general depicts the interaction between objects in a particular scenario. In an IDS system that uses ML classifiers, the sequence diagram would show the steps involved in detecting an intrusion and generating an alert. The diagram would show the flow of data between the sensors, the classifiers, and the alert generation module.



4.2.3.1. Sequence Diagram of the Proposed System

4.2.4. State chart Diagram

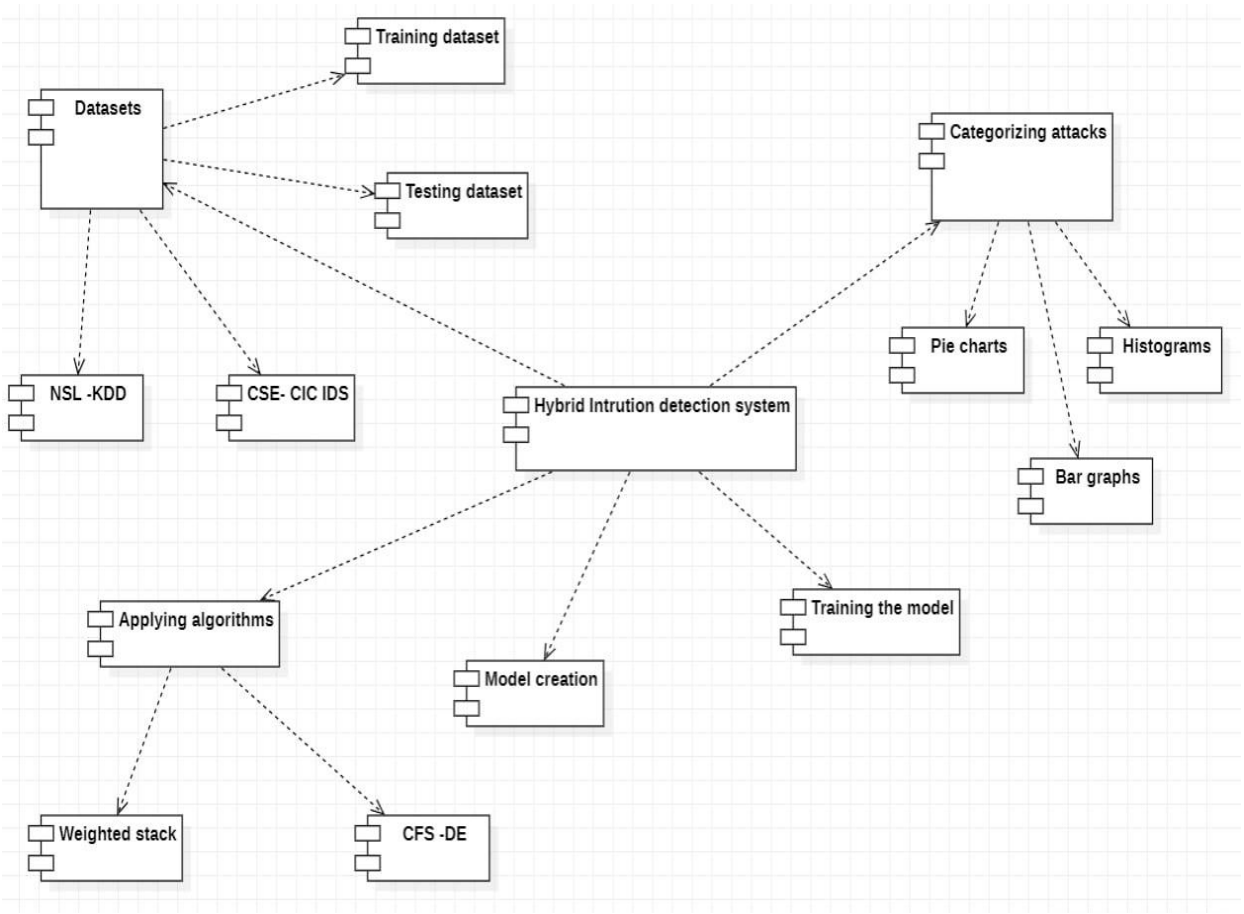
The state machine describes the behavior of an object or a system in response to external stimuli. In an IDS system that uses ML classifiers, the state machine diagram would show the different states of the system, such as idle, training, and detection. The diagram would show the transitions between these states, including the triggers that cause them.



4.2.4.1. State chart Diagram of the Proposed System

4.2.5. Component Diagram

It describes a group of classes that stand in for separate systems or subsystems that can communicate with other parts of the system.



4.2.5.1.Component Diagram of the Proposed System

4.3. Overview of Technologies

The technologies which we used in the project are written below with description about them in detail.

4.3.1. Spyder

Spyder provides a powerful and interactive environment for writing, testing, and debugging Python code, as well as for exploring data sets and visualizing results.

Some of the key features of Spyder include:

1. **Interactive console**: Spyder provides an interactive console that allows users to execute Python code line-by-line, and to see the results immediately.
2. **Code editor**: Spyder provides a code editor with syntax highlighting, auto-completion, and other features that help to make coding easier and more efficient.
3. **Variable explorer**: Spyder provides a variable explorer that allows users to view and manipulate data sets in a spreadsheet-like interface.
4. **Debugger**: Spyder provides a debugger that allows users to step through code and to inspect variables and objects at each step.
5. **Profiler:** Spyder provides a profiler that allows users to identify performance bottlenecks in their code and to optimize it for speed.

   Spyder is designed to be user-friendly and intuitive, and it is especially well-suited for scientific computing and data analysis in Python. It is also highly customizable, with a wide range of plugins and settings that allow users to tailor the IDE to their specific needs. Spyder is available for Windows, macOS, and Linux, and it is distributed under the open-source MIT license.

4.3.2. Python 3.6

Python is a very popular programming language. It was discovered by Guido Van Rossum. Python is an opensource language while has many inbuilt functions. It is the most easiest language becauseit has huge libraries and modules. Opencv ,tesseract, Matplotlib and Numpy are mostly used in the project for image recognition and text extraction .

4.3.3. Microsoft Power BI

Microsoft Power BI is designed to be user-friendly and accessible, even for users without a background in data analysis or programming. It is also highly customizable, with a wide range of settings and options. Power BI is available as a cloud-based service or as a desktop application, and it is included in some Microsoft Office 365 subscriptions.

4.3.4. Machine Learning

Machine learning (ML) is a powerful tool that can be used in Intrusion Detection Systems (IDS) to improve the accuracy and effectiveness of intrusion detection. ML classifiers are algorithms that learn from data and can be used to classify new instances into predefined categories. In the context of IDS, ML classifiers are trained to identify whether a network event or activity is benign or malicious.

One of the main advantages of using ML classifiers in IDS is their ability to learn and adapt to new types of attacks and security threats. Traditional IDS, such as signature-based IDS, rely on predefined rules and patterns to detect attacks. However, these rules can become outdated as attackers develop new techniques to evade detection. In contrast, ML classifiers can learn from new attack patterns and adjust their detection algorithms accordingly, making them more effective in detecting previously unknown attacks.

ML classifiers can be trained using various types of data, including security events. The data is labeled with information about whether each instance is benign or malicious, and the classifier learns to recognize patterns in the data that are indicative of an attack.

There are several types of ML classifiers that can be used in IDS each type of classifier has its strengths and limitations, and the choice of the classifier depends on the specific requirements of the IDS.

ML classifiers can be used in both standalone IDS and hybrid IDS systems. In standalone IDS, ML classifiers can be used to complement or replace existing detection mechanisms. In hybrid IDS, ML classifiers can be combined with other detection mechanisms, to improve the overall detection accuracy.

4.3.5. Neural Networks

Neural networks are useful in IDS because they can learn complex patterns in an attack. We use neural networks in IDS is their ability to adapt to new types of attacks and security threats. This makes them more effective in detecting previously unknown attacks.

Neural networks can also be used to detect subtle variations in network traffic that may indicate an attack.
Another advantage of neural networks in IDS is their ability to handle large volumes of data. Neural networks can process massive amounts of data quickly and accurately, making them useful in detecting attacks in real-time.

Overall, neural networks are highly effective in solving problems in Intrusion Detection Systems because they can learn from new attack patterns, adapt to evolving threats, and detect subtle variations in network traffic.

4.3.6 Web Frameworks

A web framework provides a set of libraries, tools, and best practices to help developers build web applications more efficiently.

Web frameworks typically provide a set of tools and libraries to simplify common web development tasks such as routing, authentication, database connectivity, and templating. They also provide a structure for organizing code and separating concerns between different parts of a web application.

There are many different web frameworks available for different programming languages, including popular frameworks such as Django and Flask for Python, Ruby on Rails for Ruby, and Laravel for PHP.

4.3.6.1. Flask

Flask is a popular web framework for building web applications using the Python programming language. It is a lightweight and flexible framework that provides developers with the tools and libraries necessary to build web applications quickly and easily.

Flask was developed with simplicity in mind, which makes it an excellent choice for small to medium-sized projects. It is also popular among beginners because of its minimalistic approach and ease of use.

Flask provides developers with a wide range of features, including URL routing, template rendering, request and response handling, and support for cookies and sessions. It also supports the use of extensions to add additional functionality to the framework.

One of the most significant advantages of Flask is its extensive documentation and active community, which provides developers with access to a wealth of knowledge and resources.

4.4. Libraries Used

4.4.1. Numpy

It is a fundamental library for scientific computing and data analysis in Python and is commonly used in machine learning applications.In the context of hybrid intrusion detection using ML classifiers, NumPy can be used for various tasks such as data preprocessing, feature selection, and model evaluation.

4.4.2. pandas

Pandas provides a wide range of functions for data manipulation and analysis, such as filtering,sorting, grouping, merging, and joining datasets.

4.4.3. Matplotlib

Matplotlib can be a useful tool in Intrusion Detection Systems for analyzing and presenting data related to network traffic and security events. Visualizations created with Matplotlib can help security analysts identify patterns and anomalies in the data, prioritize their response to security alerts, and evaluate the performance of the IDS.

4.4.4. sklearn

Sklearn provides can be used for classification and anomaly detection in IDS. Sklearn also provides a wide range of evaluation measures that can be used to show the performance of machine learning models in IDS

4.4.5. Seaborn

Seaborn provides several useful plotting functions, such as scatter plots, line plots, histograms, box plots, and heatmaps. They can be used to explore the data, identify patterns, and detect anomalies in the network traffic data.

4.5. Machine Learning Classifiers

Machine learning (ML) classifiers can be used for hybrid intrusion detection by training them on a large dataset of both normal and anomalous network traffic. The ML classifiers can then be used to identify anomalous traffic in real-time and trigger appropriate responses to prevent or mitigate the impact of the intrusion.

TYPES OF CLASSIFIERS

- **Logistic Regression**
- **DecisionTreeClassifier**
- **RandomForestClassifier**
- **KNeighborsClassifier**
- **MLP Classifier**
- **SVC**

4.5.1.Decision Tree Classifier

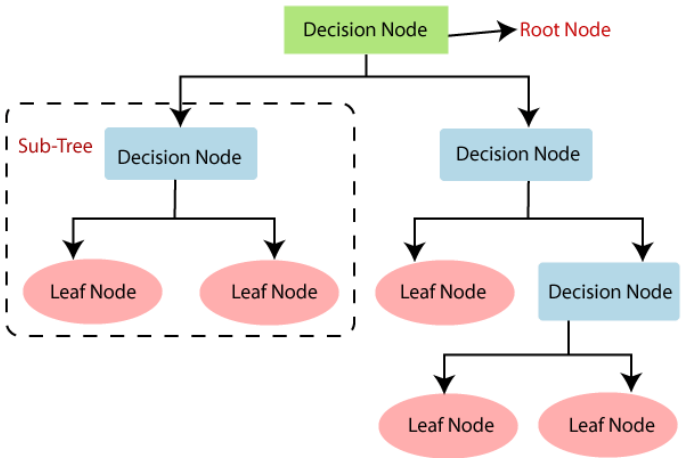


Figure 4.5.1.Working of Decision Tree Classifier

Decision tree is used in Hybrid Intrusion Detection Systems (IDS) to detect security threats. The tree consists of nodes, which represent decisions or tests, and branches, which represent the possible outcomes of each decision.

At each node, the decision tree tests a feature of the data and chooses a branch based on the outcome of the test. This process continues until the decision tree reaches a leaf node, which corresponds to a classification of the data as either normal or malicious.

One advantage of decision tree classifiers is that they are relatively easy to interpret and visualize. The decision tree can be visualized as a flowchart, where each node represents a decision or test and the branches represent the possible outcomes. This can help security analysts understand how the decision tree is making its classifications and identify potential areas for improvement.

To mitigate this, techniques such as pruning and regularization can be used to simplify the decision tree and prevent overfitting.

Overall, decision tree classifiers are a useful tool in Hybrid Intrusion Detection Systems for detecting security threats in network traffic data. They are relatively easy to interpret and visualize, and can be trained on labeled data to learn patterns in the data that are indicative of malicious activity.

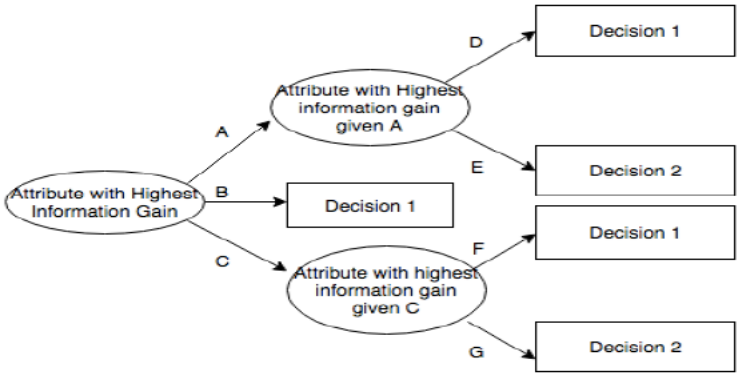


Figure 4.5.1.2. Flowchart of Decision Tree Classifier

Notations used in Decision Tree Classifier

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \left(\frac{|D_j|}{|D|} \right)$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

$$\text{Gini}(D) = 1 - \sum_{i=1}^{m} \text{Pi}^2$$

$$\text{Gini}_A(D) = \frac{|D1|}{|D|} \text{Gini}(D_1) + \frac{|D2|}{|D|} \text{Gini}(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

4.5.2. Random Forest Classifier

Random forests are used in IDS to classify network traffic data as either normal or malicious. A random forest consists of a large number of decision trees, this helps to prevent overfitting and improve the accuracy of the classification.

To classify a new data point, the random forest aggregates the predictions of all the decision trees in the forest. Each decision tree independently classifies the data point based on a subset of the features.

Random forest classifier has several advantages in IDS. One advantage is that it is less prone to overfitting than decision tree classifiers, due to the use of random subsets of features in each decision tree. Another advantage is that it can handle large and complex datasets with high dimensionality.

Furthermore, random forest classifier can also provide an estimation of feature importance, for detecting security threats. This can help security analysts focus their efforts on the most important features and improve the overall performance of the IDS.

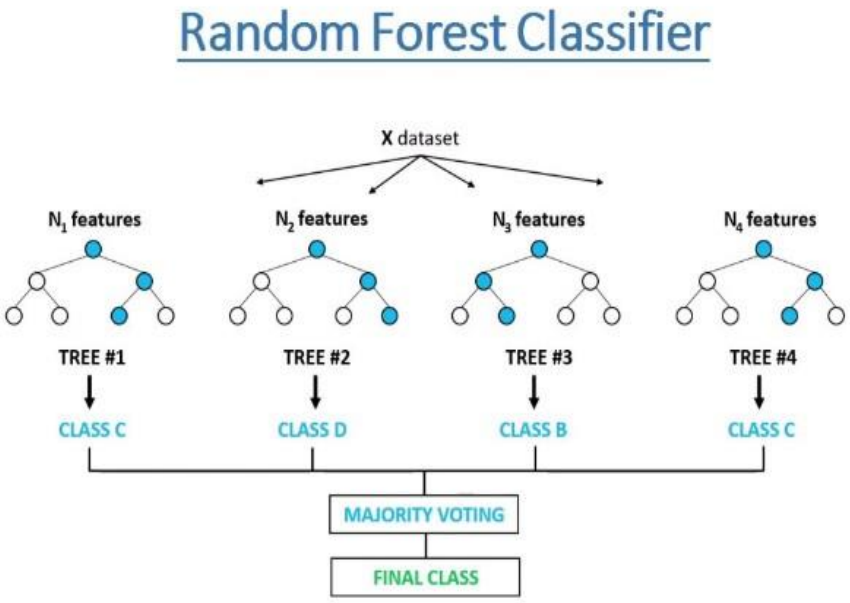


Figure 4.6.2.1.Working of Random Forest Classifier

4.5.3. Support Vector Classifier

Support vector machines (SVMs) can be useful in Intrusion Detection Systems (IDS) as a classifier algorithm. SVMs are acommonly used for classification problems, which involves separating data points into distinct classes based on their features.

In an IDS, SVMs can be trained on a labeled dataset of network traffic to learn patterns of normal and malicious traffic. This can be useful for detecting and preventing attacks on a network.

One advantage of using SVMs in IDS is their ability to handle high-dimensional data and to identify complex patterns in the data. SVMs can also be effective in handling imbalanced datasets, where there are many more examples of one class than the other, which is common in intrusion detection. Overall, SVMs can be a useful tool in IDS for classifying network traffic and identifying potential threats.

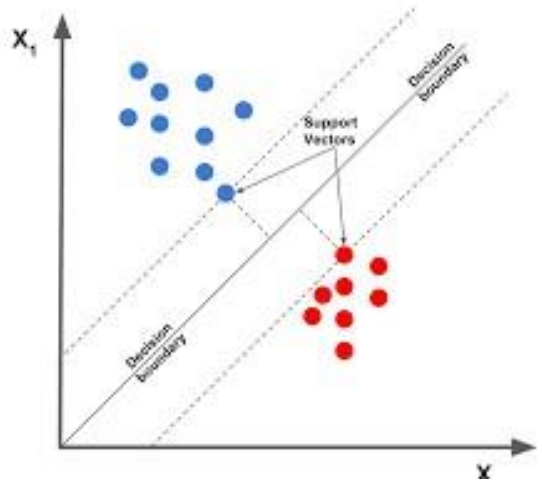


Figure 4.5.3.1.Working of SVC

4.5.4. K-Nearest Neighbor Classifier

K-nearest neighbor (KNN) algorithm can also be useful in Intrusion Detection Systems (IDS) as a classifier algorithm.

In an IDS, KNN can be used to classify network traffic by calculating the distance between the features of the incoming traffic and the features of the known labeled dataset of normal and malicious traffic.

KNN algorithm has some advantages that make it useful in IDS. It is a simple algorithm that is easy to implement and understand, which makes it ideal for real-time classification of network traffic.

However, KNN can have some limitations in IDS. It requires a lot of memory to store the labeled dataset and can be slow when classifying incoming traffic, especially if the dataset is large. It also requires a large number of training examples to achieve good performance and is sensitive to the choice of distance metric used.

Overall, KNN can be a useful tool in IDS for classifying network traffic and identifying potential threats, especially when combined with other techniques, such as feature selection and dimensionality reduction.

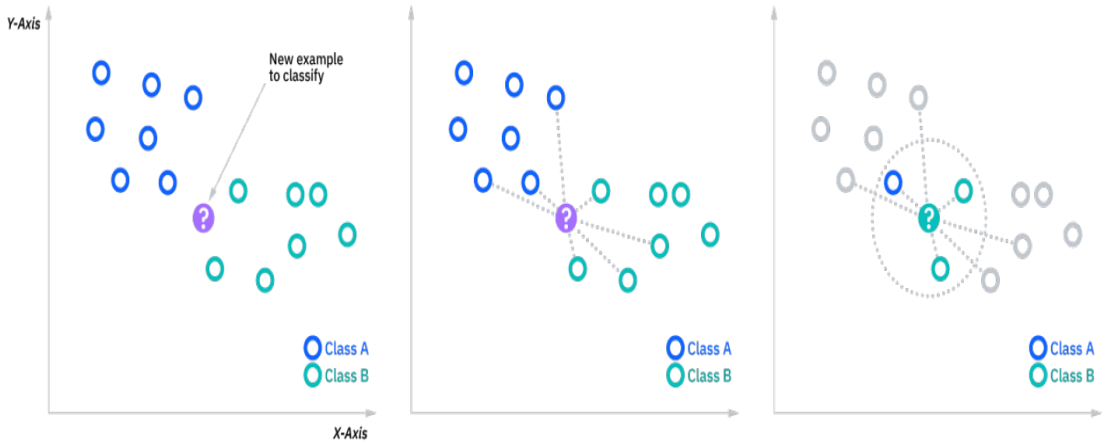
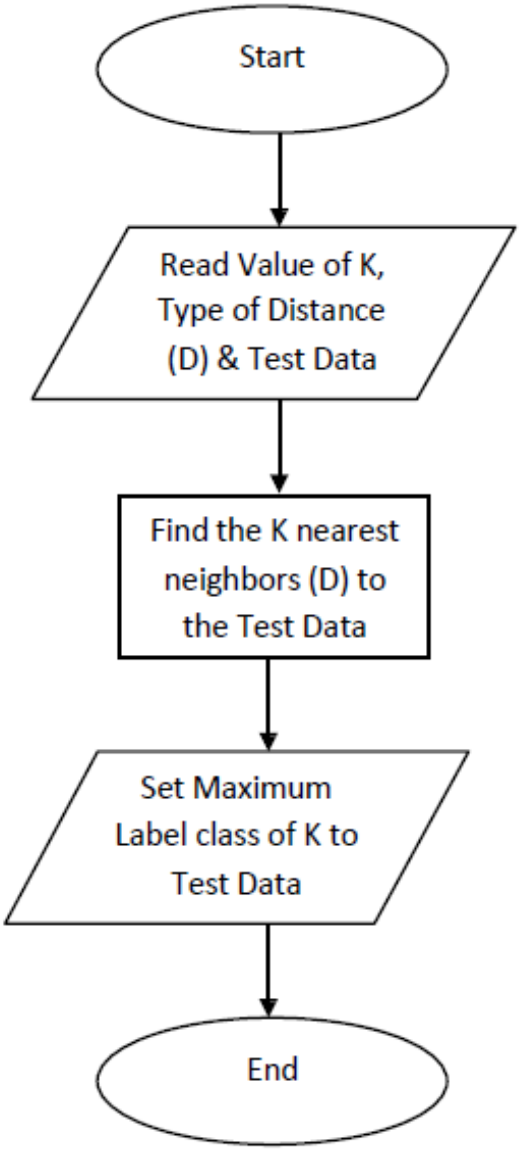


Figure 4.5.4.1.Working of KNN Classifier

Notations used in calculating distance between two points

$$d_A(x,y) = \sum_{i=1}^{N} |x_i - y_i|$$

$$d_E(x,y) = \sum_{i=1}^{N} \sqrt{x_i^2 - y_i^2}$$



4.5.4.2. Flowchart of K-Nearest Neighbor Classifier

4.5.5. Gaussian-Naïve Bayes

Gaussian Naive Bayes (GNB) can be useful in Intrusion Detection Systems (IDS) as a classifier algorithm. It is a probabilistic algorithm based on Bayes' theorem, which calculates the probability of a given instance belonging to a particular class based on its features.

In an IDS, GNB can be used to classify network traffic by modeling the probability distribution of the features of the incoming traffic using Gaussian distributions. The algorithm then calculates the probability of the incoming traffic belonging to each class based on the probability distribution of each class's features and assigns a label to the incoming traffic based on the highest probability.

GNB algorithm has some advantages that make it useful in IDS. It is a simple algorithm that requires a small amount of training data and is fast when classifying incoming traffic.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Overall, GNB can be a useful tool in IDS for classifying network traffic and identifying potential threats, especially when combined with other techniques, such as feature selection and outlier detection.

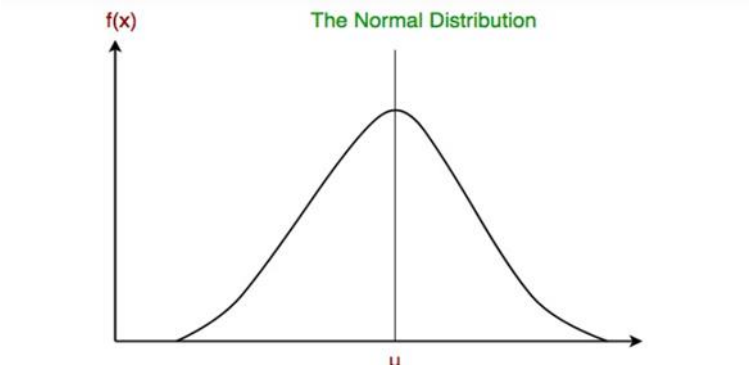


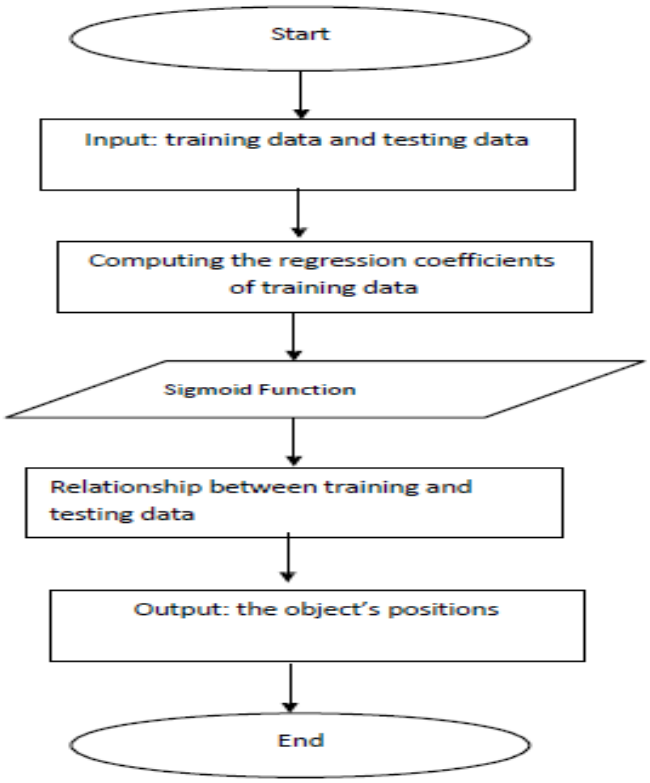
Figure 4.5.5.1. Working of Gaussian- Naive Bayes

4.5.6. Logistic Regression Classifier

In an IDS, logistic regression can be used to model the relationship between the features of the incoming network traffic and the likelihood of it being normal or malicious. The algorithm learns the coefficients of the logistic regression model by minimizing the difference between the predicted probabilities and the true labels of the training dataset. Once trained, the logistic regression model can be used to predict the label of incoming traffic based on its features.

Logistic regression has some advantages that make it useful in IDS. It is a simple algorithm that is easy to interpret and can be trained efficiently with a large number of training examples. It is also able to handle high-dimensional data and can be used for both binary and multi-class classification problems.

However, logistic regression can have some limitations in IDS. It assumes that the relationship between the features and the response variable is linear, which may not always be the case in real-world network traffic. It can also be sensitive to outliers in the data and may not perform well in cases where there is a high degree of overlap between the classes.



4.5.6.1. Flowchart of Logistic Regression

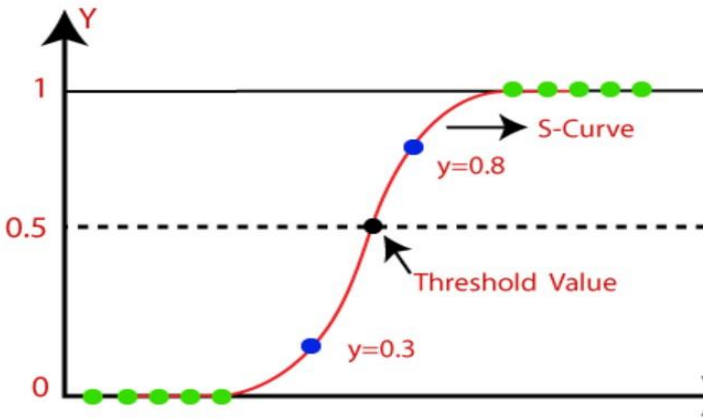


Figure 4.5.6.2. Working of Logistic Regression

4.5.7. MLP Classifier

In an IDS, MLP can be used to classify network traffic by learning a set of weights and actual labels of the data. The input layer receives the features of the incoming traffic, which are then passed through the hidden layers to the output layer, where the final classification is made.

MLP algorithm has some advantages that make it useful in IDS.It is also able to handle high-dimensional data and can be used for both binary and multi-class classification problems.

However, MLP can have some limitations in IDS. It can also be prone to overfitting, which occurs when the model learns to classify the training data too well and performs poorly on new data.

.

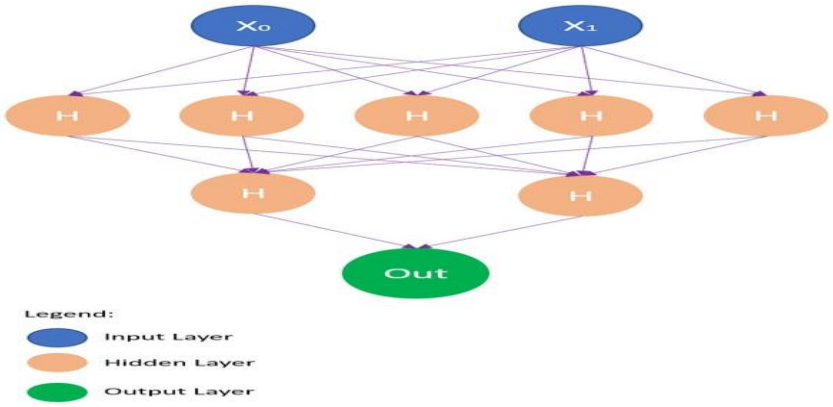


Figure 4.5.7.1.Working of MLP Classifier

4.5.8. Voting Classifier

The voting classifier is an ensemble learning technique that brings together a number of basic models to create the ultimate best result. To predict specific outputs, the base model can independently apply several algorithms like KNN, Random forests, Regression, etc. The process is known as heterogeneous assembly since it results in a diverse product. Homogeneous ensembling, in contrast, is the process of using basic models that employ the same algorithm to predict various outcomes.

There are two categories in the voting classifier: hard voting and soft voting.

Hard voting

The term "hard voting" also refers to majority voting. The classifiers in the base model are each fed a set of training data. The models independently anticipate the output class. The output class is one that the majority of the models predict.

Soft voting

Soft voting uses Classifiers or base models to forecast the classes out of m potential courses using training data. The chance of occurrence for each type is independently assigned by each base model classifier.

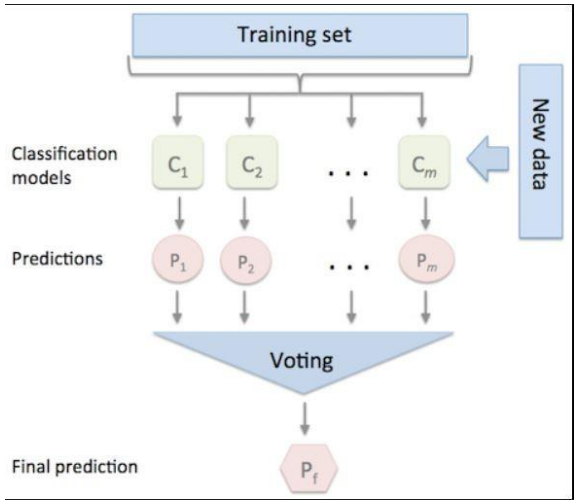


Figure 4.5.8.1.Hard Voting

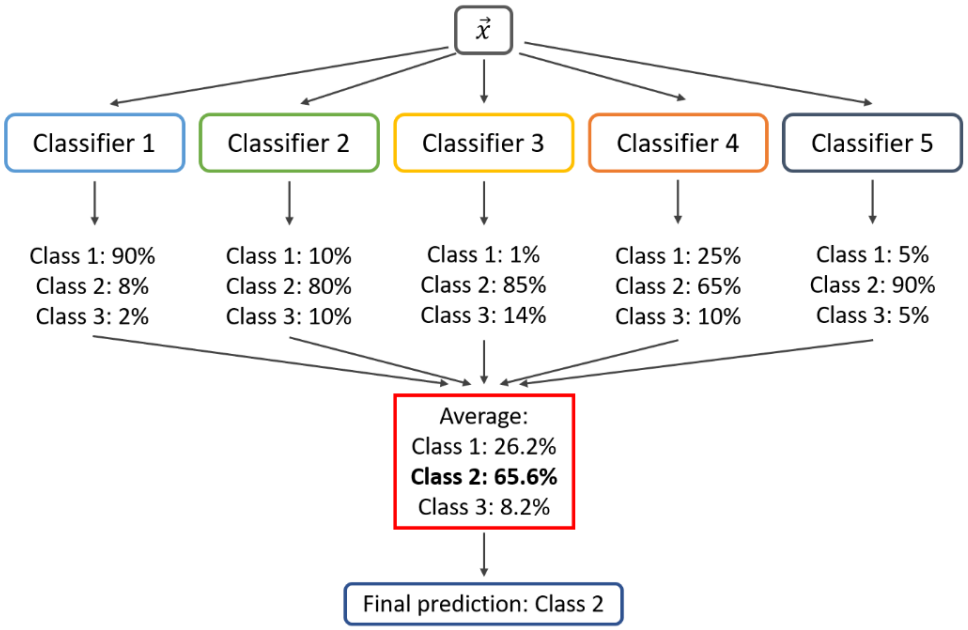


Figure 4.5.8.2. Soft voting

4.6. Creating our own models

4.6.1. Feed Forward Neural Network

FNN are a type of artificial neural network that can be useful in Intrusion Detection Systems (IDS) as a classifier algorithm. FNNs are based on a set of interconnected nodes, or neurons, arranged in layers.

In an IDS, FNN can be used to classify network traffic by learning a set of weights and actual labels of the training data. The input layer receives the features of the incoming traffic, which are then passed through the hidden layers to the output layer, where the final classification is made.

FNN algorithm has some advantages that make it useful in IDS. It can learn complex non-linear relationships between the features and the response variable, which can lead to high classification accuracy.

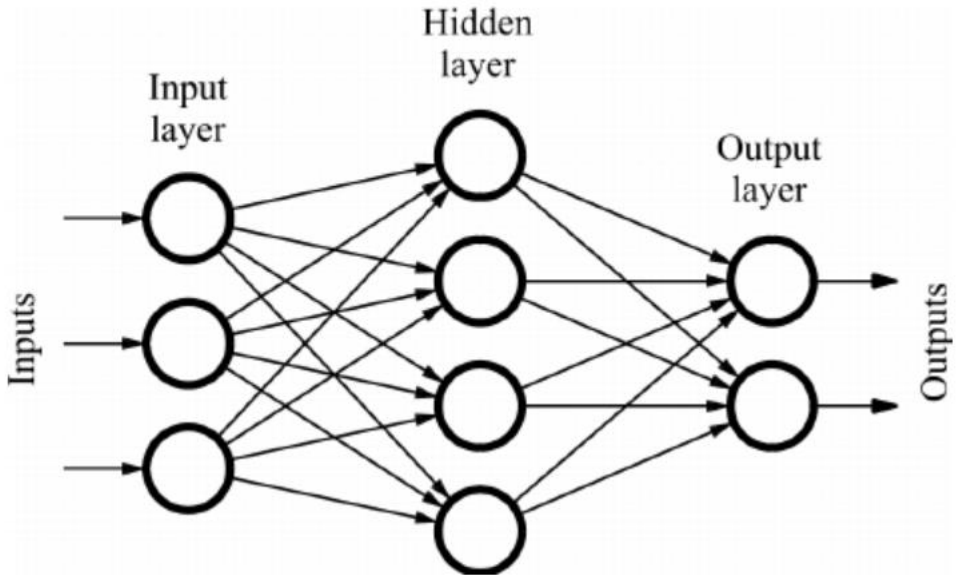


Figure 4.6.1.1.Feed Forward Neural Network

4.6.2. Recurrent Neural Network

Recurrent Neural Networks (RNNs) can be useful in Intrusion Detection Systems (IDS) for measuring the sequence of network traffic data. RNNs can handle sequential data by maintaining a hidden state that captures the temporal dependencies between the data points.

In an IDS, RNNs can be used to classify network traffic by processing the data sequentially, where the output of each time step is dependent on the previous hidden state. The RNN model consists of a set of interconnected neurons that form a feedback loop, where the output of one time step is fed back as input to the next time step.

RNN algorithm has some advantages that make it useful in IDS. It can handle variable-length input sequences and can capture the temporal dependencies between them, which is important in detecting intrusion attacks that span multiple network packets or sessions. It can also be used for both binary and multi-class classification problems.

However, RNNs can have some limitations in IDS. They can be computationally expensive and can suffer from the vanishing gradient problem, which occurs when the gradients used for backpropagation during training become too small to be effective. They can also be prone to overfitting when training on small datasets.

Overall, RNNs can be a useful tool in IDS for measuring the sequence of network traffic data and identifying potential threats, especially when combined with other techniques, such as regularization and dropout to prevent overfitting and the use of more advanced architectures, such as Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) networks, which can better handle long-term dependencies.

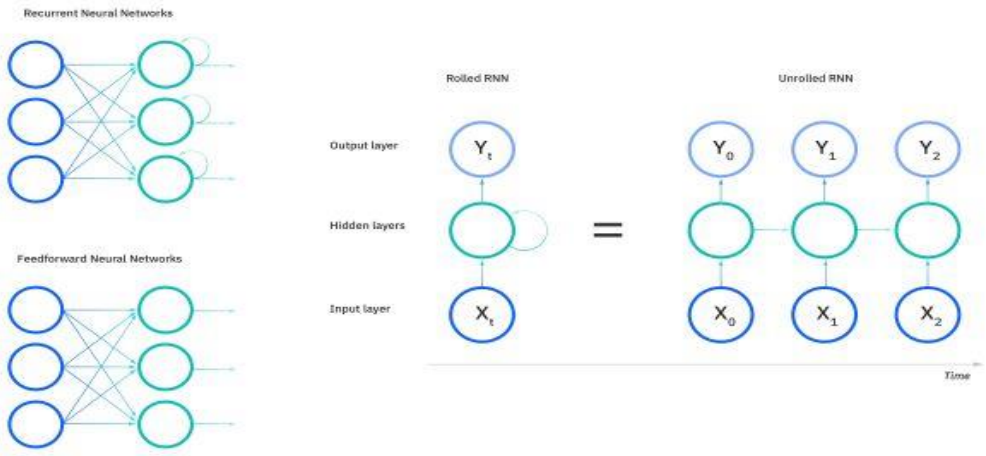


Figure 4.6.2.1.Recurrent Neural Network

4.7. TYPES OF ATTACKS:

- **DENIAL OF SERVICE (DOS)**

Denial of Service (DoS) attacks are a type of cyber attack that can impact the availability of a network or service by overwhelming it with a large volume of traffic. In an Intrusion Detection System (IDS), the role of DoS attacks is to trigger alarms or alerts that notify security personnel to mitigate the impact of the attack.

The main role of a DoS attack in an IDS is to generate traffic that exceeds the normal traffic flow and patterns, which can trigger an anomaly-based IDS to detect the attack. An anomaly-based IDS compares the incoming network traffic to a model of normal traffic behavior and raises an alarm if there is a significant deviation from the model. A DoS attack generates traffic that is significantly different from normal traffic patterns, and can, therefore, trigger the IDS to raise an alarm.

- **SURVEILLANCE OR PROBE**

Intrusion Detection Systems (IDS) rely on various techniques to monitor network traffic and detect potential security threats. One of these techniques is the use of surveillance or probes, which are tools used to monitor network traffic and identify potential security risks.

Surveillance or probes can be either passive or active. Passive surveillance involves monitoring network traffic without modifying it, while active surveillance involves

actively sending traffic to the network to test its security.

The role of surveillance or probes in IDS is to provide real-time information about the network traffic, identify patterns of traffic that may indicate a potential security breach, and help in the investigation of any security incidents that occur. They can also be used to identify any unusual or abnormal network behavior that may indicate a security threat, such as a large volume of traffic from a particular IP address or unusual traffic patterns.

- **REMOTE TO LOCAL (R2L)**

The role of remote-to-local deployment in IDS is to provide comprehensive network security by monitoring all network traffic, regardless of where it originates. Remote-to-local deployment can help in detecting security threats that may originate from outside the local network, DDoS attacks, malware, and phishing attacks. These types of attacks often originate from remote locations and are designed to target the local network, making remote-to-local deployment an essential component of network security.

- **NORMAL**

Normal attacks, also known as benign attacks or legitimate traffic, play an important role in Intrusion Detection Systems (IDS) as they help in creating a baseline for normal network behavior. By understanding normal network behavior, IDS can better detect and alert on abnormal or suspicious activity that may indicate a security threat.

Normal attacks also help in the testing and tuning of IDS. By introducing benign traffic into the network, security teams can test the IDS to ensure it is working properly and detecting suspicious activity. This testing can help identify any false positives or false negatives and ensure that the IDS is configured to detect actual security threats while minimizing false alarms.

- **USER TO ROOT (U2R)**

User to root attacks, also known as privilege escalation attacks, are a type of security threat that can be detected and prevented by Intrusion Detection Systems (IDS).

The role of IDS in detecting user to root attacks is to monitor system logs and identify any suspicious behavior that may indicate an attempt to escalate privileges.

This can include monitoring user activity, checking for unauthorized changes to system files, and identifying unusual network activity that may indicate an attacker attempting to gain access to privileged accounts.

Classes:	DoS	Probe	U2R	R2L
Sub-Classes:	• apache2 • back • land • neptune • mailbomb • pod • processtable • smurf • teardrop • udpstorm • worm	• ipsweep • mscan • nmap • portsweep • saint • satan	• buffer_overflow • loadmodule • perl • ps • rootkit • sqlattack • xterm	• ftp_write • guess_passwd • httptunnel • imap • multihop • named • phf • sendmail • Snmpgetattack • spy • snmpguess • warezclient • warezmaster • xlock • xsnoop
Total:	11	6	7	15

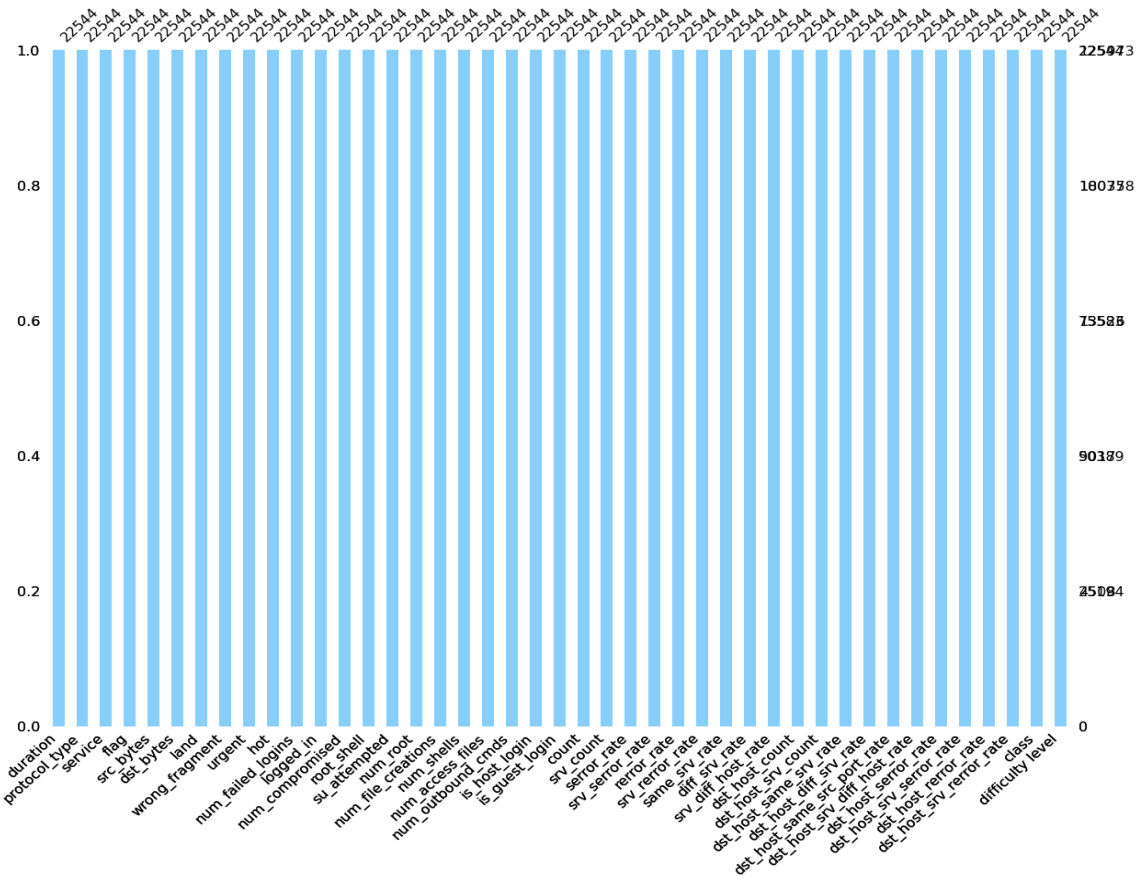
Table 4.7.1.Overview of attacks and sub-categories

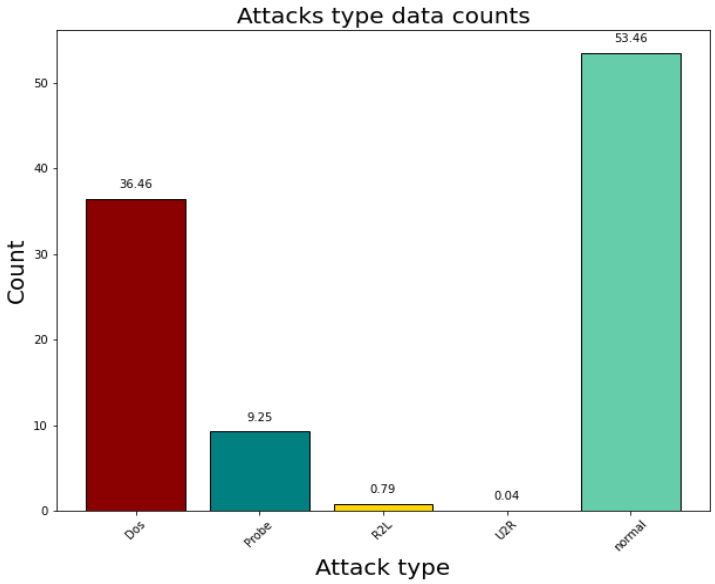
4.8. Exploratory Data Analysis

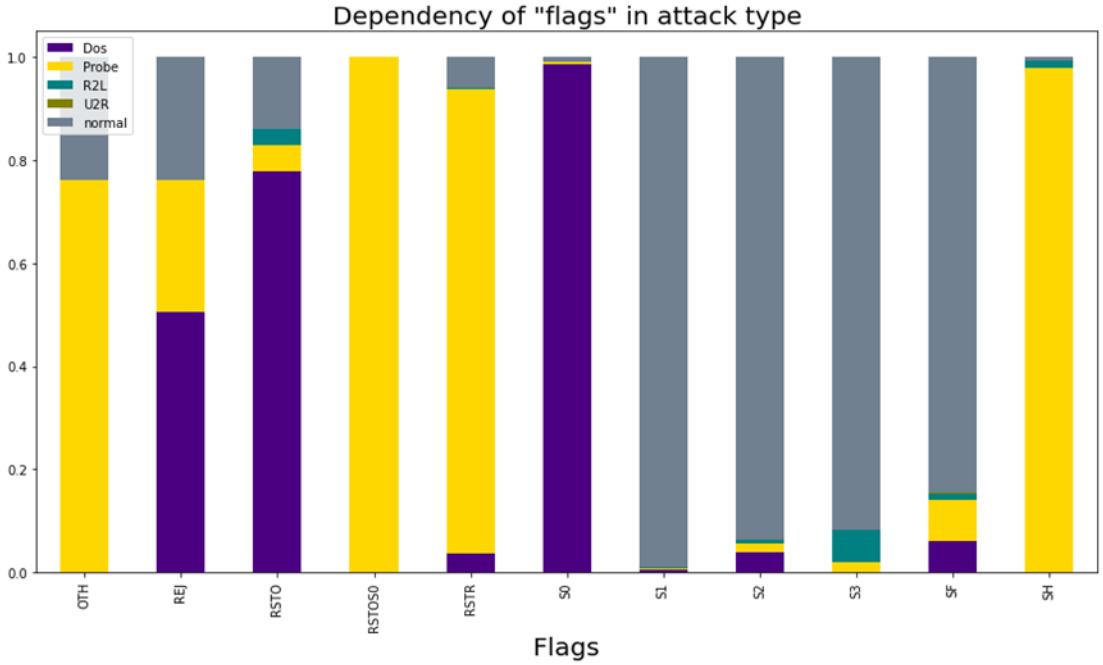
Exploratory Data Analysis (EDA) plays an important role in Intrusion Detection Systems (IDS) as it helps in understanding the characteristics of network traffic data and identifying patterns of behavior that may indicate security threats. EDA is a process of analyzing data sets to summarize their main characteristics and identify patterns and relationships between variables.

The role of EDA in IDS is to provide insights into network traffic data and identify patterns that may indicate potential security threats. EDA can help in identifying important features of the data, such as the distribution of traffic across different ports and protocols, the time of day when traffic peaks, and the patterns of traffic behavior.
By analyzing these patterns, EDA can help in the development of models and algorithms for detecting security threats.

Figure 4.8.1 Snapshots of Exploratory Data Analysis




Attacks type data counts
Count
Attack type
36.46
9.25
0.79
0.04
53.46
Dos
Probe
R2L
U2R
normal


Dependency of "flags" in attack type
Dos
Probe
R2L
U2R
normal
Flags
OTH
REJ
RSTO
RSTOS0
RSTR
S0
S1
S2
S3
SF
SH

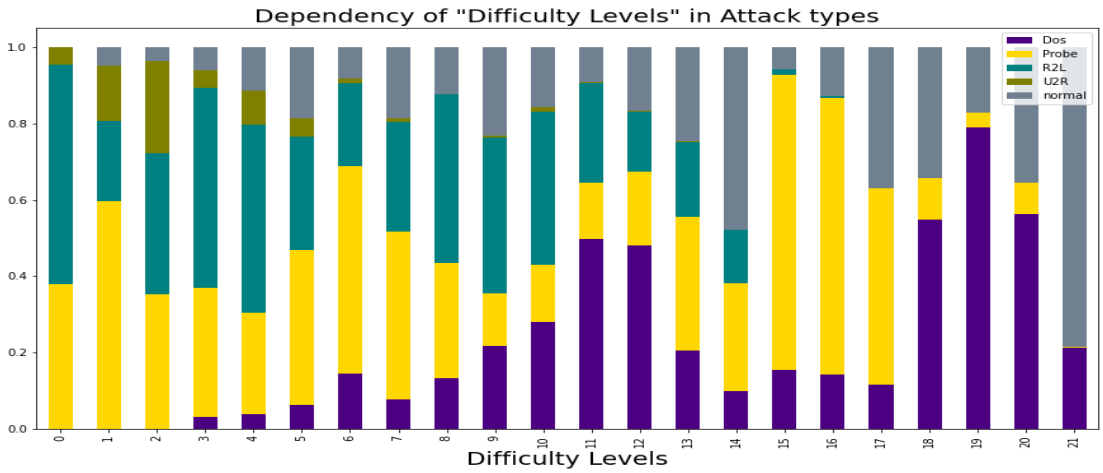
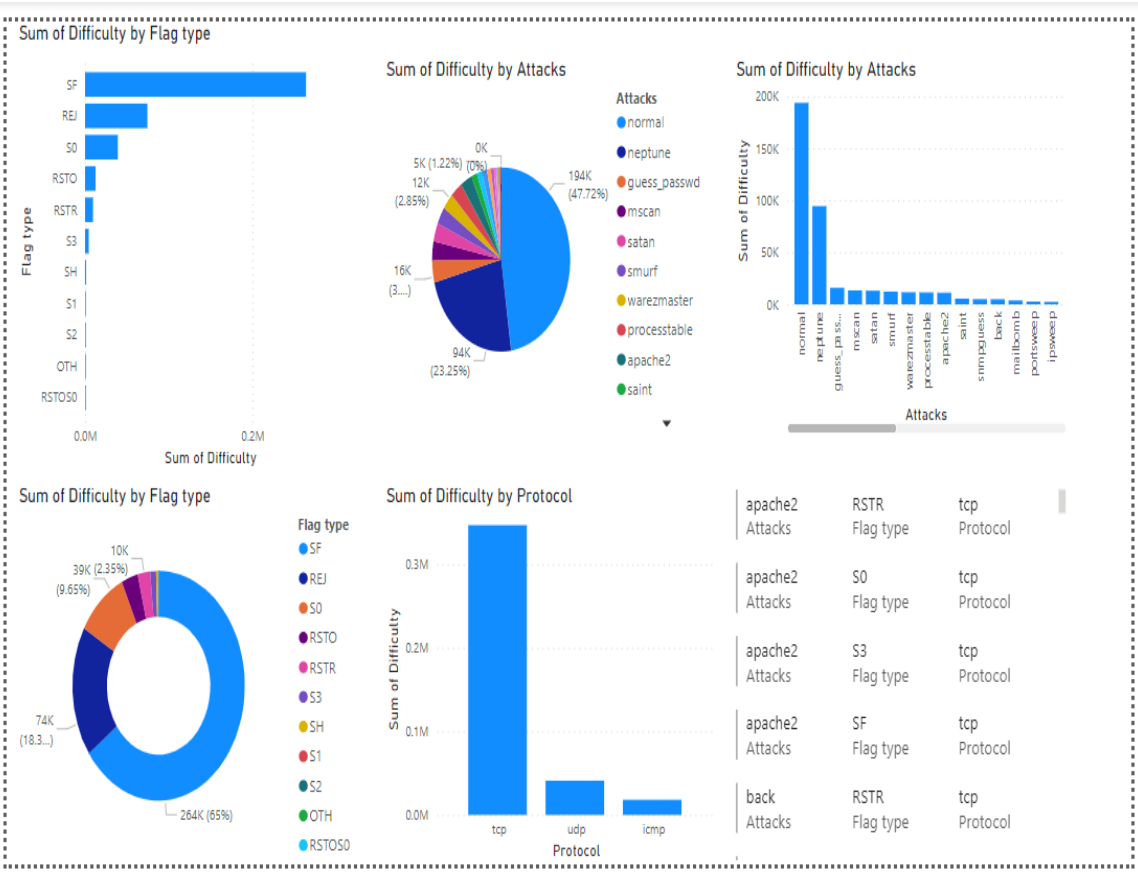
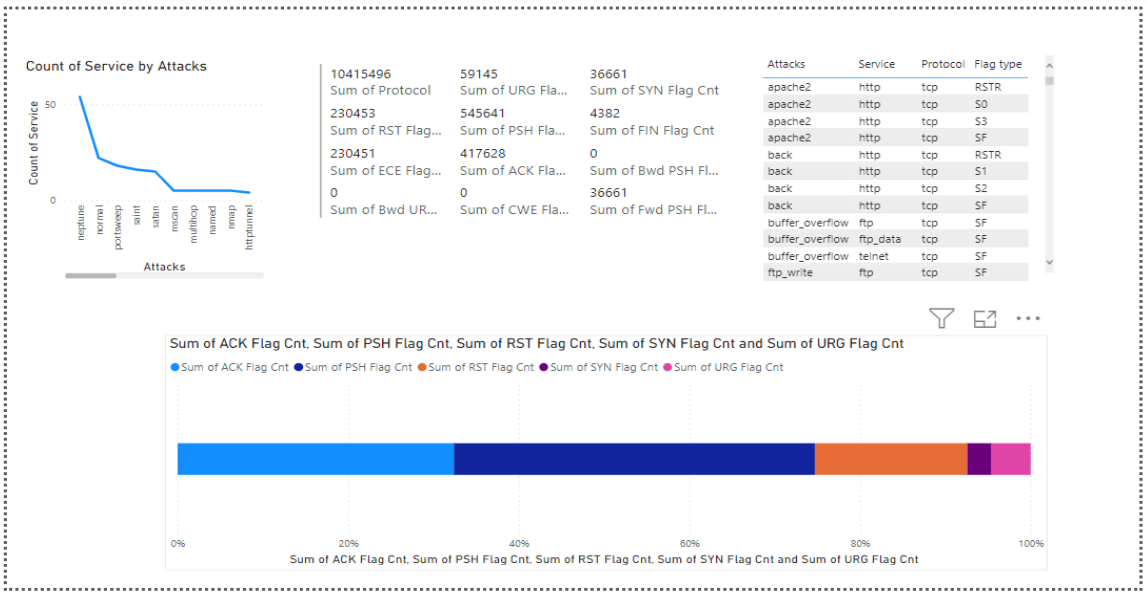
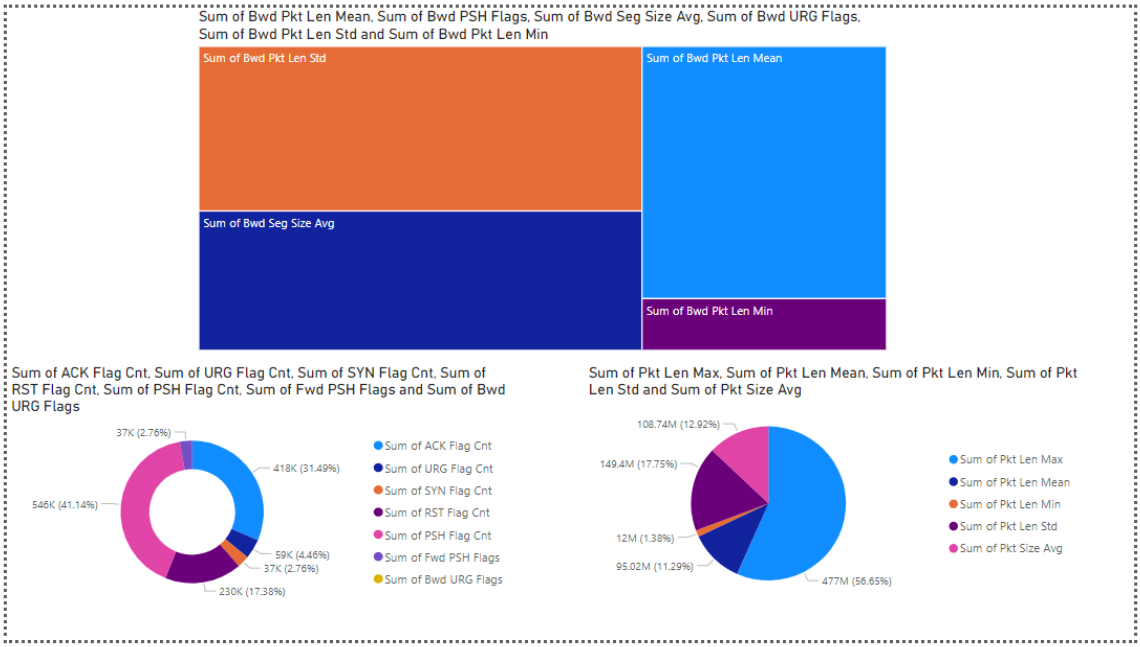


Figure 4.8.2. Visualization of specific attributes using Power BI




Count of Service by Attacks
Count of Service
50
0
neptune
normal
portsweep
saint
satan
mscan
multihop
named
nmap
httptunnel
Attacks
10415496
Sum of Protocol
59145
Sum of URG Fla...
36661
Sum of SYN Flag Cnt
230453
Sum of RST Flag...
545641
Sum of PSH Fla...
4382
Sum of FIN Flag Cnt
230451
Sum of ECE Flag...
417628
Sum of ACK Fla...
0
Sum of Bwd PSH Fl...
0
Sum of Bwd UR...
0
Sum of CWE Fla...
36661
Sum of Fwd PSH Fl...
Attacks Service Protocol Flag type
apache2 http tcp RSTR
apache2 http tcp S0
apache2 http tcp S3
apache2 http tcp SF
back http tcp RSTR
back http tcp S1
back http tcp S2
back http tcp SF
buffer_overflow ftp tcp SF
buffer_overflow ftp_data tcp SF
buffer_overflow telnet tcp SF
ftp_write ftp tcp SF
Sum of ACK Flag Cnt, Sum of PSH Flag Cnt, Sum of RST Flag Cnt, Sum of SYN Flag Cnt and Sum of URG Flag Cnt
Sum of ACK Flag Cnt
Sum of PSH Flag Cnt
Sum of RST Flag Cnt
Sum of SYN Flag Cnt
Sum of URG Flag Cnt
0%
20%
40%
60%
80%
100%
Sum of ACK Flag Cnt, Sum of PSH Flag Cnt, Sum of RST Flag Cnt, Sum of SYN Flag Cnt and Sum of URG Flag Cnt


Sum of Bwd Pkt Len Mean, Sum of Bwd PSH Flags, Sum of Bwd Seg Size Avg, Sum of Bwd URG Flags, Sum of Bwd Pkt Len Std and Sum of Bwd Pkt Len Min
Sum of Bwd Pkt Len Std
Sum of Bwd Pkt Len Mean
Sum of Bwd Seg Size Avg
Sum of Bwd Pkt Len Min
Sum of ACK Flag Cnt, Sum of URG Flag Cnt, Sum of SYN Flag Cnt, Sum of RST Flag Cnt, Sum of PSH Flag Cnt, Sum of Fwd PSH Flags and Sum of Bwd URG Flags
37K (2.76%)
418K (31.49%)
546K (41.14%)
59K (4.46%)
37K (2.76%)
230K (17.38%)
Sum of ACK Flag Cnt
Sum of URG Flag Cnt
Sum of SYN Flag Cnt
Sum of RST Flag Cnt
Sum of PSH Flag Cnt
Sum of Fwd PSH Flags
Sum of Bwd URG Flags
Sum of Pkt Len Max, Sum of Pkt Len Mean, Sum of Pkt Len Min, Sum of Pkt Len Std and Sum of Pkt Size Avg
108.74M (12.92%)
149.4M (17.75%)
12M (1.38%)
95.02M (11.29%)
477M (56.65%)
Sum of Pkt Len Max
Sum of Pkt Len Mean
Sum of Pkt Len Min
Sum of Pkt Len Std
Sum of Pkt Size Avg

5. IMPLEMENTATION

5.1. Dataset Details

- **5.1.1. Data collection**

There are several datasets available that can be used for identifying categories of attacks in Intrusion Detection Systems (IDS). These datasets provide a collection of network traffic data captured from real-world environments and labeled with different types of attacks. Here are some examples of commonly used datasets in IDS:

1. **KDD Cup 1999 Dataset:** This dataset is widely used in IDS research and contains a large collection of network traffic data captured from a simulated environment. The dataset includes various types of attacks such as Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probe attacks. The dataset also includes labeled data for normal network traffic.
2. **UNSW-NB15 Dataset:** This dataset contains a collection of network traffic data captured from a real-world environment and labeled with different types of attacks. The dataset includes nine different types of attacks such as DoS, U2R, R2L, and Probe attacks, among others. The dataset also includes labeled data for normal network traffic.
3. **CIC-IDS2018 Dataset**: This dataset contains a collection of network traffic data captured from a real-world environment and labeled with different types of attacks. The dataset includes a variety of attacks such as DoS, Brute Force, Infiltration, and Web Attacks, among others. The dataset also includes labeled data for normal network traffic.
4. **Kyoto 2006+ Dataset:** This dataset contains a collection of network traffic data captured from a real-world environment and labeled with different types of attacks. The dataset includes various types of attacks such as DoS, R2L, and Probe attacks, among others. The dataset also includes labeled data for normal network traffic.

These datasets provide a valuable resource for IDS researchers and practitioners to develop and evaluate IDS models and algorithms. By using these datasets, researchers can train and test their models on a variety of different types of attacks and normal traffic patterns, helping to improve the accuracy and efficiency of IDS systems.

Attributes	Description
1–4	Basic features of network connections
5–16	Features of network packets
17–22	Features of network flows
23–45	Statistic of network flows
46–63	Content-related traffic features
64–67	Features of network subflows
68–79	General purpose traffic features
80-83	Basic features of network connections

5.1.1.1. Table Description of features in dataset

5.2. ALGORITHMS

5.2.1.Feature selection

Feature selection is a process of selecting a subset of relevant features or variables from a larger set of input features.Here are some of the uses of feature selection in IDS:

1. **Improved accuracy and efficiency**: Feature selection can help in improving the accuracy and efficiency of IDS models by selecting only the most relevant features that are essential for detecting attacks. By reducing the number of input features, the IDS model can be more efficient and effective in identifying security threats.
2. **Reduced computational complexity**: Feature selection can also help in reducing the computational complexity of IDS models by eliminating irrelevant or redundant features. This can result in faster and more efficient detection of security threats.
3. **Improved interpretability**: Feature selection can help in improving the interpretability of IDS models by identifying the most important features that contribute to the detection of security threats. This can help in understanding the underlying patterns and characteristics of network traffic data.
4. **Robustness against noise**: Feature selection can also help in making IDS models more robust against noise and irrelevant features that may be present in the data.

   By selecting only the most relevant features, IDS models can be more accurate, efficient, and interpretable, and more robust against noise and other sources of variability in the data.

5.2.2.Weighted stacking

Here's how weighted stacking can play a crucial role in IDS:

1. **Improved accuracy:** By combining the predictions of multiple base classifiers, weighted stacking can help in improving the accuracy of IDS models. The base classifiers can be trained on different subsets of the input features or using different

learning algorithms, which can help in capturing different aspects of the data and reducing the bias in the predictions.

2. **Robustness against attacks:** Weighted stacking can also help in making IDS models more robust against attacks. By bringing together the predictions of multiple base classifiers obtained , the system can be less vulnerable to attacks that may be able to fool a single classifier. This can help in reducing the false positive and false negative rates, and improving the overall performance of the IDS.
3. **Flexibility and adaptability:** Weighted stacking can also provide a flexible and adaptable framework for IDS. The weights of the base classifiers can be adjusted dynamically based on the performance of the system and the characteristics of the input data. This can help in adapting the system to changing network environments and new types of attacks.

5.2.3. CFS-DE

Correlation-based feature selection (CFS) is a popular technique for selecting relevant features in machine learning. It works by evaluating the correlation between each feature and the target variable and selecting the features that are most strongly correlated with the target variable.

1. **Calculate the correlation matrix**: First, a correlation matrix is calculated, which measures the correlation between each feature and the target variable.
2. **Calculate the merit of each feature**: The merit of each feature is then calculated based on its correlation with the target variable and its correlation with other features. The idea is to select features that are strongly correlated with the target variable but have low inter-correlation with other features.
3. **Select the top-ranked features**: The features are then ranked based on their merit, and the top-ranked features are selected. The number of features selected can be specified by the user or can be determined automatically using a heuristic approach.
4. **Train the machine learning model**: The advantage of CFS is that it not only selects the most relevant features but also takes into account the inter-correlation between features. This helps in reducing redundancy and improving the accuracy of the machine learning model.

Step -1: Initialization

$$V_{i,G+1} = X_{r1,G} + F * (X_{r2,G} - X_{r3,G})$$

Step-2: Mutation and crossover

$$U_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \cdots, u_{Di,G+1})$$

$$u_{ji,G+1} = \begin{cases} V_{ji,G+1} & \text{if } randb(j) \leq CR \text{ or } (j = mbr(i)) \\ X_{ji,G+1} & \text{otherwise} \end{cases}$$

Step-3: Individual Selection

$$X_{i,G+1} = \begin{cases} U_{i,G+1} & \text{if } IEF(U_{i,G+1}) < IEF(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases}$$

$$M_s = \frac{k r_{cf}}{\sqrt{k + k(k-1) r_{ff}}}$$

Algorithm 1: CFS-DE Algotirhm for Feature Selection

Input: Train set and Test set
Output: Selected Feature Subset X_{best}

Initialize population of n individuals
$X_{i,G}(i = 1, 2, \cdots, N)$;
Initialize the mutation rate F and crossover rate CR;
for $G = 1$ *to* G_{max} **do**
  for $i = 1$ *to* N **do**
    Generate new individuals $V_{i,G+1}$;
    Generate new individuals $U_{i,G+1}$;
    Calculate the fitness $IEF(S)$ of feature subset;
    Generate new generation $X_{i,G+1}$;
    if $(IEF(U_{i,G+1}) < IEF(X_{i,G}))$ **then**
      $X_{i,G+1} = U_{i,G+1}$
    end
    else
      $X_{i,G+1} = X_{i,G}$
    end
  end
  $G = G + 1$;
end
$X_{best} = X_{i,G+1}$;

5.2.1. CFS-DE Algorithm

Algorithm 2: Weighted Stacking Model

Input: Train data: $train = \{X_i, Y_i\}_{i=1}^{m}$,Test data: $test = \{X_i\}_{i=1}^{m}$,Classifiers: $CLF = \{clf_i\}_{i=1}^{n}$
Output: $y_i = y_1, y_2, \cdots, y_m$

for $i = 1$ *to* n **do**
  fit the clf_i use $train$;
  calculate CK_i(Kappa coefficient) of clf_i;
  calculate the W_i,$W_i = \ln \frac{1+CK_i}{1-CK_i}$;
end
split $train$ into k fold train=$T_1, T_2, \cdots, T_k$;
for $i = 1$ *to* n **do**
  for $j = 1$ *to* k **do**
    $train' = train$-T_j;
    clf_i.fit($train'$);
    $P_j = clf_i$.predict(T_j);
    $T_j = clf_i$.predict($test$);
  end
  $\bar{T}_i = average(T_j)$ $P'_i = (P_1, P_2, \cdots, P_k)^{\mathrm{T}}$
end
$Test'_i = W_1 * \bar{T}_1, W_2 * \bar{T}_2, \cdots, W_n * \bar{T}_n$;
$Train'_i = W_1 * P'_1, W_2 * P'_2, \cdots, W_n * P'_n$;
$meta\_classifier$.fit($Train'_i$);
$y_i = meta\_classifier$.predict($Test'_i$);

5.2.2. Weighted Stacking

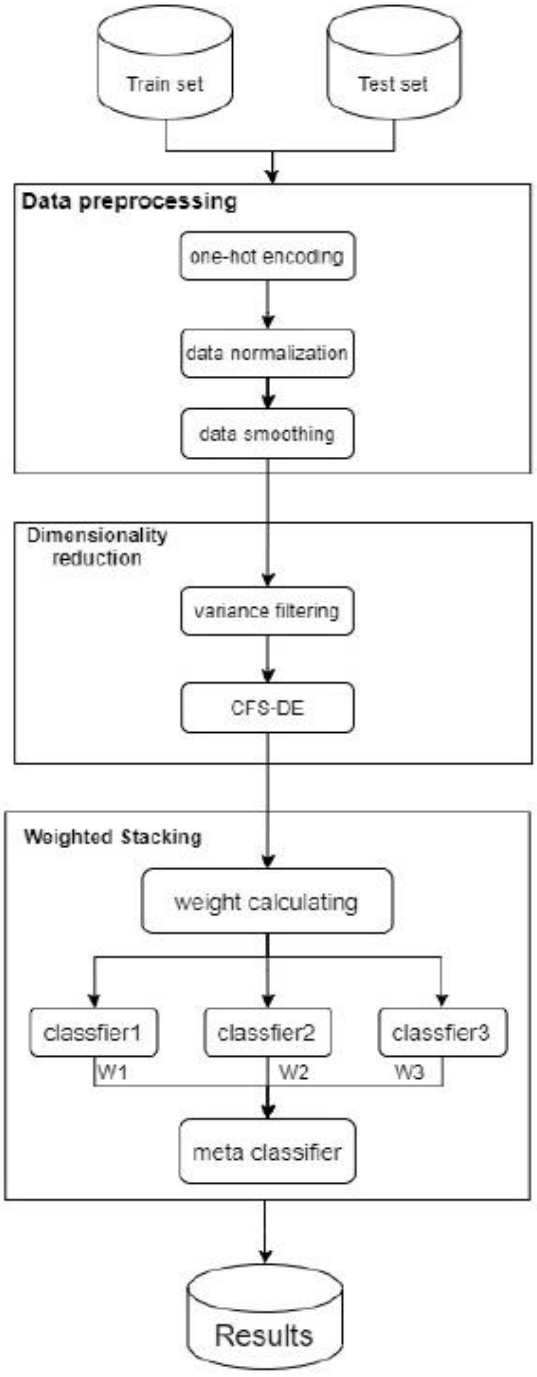


Figure 5.2.3. Feature selection weighted stacking model

Implementation Steps are as follows:

Data Collection:

1. Collect data from different sources (such as server logs, network traffic data, etc.) that are required for the intrusion detection system.
2. Ensure that the data is labeled as either "normal" or "intrusive" so that it can be used for training the machine learning classifiers.

Data Processing:

1. Preprocess the collected data to remove any noise or irrelevant information.
2. Split the data into training and testing datasets.
3. Extract relevant features from the preprocessed data.

Data Visualization or EDA:

1. Visualize the data to gain insights and identify patterns.
2. Use exploratory data analysis techniques to understand the data better.

Training Models:

1. Train the classifiers on the training dataset and fine-tune them using hyperparameter tuning techniques.

Integration with Web App using Flask:

1. Create a Flask web app that can interact with the trained intrusion detection system.
2. Integrate the trained classifiers with the Flask app.
3. Test the intrusion detection system by feeding it with live data from the web app.

Deployment:

1. Deploy the intrusion detection system on a suitable server.
2. Monitor the system for any false positives or false negatives.

5.3. Evaluation Results

5.3.1. Output Screen of Classifiers Trained

```
============================== Random Forest Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9605254370079246

Model Accuracy:
 0.9823278488726387

Confusion matrix:
 [[ 140    0    0    0    0    0    0    0]
 [   0 1359    0    0    0    0    0    0]
 [   0    1 4231    0    0    0    0    0]
 [   0    0    0  197    0    0    0    0]
 [   0    0    0    0 7034    0    0    0]
 [   0    0    0    0    0 1105    0    0]
 [   0    0    0    0    0    1  141    0]
 [   0    0    0    0    0  259    0  301]]

Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       140
           1       1.00      1.00      1.00      1359
           2       1.00      1.00      1.00      4232
           3       1.00      1.00      1.00       197
           4       1.00      1.00      1.00      7034
           5       0.81      1.00      0.89      1105
           6       1.00      0.99      1.00       142
           7       1.00      0.54      0.70       560

    accuracy                           0.98     14769
   macro avg       0.98      0.94      0.95     14769
weighted avg       0.99      0.98      0.98     14769
```

```
------------------------------ Decision Tree Classifier Model Evaluation ------------------------------

Cross Validation Mean Score:
 0.9581557228910136

Model Accuracy:
 0.9998645812174148

Confusion matrix:
 [[ 140    0    0    0    0    0    0    0]
 [   0 1359    0    0    0    0    0    0]
 [   0    0 4232    0    0    0    0    0]
 [   0    0    0  197    0    0    0    0]
 [   0    0    0    0 7034    0    0    0]
 [   0    0    0    0    0 1105    0    0]
 [   0    0    0    0    0    1  141    0]
 [   0    0    0    0    0    1    0  559]]

Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       140
           1       1.00      1.00      1.00      1359
           2       1.00      1.00      1.00      4232
           3       1.00      1.00      1.00       197
           4       1.00      1.00      1.00      7034
           5       1.00      1.00      1.00      1105
           6       1.00      0.99      1.00       142
           7       1.00      1.00      1.00       560

    accuracy                           1.00     14769
   macro avg       1.00      1.00      1.00     14769
weighted avg       1.00      1.00      1.00     14769
```

```
============================== Support Vector Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9526034700089723

Model Accuracy:
 0.9532128106168326

Confusion matrix:
 [[ 115    1    6    7    3    5    3    0]
 [   0 1350    6    0    0    1    0    2]
 [   0    1 4231    0    0    0    0    0]
 [   0    1   23  173    0    0    0    0]
 [   0    0    0    0 7026    2    4    2]
 [   0    0    0    1    2 1099    3    0]
 [   1    0    0    1    0   80   60    0]
 [   0    0    0    0    0  530    6   24]]

Classification report:
               precision    recall  f1-score   support

           0       0.99      0.82      0.90       140
           1       1.00      0.99      1.00      1359
           2       0.99      1.00      1.00      4232
           3       0.95      0.88      0.91       197
           4       1.00      1.00      1.00      7034
           5       0.64      0.99      0.78      1105
           6       0.79      0.42      0.55       142
           7       0.86      0.04      0.08       560

    accuracy                           0.95     14769
   macro avg       0.90      0.77      0.78     14769
weighted avg       0.96      0.95      0.94     14769
```

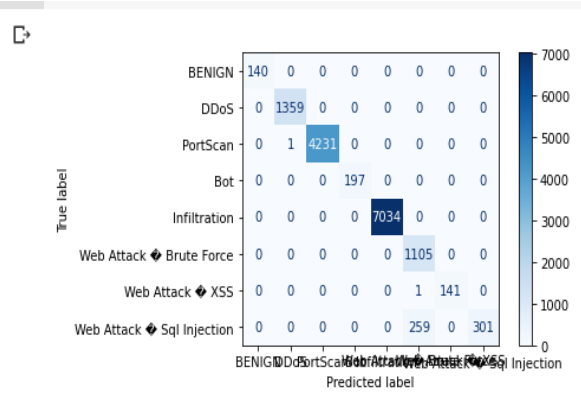
```
============================== KNeighborsClassifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9515204224486388

Model Accuracy:
 0.9695307739183425

Confusion matrix:
 [[ 126    2    1    5    2    2    2    0]
 [   0 1354    0    0    2    0    1    2]
 [   2    3 4227    0    0    0    0    0]
 [   0    1    8  188    0    0    0    0]
 [   0    0    0    0 7029    2    1    2]
 [   1    0    0    1    2  926   14  161]
 [   1    0    0    1    0   12  128    0]
 [   2    0    0    0    1  210    6  341]]

Classification report:
               precision    recall  f1-score   support

           0       0.95      0.90      0.93       140
           1       1.00      1.00      1.00      1359
           2       1.00      1.00      1.00      4232
           3       0.96      0.95      0.96       197
           4       1.00      1.00      1.00      7034
           5       0.80      0.84      0.82      1105
           6       0.84      0.90      0.87       142
           7       0.67      0.61      0.64       560

    accuracy                           0.97     14769
   macro avg       0.90      0.90      0.90     14769
weighted avg       0.97      0.97      0.97     14769
```

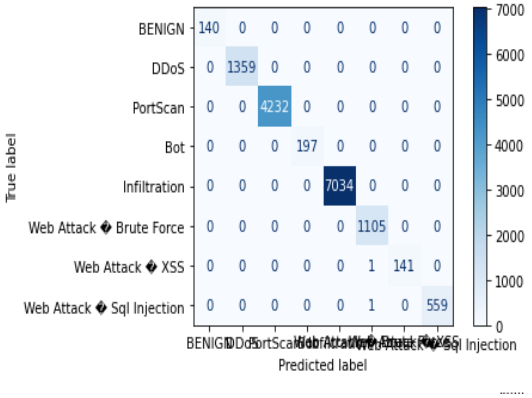
```
============================== LogisticRegression Model Evaluation ==============================

Cross Validation Mean Score:
 0.9509108039624744

Model Accuracy:
 0.9518586227909811

Confusion matrix:
 [[ 111    5    6    5    3    6    4    0]
 [   0 1348    6    0    2    2    1    0]
 [   0    1 4231    0    0    0    0    0]
 [   4    1   21  171    0    0    0    0]
 [   0    0    0    1 7025    3    3    2]
 [   0    0    0    1    1 1101    2    0]
 [   1    0    0    0   10   72   59    0]
 [   0    0    0    0    1  543    4   12]]

Classification report:
               precision    recall  f1-score   support

           0       0.96      0.79      0.87       140
           1       0.99      0.99      0.99      1359
           2       0.99      1.00      1.00      4232
           3       0.96      0.87      0.91       197
           4       1.00      1.00      1.00      7034
           5       0.64      1.00      0.78      1105
           6       0.81      0.42      0.55       142
           7       0.86      0.02      0.04       560

    accuracy                           0.95     14769
   macro avg       0.90      0.76      0.77     14769
weighted avg       0.96      0.95      0.94     14769
```

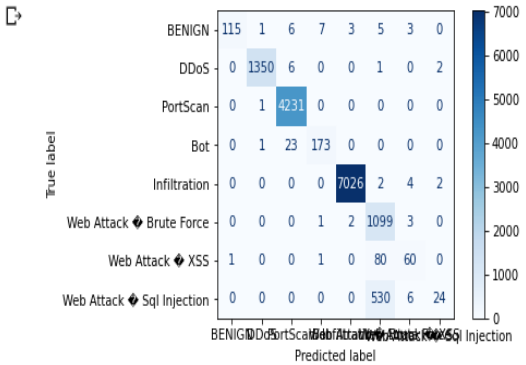
```
============================== Gaussian Naive Baye Model Evaluation ==============================

Cross Validation Mean Score:
 0.8689827123389717

Model Accuracy:
 0.8681698151533618

Confusion matrix:
 [[  50   52   22    2    1    8    5    0]
 [   0  485  519    0  353    2    0    0]
 [   4    4 4224    0    0    0    0    0]
 [   8    3   53  133    0    0    0    0]
 [   0    2   19    0 6974   36    3    0]
 [   0  110    0    0    0  504    2  489]
 [   0   73    0    0    0   12   50    7]
 [   0   17    0    0    0  136    5  402]]

Classification report:
               precision    recall  f1-score   support

           0       0.81      0.36      0.50       140
           1       0.65      0.36      0.46      1359
           2       0.87      1.00      0.93      4232
           3       0.99      0.68      0.80       197
           4       0.95      0.99      0.97      7034
           5       0.72      0.46      0.56      1105
           6       0.77      0.35      0.48       142
           7       0.45      0.72      0.55       560

    accuracy                           0.87     14769
   macro avg       0.78      0.61      0.66     14769
weighted avg       0.86      0.87      0.85     14769
```

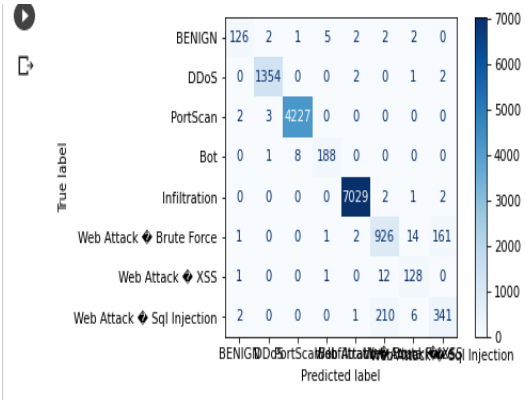
```
============================== MLP Classifier Model Evaluation ==============================

Cross Validation Mean Score:
 0.9538220647947847

Model Accuracy:
 0.9570722459205092

Confusion matrix:
 [[ 129    1    6    0    2    0    2    0]
 [   0 1358    0    0    0    0    1    0]
 [   0    2 4230    0    0    0    0    0]
 [   0    0   12  185    0    0    0    0]
 [   0    0    0    0 7031    1    0    2]
 [   0    0    0    0    2 1100    1    2]
 [   0    0    0    0    0   68   74    0]
 [   0    0    0    0    0  529    3   28]]

Classification report:
               precision    recall  f1-score   support

           0       1.00      0.92      0.96       140
           1       1.00      1.00      1.00      1359
           2       1.00      1.00      1.00      4232
           3       1.00      0.94      0.97       197
           4       1.00      1.00      1.00      7034
           5       0.65      1.00      0.78      1105
           6       0.91      0.52      0.66       142
           7       0.88      0.05      0.09       560

    accuracy                           0.96     14769
   macro avg       0.93      0.80      0.81     14769
weighted avg       0.97      0.96      0.94     14769
```

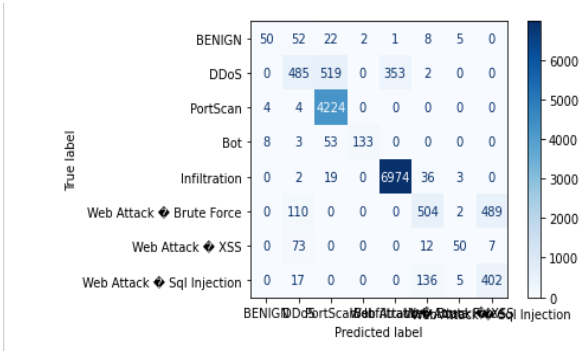
```
============================== Voting Classifier Model Test Results ==============================

Model Accuracy:
 0.958300426472911

Confusion matrix:
 [[  54    2    1    3    0    0    0    0]
 [   0  641    0    0    0    0    0    0]
 [   0    0 1768    0    0    0    0    0]
 [   0    1    1  101    0    0    0    0]
 [   0    0    0    0 2966    0    0    0]
 [   0    0    0    0    0  413    1   81]
 [   0    0    0    0    0    9   49    0]
 [   0    0    0    0    1  161    3   75]]

Classification report:
               precision    recall  f1-score   support

           0       1.00      0.90      0.95        60
           1       1.00      1.00      1.00       641
           2       1.00      1.00      1.00      1768
           3       0.97      0.98      0.98       103
           4       1.00      1.00      1.00      2966
           5       0.71      0.83      0.77       495
           6       0.92      0.84      0.88        58
           7       0.48      0.31      0.38       240

    accuracy                           0.96      6331
   macro avg       0.88      0.86      0.87      6331
weighted avg       0.96      0.96      0.96      6331
```

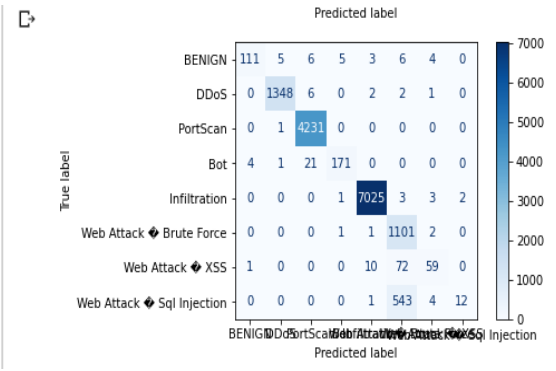

BENIGN 140 0 0 0 0 0 0 0
DDoS 0 1359 0 0 0 0 0 0
PortScan 0 1 4231 0 0 0 0 0
Bot 0 0 0 197 0 0 0 0
Infiltration 0 0 0 0 7034 0 0 0
Web Attack • Brute Force 0 0 0 0 0 1105 0 0
Web Attack • XSS 0 0 0 0 0 1 141 0
Web Attack • Sql Injection 0 0 0 0 0 259 0 301
True label
Predicted label
7000
6000
5000
4000
3000
2000
1000
0

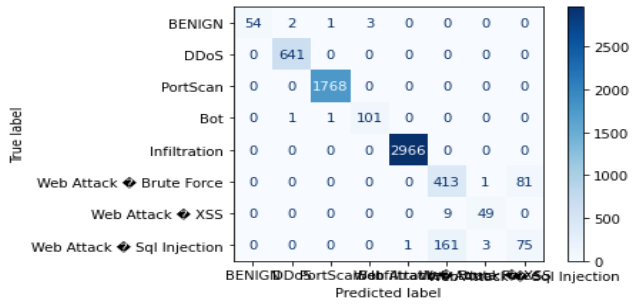

BENIGN 140 0 0 0 0 0 0 0
DDoS 0 1359 0 0 0 0 0 0
PortScan 0 0 4232 0 0 0 0 0
Bot 0 0 0 197 0 0 0 0
Infiltration 0 0 0 0 7034 0 0 0
Web Attack • Brute Force 0 0 0 0 0 1105 0 0
Web Attack • XSS 0 0 0 0 0 1 141 0
Web Attack • Sql Injection 0 0 0 0 0 1 0 559
True label
Predicted label
7000
6000
5000
4000
3000
2000
1000
0

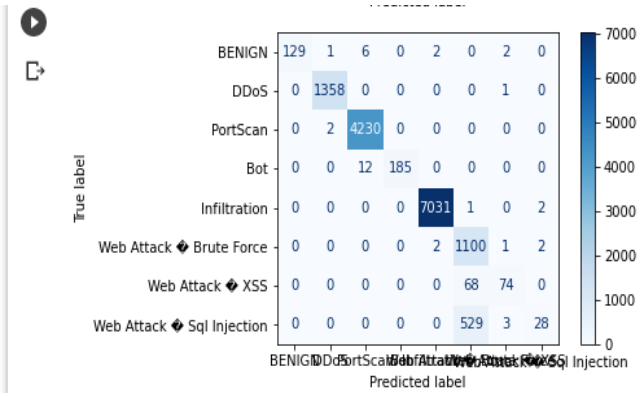

BENIGN 115 1 6 7 3 5 3 0
DDoS 0 1350 6 0 0 1 0 2
PortScan 0 1 4231 0 0 0 0 0
Bot 0 1 23 173 0 0 0 0
Infiltration 0 0 0 0 7026 2 4 2
Web Attack • Brute Force 0 0 0 1 2 1099 3 0
Web Attack • XSS 1 0 0 1 0 80 60 0
Web Attack • Sql Injection 0 0 0 0 0 530 6 24
True label
Predicted label
7000
6000
5000
4000
3000
2000
1000
0


BENIGN 126 2 1 5 2 2 2 0
DDoS 0 1354 0 0 2 0 1 2
PortScan 2 3 4227 0 0 0 0 0
Bot 0 1 8 188 0 0 0 0
Infiltration 0 0 0 0 7029 2 1 2
Web Attack • Brute Force 1 0 0 1 2 926 14 161
Web Attack • XSS 1 0 0 1 0 12 128 0
Web Attack • Sql Injection 2 0 0 0 1 210 6 341
True label
Predicted label
7000
6000
5000
4000
3000
2000
1000
0

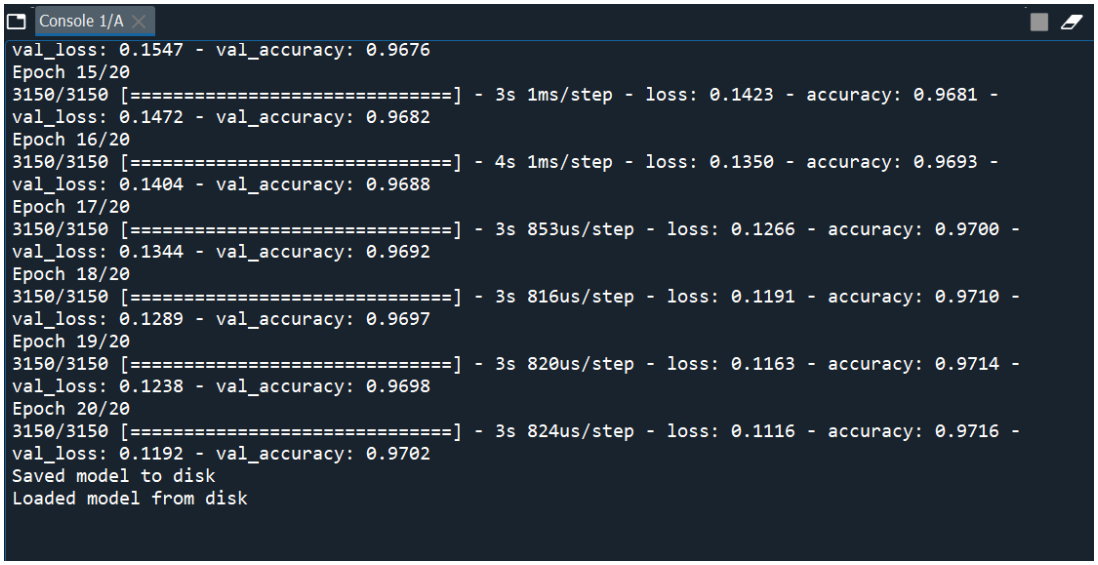

BENIGN 50 52 22 2 1 8 5 0
DDoS 0 485 519 0 353 2 0 0
PortScan 4 4 4224 0 0 0 0 0
Bot 8 3 53 133 0 0 0 0
Infiltration 0 2 19 0 6974 36 3 0
Web Attack • Brute Force 0 110 0 0 0 504 2 489
Web Attack • XSS 0 73 0 0 0 12 50 7
Web Attack • Sql Injection 0 17 0 0 0 136 5 402
True label
Predicted label
6000
5000
4000
3000
2000
1000
0

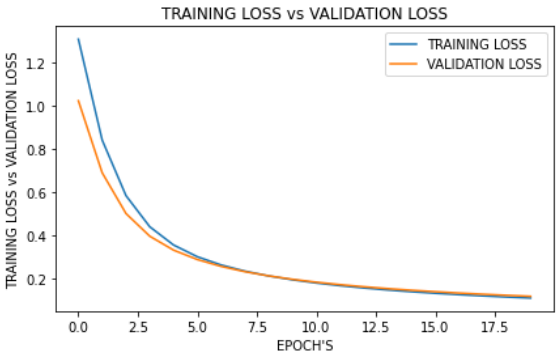

Predicted label
BENIGN 111 5 6 5 3 6 4 0
DDoS 0 1348 6 0 2 2 1 0
PortScan 0 1 4231 0 0 0 0 0
Bot 4 1 21 171 0 0 0 0
Infiltration 0 0 0 1 7025 3 3 2
Web Attack • Brute Force 0 0 0 1 1 1101 2 0
Web Attack • XSS 1 0 0 0 10 72 59 0
Web Attack • Sql Injection 0 0 0 0 1 543 4 12
True label
Predicted label
7000
6000
5000
4000
3000
2000
1000
0

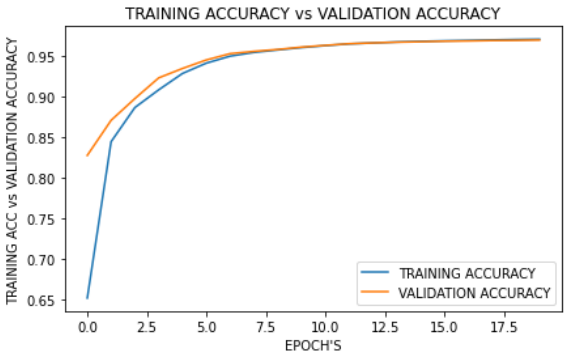




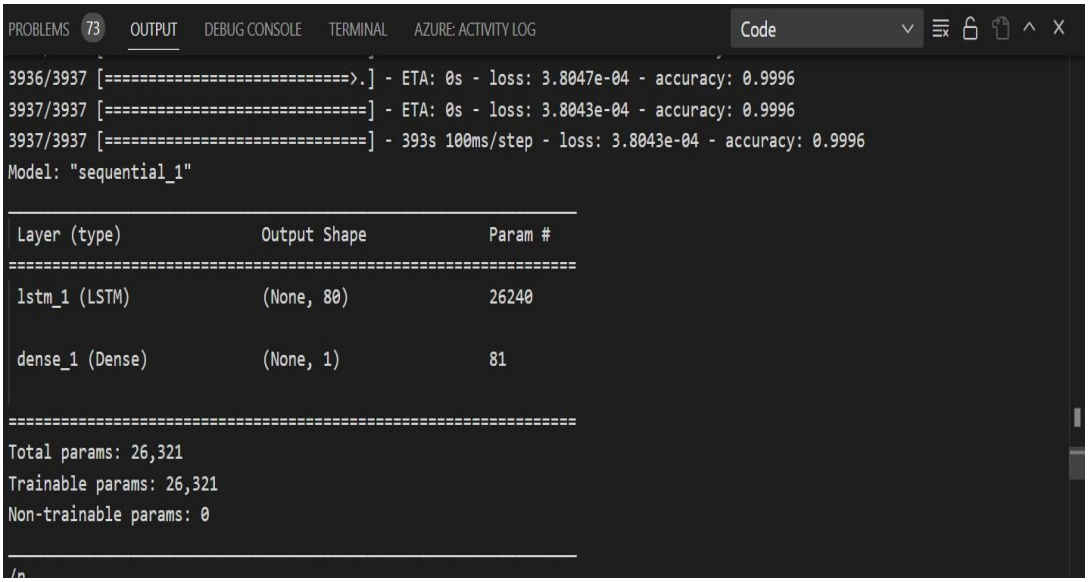
5.3.2. Output Screen of CNN Model trained

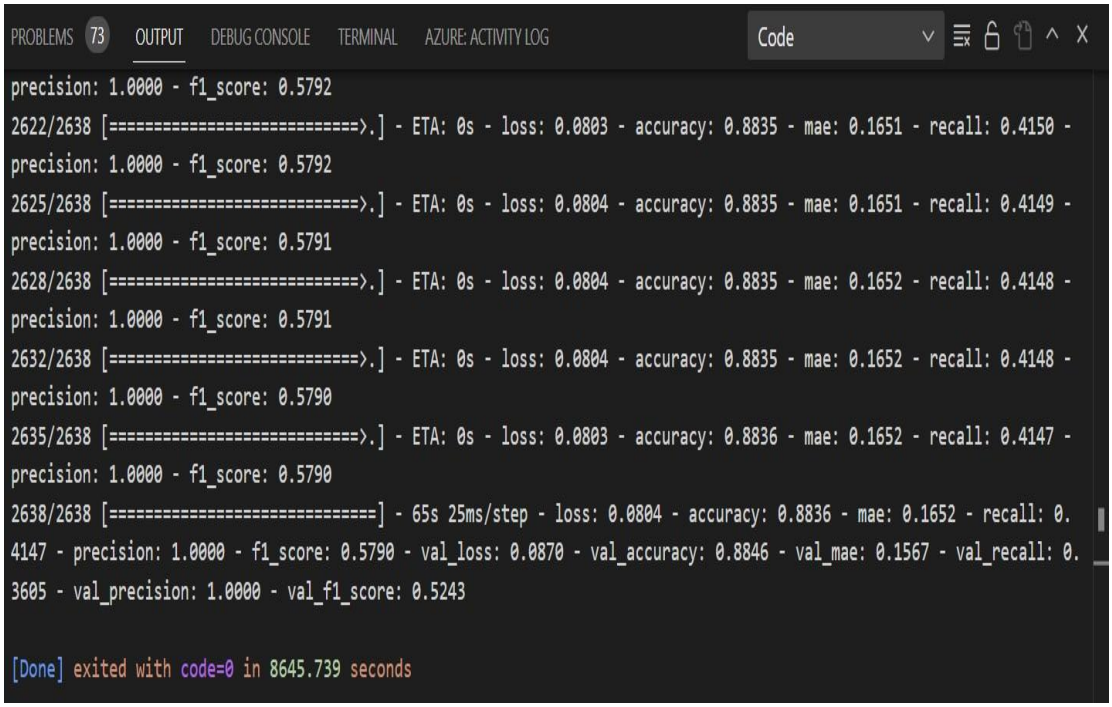


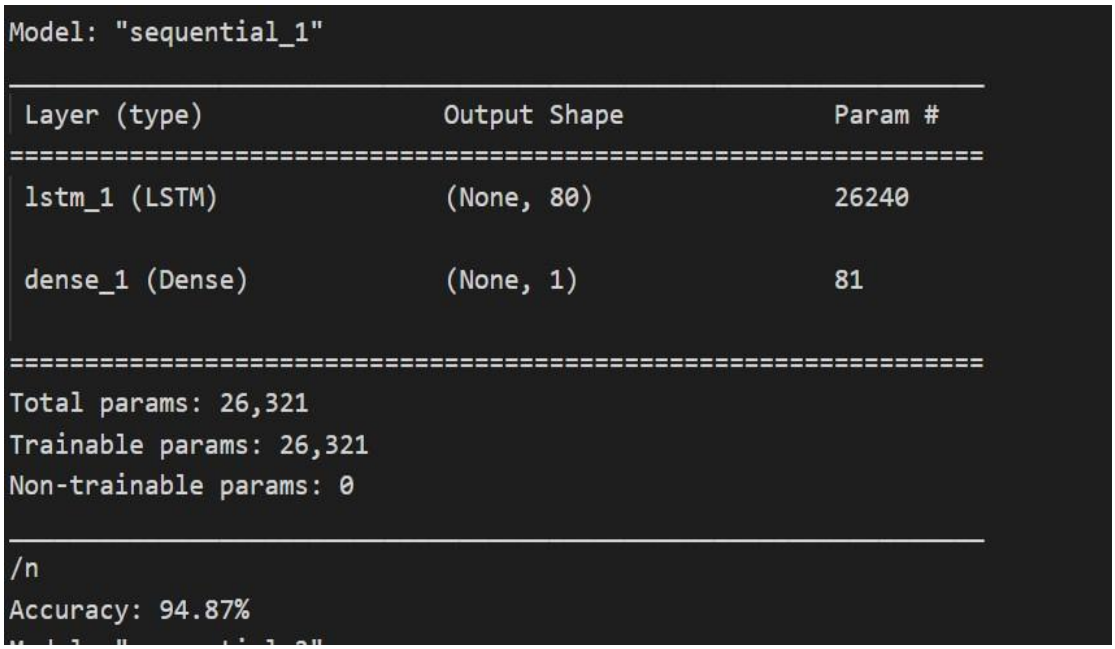




5.3.3. Output Screen of RNN Model Trained




PROBLEMS 73
OUTPUT
DEBUG CONSOLE
TERMINAL
AZURE: ACTIVITY LOG
Code
precision: 1.0000 - f1_score: 0.5792
2622/2638 [==========>.] - ETA: 0s - loss: 0.0803 - accuracy: 0.8835 - mae: 0.1651 - recall: 0.4150 - precision: 1.0000 - f1_score: 0.5792
2625/2638 [==========>.] - ETA: 0s - loss: 0.0804 - accuracy: 0.8835 - mae: 0.1651 - recall: 0.4149 - precision: 1.0000 - f1_score: 0.5791
2628/2638 [==========>.] - ETA: 0s - loss: 0.0804 - accuracy: 0.8835 - mae: 0.1652 - recall: 0.4148 - precision: 1.0000 - f1_score: 0.5791
2632/2638 [==========>.] - ETA: 0s - loss: 0.0804 - accuracy: 0.8835 - mae: 0.1652 - recall: 0.4148 - precision: 1.0000 - f1_score: 0.5790
2635/2638 [==========>.] - ETA: 0s - loss: 0.0803 - accuracy: 0.8836 - mae: 0.1652 - recall: 0.4147 - precision: 1.0000 - f1_score: 0.5790
2638/2638 [==========] - 65s 25ms/step - loss: 0.0804 - accuracy: 0.8836 - mae: 0.1652 - recall: 0.4147 - precision: 1.0000 - f1_score: 0.5790 - val_loss: 0.0870 - val_accuracy: 0.8846 - val_mae: 0.1567 - val_recall: 0.3605 - val_precision: 1.0000 - val_f1_score: 0.5243
[Done] exited with code=0 in 8645.739 seconds

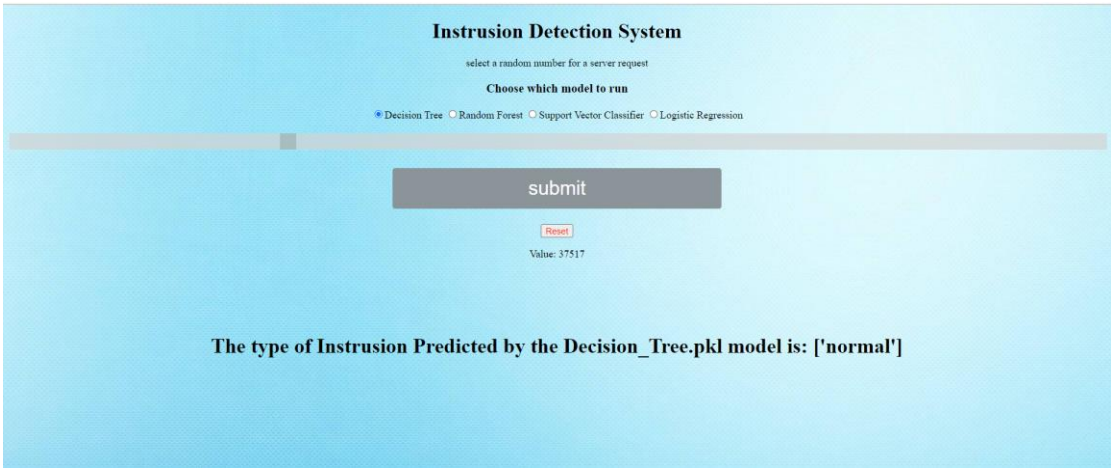

Model: "sequential_1"
Layer (type) Output Shape Param #
lstm_1 (LSTM) (None, 80) 26240
dense_1 (Dense) (None, 1) 81
Total params: 26,321
Trainable params: 26,321
Non-trainable params: 0
/n
Accuracy: 94.87%

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 60)                3720

 dense_2 (Dense)             (None, 1)                 61

=================================================================
Total params: 3,781
Trainable params: 3,781
Non-trainable params: 0
```

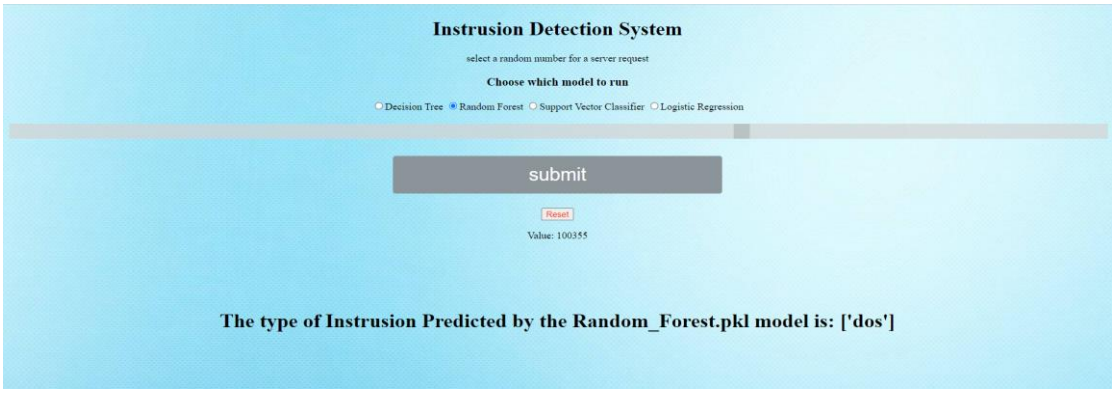
5.3.4. Execution of the Web App

```
Anaconda Prompt (Anaconda

(base) C:\Users\Admin>cd D:\PROJECTS\CODE Hybrid\APP

(base) C:\Users\Admin>d:

(base) D:\PROJECTS\CODE Hybrid\APP>python IDS.py
 * Serving Flask app "IDS" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 304-705-985
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

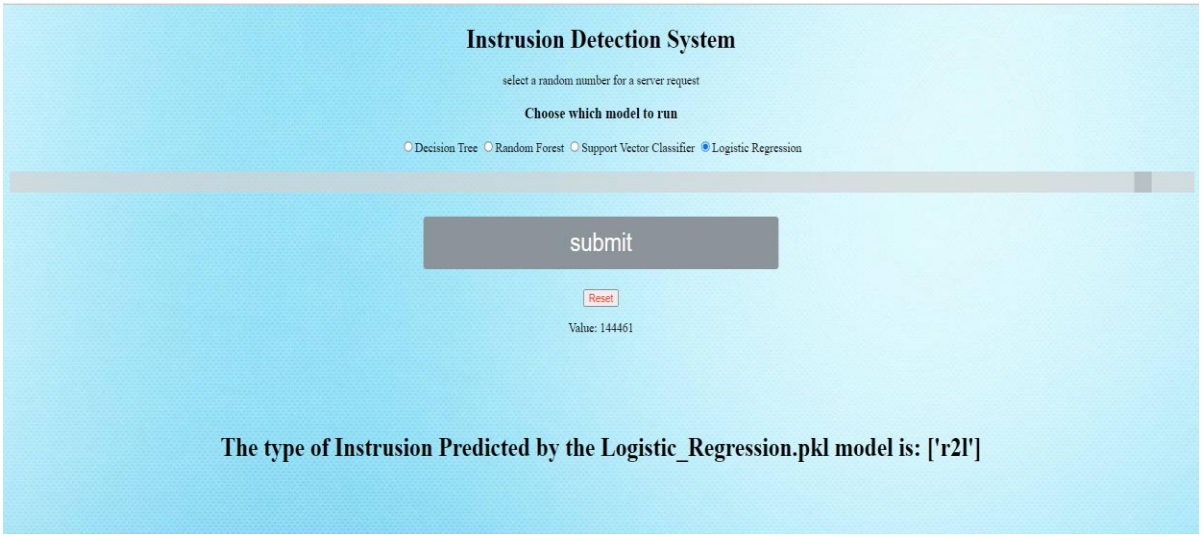
5.3.5. Output Screen of the User Interface



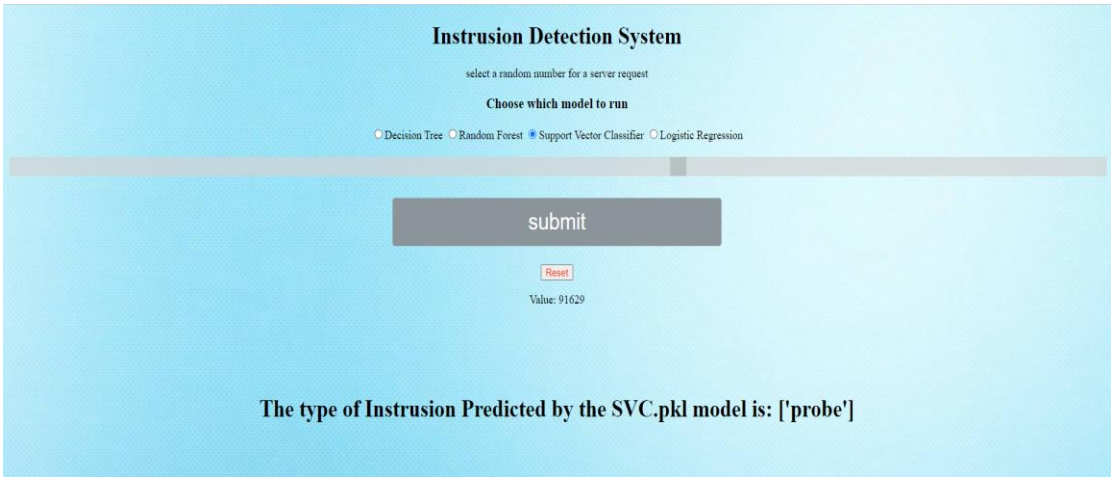
SCREEN 5.3.5.1. Web App



SCREEN 5.3.5.2. Web App



SCREEN 5.3.5.3. Web App



SCREEN 5.3.5.4. Web App

6. TESTING & VALIDATION

6.1. Introduction

Software testing is the process of evaluating a software application or system to ensure that it behaves as expected, meets its requirements, and satisfies the needs of its users. It involves executing software components or the entire software product under controlled conditions and evaluating the results to determine if the software is functioning correctly.

The primary purpose of software testing is to detect defects or bugs in the software, which could lead to system failure, incorrect output, or unexpected behavior. Each type of testing focuses on a specific aspect of the software and employs different techniques and tools to evaluate it.

Overall, software testing is critical to ensure the quality of the software and to ensure that it meets the needs of its users. It helps identify and fix defects early in the development process, reducing the cost and risk of software failure.

6.2. Objectives

- **Finding defects or bugs**: The primary objective of software testing is to find defects or bugs in the software so that they can be fixed before the software is released. This helps improve the quality of the software and reduces the risk of failure.

- **Improving software quality**: Testing helps to improve the quality of the software by identifying and fixing defects, improving usability, ensuring compliance with standards and regulations, and ensuring that the software meets user requirements.

- **Ensuring software functionality**: Testing helps to ensure that the software functions as intended, and meets its specified requirements. This includes testing the software's features, functionality, and performance under different conditions and scenarios.

- **Enhancing user experience**: Testing helps to ensure that the software provides a good user experience by testing the software's usability, accessibility, and user interface.

- **Reducing development costs and time**: By finding defects early in the development cycle, testing helps to reduce the cost and time required for software development.

- **Ensuring software security**: Testing helps to ensure that the software is secure by testing for vulnerabilities and potential security risks.

6.3. Design of Test Cases and Scenarios

A test case is a set of conditions or steps that are designed to evaluate the functionality or behavior of a software application or system. It outlines the inputs, actions, and expected outcomes for a particular test scenario, and helps to verify that the software is working as intended.

Test cases are essential for effective software testing as they help to ensure that all aspects of the software are thoroughly tested and that the software meets the expected requirements and functionality. They also serve as a basis for regression testing, which involves retesting previously tested features to ensure that they continue to function as expected after changes have been made to the software.

A test case typically includes the following elements:

- **Test case ID**: A unique identifier for the test case.
- **Test case description**: A description of the test case, including the purpose and expected outcome.
- **Preconditions:** The conditions that must be met before the test case can be executed.
- **Test steps**: The specific steps that must be taken to execute the test case.
- **Expected result**: The expected outcome of the test case.
- **Actual result**: The actual outcome of the test case.

Hybrid Intrusion Detection using Machine Learning Classifiers is a complex system, and there can be multiple test cases to validate its performance.

Test case for training the ML models:

- Generate a dataset of network traffic, including normal and malicious traffic.
- Split the dataset into training and testing sets.
- Train the ML models, such as Random Forest, SVM, and Naive Bayes on the training set.
- Evaluate the accuracy of the models on the testing set.

Test case for feature selection:

- Choose different sets of features to be used for the models.
- Train the models on each set of features.
- Compare the accuracy of the models and choose the optimal set of features for the final system.

Test case for testing the hybrid intrusion detection system:

- Generate a dataset of network traffic, including normal and malicious traffic.
- Feed the traffic to the hybrid intrusion detection system.
- Record the detection results for each traffic instance.
- Evaluate the detection accuracy of the system using metrics such as accuracy, precision, recall, and F1-score.

Test case for testing the system's performance against different types of attacks:

- Generate datasets of different types of attacks, such as DoS, Port Scanning, and SQL Injection.
- Feed the traffic to the hybrid intrusion detection system.
- Record the detection results for each traffic instance.
- Evaluate the detection accuracy of the system for each attack type using metrics such as accuracy, precision, recall, and F1-score.

Test case for testing the system's scalability:

- Generate a large dataset of network traffic, including normal and malicious traffic.
- Test the system's performance by feeding the traffic to the system at different rates.
- Record the detection results and evaluate the system's performance in terms of detection accuracy and processing time.

6.4. Acceptance criteria

1. The Hybrid Intrusion Detection System using Machine Learning Classifiers must be able to monitor both network traffic and system logs.

2. The system should be able to detect and alert on anomalies or potential security threats in real-time using machine learning algorithms.

3. The system should be able to differentiate between legitimate and malicious activity and minimize false positives.

4. The Hybrid Intrusion Detection System using Machine Learning Classifiers should be customizable to the specific needs and requirements of the organization.

5. The system should be able to generate reports on detected security incidents and provide recommendations for remediation.

6. The system should be able to continuously learn and adapt to new types of attacks and security threats.

7. The system should be able to handle large volumes of network and system data.

8. The system should be easy to deploy, configure and manage, and should not significantly impact network or system performance.

9. The system should be able to integrate with other security tools and systems in the organization's environment.

10. The accuracy and effectiveness of the Machine Learning Classifiers used in the system should be evaluated and tested regularly.

Iteration-1

Table 6.4.1.Creating a User Story Card for the project

Business Requirement	Deliverables	Status
BR_01 The system should validate the relationship among various attributes present in the dataset.	Work on Model Architecture	**In-review**
	Work on System Architecture	**Accepted**
	Working on UML diagrams	**Accepted**
BR_02 Figuring out the best case models for our project Finalizing the models to be created	Use case diagram/Component Diagram	**To be worked again**
	Finalize the requirements for the project.	**ReadyForReview**

Iteration-2

Table 6.4.2. Iteration 2 of the story card created

BR_LR_01	Assign UseCase Diagram	**ReadyForReview**
BR_LR_01	Assignment by UseCase - Add More usecases in detail.	**Reviewed**
BR_LR_01	Assignment by UseCase - Added new relationships in class Diagram	**Reviewed**
BR_LR_02	Changes done in Sequential diagram	**Reviewed**
BR_LR_02	Conceptual Design/System Architecture	**Accepted**

BR_LR_03	Model selected CNN,RNN	**Accepted**
BR_LR_04	EDA using Power BI	**Accepted**
BR_LR_05	Working on various other classifiers	**Reviewed**
BR_LR_06	Evaluation of the trained Models	**Reviewed**
BR_LR_07	Modifying the RNN model by changing the optimizers	**Reviewed**
BR_LR_08	Integration of Web App	**Reviewed**

Iteration-3

Table 6.4.3. Iteration 3 of the storycard created

BR_LR_08	**Priority :01**	Functional testing of the User Interface includes radio buttons, checkboxes, scrollbar	**Reviewed**
BR_LR_07	**Priority:02**	Checking the category of attacks displayed for various number of inputs	**ReadyForReview**
BR_LR_09	**Priority:03**	BU Assignment Rules Maint TestCase: Add New Rule(Change in the display of output)– Success	**ReadyForReview**

6.5. Design of Test Cases for this Project

Project Name :Hybrid IDS

TEST CASE

Test Case ID:	**#A001**	**Test Designed by:**	**Adithya**
TestPriority (Low/Medium/high)	**High**	**Test Designated date:**	**21/03/23**
Module Name:	**Home Page**	**Test Executed by:**	**Adithya**
Test Title:	**View items**	**Test Execution Date:**	**21/03/23**
Description:	**Check if all items are correctly displayed**		

Pre Conditions: NILL

Dependencies: No specific dependencies

STEP	TEST STEPS	TEST DATA	EXPECTED RESULTS	ACTUAL RESULT	STATUS (PASS/ FAIL)
1.	Enter http request	100355,RF	DoS	DoS	Pass
2.	Enter http request	56437,DT	normal	normal	Pass
3.	Enter http request	76589,DT	probe	probe	Pass
4.	Enter http request	91629, SVC	probe	probe	Pass
5.	Enter http request	144461, LR	r2l	R2l	Pass

Project Name : Hybrid IDS

TEST CASE

Test Case ID:	**#A002**	**Test Designed by:**	**Akhil**
Test Priority (Low/Medium/high)	**High**	**Test Designated date:**	**22/03/23**
Module Name:	**Home Page**	**Test Executed by:**	**Varsha**
Test Title:	**View items**	**Test Execution Date:**	**22/03/23**
Description:	**Check if all items are correctly displayed**		

Pre Conditions: NILL

Dependencies: No specific dependencies

STEP	TEST STEPS	TEST DATA	EXPECTED RESULTS	ACTUAL RESULT	STATUS (PASS/ FAIL)
1.	**Enter http request**	**1723,RF**	**normal**	**normal**	**Pass**
2.	**Enter http request**	**6437,DT**	**dos**	**dos**	**Pass**
3.	**Enter http request**	**7589,DT**	**probe**	**probe**	**Pass**
4.	**Enter http request**	**1629, SVC**	**U2l**	**U2l**	**Pass**
5.	**Enter http request**	**144461, LR**	**R2l**	**R2l**	**Pass**

7. EVALUATION RESULTS

Performance Metrics for Classification

Evaluation metrics are important for assessing the performance of an Intrusion Detection System (IDS) and for comparing different models. Here are some commonly used evaluation metrics for IDS:

1. **Accuracy:** This is a measure of the overall correctness of the IDS, which is defined as the number of correct predictions which when divided by the total number of predictions.
2. **Precision:** This measures the proportion of true positives (correctly identified attacks) among all the instances that were classified as attacks. A high precision indicates that the system is accurately identifying attacks and minimizing false positives.
3. **Recall:** This measures the proportion of true positives among all the actual attacks in the dataset. A high recall indicates that the system is correctly identifying a high percentage of attacks and minimizing false negatives.
4. **F1-score:** This is a harmonic mean of precision and recall and provides a balance between the two metrics. A high F1-score indicates that the system is accurately identifying attacks while minimizing both false positives and false negatives.
5. **True positive rate (TPR) or sensitivity**: This is the proportion of actual attacks that are correctly identified by the IDS. A high TPR indicates that the system is accurately identifying attacks and minimizing false negatives.
6. **False positive rate (FPR) or specificity**: This is the proportion of non-attacks that are incorrectly classified as attacks by the IDS. A low FPR indicates that the system is accurately identifying non-attacks and minimizing false positives.
7. **Area under the ROC curve (AUC-ROC):** This is a measure of the overall performance of the IDS across different threshold values. The AUC-ROC is calculated by plotting the true positive rate which we have obtained against the false positive rate at different threshold values and calculating the area under the curve. A higher AUC-ROC indicates better overall performance.

8. **Area under the precision-recall curve (AUC-PR):** This is another measure of the overall performance of the IDS, which is based on the precision-recall curve rather than the ROC curve. A higher AUC-PR indicates better overall performance.

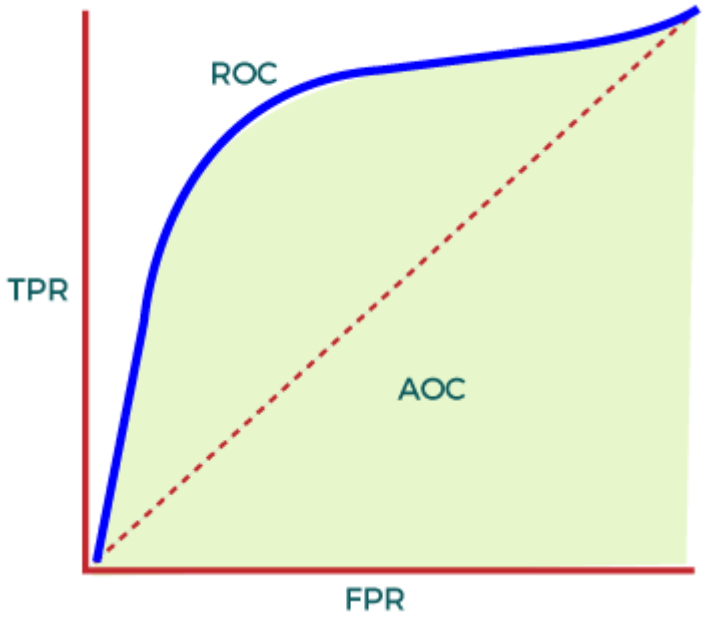


Figure 7.1. Area Under ROC Curve

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

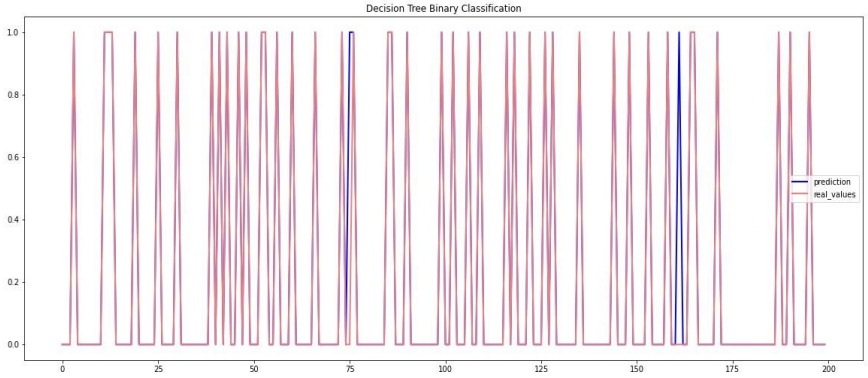
$$specificity = \frac{TN}{TN + FP}$$

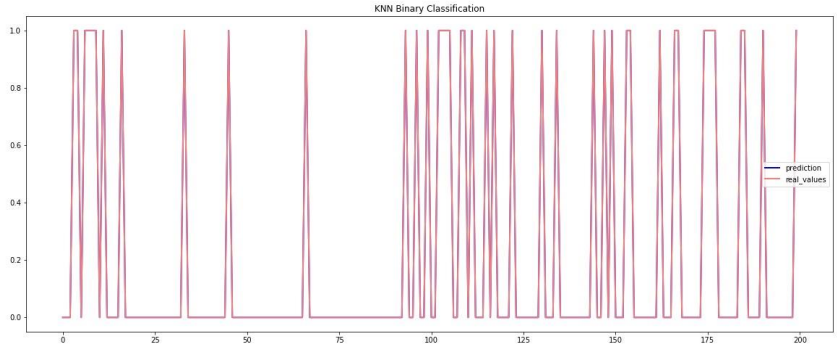
Table 7.2. Evaluation Metrics

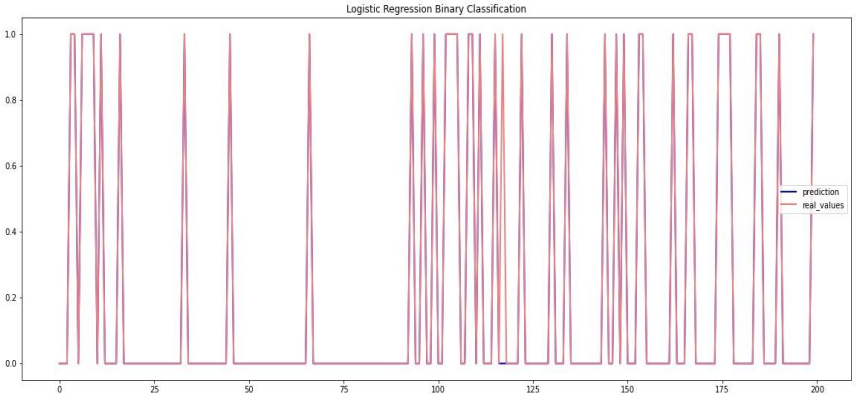
S.NO	MODEL AND CLASSIFIERS	ACCURACY
1.	Decision Tree Classifier	0.9933
2.	Random Forest Classifier	0.9834
3.	Support Vector Classifier	0.9529
4.	K Nearest Neighbor Classifier	0.9695
5.	Gaussian Naive Bayes Classifier	0.8729
6.	Logistic Regression Classifier	0.9504
7.	Voting Classifier	0.9578
8.	Recurrent Neural Network	0.9876
9.	Convolutional Neural Network	0.9976

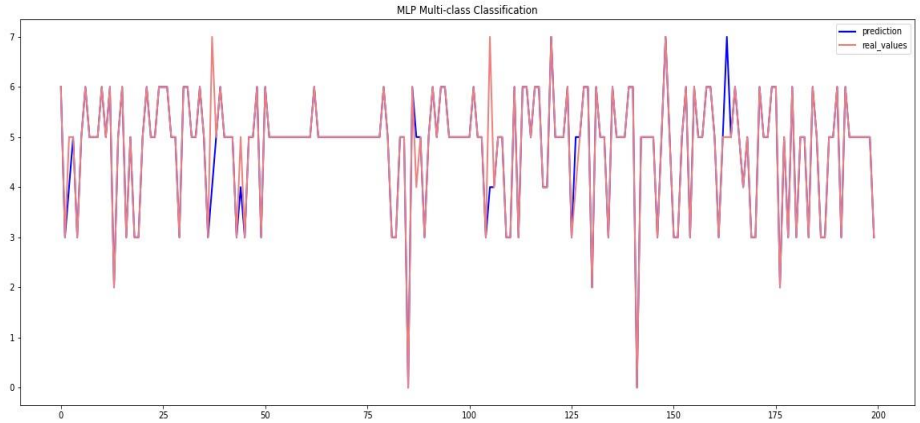
Table 7.3. Evaluation Results of our Trained Models

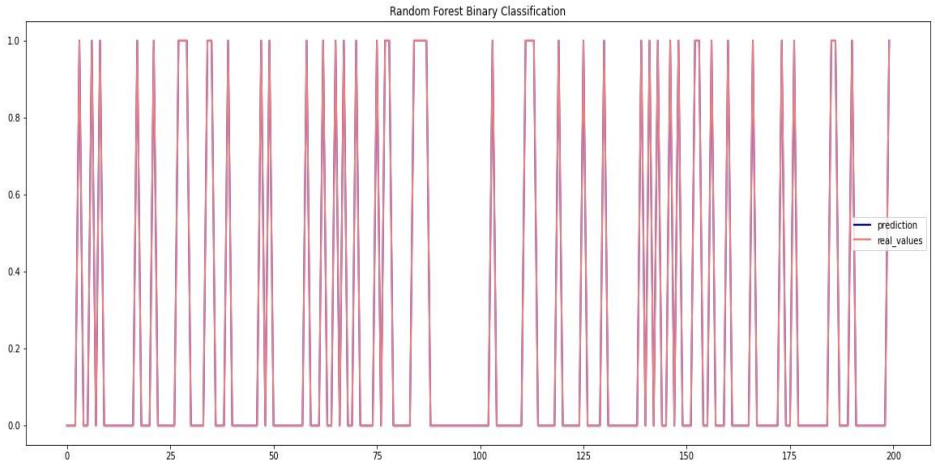
Figures 7.4. Visualization of classifiers








MLP Multi-class Classification
prediction
real_values
0
1
2
3
4
5
6
7
0
25
50
75
100
125
150
175
200


Random Forest Binary Classification
prediction
real_values
0.0
0.2
0.4
0.6
0.8
1.0
0
25
50
75
100
125
150
175
200

8. CONCLUSION AND FUTURE SCOPE

CONCLUSION

The primary goal is to detect attacks and malicious activities within a network while reducing false positives. The IDS output would be accurate, advanced, and reliable if machine learning algorithms were used. This system also displays the accuracy rate of attacks detected by the various machine learning algorithms that have been implemented. The increasing use of technology has resulted in such a massive amounts of data that must be processed and securely stored for users. Security is an important consideration for any user. In conclusion, the hybrid intrusion detection system using ML classifiers is a promising approach to enhance the security of computer networks. The system combines the strengths of different ML classifiers to improve the detection rate of both known and unknown attacks. It also reduces false positives and provides a high level of accuracy in identifying malicious activities in real-time.The implementation of the hybrid intrusion detection system demonstrated the effectiveness of the approach in detecting various types of attacks on the network. The system showed a significant improvement in detecting network anomalies and intrusion attempts when compared to traditional IDS systems.

FUTURE SCOPE

In terms of future scope, the hybrid intrusion detection system using ML classifiers can be further enhanced by integrating with other security tools, such as firewalls and antivirus software, to provide a comprehensive security solution. The system can also be extended to detect more sophisticated attacks, such as zero-day attacks, by leveraging advanced ML techniques, such as deep learning and neural networks.

Overall, the hybrid intrusion detection system using ML classifiers is a valuable contribution to the field of network security. It provides a reliable and effective approach to detect and mitigate various types of cyber threats, which is essential in today's highly interconnected and vulnerable digital world.

REFERENCES

A. Journals/Articles

[1] Ruizhe Zhao , Yingxue Mu , Long Zou , And Xiumei Wen , "A Hybrid Intrusion Detection System Based on Feature Selection and Weighted Stacking Classifier", *IEEE,* **Vol 10, Issue 2, 2022.**

[2] A B. Athira, V. Pathari, "Standardisation and Classification of Alerts Generated by Intrusion Detection Systems", *IJCI, International Journal on Cybernetics & Informatics***, Vol 5 Issue 2, 2016.**

[3] S, Vijayarani, and Maria Sylviaa S. "Intrusion Detection System – *A Study", IJSPTM, International Journal of Security, Privacy and Trust Management* ,**Vol 4, Issue 1, pp. 31–44, February 2015.**

[4] Singh Deepak Kumar, Gupta Jitendra Kumar, "An approach for Anomaly based Intrusion detection System using SNORT*", IJSER, International Journal of Scientific & Engineering Research,* **Volume 4, Issue 9, September 2013.**

[5] Chakraborty Nilotpal, "Intrusion Detection System and Intrusion Prevention System: A Comparative Study*", IJCBR, International Journal of Computing and Business Research ,* **Volume 4 Issue 2, May 2013.**

[6] Kumar Vinod, Sangwan Prakash Om, "Signature Based Intrusion Detection System Using SNORT", *IJCAIT, International Journal of Computer Applications & Information Technology*, **Vol. I, Issue III, November 2012.**

B. Books

- **"**Instant OSSEC Host-Based Intrusion Detection System" by Brad Lhotsky.
- "CCSP Self-Study: Cisco Secure Intrusion Detection System (CSIDS) (Certification Self-Study Series)" by Earl Carter and Cisco Systems Inc
- "Intrusion Detection with Snort" by Jack Koziol

C. E- Websites/ downloads

https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids

https://en.wikipedia.org/wiki/Intrusion_detection_system

https://www.knowledgehut.com/blog/security/intrusion-detection-system

ANNEXURE – A

Installations

Python 3 Installation on Windows

Step 1: Select Version of Python to Install

The installation procedure involves downloading the official Python .exe installer and running it on your system. The version you need depends on what you want to do in Python. For example, if youare working on a project coded in Python version 2.6, you probably need that version.

Step 2: Download Python Executable Installer

- Open your web browser and navigate to the Downloads for Windows section ofthe official Python website.
- Search for your desired version of Python. At the time of publishing this article, the latest Python 3 release is version 3.7.3, while the latest Python 2 release is version 2.7.16.

Step 3: Run Executable Installer

- Run the **Python Installer** once downloaded.
- Make sure you select the **Install launcher for all users** and **Add Python 3.7 to PATH** checkboxes. The latter places the interpreter in the execution path. For older versionsof Python that do not support the **Add Python to Path** checkbox.
- Select **Install Now** – the recommended installation options.

For all recent versions of Python, the recommended installation options include **Pip** and **IDLE**. Older versions might not include such additional feature.

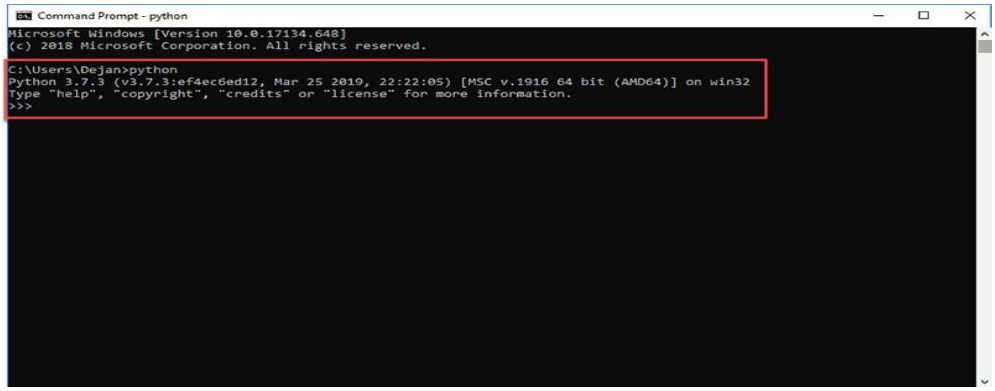
The next dialog will prompt you to select whether to **Disable path length limit**. Choosing this option will allow Python to bypass the 260-character MAX_PATH limit. Effectively, it will enablePython to use long path names.

Step 4: Verify Python Was Installed On Windows

Navigate to the directory in which Python was installed on the system. In our case, itis **C:\Users*Username*\AppData\Local\Programs\Python\Python37** since we have installed the latest version.

Double-click **python.exe**.

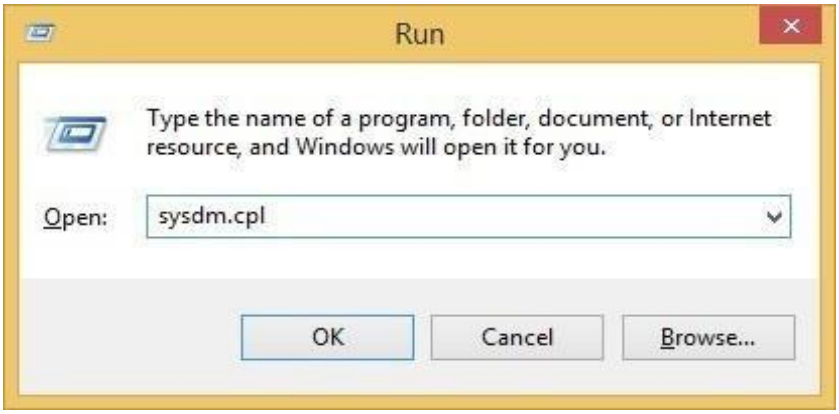
The output should be similar to what you can see below:



4.4.1.SCREEN Command prompt

Step 5: Verify Pip Was Installed

- If you opted to install an older version of Python, it is possible that it did not come with Pip preinstalled. Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.
- We recommend using Pip for most Python packages, especially when working in virtual environments.
- To verify whether Pip was installed:
- Open the **Start** menu and type "**cmd.**"
- Select the **Command Prompt** application.
- Enter **pip -V** in the console. If Pip was installed successfully, you should see the following



4.4.2. SCREEN Run terminal

.

4.4.3. SCREEN Setting Path

- Type **sysdm.cpl** and click **OK**. This opens the **System Properties** window.
- Navigate to the **Advanced** tab and select **Environment Variables**.
- Under **System Variables**, find and select the **Path** variable.
- Click **Edit**.
- Select the **Variable value** field. Add the path to the **python.exe** file preceded witha **semicolon (;)**. For example, in the image below, we have added "**;C:\Python34.**
- Click **OK** and close all windows.
- By setting this up, you can execute Python scripts like this: **Python script.py**
- Instead of this: **C:/Python34/Python script.py**
- As you can see, it is cleaner and more manageable.

ANNEXURE- B

Detail Description of features of dataset

- **5.1.1.1.1. Table 1-4: Basic features of network connections**

Feature Name	Description
fl_dur	Flow duration
tot_fw_pk	Total packets in the forward direction
tot_bw_pk	Total packets in the backward direction
tot_l_fw_pkt	Total size of packet in forward direction

- **5.1.1.1.2. Table 5-16 : Features of network packets**

fw_pkt_l_max	Maximum size of packet in forward direction
fw_pkt_l_min	Minimum size of packet in forward direction
fw_pkt_l_avg	Average size of packet in forward direction
fw_pkt_l_std	Standard deviation size of packet in forward direction
Bw_pkt_l_max	Maximum size of packet in backward direction
Bw_pkt_l_min	Minimum size of packet in backward direction
Bw_pkt_l_avg	Mean size of packet in backward direction
Bw_pkt_l_std	Standard deviation size of packet in backward direction
fl_byt_s	flow byte rate that is number of packets transferred per second
fl_pkt_s	flow packets rate that is number of packets transferred per second
fl_iat_avg	Average time between two flows
fl_iat_std	Standard deviation time two flows

- **5.1.1.1.3. Table 17-22: Features of network flows**

fl_iat_max	Maximum time between two flows
fl_iat_min	Minimum time between two flows
fw_iat_tot	Total time between two packets sent in the forward direction
fw_iat_avg	Mean time between two packets sent in the forward direction
fw_iat_std	Standard deviation time between two packets sent in the forward direction
fw_iat_max	Maximum time between two packets sent in the forward direction

- **5.1.1.1.4. Table 23-45: statistic of network flows**

fw_iat_min	Minimum time between two packets sent in the forward direction
bw_iat_tot	Total time between two packets sent in the backward direction
bw_iat_avg	Mean time between two packets sent in the backward direction
bw_iat_std	Standard deviation time between two packets sent in the backward direction
bw_iat_max	Maximum time between two packets sent in the backward direction
bw_iat_min	Minimum time between two packets sent in the backward direction
fw_psh_flag	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
bw_psh_flag	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
fw_urg_flag	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
bw_urg_flag	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
fw_hdr_len	Total bytes used for headers in the forward direction
bw_hdr_len	Total bytes used for headers in the forward direction
fw_pkt_s	Number of forward packets per second
bw_pkt_s	Number of backward packets per second
pkt_len_min	Minimum length of a flow
pkt_len_max	Maximum length of a flow

pkt_len_avg	Mean length of a flow

- **5.1.1.1.5. Table 46-63 : Content related features**

fin_cnt	Number of packets with FIN
syn_cnt	Number of packets with SYN
rst_cnt	Number of packets with RST
pst_cnt	Number of packets with PUSH
ack_cnt	Number of packets with ACK
urg_cnt	Number of packets with URG
cwe_cnt	Number of packets with CWE
ece_cnt	Number of packets with ECE
down_up_ratio	Download and upload ratio
pkt_size_avg	Average size of packet
fw_seg_avg	Average size observed in the forward direction
bw_seg_avg	Average size observed in the backward direction
fw_byt_blk_avg	Average number of bytes bulk rate in the forward direction
fw_pkt_blk_avg	Average number of packets bulk rate in the forward direction
fw_blk_rate_avg	Average number of bulk rate in the forward direction
bw_byt_blk_avg	Average number of bytes bulk rate in the backward direction
bw_pkt_blk_avg	Average number of packets bulk rate in the backward direction
bw_blk_rate_avg	Average number of bulk rate in the backward direction

- **5.1.1.1.6. Table 64-67: Features of network subflows**

subfl_fw_pk	The average number of packets in a sub flow in the forward direction
subfl_fw_byt	The average number of bytes in a sub flow in the forward direction
subfl_bw_pkt	The average number of packets in a sub flow in the backward direction
subfl_bw_byt	The average number of bytes in a sub flow in the backward direction

- **5.1.1.1.7. Table 68-79 : General purpose traffic features**

fw_win_byt	Number of bytes sent in initial window in the forward direction
bw_win_byt	# of bytes sent in initial window in the backward direction
Fw_act_pkt	# of packets with at least 1 byte of TCP data payload in the forward direction
fw_seg_min	Minimum segment size observed in the forward direction
atv_avg	Mean time a flow was active before becoming idle
atv_std	Standard deviation time a flow was active before becoming idle
atv_max	Maximum time a flow was active before becoming idle
atv_min	Minimum time a flow was active before becoming idle
idl_avg	Mean time a flow was idle before becoming active

- **5.1.1.1.8. Table 80-83 : Basic features of network connections**

idl_std	Standard deviation time a flow was idle before becoming active
idl_max	Maximum time a flow was idle before becoming active
idl_min	Minimum time a flow was idle before becoming active

ANNEXURE- C

CODE SNIPPETS

Data Preprocessing and Exploratory Data Analysis

```
[ ] df = pd.read_csv("CSE-CIC-IDS2018.csv")
    df
```

	0	491	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	25	0.17.1	0.03	0.17.2	0.24	0.25	0.26	0.05	0.27	1.1
0	0	146	0	0	0	0	0	0	0	0	...	1	0.00	0.60	0.88	0.00	0.00	0.00	0.00	0.00	1
1	0	0	0	0	0	0	0	0	0	0	...	26	0.10	0.05	0.00	0.00	1.00	1.00	0.00	0.00	2
2	0	232	8153	0	0	0	0	0	1	0	...	255	1.00	0.00	0.03	0.04	0.03	0.01	0.00	0.01	1
3	0	199	420	0	0	0	0	0	1	0	...	255	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1
4	0	0	0	0	0	0	0	0	0	0	...	19	0.07	0.07	0.00	0.00	0.00	0.00	1.00	1.00	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
125967	0	0	0	0	0	0	0	0	0	0	...	25	0.10	0.06	0.00	0.00	1.00	1.00	0.00	0.00	2
125968	8	105	145	0	0	0	0	0	0	0	...	244	0.96	0.01	0.01	0.00	0.00	0.00	0.00	0.00	1
125969	0	2231	384	0	0	0	0	0	1	0	...	30	0.12	0.06	0.00	0.00	0.72	0.00	0.01	0.00	1
125970	0	0	0	0	0	0	0	0	0	0	...	8	0.03	0.05	0.00	0.00	1.00	1.00	0.00	0.00	2

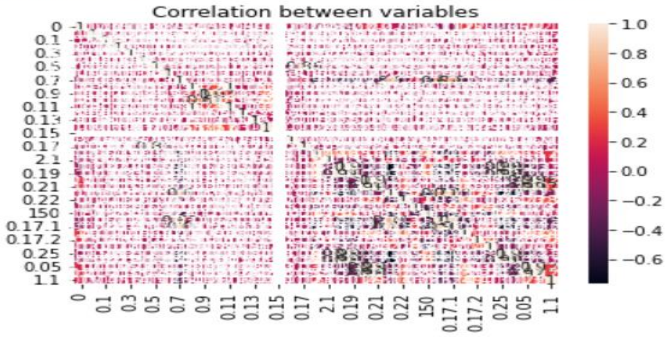
```
train_data = df.iloc[:, 0:] #reading the csv files using pandas
test_data = df.iloc[:,1:]
train_data.head()
```

	0	491	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	25	0.17.1	0.03	0.17.2	0.24	0.25	0.26	0.05	0.27	1.1
0	0	146	0	0	0	0	0	0	0	0	...	1	0.00	0.60	0.88	0.00	0.00	0.00	0.0	0.00	1
1	0	0	0	0	0	0	0	0	0	0	...	26	0.10	0.05	0.00	0.00	1.00	1.00	0.0	0.00	2
2	0	232	8153	0	0	0	0	0	1	0	...	255	1.00	0.00	0.03	0.04	0.03	0.01	0.0	0.01	1
3	0	199	420	0	0	0	0	0	1	0	...	255	1.00	0.00	0.00	0.00	0.00	0.00	0.0	0.00	1
4	0	0	0	0	0	0	0	0	0	0	...	19	0.07	0.07	0.00	0.00	0.00	0.00	1.0	1.00	2

5 rows × 39 columns

```
# correlation matrix and heatmap
sns.heatmap(df.corr(), annot=True)
plt.title('Correlation between variables')
```

```
Text(0.5, 1.0, 'Correlation between variables')
```




```
print("LOADING TRAINING AND TESTING DATA")
```

```
LOADING TRAINING AND TESTING DATA
```

```
column_name = pd.read_csv("Field Names.csv", header = None)
print(column_name)
```

```
                         0           1
0                 duration  continuous
1            protocol_type    symbolic
2                  service    symbolic
3                     flag    symbolic
4                src_bytes  continuous
5                dst_bytes  continuous
6                     land  continuous
7            wrong_fragment  continuous
8                   urgent  continuous
9                      hot  continuous
10       num_failed_logins  continuous
11              logged_in  continuous
12          num_compromised  continuous
13            root_shell  continuous
14         su_attempted  continuous
15             num_root  continuous
```

```
new_columns = list(column_name[0].values)
```

```
new_columns += ['class', 'difficulty']
```

```
train_data = pd.read_csv('CSE-CIC-IDS2018Train+.txt', names = new_columns)
print(train_data)
```

```
        duration protocol_type   service flag  src_bytes  dst_bytes  land  \
0              0           tcp  ftp_data   SF        491          0     0
1              0           udp     other   SF        146          0     0
2              0           tcp   private   S0          0          0     0
3              0           tcp      http   SF        232       8153     0
4              0           tcp      http   SF        199        420     0
...          ...           ...       ...  ...        ...        ...   ...
125968         0           tcp   private   S0          0          0     0
125969         8           udp   private   SF        105        145     0
125970         0           tcp      smtp   SF       2231        384     0
125971         0           tcp    klogin   S0          0          0     0
125972         0           tcp  ftp_data   SF        151          0     0
```

```
test_data = pd.read_csv('CSE-CIC-IDS2018Test+.txt', names = new_columns)
print(test_data)
```

```
       duration protocol_type   service  flag  src_bytes  dst_bytes  land  \
0             0           tcp   private   REJ          0          0     0
1             0           tcp   private   REJ          0          0     0
2             2           tcp  ftp_data    SF      12983          0     0
3             0          icmp     eco_i    SF         20          0     0
4             1           tcp    telnet  RSTO          0         15     0
...         ...           ...       ...   ...        ...        ...   ...
22539         0           tcp      smtp    SF        794        333     0
22540         0           tcp      http    SF        317        938     0
22541         0           tcp      http    SF      54540       8314     0
22542         0           udp  domain_u    SF         42         42     0
22543         0           tcp    sunrpc   REJ          0          0     0
```

```
[ ] train_data.tail()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same
125968	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.06	
125969	8	udp	private	SF	105	145	0	0	0	0	...	0.96	0.01	
125970	0	tcp	smtp	SF	2231	384	0	0	0	0	...	0.12	0.06	
125971	0	tcp	klogin	S0	0	0	0	0	0	0	...	0.03	0.05	
125972	0	tcp	ftp_data	SF	151	0	0	0	0	0	...	0.30	0.03	

5 rows × 43 columns

```
[ ] print("The testing data is")
    test_data.head()
```

The testing data is

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src
0	0	tcp	private	REJ	0	0	0	0	0	0	...	0.04	0.06	
1	0	tcp	private	REJ	0	0	0	0	0	0	...	0.00	0.06	
2	2	tcp	ftp_data	SF	12983	0	0	0	0	0	...	0.61	0.04	
3	0	icmp	eco_i	SF	20	0	0	0	0	0	...	1.00	0.00	
4	1	tcp	telnet	RSTO	0	15	0	0	0	0	...	0.31	0.17	

5 rows × 43 columns

```
[ ] map_attacks = [x.strip().split() for x in open('attacks.txt', 'r')]
    map_attacks = {k:v for (k,v) in map_attacks}
```

```
[ ] train_data['class'] = train_data['class'].replace(map_attacks)
    test_data['class'] = test_data['class'].replace(map_attacks)
    print(test_data['class'])
```

```
0           dos
1           dos
2        normal
3         saint
4         mscan
          ...
22539    normal
22540    normal
22541       dos
22542    normal
22543     mscan
Name: class, Length: 22544, dtype: object
```

```
[ ] train_data = shuffle(train_data)
```

```
print(train_data)
```

```
        duration protocol_type   service flag  src_bytes  dst_bytes  land  \
75163          0           tcp      http   SF        212       4132     0
108054         0           tcp      http   SF        180        614     0
30248          0           tcp      http   SF        309       1284     0
86018          0           tcp   private   S0          0          0     0
63590          2           tcp  ftp_data   SF    2194619          0     0
...          ...           ...       ...  ...        ...        ...   ...
112461         0           tcp      http   SF        234       2239     0
12866          0           tcp      http   SF        323        188     0
5207           0           tcp      smtp   SF       2226        327     0
92189          0           tcp  ftp_data   S0          0          0     0
3179           0           udp  domain_u   SF         46         78     0

        wrong_fragment  urgent  hot  ...  dst_host_same_srv_rate  \
75163                0       0    0  ...                    1.00
108054               0       0    0  ...                    1.00
30248                0       0    0  ...                    1.00
86018                0       0    0  ...                    0.03
63590                0       0    0  ...                    0.32
```

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler('majority')
X_rus, y_rus = rus.fit_resample(X, y)
y_rus.value_counts()
df = X_rus
df[' Label'] = y_rus
minor = pd.DataFrame(df[(df[' Label']!=4) & (df[' Label']!=2)])
major = pd.DataFrame(df[(df[' Label']==4) | (df[' Label']==2)])
minor[' Label'].value_counts()
```

```
1    1966
5    1507
7     652
3      36
0      21
6      21
Name:  Label, dtype: int64
```

```
from imblearn.over_sampling import SMOTE
y_rus_ =  minor[' Label']
X_rus_ =  minor.drop([' Label'],axis=1)
strategy = {1:2000, 5:1600, 7:800, 3:300, 6:200, 0:200}
sm = SMOTE(sampling_strategy=strategy)
X_sm, y_sm = sm.fit_resample(X_rus_, y_rus_)
X_min,y_min = X_sm, y_sm


major[' Label'].value_counts()
```

```
4    158930
2    128027
Name:  Label, dtype: int64
```

```
from imblearn.under_sampling import RandomUnderSampler
y_rus_ =  major[' Label']
X_rus_ =  major.drop([' Label'],axis=1)
strategy = {4:10000, 2:6000}
tom = RandomUnderSampler(sampling_strategy=strategy)
X_tom, y_tom = tom.fit_resample(X_rus_, y_rus_)
print(y_tom.value_counts())
```

```
4    10000
2     6000
Name:  Label, dtype: int64
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
cols = X.select_dtypes(include=['float32','float16','int32','int16','int8']).columns
train_X = scaler.fit_transform(X.select_dtypes(include=['float32','float16','int32','int16','int8']))
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(train_X,y,train_size=0.70, random_state=2)
```

```
print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

```
Training set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 70 categories
Feature 'flag' has 11 categories
Feature 'label' has 23 categories
```

```
print('Distribution of categories in service:')
print(df['service'].value_counts().sort_values(ascending=False).head())
# Test set
print('Test set:')
for col_name in df_test.columns:
    if df_test[col_name].dtypes == 'object' :
        unique_cat = len(df_test[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))
```

```
Distribution of categories in service:
http        40338
private     21853
domain_u     9043
smtp         7313
ftp_data     6860
Name: service, dtype: int64
Test set:
Feature 'protocol_type' has 3 categories
Feature 'service' has 64 categories
Feature 'flag' has 11 categories
Feature 'label' has 38 categories
```

```
unique_flag=sorted(df.flag.unique())
string3 = 'flag_'
unique_flag2=[string3 + x for x in unique_flag]
print(unique_flag2)
```

```
['flag_OTH', 'flag_REJ', 'flag_RSTO', 'flag_RSTOS0', 'flag_RSTR', 'flag_S0', 'flag_S1', 'flag_S2', 'flag_S3', 'flag_SF', 'flag_SH']
```

```
dumcols=unique_protocol2 + unique_service2 + unique_flag2
```

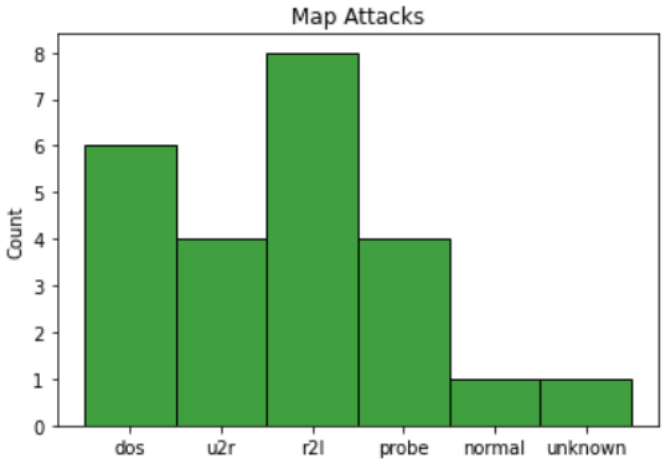
```
df_categorical_values_enc=df_categorical_values.apply(LabelEncoder().fit_transform)
print(df_categorical_values.head())
print('--------------------')
print(df_categorical_values_enc.head())
```

```
  protocol_type   service flag
0           tcp  ftp_data   SF
1           udp     other   SF
2           tcp   private   S0
3           tcp      http   SF
4           tcp      http   SF
--------------------
   protocol_type  service  flag
0              1       20     9
1              2       44     9
2              1       49     5
3              1       24     9
4              1       24     9
```

```
enc = OneHotEncoder(categories='auto')
df_categorical_values_encenc = enc.fit_transform(df_categorical_values_enc)
df_cat_data = pd.DataFrame(df_categorical_values_encenc.toarray(),columns=dumcols)
```

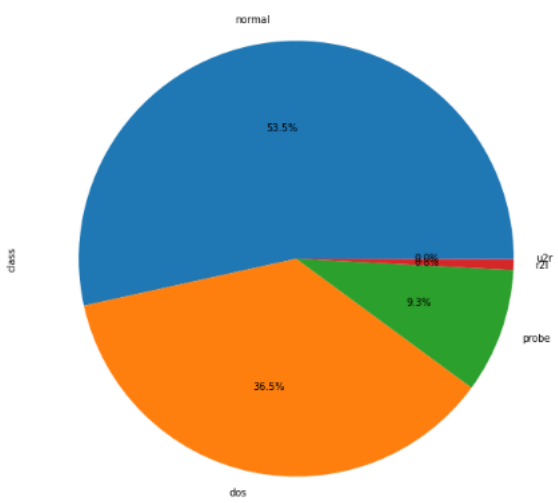
```
[ ]  sns.histplot(x=map_attacks, color='g')
     plt.title('Map Attacks')
```

```
Text(0.5, 1.0, 'Map Attacks')
```

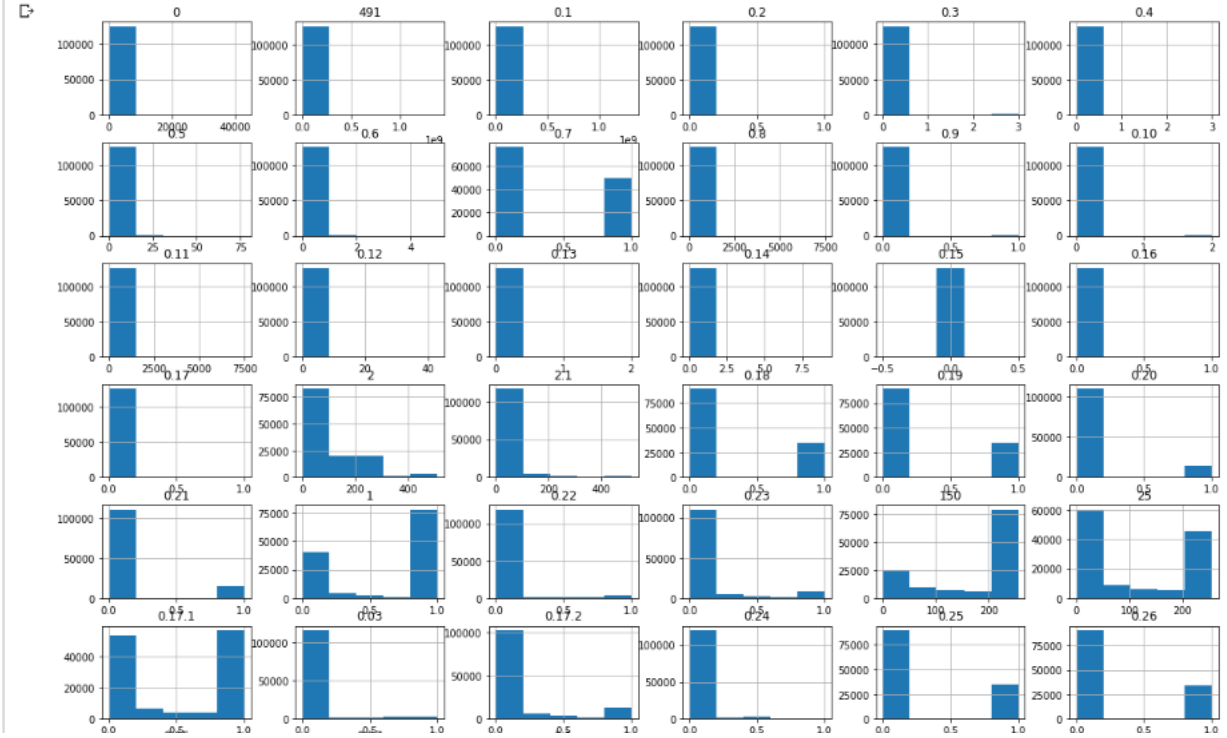



```
plt.figure(figsize=(10,10))
train_data['class'].value_counts().plot.pie(autopct="%1.1f%%")
plt.title('Different category of attacks')
```

```
Text(0.5, 1.0, 'Different category of attacks')
```

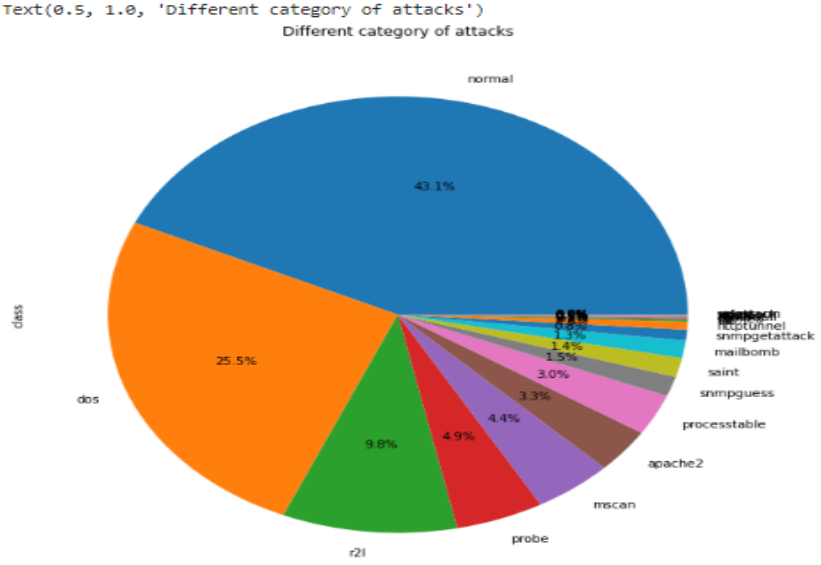


```
df.hist(bins=5,figsize=(20,15))
plt.show()
```




```
plt.figure(figsize=(10,10))
test_data['class'].value_counts().plot.pie(autopct="%1.1f%%")
plt.title('Different category of attacks')
```

```
Text(0.5, 1.0, 'Different category of attacks')
```



Working on Training the dataset using various Classifiers

```
[ ]  from sklearn import tree
     from sklearn.model_selection import cross_val_score
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.svm import SVC
     from sklearn.naive_bayes import BernoulliNB
     from sklearn.model_selection import cross_val_score
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn.neural_network import MLPClassifier
```

```
[ ]  RFC_Classifier = RandomForestClassifier(max_depth=40)
     RFC_Classifier.fit(X_train, Y_train)
```

```
     RandomForestClassifier(max_depth=40)
```

```
     SVM_Classifier = SVC()
     SVM_Classifier.fit(X_train, Y_train)
```

```
     SVC()
```

```
[ ] DTC_Classifier = tree.DecisionTreeClassifier(criterion='gini', max_depth=33, random_state=20, max_features=12, splitter='random')
    DTC_Classifier.fit(X_train, Y_train)
```

```
    DecisionTreeClassifier(max_depth=33, max_features=12, random_state=20,
                           splitter='random')
```

```
[ ] KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
    KNN_Classifier.fit(X_train, Y_train)
```

```
    KNeighborsClassifier(n_jobs=-1)
```

```
[ ]  LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0, max_iter=5000)
     LGR_Classifier.fit(X_train, Y_train)
```

```
     LogisticRegression(max_iter=5000, n_jobs=-1, random_state=0)
```

```
     BNB_Classifier = BernoulliNB()
     BNB_Classifier.fit(X_train, Y_train)
```

```
     BernoulliNB()
```

```
[ ]  mlp_multi = MLPClassifier(random_state=123, solver='adam', max_iter=8000)
     mlp_multi.fit(X_train,Y_train)


     y_pred = mlp_multi.predict(X_test)
```

```
[ ]  y_pred = mlp_multi.predict(X_test)
```

```
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
import time

models = []
models.append(('Random Forest Classifier', RFC_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
models.append(('Support Vector Classifier',SVM_Classifier))
models.append(('KNeighborsClassifier',KNN_Classifier))
models.append(('LogisticRegression',LGR_Classifier))
models.append(('Gaussian Naive Baye',BNB_Classifier))
models.append(('MLP Classifier',mlp_multi))
```

```
for i, v in models:
    Xpred =  v.predict(X_train)
    scores = cross_val_score(v, X_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, Xpred)
    confusion_matrix = metrics.confusion_matrix(Y_train, Xpred)
    cmd = metrics.ConfusionMatrixDisplay(confusion_matrix, display_labels=attackType)
    classification = metrics.classification_report(Y_train, Xpred)
    print()
    print('============================== {} Model Evaluation =============================='.format(i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
    cmd.plot(cmap='Blues', include_values=True)
    time.sleep(1)
```

```
from sklearn.ensemble import VotingClassifier

clf1 = tree.DecisionTreeClassifier(criterion='gini', max_depth=33, random_state=20, max_features=12, splitter='random')
clf2 = RandomForestClassifier(criterion='gini', max_depth=40, random_state=20)
clf3 = SVC()
clf4 = KNeighborsClassifier(n_jobs=-1)
clf5 = LogisticRegression(n_jobs=-1, random_state=0, max_iter=5000)
clf6 = BernoulliNB()

votingC = VotingClassifier(estimators=[('dc',clf1), ('rf', clf2),('svc',clf3),('knn',clf4),('lgr',clf5),('bnb',clf6)],voting='hard', weights=[2,2,1,2,1,1],
                           flatten_transform=True)
votingC.fit(X_train,Y_train)


pred = votingC.predict(X_test)
accuracy = metrics.accuracy_score(Y_test,pred)
confusion_matrix = metrics.confusion_matrix(Y_test, pred)
cmd = metrics.ConfusionMatrixDisplay(confusion_matrix, display_labels=attackType)
classification = metrics.classification_report(Y_test, pred)
print()
print('========================== {} Model Test Results ==========================='.format('Voting Classifier'))
print()
print ("Model Accuracy:" "\n", accuracy)
print()
print("Confusion matrix:" "\n", confusion_matrix)
print()
print("Classification report:" "\n", classification)
print()
cmd.plot(cmap='Blues', include_values=True)
time.sleep(1)
```

Working on training the CNN Model

CNN.py

```
import warnings
import pandas as pd
import numpy as np
import sklearn
from sklearn.utils import shuffle
from sklearn.metrics import *
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import warnings

from keras.preprocessing import sequence
from tensorflow.keras import optimizers
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Embedding, SimpleRNN, BatchNormalization
from keras.models import model_from_json

warnings.filterwarnings("ignore")
print("LOADING TRAINING AND TESTING DATA")
#load the csv file containing the column names
column_name = pd.read_csv("Field Names.csv", header = None)
#Convert the array into list
new_columns = list(column_name[0].values)
#adding difficulty
new_columns += ['class', 'difficulty']
#loading train and test data files
train_data = pd.read_csv('CSE-CIC-IDS2017Train+.txt', names = new_columns)
test_data = pd.read_csv('CSE-CIC-IDS2017Test+.txt', names = new_columns)
#Training data sample
print("The training data is")
train_data.tail()
#Output total rows and columns of dataframe
print(f"The shape of the training dataframe is : {train_data.shape}")
#Same for testing
print("The testing data is")
test_data.head()
print(f"The shape of the testing dataframe is : {test_data.shape}")
#Load attacks.txt containing the attack categories
map_attacks = [x.strip().split() for x in open('attacks.txt', 'r')]
```

```
map_attacks = {k:v for (k,v) in map_attacks}

#Replace the "class" column values to 5 attack categories in training and testing dataframe
train_data['class'] = train_data['class'].replace(map_attacks)
test_data['class'] = test_data['class'].replace(map_attacks)
train_data = shuffle(train_data)

print("DATA PREPROCESSING")
y_new = train_data['class']
y_new = pd.get_dummies(y_new)
#Split data: 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X_new, y_new, test_size = 0.2, random_state = 101)
#Use StandardScaler() to standardize data - explained in Honours Project
sc = StandardScaler()
sc.fit(np.array(X_train))
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
#Use the keras's sequential API
#First dense layer takes an input parameter as 256 (number of neurons in the first layer).
#The second parameter "input_dim" corresponds to the input features.
#Use "relu" as activation function.
#The activation function for last dense layer is "softmax" because of the multiple classes, further explained in document.
#Set dropout for 10%.
model1 = Sequential()
model1.add(Dense(64, input_dim = 120, activation = "relu", kernel_initializer = "lecun_normal"))
model1.add(Dense(128, activation = "relu"))
model1.add(Dense(5, activation = "softmax"))
#Summary of model architecture listing information about parameters per layer.
model1.summary()
#Three paramaters:
#Loss - The loss function.
#Optimizer - To minimize the loss function.
#Metrics - The mode of evaluation for our model.
#"categorical_loss" - is used because of the multi-class classifcation problem.
#"adam" - The updated version of SGD.
optim = optimizers.SGD(lr = 0.0001)
model1.compile(loss = 'categorical_crossentropy', optimizer = optim, metrics = ['accuracy'])
#Fit the model on our data.
#X_train - The feature columns of the training data.
#y_train - The labels columns of the training data.
#validation_data - The validation data
#batch_size and epochs further explained in document.
history = model1.fit(X_train, y_train,
        validation_data = (X_test, y_test),
        batch_size = 32,
        epochs = 20)
#use matplitlib to draw the plots
plt.figure(figsize = (15, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label = "TRAINING LOSS")
```

```
plt.plot(history.history['val_loss'], label = "VALIDATION LOSS")
plt.title("TRAINING LOSS vs VALIDATION LOSS")
plt.xlabel("EPOCH'S")
plt.ylabel("TRAINING LOSS vs VALIDATION LOSS")
plt.legend(loc = "best")

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label = "TRAINING ACCURACY")
plt.plot(history.history['val_accuracy'], label = "VALIDATION ACCURACY")
plt.title("TRAINING ACCURACY vs VALIDATION ACCURACY")
plt.xlabel("EPOCH'S")
plt.ylabel("TRAINING ACC vs VALIDATION ACCURACY")
plt.legend(loc = "best")
#Serialize model 1 , save with json.
model_json = model1.to_json()
with open("model1.json", "w") as json_file:
    json_file.write(model_json)
model1.save_weights('model1_weights.h5')
print("Saved model to disk")
# load model 1.
json_file = open("model1.json", "r")
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights("model1_weights.h5")
print("Loaded model from disk")
```

Working on training the RNN Model

RNN.py

```
# import libraries
import pandas as pd
import numpy as np
import sys
import sklearn
import io
import random
from tensorflow.keras.layers import Dense, Flatten, Dropout,BatchNormalization ,Activation
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import plot_model
from IPython.display import Image
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.pipeline import Pipeline
from keras.layers import SimpleRNN
```

```
## add the columns' name and read the KDDTrain+ and KDDTest+ datasets
col_names = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","label"]

#training set
df = pd.read_csv("CSE-CIC-IDS2018_Train.csv",header=None, names = col_names)
#testing set
df_test = pd.read_csv("CSE-CIC-IDS2018_Test.csv", header=None, names = col_names)

print('Dimensions of the Training set:',df.shape)
print('Dimensions of the Test set:',df_test.shape)
print('Label distribution in the Training set:')
print("*****************************************")
print(df['label'].value_counts())
print('Label distribution in the Test set:')
print("*****************************************")
print(df_test['label'].value_counts())
# columns are categorical, not yet binary: protocol_type (column 2), service (column 3), flag (column 4).

print('Training set:')
for col_name in df.columns:
    if df[col_name].dtypes == 'object' :
        unique_cat = len(df[col_name].unique())
        print("Feature '{col_name}' has {unique_cat} categories".format(col_name=col_name, unique_cat=unique_cat))


from tensorflow import keras
import numpy as np
import datetime
import time
x=pd.DataFrame(x)
x = x.values
sample = x.shape[0]
features = x.shape[1]
#Train: convert 2D to 3D for input RNN
x_train = np.reshape(x,(sample,features,1)) #shape  = (125973, 18, 1)
#Test: convert 2D to 3D for input RNN
x_test=pd.DataFrame(xtest)
x_test = x_test.values
x_test = np.reshape(x_test,(x_test.shape[0],x_test.shape[1],1))
Model = keras.Sequential([
```

```
        keras.layers.LSTM(80,input_shape=(features,x_train.shape[2]),
                          activation='sigmoid',recurrent_activation='hard_sigmoid'),
        keras.layers.Dense(1,activation="softmax")
    ])

Model.compile(optimizer='rmsprop',loss='mse', metrics=['accuracy'])

#Training the model

Model.fit(x_train, y, epochs=10, batch_size= 32)
Model.summary()

# Final evaluation of the model
scores = Model.evaluate(x_test, ytest, verbose=0)
print('/n')
print("Accuracy: %.2f%%" % (scores[1]*100))
# Using tanh and sigmoid as activation functions

Model = keras.Sequential([

        keras.layers.LSTM(80,input_shape=(features,x_train.shape[2]),
                          activation='tanh',recurrent_activation='hard_sigmoid'),
        keras.layers.Dense(1,activation="tanh")
    ])

Model.compile(optimizer='rmsprop',loss='mse', metrics=['accuracy'])

#Training the model
Model.fit(x_train, y, epochs=10, batch_size= 32)
Model.summary()

# Final evaluation of the model
scores = Model.evaluate(x_test, ytest, verbose=0)
print("/n")
print("Accuracy: %.2f%%" % (scores[1]*100))

import tensorflow as tf
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall')
]

import tensorflow.keras.backend as K
def recall(y_true, y_pred):
    y_true = K.ones_like(y_true)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    all_positives = K.sum(K.round(K.clip(y_true, 0, 1)))

    recall = true_positives / (all_positives + K.epsilon())
```

```
    return recall

def precision(y_true, y_pred):
    y_true = K.ones_like(y_true)
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))

    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_score(y_true, y_pred):
    prec = precision(y_true, y_pred)
    rec = recall(y_true, y_pred)
    return 2*((prec*rec)/(prec+rec+K.epsilon()))

from keras import optimizers
model = Sequential()
model.add(SimpleRNN(60,input_shape=(features,x_train.shape[2]), activation='sigmoid'))
model.add(Dense(1))
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['accuracy', 'mae', recall,precision,f1_score])
model.summary()

mcp = ModelCheckpoint('RNN_binary2.h5')

history = model.fit(x_train, y, validation_split=0.33, epochs=15, batch_size= 32,callbacks=[mcp])
```

INTEGRATION OF WEBAPP

IDS.py

```
from flask import Flask, render_template, request, redirect, url_for
import output

app = Flask(__name__)
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0


@app.route("/")
def home():
    return render_template('home.html', val = 0, tab = (0, 0, 0, ""), num = 0)
@app.route("/home")
def hello():
    return render_template('home.html', val = 0, tab = (0, 0, 0, ""), num = 0)
@app.route('/', methods=["POST", "GET"])
def login():
    if request.method == "POST":
```

```
        user = request.form['slid_val']
        model_number = request.form['optradio']

        return redirect(url_for('user', usr=user, mod_number = model_number))
    else:
        return render_template('home.html')

@app.route('/<usr>/<mod_number>', methods=["POST", "GET"])
def user(usr, mod_number = 0):
    tab = output.table(int(usr), int(mod_number))
    return render_template('home.html', val = usr, tab = tab, num = mod_number)
@app.route("/about")
def about():
    return load_model.code()

@app.after_request
def add_header(response):
    # response.cache_control.no_store = True
    response.headers['Cache-Control'] = 'no-store, no-cache, must-revalidate, post-check=0, pre-check=0, max-age=0'
    response.headers['Pragma'] = 'no-cache'
    response.headers['Expires'] = '-1'
    return response

if __name__ == '__main__':
    app.run(debug=True)
```

output.py

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
import pickle
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import dataframe_image as dfi


def table(val, model = 0):

    pkl_filename = "Decision_Tree.pkl"
```

```
    if(model != 0):

        if(model == 1):
            pkl_filename = "Decision_Tree.pkl"
        if(model == 2):
            pkl_filename = "Random_Forest.pkl"
        if(model == 3):
            pkl_filename = "SVC.pkl"
        if(model == 4):
            pkl_filename = "Logistic_Regression.pkl"



    with open(pkl_filename, 'rb') as file:
        clfg = pickle.load(file)

    X_test = np.load('new_data.npy',allow_pickle=True)
    newdf =  pd.read_pickle('newdf.pkl')

    newdf1 = newdf[['duration', 'protocol_type', 'flag', 'src_bytes', 'dst_bytes','land', 'wrong_fragment', 'urgent',
'num_failed_logins', 'logged_in', 'num_compromised', 'root_shell', 'su_attempted', 'num_file_creations', 'is_guest_login'
,'num_shells' , 'num_access_files']]

    newdf2 = newdf[[ 'srv_count', 'count', 'hot',
    'serror_rate',
    'rerror_rate',
    'same_srv_rate',
    'diff_srv_rate',
    'srv_diff_host_rate',
    'dst_host_count',
    'dst_host_srv_count',
    'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate',
    'dst_host_srv_diff_host_rate',
    'Attack Type']]

    p = newdf1[val:val + 1]
   dfi.export(p, 'static/df_styled1.png', max_cols=-1)

    q = newdf2[val:val + 1]
    dfi.export(q, 'static/df_styled2.png', max_cols=-1)



    display = X_test[val]


    y_test_pred = clfg.predict([display])

    return display, y_test_pred, model, pkl_filename
```

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Intusion Detection</title>

   <link rel= "stylesheet" type= "text/css" href= "{{ url_for('static',filename='style.css') }}">
</head>
<body onload="checkbox( {{ tab[2] }}) ">

   <div class='middle'>

   <h1> Instrusion Detection System </h1>


   <p> select a random number for a server request </p>

   <div class="slidecontainer">

      <div class="container">

         <form>

         </form>
        </div>


      <form action="#" method="post" onsubmit="return checkbox( {{ tab[2] }})">

         <h3>Choose which model to run</h3>


           <label class="radio-inline">
             <input type="radio" name="optradio" value = 1 id='1'>Decision Tree
           </label>
           <label class="radio-inline">
             <input type="radio" name="optradio" value = 2 id='2'>Random Forest
           </label>
           <label class="radio-inline">
             <input type="radio" name="optradio" value = 3 id='3'>Support Vector Classifier
           </label>
           <label class="radio-inline">
             <input type="radio" name="optradio" value = 4 id='4'>Logistic Regression
           </label>
```

```
        {% if val != 0 %}
        <p><input type="range" name='slid_val' min="0" max="149999" value="{{ val }}" class="slider"
id="myRange"></p>
        {% else %}
        <p><input type="range" name='slid_val' min="0" max="149999" value="1" class="slider" id="myRange"></p>
        {% endif %}

        <p><input type='submit' value='submit' /></p>
      </form>

      <a href="{{ url_for('home') }}"><button class="btn btn-primary" style="color:rgb(255, 38, 0);text-
align:center;">Reset</button></a>

   </div>

    <p>Value: <span id="demo"></span></p>
    <script src="{{url_for('static', filename='script.js')}}">

    </script>

    <br><br><br><br>
    {% if val != 0 %}
    <h1> The type of Instrusion Predicted by the {{ tab[3] }} model is: {{ tab[1] }} </h1>

    {% endif %}


</body>
</html>
```

ANNEXURE- C

User Manual of the Project

User Manual for Hybrid Intrusion Detection System Using Machine Learning Classifiers

Introduction:

Hybrid Intrusion Detection System (IDS) using Machine Learning (ML) classifiers is a system that combines the strengths of rule-based and ML-based IDS to provide a more comprehensive detection system for cyberattacks. This user manual provides instructions on how to use this system.

System Requirements:

Operating System: Windows

Python 3.6

Required Python packages: Scikit-learn, NumPy, Pandas, Matplotlib, Seaborn,Flask

Installation:

1. Install Python 3.6 from https://www.python.org/downloads/.
2. Install required Python packages by running the following command in the terminal or command prompt: pip install scikit-learn numpy pandas matplotlib
3. Download the Hybrid IDS code from the GitHub repository or obtain it from your administrator.
4. Extract the files from the downloaded ZIP file.

Usage:

1. Open a terminal or command prompt and navigate to the folder containing the extracted files.
2. Run the following command to train the model on the training dataset: python train.py
3. After training is complete, run the following command to test the model on the test dataset: python test.py

4. The output will display the accuracy of the trained model and the confusion matrix.

Configuration:

The configuration file contains the parameters used by the system. These parameters include the file paths for the training and test datasets, the type of ML classifier to be used, and the hyperparameters for the ML classifier. The configuration file is named config.ini and is located in the same folder as the main Python files.

To modify the configuration, open the config.ini file with a text editor and modify the values as required. The parameters are explained below:

- **Data section:** It contains the data present for working on our project i.e the dataset folder.
  - train_path: path to the training dataset file include the path where it is stored
  - test_path: path to the test dataset file include the path where it is stored

- **ML section:**
  - classifier: the type of ML classifier to be used (e.g., Decision Tree, Random Forest, SVM)
  - hyperparameters: the hyperparameters for the chosen ML classifier, specified as a JSON object (e.g., {"max_depth": 5, "n_estimators": 100})

Note: Modifying the configuration file requires knowledge of the ML algorithms and their corresponding hyperparameters.

Troubleshooting:

- If an error occurs during installation or execution, make sure that the system requirements are met and that the required Python packages are installed.
- If the training or testing process takes too long or crashes, consider using a smaller dataset or modifying the hyperparameters in the configuration file.
- If the accuracy of the trained model is low, try using a different ML classifier or modifying the hyperparameters in the configuration file.

Conclusion: The Hybrid Intrusion Detection System using Machine Learning classifiers is a powerful tool for detecting cyberattacks. This user manual provides instructions for using the system, configuring the parameters, and troubleshooting any issues that may arise.