IDS FOR IOT NETWORKS USING PPGO FEATURE SELECTION AND DEEP LEARNING

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

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING - CYBER SECURITY

Submitted by

SUBHASH KRISHNA VEER BUDDHI (20071A6253)

Under the Guidance of

Dr. Spoorthy G

(Assistant Professor, Department of CSE-(CyS,DS) and AI&DS)

DEPARTMENT OF CSE-(CyS,DS) and AI&DS

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade, NBA Accredited for CE, EEE, ME, ECE, CSE, EIE, IT B. Tech Courses, Approved by AICTE, New Delhi, Affiliated to JNTUH, Recognized as "College with Potential for Excellence" by UGC

ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (S.O), Hyderabad – 500 090, TS, India

April 2024

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade, NBA Accredited for CE, EEE, ME, ECE, CSE, EIE, IT B. Tech Courses, Approved by AICTE, New Delhi, Affiliated to JNTUH, Recognized as "College with Potential for Excellence" by UGC, ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet(SO), Hyderabad-500090, TS, India

DEPARTMENT OF CSE- (CyS, DS) and AI&DS

CERTIFICATE

This is to certify that the project report entitled "**IDS FOR IOT NETWORKS USING PPGO FEATURE SELECTION AND DEEP LEARNING**" is bonafide work done under our supervision and is being submitted by **Mr. Subhash Krishna Veer Buddhi (20071A6253),** in partial fulfilment for the award of the degree of **Bachelor of Technology** in COMPUTER SCIENCE AND ENGINEERING - CYBER SECURITY **Department of CSE-(CyS, DS) and AI&DS**, of the VNRVJIET, Hyderabad during the academic year 2023-2024.

Certified further that to the best of our knowledge, the work presented in this thesis has not been submitted to any other University or Institute for the award of any Degree or Diploma.

Dr. Spoorthy G
Assistant Professor
Department of CSE-(CyS,DS) and AI&DS
VNR VJIET

Dr. M Raja Sekar
Professor & Head
Department of CSE-(CyS,DS) and AI&DS
VNR VJIET

VALLURUPALLI NAGESWARA RAO VIGNANA JYOTHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

An Autonomous Institute, NAAC Accredited with 'A++' Grade, NBA Accredited for CE, EEE, ME, ECE, CSE, EIE, IT B. Tech Courses, Approved by AICTE, New Delhi, Affiliated to JNTUH, Recognized as "College with Potential for Excellence" by UGC, ISO 9001:2015 Certified, QS I GUAGE Diamond Rated

Vignana Jyothi Nagar, Pragathi Nagar, Nizampet (SO), Hyderabad-500090, TS, India

DEPARTMENT OF CSE- (CyS, DS) and AI&DS

DECLARATION

We declare that the major project work entitled "**IDS FOR IOT NETWORKS USING PPGO FEATURE SELECTION AND DEEP LEARNING**" submitted in the department of CSE- (CyS, DS) and AI&DS Vallurupalli Nageswara Rao Vignana Jyothi Institute of Engineering and Technology, Hyderabad, in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING - CYBER SECURITY** is a bonafide record of our work carried out under the supervision of **Dr. Spoorthy G, Assistant Professor, Department of CSE-(CyS, DS) and AI&DS, VNRVJIET.** Also, we declare that the matter embodied in this thesis has not been submitted by us in full or in any part thereof for the award of any degree/diploma of any other institution or university previously.

Place: Hyderabad.

Subhash Krishna Veer Buddhi

(20071A6253)

ACKNOWLEDGEMENT

Firstly, we would love to express our substantial gratitude toward our institution VNR Vignana Jyothi Institute of Engineering and Technology, which created a notable platform to attain profound technical skills within the field of Computer Science, thereby pleasurable our most cherished goal.

We are very much thankful to our Principal, **Dr. Challa Dhanunjaya Naidu,** and our Head of Department, **Dr. M. Raja Sekar**, for extending their cooperation in doing this project within the stipulated time.

We extend our heartfelt thanks to our guide, **Dr. Spoorthy G,** and the project coordinators **Dr. P. Subhash**, and **Mr. A. Pramod Kumar** for their enthusiastic guidance throughout our project.

Last but not least, our appreciable obligation also goes to all the staff members of the Computer Science & Engineering department of CSE-(CyS, DS) and AI&DS and to our classmates who directly or indirectly helped us.

Miss Aleti Shriya Reddy	(20071A6201)
Mr. Bhushanam Vishnukanth Varma	(20071A6205)
Mr. Subhash Krishna Veer Buddhi	(20071A6253)
Miss Chennupati Dwijitha	(20951A6211)

ABSTRACT

Now-a-days, there has been a huge surge in intrusion detection and classification systems due to widespread usage in networks. The traditional intrusion detection systems fail to address high complexity, inefficiency in handling large datasets, so on. The proposed Intrusion Detection System outlines a progressive solution in network security, by employing a sophisticated approach that combines Perceptual Pigeon Galvanised Optimization for feature extraction. By effectively extracting features and detecting subtle intricacies in the patterns, the IDS demonstrates its adaptability to evolving cyber threats. These feature extraction techniques along with machine learning classifiers, Deep Learning techniques like Convolutional Neural Networks and Artificial Neural Networks harness cutting-edge optimization algorithms. The traditional deep learning architecture enhances intrusion detection efficiency and accuracy throughout the diverse network environments. Through thorough experimentation on various datasets such as Ton-IOT, NSL-KDD, UNSW and Cyber Intrusion data have been done. The results prove the efficiency in identifying and mitigating various types of intrusions. Our approach contributes in enhancing the network security of complex and dynamic environment, alongside safeguarding critical data from unauthorised exploitation and access.

Key words: Cybersecurity, threat modelling, IDS, PPGO, CNN, ANN, NSL-KDD, Random Forest, UNSW, BOT-IOT, K-Nearest-Neighbour, LSTM, LDA, QDA, Decision Making, Naive Bayes, GBC, Multi-Layer Perceptron.

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Now-a-days in an interconnected digital world, secured computer networks and systems are of Supreme importance due to the improved complexity of cyber threats. An Intrusion Detection Systems(IDS) serves as a primary defence system against malicious activities by continuously monitoring network traffic to identify and prevent potential security breaches. The intrusion detector systems [1,2] learn to build a predictive model (such as a classifier) which can be able to distinguish between ‘good connections’ and ‘bad connections’. The bad connections might be intruders or attacks. However, traditional IDS approaches often lag in adapting to evolving threats, prompting the need for innovative solutions. This paper suggests an advanced IDS that combines optimization techniques and deep learning methodologies customised to specific datasets, with an objective to enhance intrusion detection efficiency and accuracy across varied network environments.

There are three types of IDS based on the installation in the system: Host-based IDS (HIDS), Hybrid IDS and Network-based IDS (NIDS). In NIDS, malicious events or attacks are detected from different networks as it lies between computers, and NIDS is deployed on switches or routers in the network [7] The main purpose of NIDS is to detect malicious log events and then report them to the network manager. HIDS is implemented on a single system or host. In HIDS, attacks or attacks are detected from a single host computer system, and critical files of the operating system are examined [10] therefore, such attacks are generally easy except for very complex malware attacks if they will be found. Hybrid IDS can be deployed over a network as well as on host systems. The attack detection system generates an alarm when an intrusion is detected based on which the network or host manager attempts to prevent the attack.

In customary NIDS frameworks, intrusion detection is grounded on varied dataset features such as the way data flows, patterns of traffic and packet information. Nevertheless, this framework has some issues like single point failure factor, algorithmic complexity, signature dependence and timing [2]. Therefore; there are several other intrusion detection techniques used including: classification, optimization, clustering etc.

First, feature selection helps to reduce the curse of dimensionality [1] . IDS datasets typically contain multiple attributes, including network traffic attributes, packet information, and system

logs. However, not all factors contribute to accurate attack detection. In fact, the inclusion of irrelevant or irrelevant features can lead to overfitting and deterioration of the performance of the model. Feature selection by choosing only the most informative features reduces the dimensionality of the data set, thereby improving the performance and efficiency of ML models

Second, feature selection increases the interpretability of IDS models [11] . In cybersecurity, it is important for security researchers and practitioners to understand the logic behind research decisions. By identifying and selecting the most appropriate features, feature selection enables explicit examination of the factors affecting intrusion detection. This facilitates better understanding and interpretation of the model's behavior and predictions.

In addition, feature selection contributes to the generalizability and robustness of the IDS model. By focusing on the most discriminating features, ML models are less prone to noise and redundant information in the data structure [4] . This leads to more robust and reliable attack detection, as the model can better distinguish between genuine security threats and false alarms.

Moreover, feature selection helps to optimize computational resources and reduces training time. The removal of redundant features reduces the size of the data set, resulting in faster model training and reduced computational costs. This is particularly useful in real-time IDS environments, where timely detection and response to security threats is essential.

Overall, the choice of features plays an important role in the performance, interpretability, generalizability, and efficiency of ML-NDL [3] models for intrusion-detection systems of the By choosing the most informative features, the choice of features ensures that IDS models accurately detect security threats in different network environments And they can, and thus strengthen the overall cybersecurity posture of organizations

CHAPTER 2

LITERATURE SURVEY / EXISTING WORK

2.1 EXISTING WORK

2.1.1 Some of the Existing Models

Types of datasets used: Datasets such as KDD 99, NSL KDD, TON IoT, UNSW-NB15 have been widely used in previous works to train, validate and test Intrusion Detection Systems (IDS) These datasets provide fine granular collection of internal networks Data no . Moreover, this data set has been used to develop and evaluate classifiers with different algorithms that help to cover all possible types of attacks on different networks

Advanced embedded features: Meanwhile researchers in intrusion detection system (IDS) have adopted many modern algorithms in IDS design to improve their detection capabilities Decision trees, random forests, gradient boosting, logistic regression, K-nearest neighbours (KNN), naïve Bayes classifier linear and quadratic discriminant analysis (LDA/QDA), recurrent neural networks (RNNs) with LSTM networks are some of the algorithms that have been integrated into these systems. It is this integration which constitutes an attempt to harness varying strengths of different algorithms thereby improving the accuracy, efficiency and robustness of intrusion detection mechanisms for various network settings.

Deep Learning Approaches: There has been a remarkable increase in interest in applying deep learning techniques to input recognition tasks. Recurrent neural networks (RNNs) with deep learning mechanisms, especially long-term and short-term memory (LSTM) networks, have shown high efficiency in terms of time stability and pattern capture in between communication data in a sequence. This highlights the potential of deep learning to enhance the capabilities of intrusion detection systems, especially in detecting sophisticated and evolving threats.

2.1.2 Drawbacks of the Existing System

Reliance on specific data: One limitation of existing research is the reliance on specific data for training and evaluation. While datasets such as KDD 99, NSL KDD, TON IoT, UNSW-NB15 provide valuable insights and perspectives, they do not adequately represent real-world network scenarios and moreover, these datasets may have biases or limitations of their collection strategies They make overly adequate or incomplete descriptions of emerging threats and so caution is needed to generalize findings from research based on specific cases to broader contexts.

Complexity and definition: Some modern algorithms, especially deep learning models, exhibit high complexity and lack definition. While these models can have impressive detection accuracy, understanding the logic behind their decisions can be difficult. The black box nature of deep learning models complicates validation, troubleshooting, and reliability building of systems. In addition, complex models may require significant computational resources for training and estimation, posing practical challenges in resource-constrained environments or real-time process environments

Scalability and adaptability: Another challenge is the scalability and adaptability of intrusion detection systems (IDS) developed with modern algorithms. While these systems may prove effective under controlled conditions, their performance is less assured in real-world scenarios with evolving threats and dynamic traffic systems Monitoring that IDS can scale to handle more network traffic and adapt to emerging threats Seamless integration of IDS to achieve is a complex and ongoing challenge.usion detection systems in particular have sophisticated and evolving threats detected in.

2.2 LITERATURE SURVEY

The method proposed in the first paper combines various methods such as anticipated distance-based clustering (ADC), DBScan clustering, Perpetual Pigeon Galvanized Optimization (PPGO), and Likelihood Naïve Bayes (LNB) classification to generate intrusion detection systems (IDS) are accurate). By combining these methods, the research aims to optimise the product selection and classification processes to improve fraud detection, however, the paper neglects to address ethical issues and potential privacy issues associated with the

implementation of IoT CAD security systems, which may pose challenges related to data privacy and user consent .

On the other hand, Paper II provides a comprehensive analysis of the applicability of machine and deep learning techniques in Internet of Things (IoT) security It encompasses various aspects of IoT computing infrastructure, such as technology development, . testing, implementation, program development, training, and user acceptance Shining a light on potential risks, especially to compromise cryptographic functionality, this highlights the need for robust security measures to mitigate such risks underscoring the evolving IoT cybersecurity landscape.

The study of intrusion detection in IoT using deep learning goes deeper into preprocessing, feature selection, and classification techniques using deep learning models such as Convolutional Neural Networks (CNNs), Long Short-Term Memory (1999). LSTM), and the Gated Recurrent Units operate within (GRU). Using these techniques, the research aims to enhance the security of IoT networks by detecting and classifying botnet attacks more effectively.

However, the heavy reliance on supervised learning processes raises concerns about the system's ability to adapt to emerging or unknown IoT botnet attack patterns, potentially limiting its effectiveness in real-world scenarios

In contrast, the paper on hybrid intrusion detection systems for IoT addresses the limitations of existing IDSs by providing a comprehensive approach. Exploring the shortcomings of current intrusion detection methods, the research aims to develop a hybrid algorithm that can effectively detect and mitigate security threats in IoT environments but the proposed algorithm may face challenges in addressing multiple attack scenarios and adapting evolving threats To ensure that the need for further research and development in this area is emphasised

Machine Learning Security Framework The course of IoT Framework examines a combination of Network Function Virtualization (NFV), Software Defined Networking (SDN), and machine learning techniques to enhance the security and performance of IoT networks Concerns Despite the raised using machine learning algorithms for network management purposes can pose

additional security risks, underscoring the importance of strong cybersecurity measures to protect IoT systems

Meanwhile, the paper on IoT Cybersecurity Documentation Review and Risk Management proposes a four-pronged framework for managing cybersecurity risks in the IoT ecosystem. IOT Cyber Basic Systems that identify and assess the potential and Khataras to generate cost-effectiveness impacts to study the purpose of studying research on cybers security but the iot is iOT asset management and greater security. Ongoing assessment and empowerment.

CHAPTER 3

SOFTWARE REQUIREMENTS

3.1 Functional Requirements

Specifying the specific behaviours and functions that a system must have to achieve its intended goals in functional requirements is a methodology for system development, providing clear guidelines for the design and implementation of features intended and functional Functional requirements are essential to accurately represent user expectations. Functional requirements provide stakeholders, developers, and testers with shared knowledge of expected system behaviour, which enhances collaboration and communication throughout the project lifecycle Finally, functional requirements are important to guide the development process and help teams develop systems that meet user needs and expectations.

1. Data Preprocessing
    - The system must be able to preprocess input data sets, including handling missing values, encoding categorical variables, and scaling numeric features.
    - It should support various pre-processing techniques such as single-hot encoding, normalisation, label encoding, feature selection.
2. Feature Selection
    - The system should provide selection methods such as linear discriminant analysis, decision trees, gradient enhancement and correlation analysis.
    - It enables users to select appropriate features based on their impact on model performance.
3. Attack Classification
    - The system must be able to train and test machine learning models for attack classification.
    - It should support various classification algorithms such as decision trees, random forests, logistic regression, and neural networks.
    - To evaluate the performance of the model, analytical parameters such as accuracy, precision, recall, and F1-score must be calculated.

3.2 Non-functional Requirements

On the other hand, passive requirements satisfy system characteristics and characteristics. For example, usability, functionality, reliability and security are some of these features. In other words, functional requirements describe what the system should do while non-functional requirements show how it should be done correctly. These are necessary to enable the system to operate at the desired level of performance and to provide a good user experience. Passive requirements such as response time, scalability, security measures help stakeholders to assess the suitability of the system for its overall purpose In summary, functional and passive requirements is also important to guide the development process.

1. Performance
    - The system should be able to handle large datasets efficiently and perform preprocessing, feature selection, and attack classification in a reasonable amount of time.
    - It should be scalable to accommodate increasing amounts of data and computational resources.
2. Accuracy
    - The system's classification models should achieve high accuracy in detecting and classifying intrusion activities.
    - It should minimise false positives and false negatives to ensure reliable intrusion detection.
3. Usability
    - The system should have a user-friendly interface that allows administrators to easily interact with and configure preprocessing, feature selection, and attack classification tasks.
    - It should provide clear feedback on the progress and results of each stage of the intrusion detection process.
4. Security
    - Systems must ensure the confidentiality and accuracy of critical identification information.
    - Access to the system and its components must be implemented to prevent unauthorised access.
5. Reliability

- The system should be robust and resilient to failures, ensuring continuous operation even in the event of hardware or software failures.
- It should be able to recover gracefully from errors and provide mechanisms for data backup and recovery.

6. Scalability
    - On another note if there are many users or data that is being processed by this system then at least its performance should not be affected significantly.
    - Furthermore this should allow distributed computing as well as parallel processing to take place so that future demands from intrusion detection might be taken care off

CHAPTER 4

SOFTWARE DESIGN

4.1 UML Diagrams

4.1.1 Class Diagram

The learning diagram is important because it helps to visualise and explain the system structure associated with a threat model. It describes how the different objects (classes) in the system interact and communicate with each other, and their responsibilities in the system. There are categories in our sample:

Classrooms:

1. DataPreprocessing: It is responsible for preprocessing the input dataset.
2. FeatureSelection: Performs the feature selection process.
3. AttackClassification: It takes care of training and evaluating machine learning (ML) systems for attack segregation.
4. Dataset: Represents the datasets used in the system (TON-IOT, NSL-KDD, UNSW, Cyber Intrusion).
5. MachineLearningModel: Represents the machine learning(ML) model used for attack grouping.
6. EvaluationMetrics: Represents metrics (e.g., F1-score, recall, accuracy, precision,,) used to assess system performance.

The production method

Each category contains strategies that improve performance:

1. DataPreprocessing method : preprocess (dataset : Dataset): void
    - This method takes a Dataset object as input and uses preprocessing techniques to prepare the dataset for further processing.
2. Selection method : selectFeatures (dataset : Dataset): void
    - This method takes a Dataset object as input and selects the most informative objects from the dataset for further analysis.
3. AttackClassification method : classifyAttacks (dataset : dataset , model : Machine LearningModel): null

- This method takes a Dataset object and a MachineLearningModel object as input and performs an attack classification using the given dataset and model.

4. Machine learning modelling methods:
   - train(dataset: Dataset): void: This method trains a machine learning(ML) system on the given dataset.
   - evaluate(dataset: Dataset, metrics: EvaluationMetrics): void: This method uses the specified evaluation metrics to assess the performance of the model trained for the given dataset
5. Research metric methods:
   - getAccuracy(): float: This method returns the system accuracy.
   - getPrecision(): float: This method returns the exactness of the model.
   - getRecall(): float: This method returns the memory of the model.
   - getF1Score(): float: This method returns the F1-score of the model.

Together, these classes provide important applications for preprocessing datasets, selecting features, classifying attacks, training machine learning models, and testing model performance in an intrusion detection system

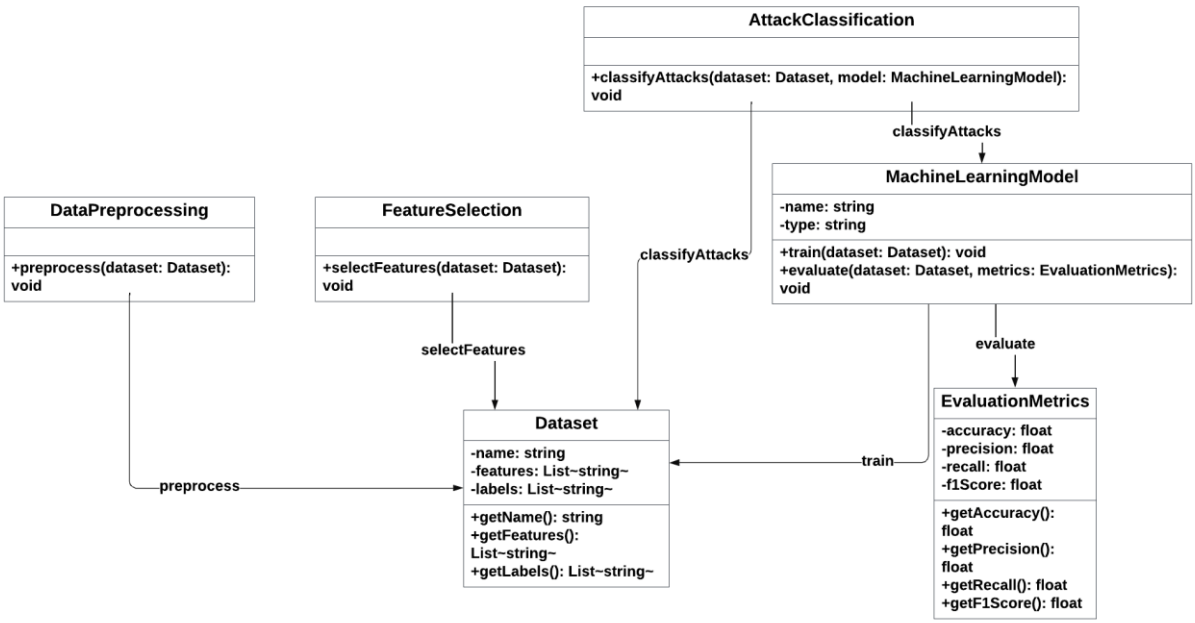


Fig 4.1.1. Class Diagram

4.1.2 Use case Diagram

Use information diagrams to illustrate how these actors interact to enable users to access similar content and desired information for users to search.

- The administrator (operator) contacts the Data Preprocessing task to initiate preprocessing of the data. The administrator assigns data sets to be processed first.
- The machine learning instance interacts with the Data Preprocessing task to provide data types required for preprocessing.
- The administrator interacts with the Feature Selection function to choose applicable properties from the preprocessed dataset.
- The machine learning(ML) system interacts with the Feature Selection task to provide the data types required for feature selection.
- The administrator interacts with the attack classification function to classify attacks using pre-generated and feature-selected data.
- The machine learning model interacts with the attack classification task to provide data structures and trained models for classification.

There are other credentials, users and roles as shown in the figure.

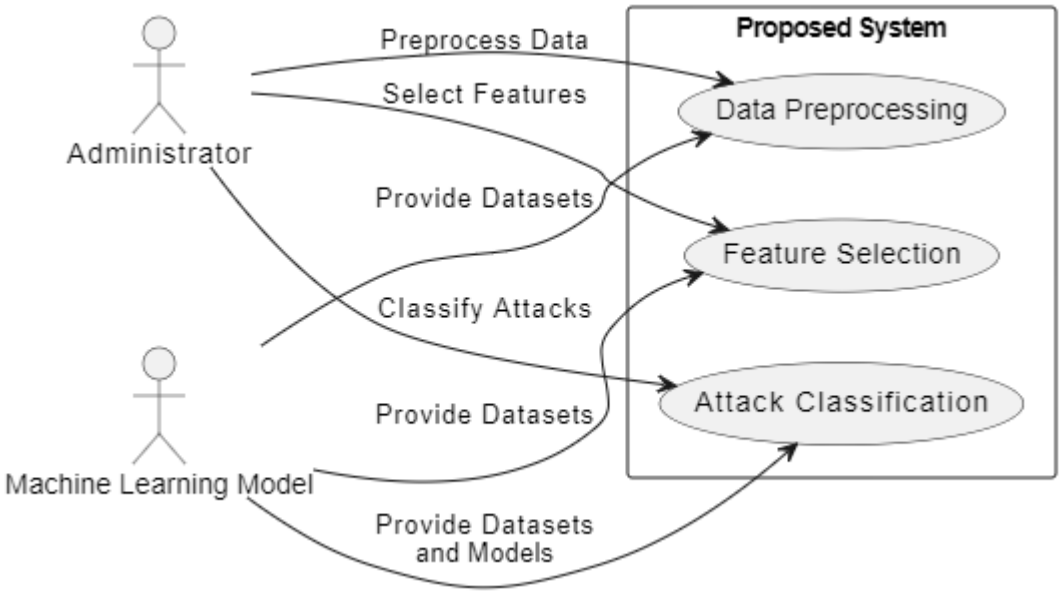


Fig 4.1.2. Use case diagram of the proposed system

4.1.3 Sequence Diagram

The sequence diagram shows the interaction between a user and a named part of the system

- The administrator initiates the process by requesting that the data be processed first.
- The administrator provides data sets to the machine learning(ML) instance.
- The Data Preprocessing module preprocesses given datasets.
- The machine learning model provides preprocessed data for the Feature Selection module.
- The Feature Selection module selects relevant features from a preprocessed data set.
- The machine learning model provides a pre-processed data set and feature selection with trained models for the attack classification module.
- The attack classification module classifies attacks using the data types and patterns provided.

This series of diagrams illustrates the interactions between actors and modules in the proposed framework, and illustrates the data preprocessing, feature selection, and attack classification tasks

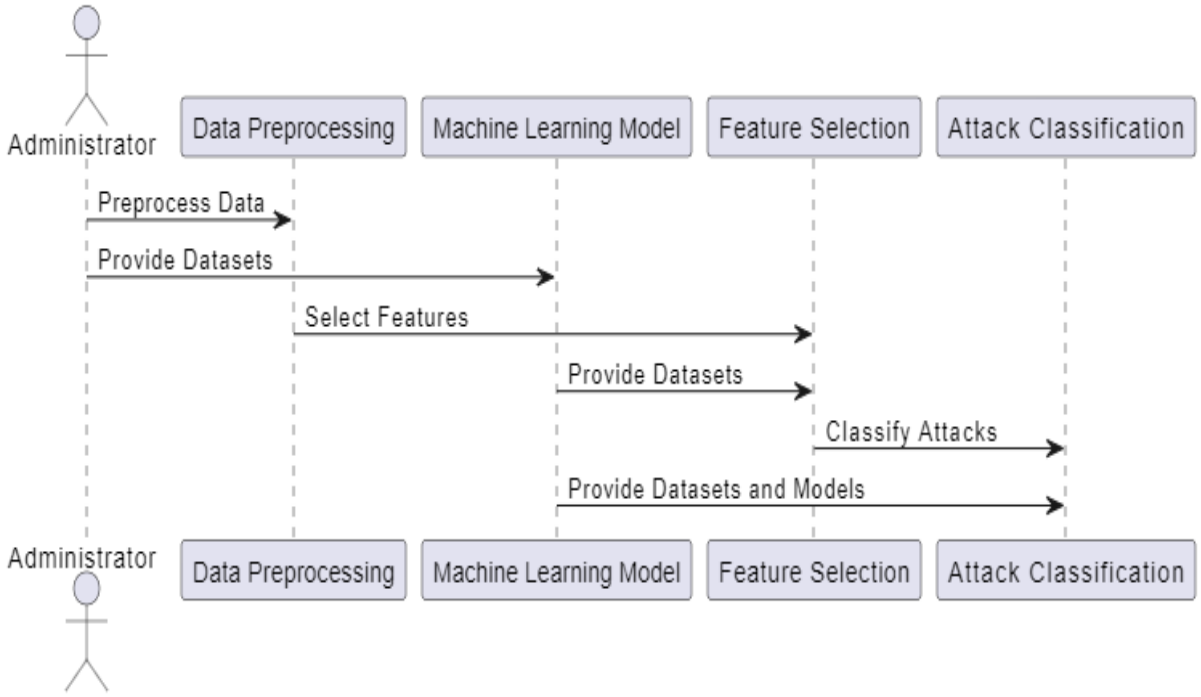


Fig 4.1.3. Sequence diagram of the Proposed system

4.1.4 Activity Diagram

A functional diagram shows the functionality that goes into a system or process. It allows us to create a visual representation of the sequence of actions or steps in a particular business process, which helps to understand the overall behaviour of the system or process

- The process begins immediately.
- The system checks if the data set is TON-IOT. If yes, then one hot encoding and normalisation goes, followed by feature selection and attack segregation.
- If the data set is not TON-IOT, the system checks to see if it is NSL-KDD. If yes, it handles missing values, encodes categorical variables, scales numeric features, and then performs feature selection and attack grouping.
- If the data set is not NSL-KDD, the system checks to see if it is UNSW. If yes, it handles missing values and label encoding, performs feature selection based on Pearson correlation, and then performs feature selection and attack grouping.
- If the data set is not UNSW, the system checks to see if it is a Cyber Intrusion data set. If yes, it imports data, removes unnecessary attributes, and performs single-hot encoding, normalisation, feature selection, and attack classification.
- Eventually, the process comes to an end.

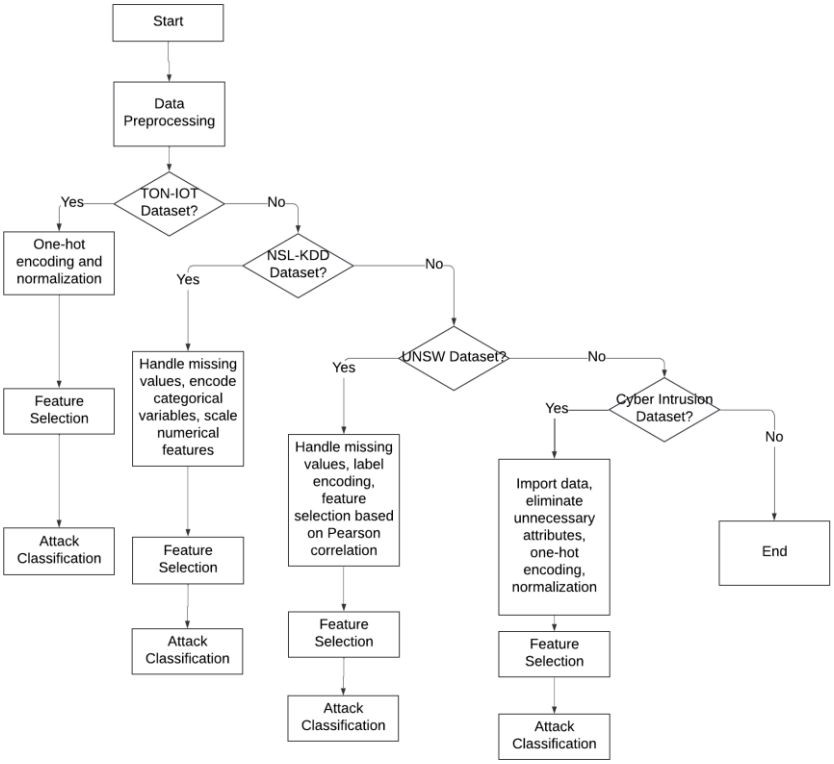


Fig 4.1.4. Activity diagram

4.1.5 Work Flow Diagram

A workflow diagram visually represents the steps, tasks, activities, and decisions associated with the completion of a specific activity or business process in an organisation or system. It shows a sequence of activities from start to finish, showing how the work goes systematically to various sectors and stakeholders. Business process diagrams are used to analyse, plan, optimise, and communicate business processes and business processes.

- The process starts in the "Start" area.
- Checks if the data set is TON-IOT. If yes, it proceeds with one-hot encoding normalisation, then proceeds to feature selection and attack classification.
- If the data set is not TON-IOT, it checks if it is NSL-KDD. If yes, it handles missing values, encodes categorical variables, scales numeric features, and then proceeds to feature selection and attack classification.

- If the data set is not NSL-KDD, it checks if it is UNSW. If yes, it handles missing values and label encoding, performs feature selection based on Pearson correlation, and then performs feature selection and attack classification
- If the data set is not UNSW, it checks to see if it is a Cyber Intrusion data set. If yes, it imports the data, removes unnecessary attributes, and performs single-hot encoding, normalization, feature selection, and attack classification.
- Finally, the process ends at the "End" point.

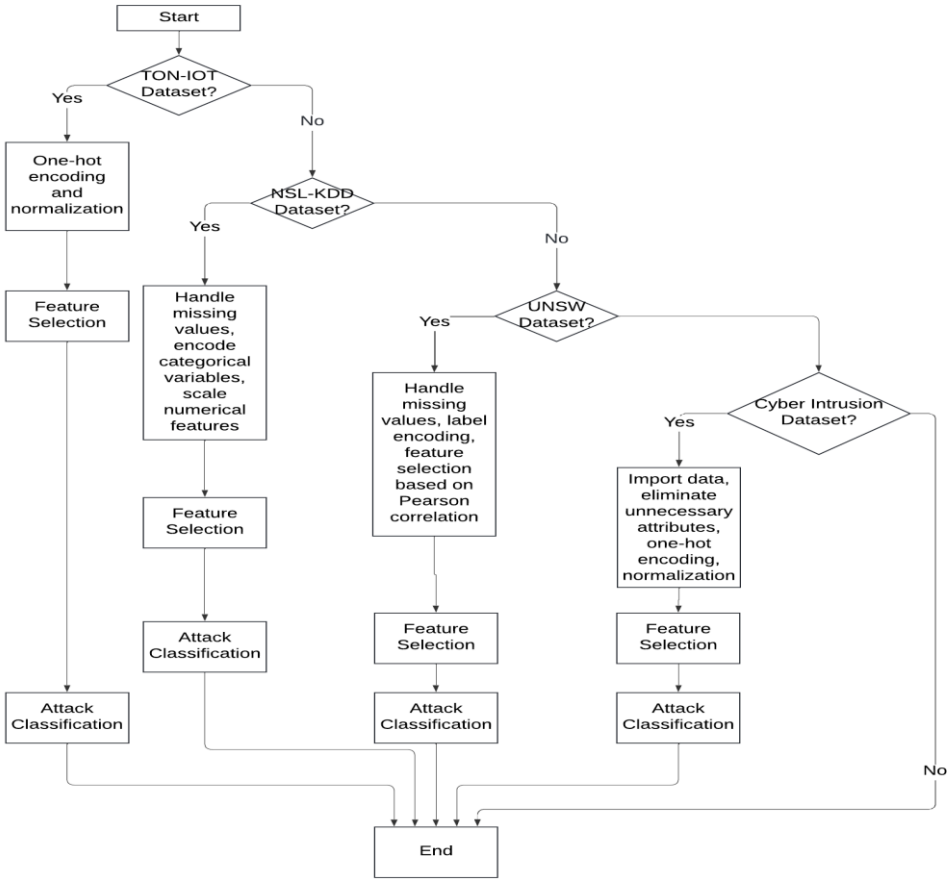


Fig 4.2. Workflow diagram

CHAPTER 5

PROPOSED SYSTEM

The TON-IOT, NSL-KDD, UNSW, and Cyber Intrusion databases are four data types used to report subsequent intrusion detections using the proposed systems described in this work. It is important to ensure systems are secure and accurate. By accurately predicting and naming common attacks through a rigorous process that includes data analysis, model training and analysis, this approach seeks to strengthen network security

The creation of data entry is an important aspect of the proposed framework. This step should be done to ensure that the data is clean, standardized, and properly prepared for further testing. The data are prepared separately for each data set. For example, the preprocessing includes adjusting the numerical representations of categorical variables and normalizing numerical characteristics to standard ranges in the TON-IOT dataset, which includes IoT devices Such as, the NSL-KDD data set-which contains network traffic data-needs scaling numerical characteristics , and resolving missing values SW Data Collection A rigorous preliminary step in selecting objects based on their relationship shapes and written characters. Conversely, preparation procedures compared to computerised input datasets are used to ensure data integrity and purity. The main feature of the proposed scheme is feature selection, which generates the most appropriate sub attributes for accurate penetration detection. Rather than focusing only on specific algorithms, the system explores multiple methods optimised for each data set. For example, features selected in the TON-IOT dataset are using various methods such as logistic regression, decision trees, gradient enhancement to determine the importance of attributes like this in the NSL-KDD dataset gradient boosting, correlation matrix a depending on how well features correlate with attack activities -Selection is made using methods such as analysis Although the Cyber Intrusion database may use methods such as principal component analysis or repetitive feature extraction to find appropriate characteristics, the UNSW database uses similar methods.

The final step of the proposed framework to classify network traffic into specific intrusion classes involves training and testing machine learning models, which focus on attack

classification Here various machine learning techniques are used roles and data types, each customized to the distinct qualities of the data. For example, the TON-IOT data set can be segmented using logistic regression, decision trees, and random forests; The NSL-KDD data set can be well suited for algorithms such as naive Bayes and linear discriminant analysis(LDA). Each algorithm is evaluated using criteria including F1-score,accuracy, precision, and recall performance monitoring to determine the most effective method for accurately detecting and classifying attack activities in each dataset

In summary, the proposed scheme provides a complete attacker detection scheme. This includes feature selection, data preparation, and attack classification tailored to the specific characteristics of each data set. Using a variety of techniques, the system seeks to increase the accuracy and simplicity of intrusion detection, thus strengthening the overall security of the network and combating cyberattacks attacks

5.1 ADVANTAGES OF THE PROPOSED SYSTEM

1. All-inclusive methodology: The system uses a multi-dataset approach for attack classification, feature selection, and data generation in its intrusion detection process The technology provides a comprehensive framework for network detection and segmentation environments accurately by handling multiple cases

2. Flexibility for different datasets: One of its strengths is the ability of the proposed framework to adapt to different datasets, such as TON-IOT, NSL-KDD, UNSW, and different cyber-intrusion databases and intervention issues

3. Advanced Data Preprocessing: The system uses careful data preprocessing techniques to ensure that the incoming data is clean, standardized and organized for analytical technology to improve the quality and usability of the data system by troubleshooting such as missing values, category coding, attribute measurement

4. Optimization of feature selection: Increasing the efficiency and accuracy of input detection models is highly dependent on feature selection. The proposed method uses several techniques including principal component analysis, gradient enhancement, and correlation matrix analysis to find the most suitable subset subset for admission detection accuracy The scheme reduces model complexity and improves performance by retaining only relevant items and eliminating the useless

5. Variety of Machine Learning strategies: The device makes use of loads of devices gaining knowledge of techniques, every specially designed to fulfil the precise necessities of a given dataset, to classify attacks. The system can successfully categorise community traffic into numerous intrusion kinds by using strategies including logistic regression, selection timber, random forests, and linear discriminant evaluation. This improves the device's universal detection skills.

6. Evaluation Metrics for Performance Assessment: To evaluate the effectiveness of intrusion detection models, the advised system uses an extensive range of assessment metrics, inclusive of accuracy, precision, bear in mind, and F1-rating. The era facilitates properly-knowledgeable choice-making and intrusion detection procedure optimization through methodically assessing the efficacy of various algorithms and methodologies.

CHAPTER 6

CODING

We developed four independent models using different datasets: UNSW, Ton-IoT, Cyber Intrusion, and NSL-KDD. These models were put into the frontend of our website.

6.1 Model - Cyber Intrusion Dataset

Data Preprocessing

This code fragment is designed to pick up intrusions.It starts by sorting out the dataset by storing attribute names and labelling different kinds of invasions. Labelling is done using one-hot encoded to ensure compatibility with machine learning methods. Various marginal features are deleted so as to enhance effectiveness of the model.Next, the dataset is divided into testing and training datasets.The Shapiro-Wilk test was used to check whether each variable follows a normal distribution.Since it doesn't, Min-Max scaling is used for data standardisation in order to obtain uniform ranges for input variables in the model.In brief, this preprocessing make sure that the dataset is properly formatted and prepared for training intrusion detection models which are crucial in detecting and mitigating various cyber security risks.

```
# save attribute and label strings
all_attribute_names = intru_table.columns.drop('label')

intru_label_names = ['Backdoor_Malware', 'BenignTraffic', 'BrowserHijacking', 'CommandInjection', 'DDoS-ACK_Fragmentation', 'DDoS-HTTP_Flood', 'DDoS-ICMP_Flood', 'DDoS-ICMP_Fragmentation',
                     'DDoS-PSHACK_Flood', 'DDoS-RSTFINFlood', 'DDoS-SYN_Flood', 'DDoS-SlowLoris', 'DDoS-SynonymousIP_Flood', 'DDoS-TCP_Flood', 'DDoS-UDP_Flood', 'DDoS-UDP_Fragmentation',
                     'DNS_Spoofing', 'DictionaryBruteForce', 'DoS-HTTP_Flood', 'DoS-SYN_Flood', 'DoS-TCP_Flood', 'DoS-UDP_Flood', 'MITM-ArpSpoofing', 'Mirai-greeth_flood', 'Mirai-greip_flood',
                     'Mirai-udpplain', 'Recon-HostDiscovery', 'Recon-OSScan', 'Recon-PingSweep', 'Recon-PortScan', 'SqlInjection', 'Uploading_Attack', 'VulnerabilityScan', 'XSS']
```

Python

```
# complete one-hot encoding on intrusion labels
ohe = OneHotEncoder()
ohe_intru_labels = pd.DataFrame(ohe.fit_transform(intru_table['label'].values.reshape(-1,1)).toarray(), columns=intru_label_names)
```

Python

```
# we decided to remove the following attributes since they provide no meaningful contributions to the model
exclude = ['ece_flag_number', 'cwr_flag_number', 'SMTP', 'Telnet', 'IRC', 'Tot sum', 'Min', 'Max', 'AVG', 'Std', 'Tot size', 'Covariance', 'Variance']

relevant = ['flow_duration', 'Header_Length', 'Protocol Type', 'Duration', 'Rate', 'Srate', 'Drate',
            'syn_count', 'fin_count', 'urg_count', 'rst_count', 'IPv', 'LLC', 'DNS', 'SSH', 'TCP', 'UDP',
            'DHCP', 'ARP', 'ICMP', 'IAT', 'Number', 'Magitue', 'Radius', 'Weight']

# remove unwanted data attribute columns
attribute_data = intru_labels_removed.drop(columns=exclude)

# split data for all model runs
X_train, X_test, y_train, y_test = train_test_split(attribute_data, ohe_intru_labels, train_size=0.9, random_state=5)
```

Python

```
# check if each attribute is normally distributed
for col_name in attribute_data:
    print(col_name, shapiro(attribute_data[col_name]), "\n\tmin:", np.min(attribute_data[col_name]), "; max:", np.max(attribute_data[col_name]))
```

Python

```
flow_duration ShapiroResult(statistic=0.0039147138595581055, pvalue=0.0)
        min: 0.0 ; max: 99435.76178
Header_Length ShapiroResult(statistic=0.1560286283493042, pvalue=0.0)
```

Fig 6.1.1. Code Snippet of Data Preprocessing in Cyber Intrusion Dataset

Data Visualisation

In this case, we aimed at displaying how each input feature relates with different intrusion types and showing their prevalence in the data set. Firstly, it gives counts of true labels for each type of intrusion and makes a bar plot to visualise this spread. Each bar indicates how many times an individual intrusion appears within the dataset.It then iterates over each input attribute (except the 'label' column) and generates categorical graphs (cat plots) to show how each attribute's values are distributed across the various intrusion types. These plots aid in understanding the relationship between input properties and intrusion kinds by revealing which features may be more indicative of specific types of invasions. Overall, these visualisations provide a better understanding of the dataset's composition and can help uncover trends and links between input variables and intrusion types, hence assisting in the creation of effective intrusion detection models.

Fig: 6.1.2. Code Snippet of Data Visualisation in Cyber Intrusion Dataset

Feature Selection

PPGO algorithm is a straightforward optimization method that iteratively adjusts parameters to find the optimal solution to the problem Parameters are first randomly initialised, then modified using default logic so, which may include operations such as mutation or crossover. The revised model is then evaluated using an objective function representing the characteristics of the solution. This process is performed recursively for a predetermined number of times, with the goal of improving the performance of the solution over time. PPGO monitors the optimization process by publishing the number of iterations and the corresponding loss value. Overall, PPGO

continues to refine the theory, gradually converging on an optimal or nearly optimal solution to a problem.

```python
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Define PPGO algorithm
def ppgo_initialize(num_parameters):
    # Initialize parameters randomly or using some strategy
    initial_parameters = np.random.randn(num_parameters)  # Example: Random initialization
    return initial_parameters

def ppgo_update(current_parameters):
    # Implement parameter update logic based on PPGO algorithm
    # This could involve various operations such as mutation, crossover, etc.
    updated_parameters = current_parameters + np.random.randn(len(current_parameters)) * 0.1  # Example: Random update
    return updated_parameters

def perpetual_pigeon_galvanized_optimization(objective_function, initial_parameters, num_iterations):
    current_parameters = initial_parameters
    for i in range(num_iterations):
        updated_parameters = ppgo_update(current_parameters)
        loss = objective_function(updated_parameters)
        print(f"Iteration {i+1}, Loss: {loss}")
        current_parameters = updated_parameters
```

Fig: 6.1.3. Code Snippet of PPGO Feature Selection in Cyber Intrusion Dataset

ANN Model

This part of the code trains and tests a neural network model to predict intrusion classifications. The neural network consists of three macro layers with different units and with different activation functions. The model was constructed using a stochastic gradient descent (SGD) optimizer and categorical cross-entropy loss functions. It separates to 50 epochs with training data. After training, the functioning of the model is tested on the testing and training datasets. Performance metrics for both data sets including precision, accuracy, recall, mean square error (MSE), and classification reports are calculated and published These metrics help determine how accurately the model can classify attacks. Overall, the code outlines a procedure for training and evaluating neural network models for intrusion detection in simple steps.

```python
# HIGH2 FIT
# setup neural net for predicting intrusion class
classifier = Sequential()
classifier.add(Dense(units = 56, activation = 'relu', input_dim = 33))
classifier.add(Dense(units = 55, activation = 'relu'))
classifier.add(Dense(units = 34, activation = 'softmax'))
classifier.compile(optimizer = "SGD", loss = 'CategoricalCrossentropy')
classifier.fit(X_train.astype(float), y_train, epochs = 50)
```

Python

```
Epoch 1/50
29492/29492 [==============================] - 32s 1ms/step - loss: 0.7240
Epoch 2/50
29492/29492 [==============================] - 30s 1ms/step - loss: 0.5340
Epoch 3/50
29492/29492 [==============================] - 29s 998us/step - loss: 0.4991
Epoch 4/50
29492/29492 [==============================] - 31s 1ms/step - loss: 0.4839
Epoch 5/50
29492/29492 [==============================] - 31s 1ms/step - loss: 0.4767
Epoch 6/50
29492/29492 [==============================] - 33s 1ms/step - loss: 0.4722
Epoch 7/50
29492/29492 [==============================] - 31s 1ms/step - loss: 0.4684
Epoch 8/50
29492/29492 [==============================] - 32s 1ms/step - loss: 0.4651
Epoch 9/50
29492/29492 [==============================] - 32s 1ms/step - loss: 0.4619
Epoch 10/50
29492/29492 [==============================] - 31s 1ms/step - loss: 0.4592
Epoch 11/50
29492/29492 [==============================] - 30s 1ms/step - loss: 0.4568
Epoch 12/50
29492/29492 [==============================] - 30s 1ms/step - loss: 0.4548
Epoch 13/50
...
Epoch 49/50
29492/29492 [==============================] - 30s 1ms/step - loss: 0.4281
Epoch 50/50
29492/29492 [==============================] - 30s 1ms/step - loss: 0.4277
```

```python
# HIGH2 STATS
# measure performance on training data
intru_predictions = pd.DataFrame(((classifier.predict(X_train.astype(float))) > 0.5).astype(int), columns=ohe_intru_labels.columns)
print("Model Training Accuracy (avg.):", accuracy_score(y_train, intru_predictions))
print("Model Training Precision (avg.):", precision_score(y_train, intru_predictions, average='macro'))
print("Model Training Recall (avg.):", recall_score(y_train, intru_predictions, average='macro'))
high2_MSE_train = mean_squared_error(y_train, intru_predictions)
print("Model Training MSE:", high2_MSE_train)
print(classification_report(y_train, intru_predictions, target_names=ohe_intru_labels.columns))

# measure performance on testing data
intru_predictions = pd.DataFrame(((classifier.predict(X_test.astype(float))) > 0.5).astype(int), columns=ohe_intru_labels.columns)
print("Model Testing Accuracy (avg.):", accuracy_score(y_test, intru_predictions))
print("Model Testing Precision (avg.):", precision_score(y_test, intru_predictions, average='macro'))
print("Model Testing Recall (avg.):", recall_score(y_test, intru_predictions, average='macro'))
high2_MSE_test = mean_squared_error(y_test, intru_predictions)
print("Model Testing MSE:", high2_MSE_test)
print(classification_report(y_test, intru_predictions, target_names=ohe_intru_labels.columns))
```

Python

```
29492/29492 [==============================] - 24s 815us/step
Model Training Accuracy (avg.): 0.8100500467830928
Model Training Precision (avg.): 0.6449885638412912
Model Training Recall (avg.): 0.4971693039195549
Model Training MSE: 0.010798383039681201
                      precision    recall  f1-score   support

      Backdoor_Malware       0.00      0.00      0.00        68
         BenignTraffic       0.74      0.94      0.82     22033
      BrowserHijacking       0.00      0.00      0.00       126
      CommandInjection       0.00      0.00      0.00        88
DDoS-ACK_Fragmentation       0.99      0.98      0.98      5796
       DDoS-HTTP_Flood       0.62      0.56      0.59       579
       DDoS-ICMP_Flood       1.00      1.00      1.00    145241
DDoS-ICMP_Fragmentation       1.00      0.97      0.99      9235
      DDoS-PSHACK_Flood       1.00      1.00      1.00     83181
      DDoS-RSTFINFlood       1.00      1.00      1.00     81774
        DDoS-SYN_Flood       0.66      0.96      0.78     82578
       DDoS-SlowLoris       0.89      0.14      0.24       448
DDoS-SynonymousIP_Flood       0.90      0.58      0.71     72679
        DDoS-TCP_Flood       0.64      0.97      0.77     90906
```

Fig: 6.1.4. Code Snippet of ANN Model

6.2 Model - UNSW-nb15 Dataset

1. Importing Modules and Libraries: Necessary libraries and modules are imported, including those for data manipulation, visualization, and machine learning algorithms.

Importing Modules and Libraries

```
# importing required libraries
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

import pickle
from os import path

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

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

from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
```

Fig: 6.2.1. Import modules and libraries

2. Importing Datasets: The UNSW-NB15 dataset is imported and examined.

Importing Datasets

```
data = pd.read_csv('D:/Project_2024/Intrusion_Detection_2024/datasets/UNSW_NB15.csv')
```

Fig: 6.2.2. Imporing datasets

3. Data Preprocessing:
   - Missing values in the dataset are handled.
   - Categorical variables are encoded using one-hot encoding and label encoding.
   - Data normalisation is performed.
4. Data Visualization: Visualisations are created to understand the distribution of binary and multi-class labels.
5. Feature Selection: Attributes are selected based on correlation with labels for both multi-class classification and binary .

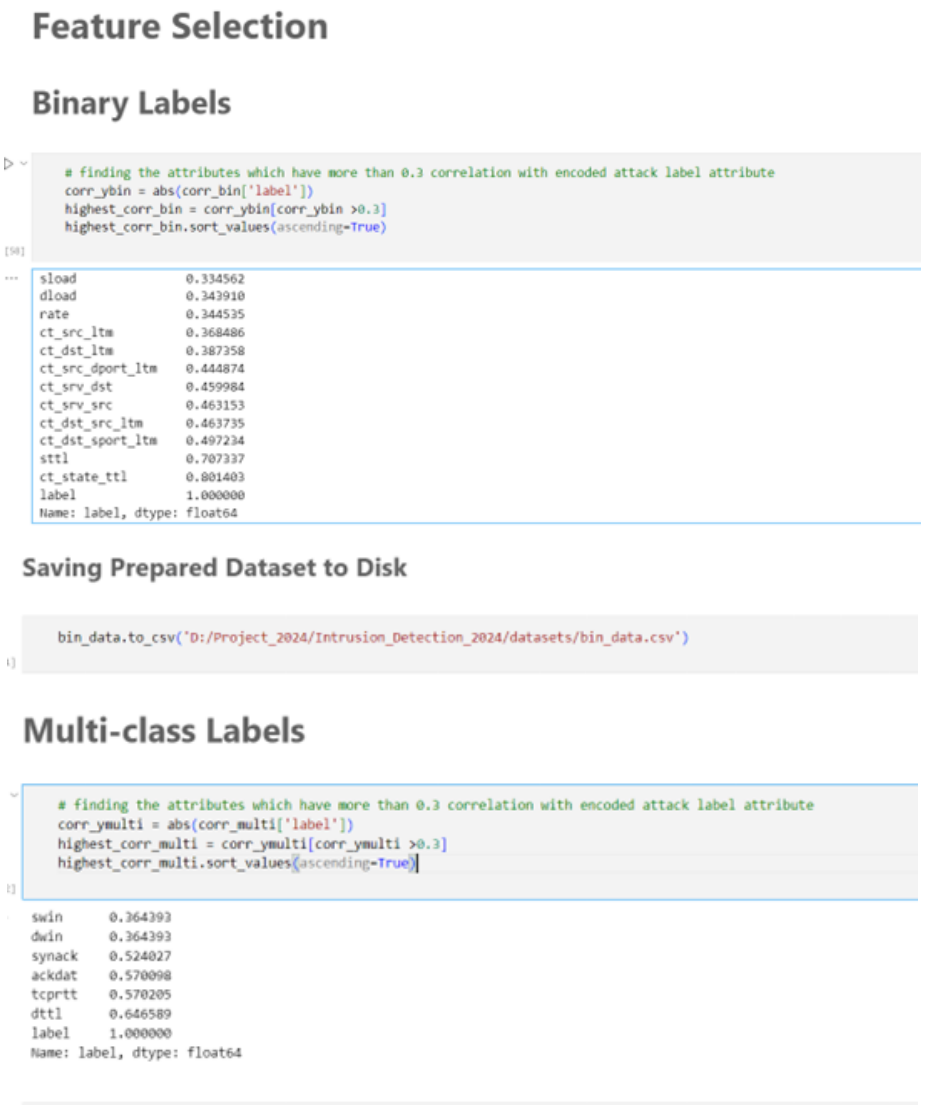


Fig: 6.2.3. Feature selection

6. Model Training and Evaluation:

○ Models such as Linear Regression, Logistic Regression, Support Vector Machine, K Nearest Neighbors, Random Forest, Decision Tree, and Multi-Layer Perceptron are trained and evaluated for both binary and multi-class classification tasks.

○ Evaluation metrics like Mean Absolute Error, Root Mean Squared Error, Mean Squared Error, R2 Score, Accuracy, and Classification Report are provided.

BINARY CLASSIFICATION

Data Splitting

```
X = bin_data.drop(columns=['label'],axis=1)
Y = bin_data['label']
```

66]

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.20, random_state=50)
```

67]

Linear Regression

```
lr_bin = LinearRegression()
lr_bin.fit(X_train, y_train)
```

69]

▾ LinearRegression
LinearRegression()

Real and Predicted Data

```
lr_bin_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
lr_bin_df.to_csv('D:/Project_2024/Intrusion_Detection_2024/predictions/lr_real_pred_bin.csv')
lr_bin_df
```

?]

	Actual	Predicted
159889	0	0
125506	0	0
158979	0	0
8148	1	1
39918	1	1
...	...	...
68470	0	0
159674	0	0
171321	0	0
125777	0	0
39178	1	1

Plot between Real and Predicted Data

```
plt.figure(figsize=(20,8))
plt.plot(y_pred[:200], label="prediction", linewidth=2.0,color='blue')
plt.plot(y_test[:200].values, label="real_values", linewidth=2.0,color='lightcoral')
plt.legend(loc="best")
plt.title("Linear Regression Binary Classification")
plt.savefig('D:/Project_2024/Intrusion_Detection_2024/plots_UNSW/lr_real_pred_bin.png')
plt.show()
```

Logistic Regression

```
logr_bin = LogisticRegression(random_state=123, max_iter=5000)
logr_bin
```

```
LogisticRegression(max_iter=5000, random_state=123)
```

```
logr_bin.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=5000, random_state=123)
```

Linear Support Vector Machine

```
lsvm_bin = SVC(kernel='linear',gamma='auto')
lsvm_bin.fit(X_train,y_train)
```

```
SVC(gamma='auto', kernel='linear')
```

K Nearest Neighbor Classifier

```
knn_bin=KNeighborsClassifier(n_neighbors=5)
knn_bin.fit(X_train,y_train)
```

```
KNeighborsClassifier()
```

Random Forest Classifier

```
rf_bin = RandomForestClassifier(random_state=123)
rf_bin.fit(X_train,y_train)
```

```
RandomForestClassifier(random_state=123)
```

Decision Tree Classifier

```
dt_bin = DecisionTreeClassifier(random_state=123)
dt_bin.fit(X_train,y_train)
```

```
DecisionTreeClassifier(random_state=123)
```

Fig: 6.2.4. Code snippets of binary classification

7. Saving Trained Models to Disk: Trained models are saved to disk for future use.

MULTI-CLASS CLASSIFICATION

Data Splitting

```
X = multi_data.drop(columns=['label'],axis=1)
Y = multi_data['label']
```

Linear Regression

```
lr_multi = LinearRegression()
lr_multi.fit(X_train, y_train)
```

```
LinearRegression()
```

Multi Layer Perceptron

```
mlp_bin = MLPClassifier(random_state=123, solver='adam', max_iter=8000)
```

```
mlp_bin.fit(X_train,y_train)
```

```
MLPClassifier(max_iter=8000, random_state=123)
```

Linear Support Vector Machine

```
lsvm_multi = SVC(kernel='linear',gamma='auto')
lsvm_multi.fit(X_train,y_train)
```

```
SVC(gamma='auto', kernel='linear')
```

K Nearest Neighbor Classifier

```
knn_multi = KNeighborsClassifier(n_neighbors=5)
knn_multi.fit(X_train,y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

Random Forest Classifier

```
rf_multi = RandomForestClassifier(random_state=50)
rf_multi.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=50)
```

Decision Tree Classifier

```
dt_multi = DecisionTreeClassifier(random_state=123)
dt_multi.fit(X_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=123)
```

Multi Layer Perceptron

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

```
MLPClassifier
MLPClassifier(max_iter=8000, random_state=123)
```

Fig: 6.2.5. Code snippets of multi-class classification

8. Real and Predicted Data: Real and predicted data are saved to CSV files for analysis.
9. Plotting: Plots between real and predicted data are created and saved.

6.3 Model - TON_IOT Dataset

Data preprocessing:

During the data preprocessing phase, several steps are usually performed to prepare the data set for modeling. These steps may include dealing with missing values, recording categorical variables, and maximizing numerical functions. In the prepared snippet, the StandardScaler from scikit-research is used to scale the numeric features to zero mean and unit variance, a common preprocessing step to ensure that the power is at a comparable scale for model overall performance of high quality

Logistic Regression

```
logr_multi = LogisticRegression(random_state=123, max_iter=5000,solver='newton-cg',multi_class='multinomial')
logr_multi.fit(X_train,y_train)
```

```
LogisticRegression(max_iter=5000, multi_class='multinomial', random_state=123,
                   solver='newton-cg')
```

Out[36]:

	door_state	sphone_signal	time_seconds
5519	0	4	14747
347625	1	6	65384
342985	0	4	70499
13637	0	4	45737
102199	0	4	15087
...	...	...	...
110268	0	4	15013
259178	0	4	62517
365838	0	4	83946
131932	0	4	15150
121958	1	6	15112

473156 rows × 3 columns

In [37]:
```
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
dt_predictions = dt_classifier.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_predictions)
print("Decision Tree Accuracy:", dt_accuracy)
```

```
Decision Tree Accuracy: 0.8651365288697269
```

In [38]:
```
# Get classification report
classification_rep = classification_report(y_test, dt_predictions)

# Print classification report
print("Decision Tree Classification Report:")
print(classification_rep)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

    backdoor       0.36      0.37      0.36      7228
        ddos       0.31      0.36      0.34      2032
   injection       0.19      0.21      0.20      1199
      normal       0.94      0.94      0.94    103123
    password       0.42      0.38      0.40      3792
  ransomware       0.15      0.12      0.14       577
    scanning       0.00      0.00      0.00       114
         xss       0.44      0.48      0.46       225

    accuracy                           0.87    118290
   macro avg       0.35      0.36      0.35    118290
weighted avg       0.87      0.87      0.87    118290
```

In [42]:
```
# Get classification report
classification_rep = classification_report(y_test, gb_predictions)

# Print classification report
print("Random Forest Classification Report:")
print(classification_rep)
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

    backdoor       0.90      0.28      0.43      7228
        ddos       0.33      0.00      0.00      2032
   injection       0.41      0.07      0.11      1199
      normal       0.93      0.98      0.95    103123
    password       0.37      0.74      0.49      3792
  ransomware       0.00      0.00      0.00       577
    scanning       0.00      0.00      0.00       114
         xss       0.50      0.01      0.03       225

    accuracy                           0.89    118290
   macro avg       0.43      0.26      0.25    118290
weighted avg       0.89      0.89      0.87    118290
```

In [44]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
k = 5  # Choose an appropriate value for K
knn_classifier = KNeighborsClassifier(n_neighbors=k, metric='euclidean')
knn_classifier.fit(X_train_scaled, y_train)
y_pred = knn_classifier.predict(X_test_scaled)
knn_accuracy = accuracy_score(y_test, y_pred)
print("KNNs Accuracy:", knn_accuracy)
```

```
KNNs Accuracy: 0.8611294276777411
```

In [45]:
```
classification_rep = classification_report(y_test, y_pred)
print("KNNs Classifier Report:")
print(classification_rep)
```

```
KNNs Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.42      0.38      0.40      7228
        ddos       0.02      0.01      0.02      2032
   injection       0.21      0.24      0.22      1199
      normal       0.93      0.95      0.94    103123
    password       0.29      0.25      0.27      3792
  ransomware       0.20      0.12      0.15       577
    scanning       0.00      0.00      0.00       114
         xss       0.26      0.21      0.23       225

    accuracy                           0.86    118290
   macro avg       0.29      0.27      0.28    118290
weighted avg       0.85      0.86      0.86    118290
```

In [ ]:
```
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.utils import to_categorical

# Encode target labels using one-hot encoding
y_train_encoded = to_categorical(LabelEncoder().fit_transform(y_train))
y_test_encoded = to_categorical(LabelEncoder().fit_transform(y_test))

# Determine the number of classes in your dataset
num_classes = len(np.unique(y_train))  # Assuming y_train contains the target labels

# Define LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(units=num_classes, activation='softmax'))  # Adjust for the number of classes
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_lstm, y_train_encoded, epochs=100, batch_size=32, validation_data=(X_test_lstm, y_test_encoded))

# Evaluate the model
loss, accuracy = model.evaluate(X_test_lstm, y_test_encoded)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

```
Epoch 1/100
16257/16257 [==============================] - 140s 8ms/step - loss: 0.6071 - accuracy: 0.8605 - val_loss: 0.6041 - val_accurac
y: 0.8607
Epoch 2/100
16257/16257 [==============================] - 131s 8ms/step - loss: 0.6043 - accuracy: 0.8608 - val_loss: 0.6028 - val_accurac
y: 0.8607
Epoch 3/100
16257/16257 [==============================] - 134s 8ms/step - loss: 0.6039 - accuracy: 0.8608 - val_loss: 0.6046 - val_accurac
y: 0.8607
Epoch 4/100
16257/16257 [==============================] - 135s 8ms/step - loss: 0.6039 - accuracy: 0.8609 - val_loss: 0.6074 - val_accurac
y: 0.8607
Epoch 5/100
16257/16257 [==============================] - 136s 8ms/step - loss: 0.6037 - accuracy: 0.8609 - val_loss: 0.6069 - val_accurac
y: 0.8607
Epoch 6/100
 7073/16257 [============>.................] - ETA: 1:04 - loss: 0.6030 - accuracy: 0.8612
```

In [ ]:
```
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.utils import to_categorical

# Encode target labels using one-hot encoding
y_train_encoded = to_categorical(LabelEncoder().fit_transform(y_train))
y_test_encoded = to_categorical(LabelEncoder().fit_transform(y_test))

# Determine the number of classes in your dataset
num_classes = len(np.unique(y_train))  # Assuming y_train contains the target labels

# Define LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(units=num_classes, activation='softmax'))  # Adjust for the number of classes
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_lstm, y_train_encoded, epochs=100, batch_size=32, validation_data=(X_test_lstm, y_test_encoded))

# Evaluate the model
loss, accuracy = model.evaluate(X_test_lstm, y_test_encoded)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

In [117]:
```
thermostat.head()
thermostat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442228 entries, 0 to 442227
Data columns (total 6 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   date                 437607 non-null  object
 1   time                 437607 non-null  object
 2   current_temperature  442228 non-null  float64
 3   thermostat_status    442228 non-null  int64
 4   label                442228 non-null  int64
 5   type                 442228 non-null  object
dtypes: float64(1), int64(2), object(3)
memory usage: 20.2+ MB
```

Fig: 6.3.1 Code snippets of data preprocessing in TON-IOT dataset

Data visualisation:

Data visualisation performs a vital function in knowing the characteristics and styles inside the dataset. Visualisation techniques like scatter plots, histograms, and pair plots are generally used to discover records distribution, pick out correlations, and determine feature importance. In the supplied snippet, a pair plot is generated the usage of seaborn to visualise relationships among pairs of features in the binary dataset, with unique shades representing distinctive classes.

```
In [142]: rf_classifier = RandomForestClassifier(random_state=42)
          rf_classifier.fit(X_train, y_train)
          rf_predictions = rf_classifier.predict(X_test)
          rf_accuracy = accuracy_score(y_test, rf_predictions)
          print("Random Forest Accuracy:", rf_accuracy)

          Random Forest Accuracy: 0.9944943828864505

In [143]: # Get classification report
          classification_rep = classification_report(y_test, rf_predictions)

          # Print classification report
          print("Random Forest Classification Report:")
          print(classification_rep)

          Random Forest Classification Report:
                        precision    recall  f1-score   support

              backdoor       0.97      0.98      0.98      7068
                  ddos       0.95      0.97      0.96      3054
             injection       0.99      1.00      1.00      1913
                normal       1.00      1.00      1.00    111932
              password       0.99      0.99      0.99      5186
            ransomware       0.96      0.94      0.95       596
              scanning       0.98      0.95      0.97       105
                   xss       1.00      1.00      1.00       195

              accuracy                           0.99    130049
             macro avg       0.98      0.98      0.98    130049
          weighted avg       0.99      0.99      0.99    130049

In [144]: gb_classifier = GradientBoostingClassifier(random_state=42)
          gb_classifier.fit(X_train, y_train)
          gb_predictions = gb_classifier.predict(X_test)
          gb_accuracy = accuracy_score(y_test, gb_predictions)
          print("Gradient Boosting Accuracy:", gb_accuracy)

          Gradient Boosting Accuracy: 0.8996224499996155

In [145]: # Get classification report
          classification_rep = classification_report(y_test, gb_predictions)

          # Print classification report
          print("Gradient Boosting Classification Report:")
          print(classification_rep)

          Gradient Boosting Classification Report:
                        precision    recall  f1-score   support

              backdoor       0.90      0.25      0.39      7068
                  ddos       0.85      0.35      0.50      3054
             injection       0.79      0.75      0.77      1913
                normal       0.90      0.99      0.95    111932
              password       0.84      0.16      0.26      5186
            ransomware       0.90      0.75      0.82       596
              scanning       0.57      0.52      0.54       105
                   xss       1.00      0.93      0.96       195

              accuracy                           0.90    130049
             macro avg       0.84      0.59      0.65    130049
          weighted avg       0.90      0.90      0.87    130049
```

Fig: 6.3.2. Code snippets of data visualization in ton-iot dataset

Feature selection and accuracy:

Feature selection is an important step in system studying to discover the most relevant capabilities that make a contribution to the predictive energy of the version while lowering dimensionality and computational complexity. Various strategies along with statistical exams, feature importance from tree-primarily based models, or model-based totally selection strategies may be hired. In the furnished snippet, the SelectKBest approach with chi-squared check is used to choose the top of features which might be maximum predictive of the target variable inside the binary dataset, aiming to enhance version performance and performance.

```
In [153]: lda_classifier = LinearDiscriminantAnalysis()
          lda_classifier.fit(X_train, y_train)
          lda_predictions = lda_classifier.predict(X_test)
          lda_accuracy = accuracy_score(y_test, lda_predictions)
          print("Linear Discriminant Analysis Accuracy:", lda_accuracy)

          Linear Discriminant Analysis Accuracy: 0.8606909703265693

In [154]: classification_rep = classification_report(y_test, lda_predictions)

          # Print classification report
          print("Linear Discriminant Analysis Report:")
          print(classification_rep)

          Linear Discriminant Analysis Report:
                        precision    recall  f1-score   support

              backdoor       0.00      0.00      0.00      7068
                  ddos       0.00      0.00      0.00      3054
             injection       0.00      0.00      0.00      1913
                normal       0.86      1.00      0.93    111932
              password       0.00      0.00      0.00      5186
            ransomware       0.00      0.00      0.00       596
              scanning       0.00      0.00      0.00       105
                   xss       0.00      0.00      0.00       195

              accuracy                           0.86    130049
             macro avg       0.11      0.12      0.12    130049
          weighted avg       0.74      0.86      0.80    130049

In [155]: qda_classifier = QuadraticDiscriminantAnalysis()
          qda_classifier.fit(X_train, y_train)
          qda_predictions = qda_classifier.predict(X_test)
          qda_accuracy = accuracy_score(y_test, qda_predictions)
          print("Quadratic Discriminant Analysis Accuracy:", qda_accuracy)

          Quadratic Discriminant Analysis Accuracy: 0.8487339387461649

In [156]: classification_rep = classification_report(y_test, qda_predictions)

          # Print classification report
          print("Quadratic Discriminant Analysis Report:")
          print(classification_rep)

          Quadratic Discriminant Analysis Report:
                        precision    recall  f1-score   support

              backdoor       0.00      0.00      0.00      7068
                  ddos       1.00      0.00      0.00      3054
             injection       0.00      0.00      0.00      1913
                normal       0.86      0.98      0.92    111932
              password       0.00      0.00      0.00      5186
            ransomware       0.13      0.47      0.20       596
              scanning       0.59      0.70      0.64       105
                   xss       1.00      0.30      0.46       195

              accuracy                           0.85    130049
             macro avg       0.45      0.31      0.28    130049
          weighted avg       0.77      0.85      0.79    130049
```

Fig: 6.3.3. Code snippets of feature selection in ton-iot dataset

6.4 Model - PPGO_IOT Dataset

Data Preparation:

The furnished code begins with information education steps. It splits the data as training and checking out puts the usage of `train_test_split` feature from scikit-learn. It additionally encodes the goal labels using one-warm encoding to prepare them for LSTM model schooling. Additionally, the entered facts is reshaped to fit the LSTM model's input shape, assuming a time series shape.

```
In [11]: dt_classifier = DecisionTreeClassifier(random_state=42)
         dt_classifier.fit(X_train, y_train)
         dt_predictions = dt_classifier.predict(X_test)
         dt_accuracy = accuracy_score(y_test, dt_predictions)
         print("Decision Tree Accuracy:", dt_accuracy)

         Decision Tree Accuracy: 0.7756694147305307

In [12]: # Get classification report
         classification_rep = classification_report(y_test, dt_predictions)

         # Print classification report
         print("Decision Tree Classification Report:")
         print(classification_rep)

         Decision Tree Classification Report:
                       precision    recall  f1-score   support

            backdoor       0.12      0.12      0.12      7192
                ddos       0.08      0.07      0.08      2090
           injection       0.22      0.24      0.23      1415
              normal       0.88      0.89      0.88    100027
            password       0.19      0.17      0.18      5707
          ransomware       0.12      0.13      0.13       577
                 xss       0.14      0.14      0.14       408

            accuracy                           0.78    117416
           macro avg       0.25      0.25      0.25    117416
        weighted avg       0.77      0.78      0.77    117416

In [13]: rf_classifier = RandomForestClassifier(random_state=42)
         rf_classifier.fit(X_train, y_train)
         rf_predictions = rf_classifier.predict(X_test)
         rf_accuracy = accuracy_score(y_test, rf_predictions)
         print("Random Forest Accuracy:", rf_accuracy)

In [15]: gb_classifier = GradientBoostingClassifier(random_state=42)
         gb_classifier.fit(X_train, y_train)
         gb_predictions = gb_classifier.predict(X_test)
         gb_accuracy = accuracy_score(y_test, gb_predictions)
         print("Gradient Boosting Accuracy:", gb_accuracy)

         Gradient Boosting Accuracy: 0.8518941200517817

In [16]: classification_rep = classification_report(y_test, gb_predictions)

         # Print classification report
         print("Gradient Boosting Classifier Report:")
         print(classification_rep)

         Gradient Boosting Classifier Report:
                       precision    recall  f1-score   support

            backdoor       0.00      0.00      0.00      7192
                ddos       1.00      0.00      0.00      2090
           injection       0.00      0.00      0.00      1415
              normal       0.85      1.00      0.92    100027
            password       1.00      0.00      0.00      5707
          ransomware       0.00      0.00      0.00       577
                 xss       0.00      0.00      0.00       408

            accuracy                           0.85    117416
           macro avg       0.41      0.14      0.13    117416
        weighted avg       0.79      0.85      0.78    117416
```

Fig: 6.4.1. Data preparation in PPGO- iot model

Both Decision Tree and Random Forest are ensemble learning methods used for category duties.Random Forest is an extension of Decision Trees, wherein multiple decision bushes are trained on distinct subcategory of the information and their forecasts are averaged to enhance generalization and decrease overfitting.In this situation, the Random Forest model outperforms the Decision Tree version in phrases of correctneses, reaching about seventy eight.55% accuracy compared to 77.57% for the Decision Tree.The development in accuracy may be attributed to the ensemble nature of Random Forest, which mixes more than one selection

bushes to make more strong predictions and reduce the variance related to individual decision timber.The classification document gives specific insights into the version's performance throughout distinct classes, highlighting precision, take into account, and F1-rating metrics for every class, in conjunction with average accuracy.

```
In [21]: from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = accuracy_score(y_test, nb_predictions)
print("Naive Bayes Accuracy:", nb_accuracy)

Naive Bayes Accuracy: 0.8519026367786332

In [22]: classification_rep = classification_report(y_test, nb_predictions)

# Print classification report
print("Naive Bayes Classifier Report:")
print(classification_rep)

Naive Bayes Classifier Report:
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7192
        ddos       0.00      0.00      0.00      2090
   injection       0.00      0.00      0.00      1415
      normal       0.85      1.00      0.92    100027
    password       0.00      0.00      0.00      5707
  ransomware       0.00      0.00      0.00       577
         xss       0.00      0.00      0.00       408

    accuracy                           0.85    117416
   macro avg       0.12      0.14      0.13    117416
weighted avg       0.73      0.85      0.78    117416

In [23]: from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda_classifier = LinearDiscriminantAnalysis()
lda_classifier.fit(X_train, y_train)
lda_predictions = lda_classifier.predict(X_test)
lda_accuracy = accuracy_score(y_test, lda_predictions)
print("Linear Discriminant Analysis Accuracy:", lda_accuracy)

Linear Discriminant Analysis Accuracy: 0.8519026367786332

In [24]: classification_rep = classification_report(y_test, lda_predictions)

# Print classification report
print("Linear Discriminant Analysis Report:")
print(classification_rep)

Linear Discriminant Analysis Report:
              precision    recall  f1-score   support

    backdoor       0.00      0.00      0.00      7192
        ddos       0.00      0.00      0.00      2090
   injection       0.00      0.00      0.00      1415
      normal       0.85      1.00      0.92    100027
    password       0.00      0.00      0.00      5707
  ransomware       0.00      0.00      0.00       577
         xss       0.00      0.00      0.00       408

    accuracy                           0.85    117416
   macro avg       0.12      0.14      0.13    117416
weighted avg       0.73      0.85      0.78    117416
```

Fig: 6.4.2. LDA in PPGO-iot model

LSTM Model Definition:

In this phase, a Sequential model describes the usage of Keras. The version architecture consists of LSTM layers followed by means of a Dense layer with softmax activation for multi-elegance class. The version is gathered with the categorical cross-entropy loss and Adam optimizer, which might be common choices for training neural networks.

Perpetual Pigeon Galvanized Optimization (PPGO) Algorithm:

The PPGO set of rules is employed to optimize the weights of the LSTM version iteratively. The `perpetual_pigeon_galvanized_optimization` characteristic orchestrates this method, updating the model's parameters for a distinctive range of iterations. During each new release, the objective function evaluates the version's overall performance at the check set the use of the modern-day set of weights.

Evaluation

After optimization, the model's overall performance is evaluated when taking a look at the usage of the optimized weights. Take a look at loss and accuracy metrics are computed and revealed to assess the very last overall performance of the LSTM model. This step gives insights into how properly the model generalizes to unseen data and demonstrates the effectiveness of the PPGO set of rules in optimizing the LSTM version for progressed accuracy.

In [27]:

```
import numpy as np
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Define PPGO algorithm
def ppgo_initialize(num_parameters):
    # Initialize parameters randomly or using some strategy
    initial_parameters = np.random.randn(num_parameters)  # Example: Random initialization
    return initial_parameters

def ppgo_update(current_parameters):
    # Implement parameter update logic based on PPGO algorithm
    # This could involve various operations such as mutation, crossover, etc.
    updated_parameters = current_parameters + np.random.randn(len(current_parameters)) * 0.1  # Example: Random update
    return updated_parameters

def perpetual_pigeon_galvanized_optimization(objective_function, initial_parameters, num_iterations):
    current_parameters = initial_parameters
    for i in range(num_iterations):
        updated_parameters = ppgo_update(current_parameters)
        loss = objective_function(updated_parameters)
        print(f"Iteration {i+1}, Loss: {loss}")
        current_parameters = updated_parameters

# Example data (replace with your own)
# Assuming X and y are your input features and target labels
# Assuming X_train, X_test, y_train, y_test are defined

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Encode target labels using one-hot encoding
label_encoder = LabelEncoder()
y_train_encoded = to_categorical(label_encoder.fit_transform(y_train))
y_test_encoded = to_categorical(label_encoder.transform(y_test))

# Reshape the input data for LSTM
X_train_lstm = X_train.values.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test_lstm = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

Fig: 6.4.3. PPGO model

CHAPTER 7

TESTING

In a cyber security context, we cannot avoid testing the windows that are connected to the system of the machine to find and fix the vulnerabilities surfaced beforehand. The above tests would help determine whether the security system is affected by current attacks or not as well as to what extent the system can foresee the attacks that could happen under the given operating environmental and security conditions. Cyber Security is a process which takes cyber evaluations of a system’s ability to operate under various attacks, and then builds a model on what measures need to be taken for a successful defence. We are very confident that we will be able to solve the problem by revealing the level of possible harm within the named system. These consequences may be immediately seen post an attack or they might remain hidden until discovered after a life under threat, upon which purposeful screening of the comprehensive system will be executed to prove its safety.

7.1 TESTING TYPES

7.1.1 MANUAL TESTING

Traditional testing is test cases involving testers executing manual tests without any automated tools support. To find out all sorts of bugs, problems and defects within the soft-ware application, the ain of Manual Testing is perform testing. It performs the fundamental purpose of the black-box testing and helps to unearth the crucial loopholes. Manual testing must be the first test step for any new applications, in order before automation can be implemented. It is definitely more complicated, but it is indispensable to analyse automation possibilities as well as the need of learning test tools is not necessarily here. As a result, it's indicated within Soft-ware Testing Fundamentals that "100% Automation is impossible," and thereby manual testing is carried out.

- **White Box Testing:** In this test environment we essentially know the way the system is expected to work leveraging open-ended nature of the program to better understand how different components interact in their process. This approach entails running the individual patterns along with the lines and the respective branches since the ultimate goal is the confirmation of its correctness. In White Box Testing, however, it is the developers who execute the testing. Developers need to know the programming languages and code structures, before being able to perform this kind of testing.
- **Black Box Testing:** White Box Testing as compared to Black Box Testing is implemented like an isolated black box where testers are unable to know any internal details. Tester's shift focuses on the fundamental steps involved in a given business process and critically assess whether appropriate input processes have been successfully accomplished to meet the desired outputs. Black Box Testing is mainly user-centered and does not have the program code familiarity requirement, therefore it is aimed, in most cases, at the testers without programming expertise.

7.1.2 AUTOMATED TESTING

Through the employment of automatic testing the system utilised automated test software tools for conducting the test cases. Automation contrasts manual testing only with the automation being the process where the test cases are executed using automated test execution. Automation tools can help developers execute tests that intake test data, compare expected and actual results, and produce detailed test reports. Yet, the Utilisation of Software Test Automation should be noted, because it would potentially require investments in terms of both finances and resources.

- **White Box Testing:** Automated WhiteBox Testing is a process of applying automated tools that are designed to work based on the internal architecture of the software systems. Through this process, the internal structure is analysed to detect issues like code coverage and adhesion to the coding standards. These tools can therefore automatically generate test cases based on the architecture and logic of code to ensure the code for each code path is fully attempted.
- **Black Box Testing:** In case of Automated Blackbox Testing we will be using some tools that mimic the user behaviours directly interacting with the software. These tools

typically perform automated execution of the previously put in tests, validating outputs against the anticipated results and follow-up with generating reports. Such technology is extremely beneficial for regression testing, which is the process of checking that any changes didn't cause problems in the code and thus automated Black Box Testing can certainly save time and energy of test developers.

7.2 SOFTWARE TESTING METHODS

7.2.1 BLACKBOX TESTING

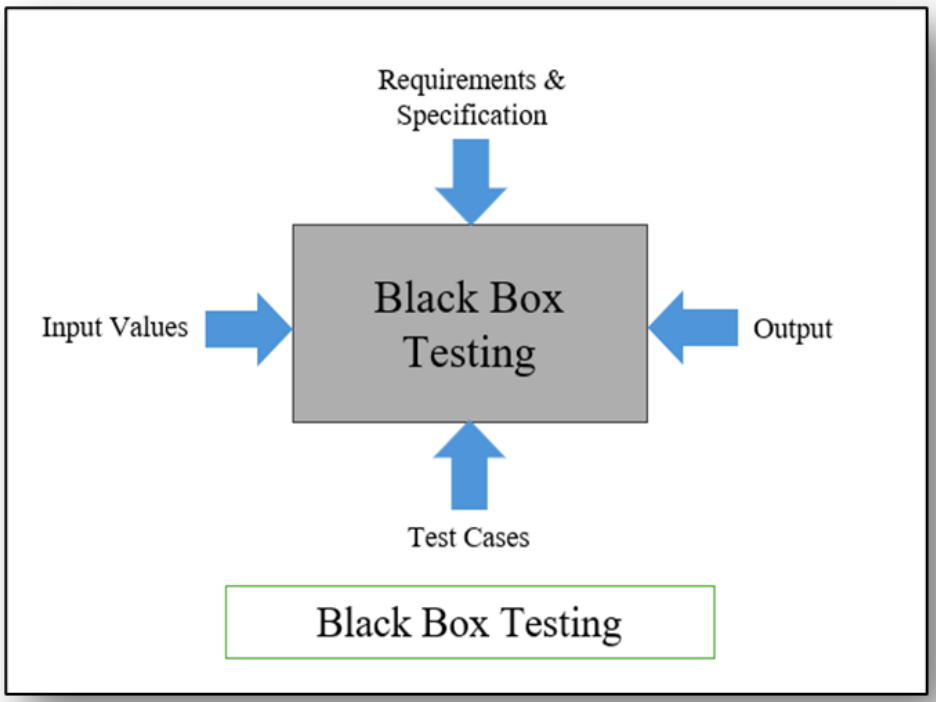


Fig 7.2.1. Black Box Testing

Black box testing is a type of software testing that checks how a program works without checking its core coding or structure. The customer stated requirements specification acts as the primary source of black box testing data. This approach, as shown in Figure 7.2.1, requires that a processor chooses a function, provides an input value to evaluate its efficiency, and determines whether the function provides the required output if they are correctly entered, the function passes the test ; Otherwise, it fails. After the results are reported to the development

team, the test team proceeds to test the next program. If serious issues are found after testing all the components, they are reported to the development team so that they can be addressed.

7.2.2 GRAY BOX TESTING

The "gray box" testing, a software testing technique, allows users to test an application while only partially knowing its inner workings. Because test programs are executed at the same level of performance as black box testing and access to the internal code is required to build test cases, it is a hybrid of white box and black box testing. For example, if a tester finds a bug during testing, it stops the code and reruns the test in real time. To increase the cost of testing, it focuses on every aspect of any complex software program. It provides the ability to analyse the internal regulatory framework in addition to the business level. It is commonly used in penetration and integration testing. This test method combines black box and white box testing. When performing a black box test, the experimenter is unsighted to the code. They have data about the intended outcome of the information provided. When conducting a white box test, the tester is familiar with the code. Gray-box testers have some knowledge of code, but not all of it. An overview of the gray box test is given in Figure 7.2.2.

Fig 7.2.2 Grey Box Testing

7.2.3 WHITE BOX TESTING

Using this technique called "white box testing," testers can test and validate the functionality of a software system, including its code, infrastructure, and system integration White box testing is a key component of an automated build processes in a modern continuous integration/ development product for continuous delivery (CI/CD). SAST), a technique that automatically validates the source code or binaries and provides input for potential bugs and vulnerabilities, is often referred to in terms of white-box testing White-box testing is a form of acquisition

testing test data comes from the structure of a program and its logic and code . Clear box testing, open box testing, rationality testing, methodological testing, and systematic testing are new terms for glass box testing. Fig. 7.3 gives the flow of white box testing.

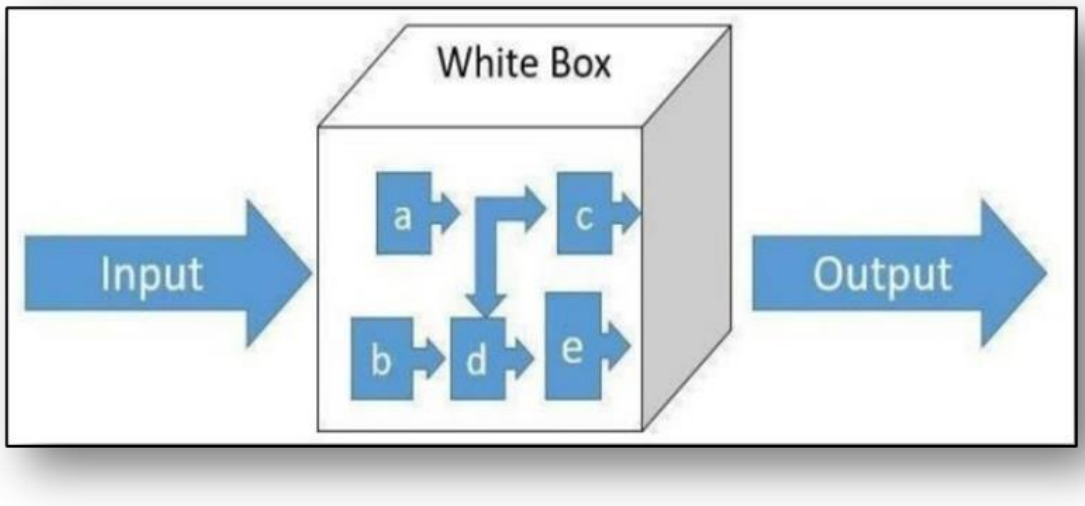


Fig 7.2.3. White Box Testing

7.3 TESTING LEVELS

7.3.1 NON-FUNCTIONAL TESTING

Non-functional testing involves the assessment of the dependability, performance, and accountability of the software from user's point of view needs. The main purpose of the assessment is measuring performance as opposed to functionality based on specific KPIs. Functional testing is different from non-functional testing because the latter looks at what resides beyond functionality and is vital since customer satisfaction is difficult to attain.

7.3.1.1 PERFORMANCE TESTING

Performance testing is a tool for testing how a system works under given loads because it does not look for finding bugs in the software. Through using different types of evaluation against specific criteria, it helps developers get diagnostic insights and solve inefficiency issues efficiently.

7.3.1.2 STRESS TESTING

Stress Testing ensures the robustness and adaptability of software platforms working in very stressful environments with high system load. The main agenda of it is to evaluate the software robustness and error handling competences, as a result software stability even in hostile conditions.

7.3.1.3 SECURITY TESTING

Security Testing offers solutions for system vulnerabilities to keep data and properties secure from any intrusions. It identifies vulnerabilities and gaps in security and shields information or company's reputation against unlawful access or cyberattacks.

7.3.1.4 PORTABILITY TESTING

The portability testing techniques provide information about the appropriateness of a software or app for the transfer between different systems be it hardware, software, or operating environments. It defines how gracefully an application can be embedded into other systems unconditionally and aligns the non-functional requirement of portability.

7.3.1.5 USABILITY TESTING

Usability Testing, in short, User Experience (UX) Testing, is a process of evaluating the user-friendliness and handy nature of the software applications through the user's point of view or through the end-users. This approach keeps target end-users at the middle of the development process, consequently discovering usability errors early in the development, which also sheds some light on the customers' expectations during the initial design condition of the software development life cycle (SDLC).

7.3.2 FUNCTIONAL TESTING

Functional Testing is used to ensure that an application's functionality is suitable for the requirements. Black-box testing gives the assurance that every function operates as it must, thus the application's functional nature is confirmed. Functional Testing is an indispensable tool for ensuring that an application accomplishes its intended purposes.

- **Whitebox Testing:** In Whitebox Testing, part of Functional testing, we examine the inner structure and the logic of the software, making sure every function works as it should. Testers review code paths and logic flows substantiation that the software processes correctly.
- **Black Box Testing:** Besides Functional Testing applies also Black Box Testing and the software is checked from the view of a user. Testers pay attention to inputs and outputs, denying access to the internal processes and only making sure that the software acts according to given conditions.

7.3.2.1 INTEGRATION TESTING

Integration testing is the process of evaluating the modules or components after they have been integrated to ensure that they function as intended, that is, to test the modules that function well separately and do not have problems when combined. Testing the interfaces between the units/modules is the primary purpose or objective of this testing. First, the different modules are examined separately. Following unit testing, the modules are combined one at a time until all of them are integrated. This allows us to verify whether the requirements are implemented appropriately and examine combinational behaviour.

7.3.2.2 REGRESSION TESTING

Regression testing is a kind of software testing used to check that recent changes to the code or program have positively impacted the existing features. Regression testing is just selecting some or all of the previously run test cases and running them again to make sure the features still function as intended. The purpose of this testing is to make sure that updates the code won;t negatively impact existing features. By ensuring that the most recent code updates are implemented, the old code remains functional.

7.3.2.3 UNIT TESTING

Unit testing is the process for testing software in which each software units—that is, a collection of computer programme modules, usage guidelines, and operating steps—are examined to see if they meet the requirements to work.. It is a kind of testing that involves testing each programme elements .. Software product unit testing is done when an application is being developed. A technique or a single function might be looked as individual components.

7.3.2.4 ALPHA TESTING

Software is tested for problems using a technique called "alpha testing" before the general public or actual users are able to use the product. One type of user acceptability testing is called alpha testing. Just because it was conducted before, close to the conclusion of software development, is known as alpha testing. Quality assurance personnel or homestead software engineers are typically in charge of alpha testing. It is the final testing phase prior to the software's public release.

7.3.2.5 BETA TESTING

Real software programme users in real environments conduct beta testing. Among the several forms of User Acceptance Testing is beta testing. A small number of product end customers are given access to a beta version of the programme, which is intended to gather input on its quality. By obtaining customer approval, beta testing reduces the likelihood that a product will fail and improves its quality. This is the final test run before a product is shipped to clients. Receiving direct feedback from clients is one of the main benefits of beta testing.

7.3 TEST CASES

7.3.1 Testcase-1

Our efforts focus on understanding network behaviour to distinguish between normal operations and potential device attacks. To start with, we used open data that gave us important information about IoT device performance, especially in the face of distributed denial-of-service (DDOS) attacks. This information was collected using Wireshark, a network protocol analyzer that allows us to analyse traffic crossing a network interface. We analysed a variety of factors in this dataset to better understand the nature of the attack and the behaviour of the device. These parameters include type of attack, number of times connections to the same location within a short period of time, status of these connections (successful or failed), types of connection flags, distribution of connections to different services, path access status, and a number of other elements of network security.

To make sense of this amount of data, we tested machine learning models to correctly classify any network behaviour as normal or indicative of an attack. After careful consideration, we chose the Logistic Regression model due to its high accuracy rate, which remained at around 98%. We selected a Logistic Regression model and gave it additional data instances, allowing it to predict whether a device is currently under DOS attack. Surprisingly, the model tended to make accurate predictions, finding some cases of DOS attacks in the data set.

Essentially, our work is a collaborative effort to harness the capabilities of machine learning and network analytics to improve cybersecurity. Using modern methods and techniques, we developed a robust system that can instantly detect and mitigate potential threats, thereby improving the overall security level of the network We hope to improve our approach to network design is administrators' cyber threats effective and reliable by continuing to improve and adapt We will provide the tools and insights needed to protect ourselves from widespread threats.

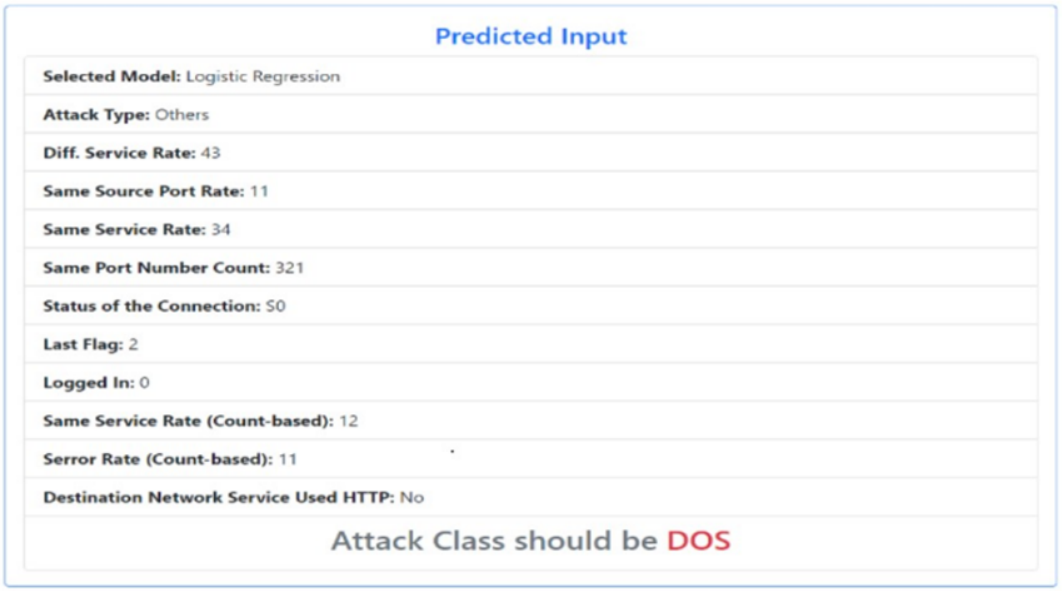


Fig 7.3.1. Predicted DOS Attack

7.3.2 Testcase-2

We conducted additional tests on an open dataset that provided detailed information on IoT device activity during the PROBE attack. This data was obtained using Wireshark, a web traffic analysis tool. To better understand attacks and device responses, we analysed several aspects of the dataset, such as attack type, destination connection frequency, connection status, flags, service distribution, access , and other safety-related attributes

To make sense of the data, we tested different machine learning algorithms to see which could accurately classify deceptive behaviour as normal or indicative of attack. After careful consideration, we chose the Logistic Regression model because of its excellent accuracy (about 98%). We used this model to test new data models and predict whether a device is now under PROBE attack. Surprisingly, the model consistently detected PROBE attacks in the sample with good prediction.

Overall, our program is a concerted effort to use machine learning and network analytics to improve cybersecurity. Using modern techniques, we developed a robust system that can quickly identify and mitigate potential threats, thus improving the security of the entire

network. We intend to continue to enhance and adapt our methods to provide network administrators with the tools and insights they need to address a wide range of cyber threats.

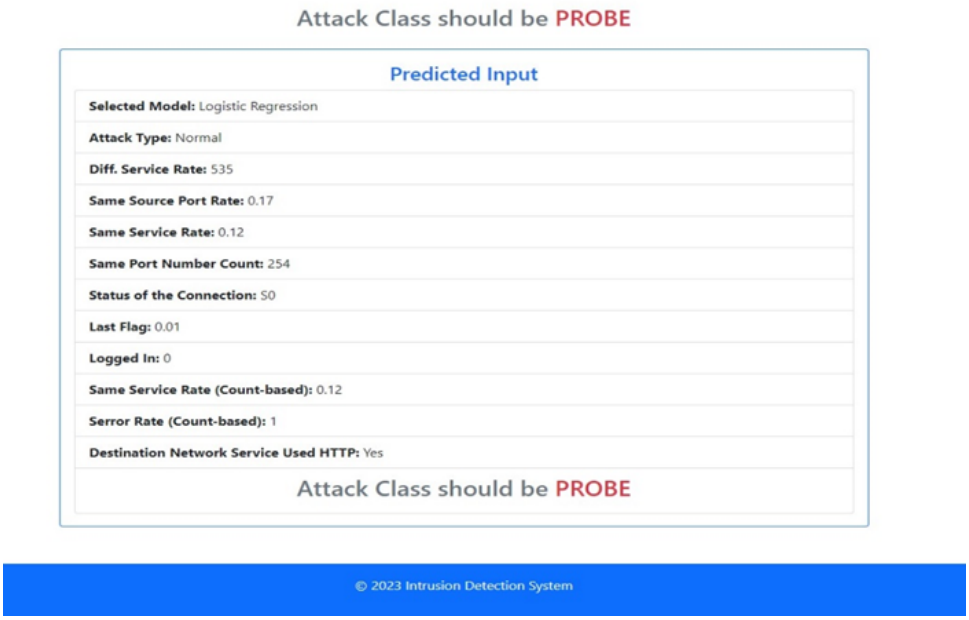


Fig 7.3.2. Predicted PROBE Attack

CHAPTER 8

OUTPUT SCREENS/ RESULTS

8.1 OUTPUT SCREENS

Intrusion Detection System Page: Everything occurs at the "Intrusion Detection System" web page, which is the primary page of our venture. When you go to this web page, you may see a form where you could enter vital information. This data assists us in determining whether an assault is happening on a networked tool. We consult with this as the expected assault elegance. The shape gives many fields where you may enter records inclusive of the sort of attack, how many connections have come about, and different information. Once you've entered all the necessary facts into the form, in reality click the "Predict" button. After you click that button, our system goes to paintings and tells you what sort of attack it believes is taking the vicinity based on the data you have furnished. You can also specify which approach or version to apply for this prediction. So, basically, this page functions as a control centre, allowing you to enter information and benefit insight into capability community threats.

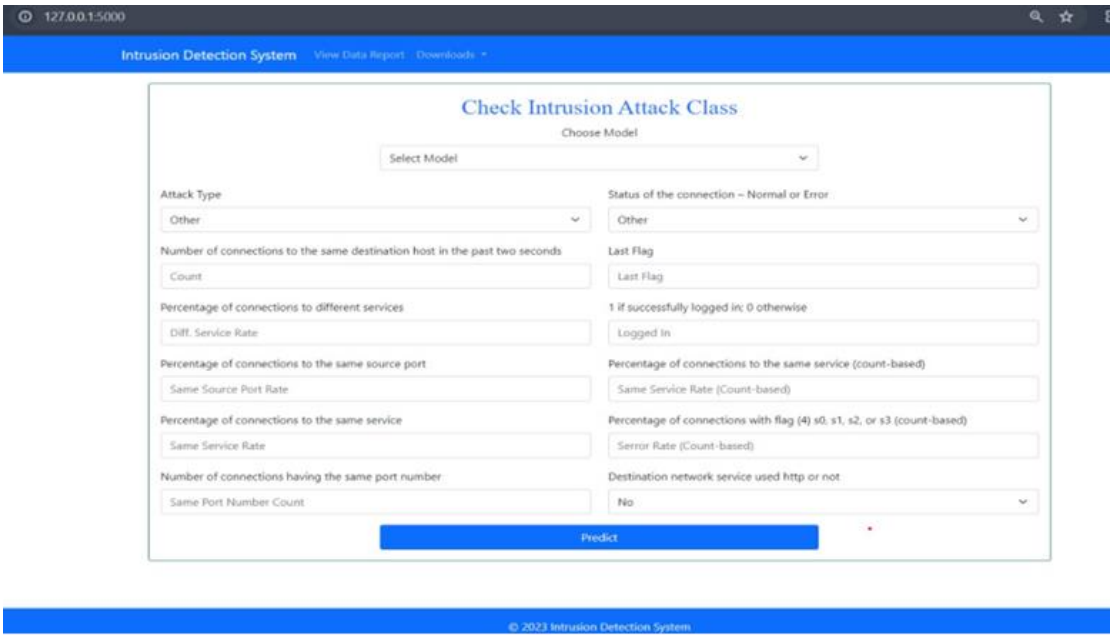
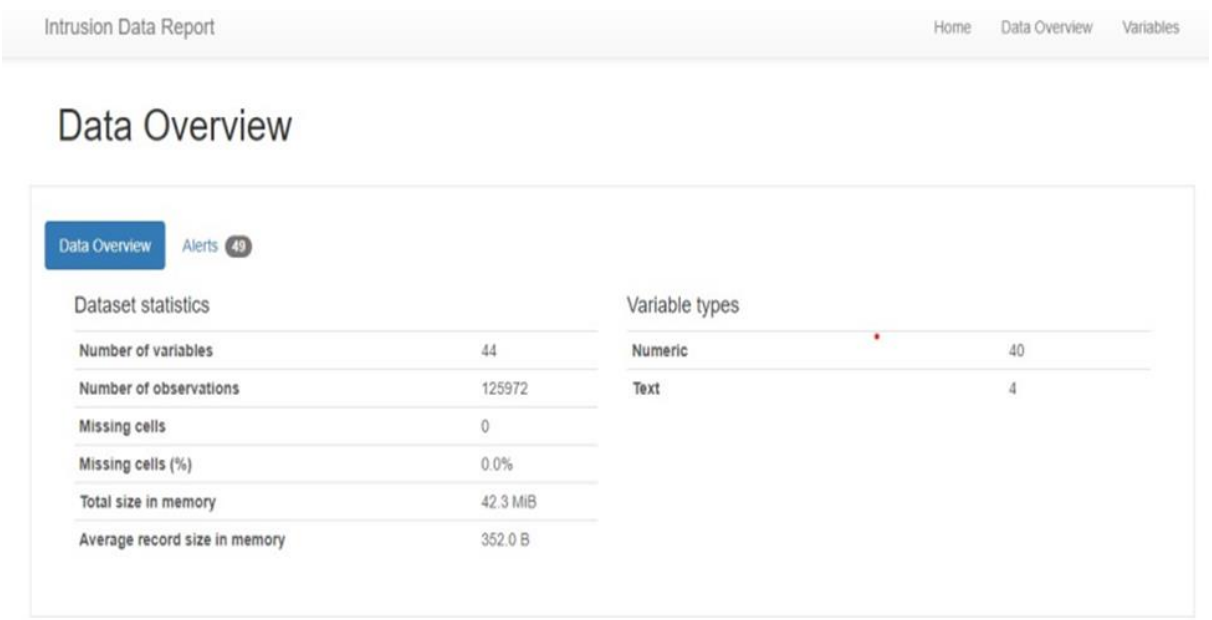


Fig 8.1.1 Intrusion Detection Page

View Data Report Page: The Intrusion Data Report page is vital for making judgments. Here is how it works: When a person chooses a model from the Intrusion Detection System page, they could navigate to this document page to view essential records approximately the dataset

on which the model became skilled. It starts with a desk that shows statistics approximately the dataset, such as how many awesome matters it's far searching at, whether or not any statistics is lacking, how plenty memory it occupies and the variable sorts together with numerical or text. It then makes a speciality of the dataset's crucial features, which include dst_bytes, land, num_failed_logins, logged_in, last_flag, and attack_class. It offers us things like what number of special values there are, the biggest cost, the common fee, variety of zeros and infinite values and what kind of memory everyone takes up. Furthermore, there are bar graphs that illustrate which values appear the most often for every characteristic. So, this page gives us an in depth check out the records on which our version is built, permitting us to better recognize how it generates predictions.



Intrusion Data Report

Home Data Overview Variables

dst_bytes

Real number (ℝ)

SKEWED ZEROS

Distinct	9326	Minimum	0
Distinct (%)	7.4%	Maximum	1309937401
Missing	0	Zeros	67966
Missing (%)	0.0%	Zeros (%)	54.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	19779.27143	Memory size	984.3 KiB

More details

land

Real number (ℝ)

SKEWED ZEROS

Distinct	2	Minimum	0
Distinct (%)	< 0.1%	Maximum	1
Missing	0	Zeros	125947
Missing (%)	0.0%	Zeros (%)	> 99.9%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.0001984567999	Memory size	984.3 KiB

More details

Intrusion Data Report

Home Data Overview Variables

num_failed_logins

Real number (ℝ)

SKEWED ZEROS

Distinct	6	Minimum	0
Distinct (%)	< 0.1%	Maximum	5
Missing	0	Zeros	125850
Missing (%)	0.0%	Zeros (%)	99.9%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.001222493888	Memory size	984.3 KiB

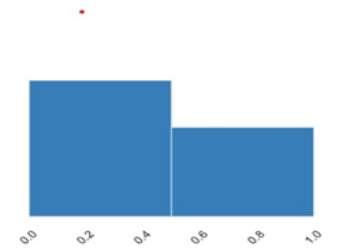
More details

logged_in

Real number (ℝ)

ZEROS

Distinct	2	Minimum	0
Distinct (%)	< 0.1%	Maximum	1
Missing	0	Zeros	76120
Missing (%)	0.0%	Zeros (%)	60.4%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	0.3957387356	Memory size	984.3 KiB



More details

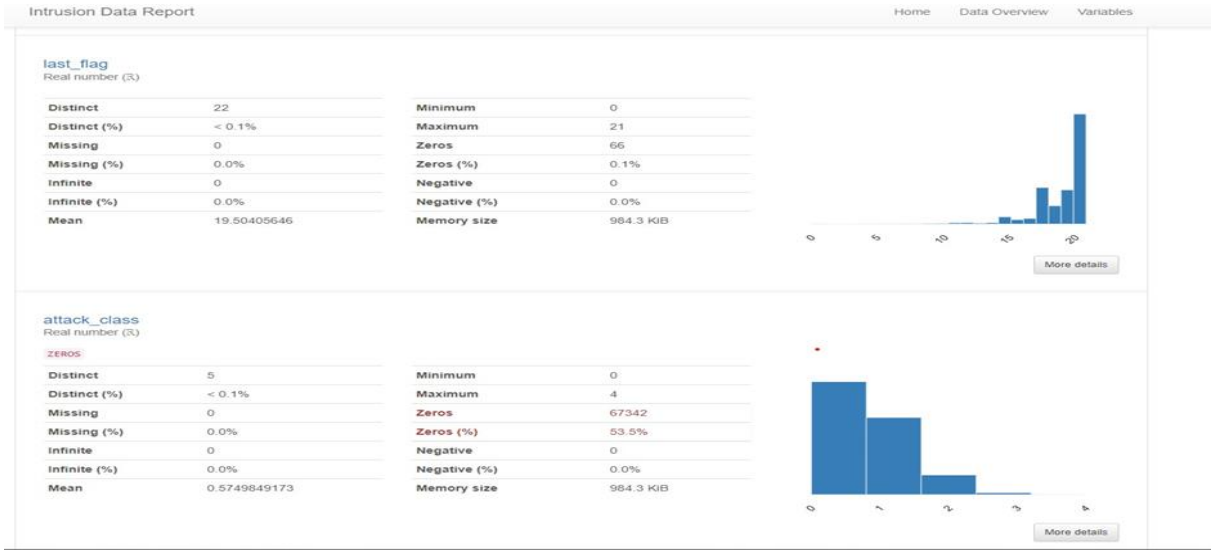


Fig 8.1.2. Intrusion Data Report Page

Alerts Page: The Alert Page, that is included inside the Intrusion Data Report, may be very useful for better comprehending the statistics. It displays us which variables have unbalanced values, indicating that they are no longer balanced. It additionally identifies which variables encompass many zeros and which stay steady, not changing an awful lot. Knowing all of this lets us make better predictions about what is going to appear with the facts. So, basically, it's a heads-up approximately which regions of the facts might also require in addition attention whilst we're attempting to discern things out.

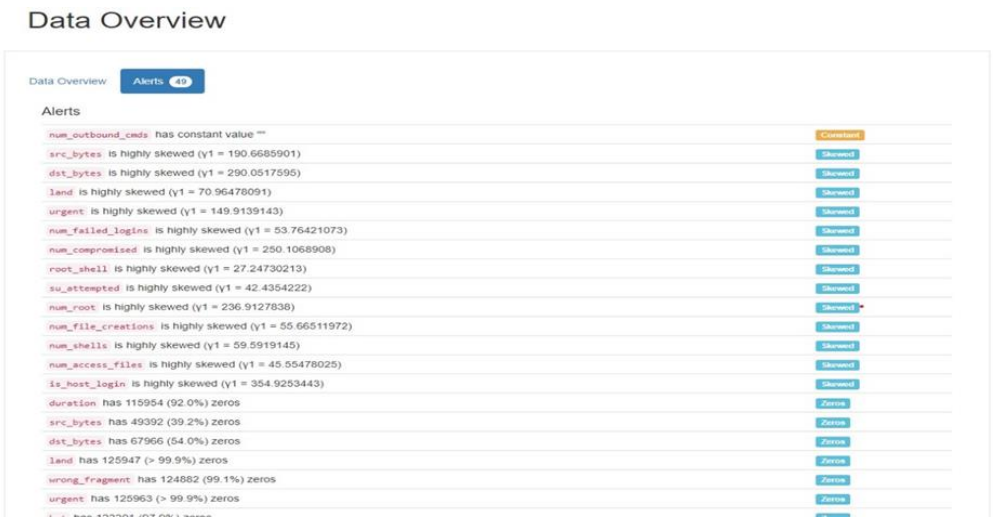


Fig 8.1.3. Alerts Page

Downloads Page: The Downloads Page incorporates crucial files for our venture. You can download the schooling and test data, which might be basically sets of information that our set of rules learns and practices on. You can also achieve correlation data, which shows how diverse portions of records in the dataset are associated with one another. In addition, there may be a facts precis, which presents a speedy overview of the dataset's contents and shape. All of the fabric is in Excel sheet layout, which makes it easy to get right of entry to and apprehend. So, it's a handy approach to achieve a higher know-how of both the dataset and how our version operates.

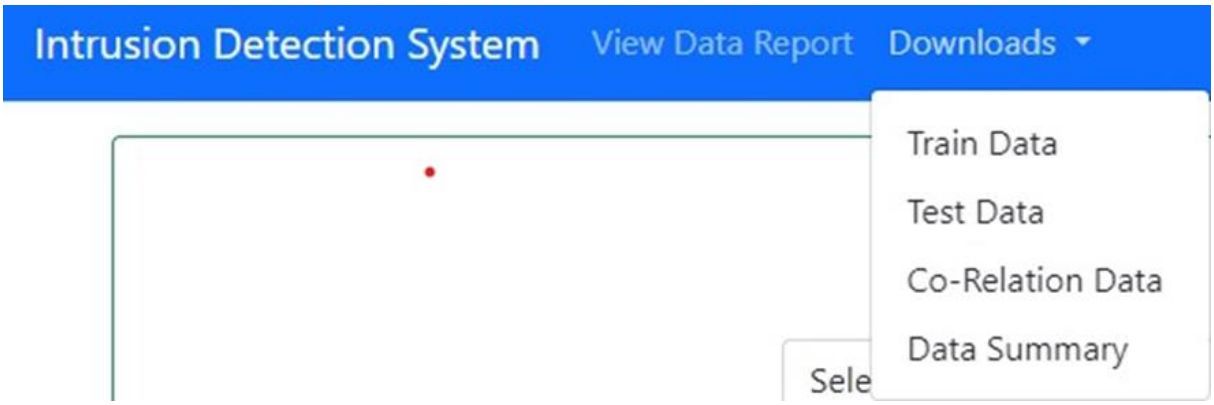


Fig 8.1.4. Downloads Page

These pages work together to assist customers expecting extraordinary types of attacks. They stroll purchasers through the procedure grade by grade, supporting them in identifying capability flaws and better knowledge of the information. It's like a useful adventure in which humans can study information and make predictions in an understandable manner.

8.2 RESULTS

Our undertaking used four datasets: Cyber Intrusion, UNSW, NSL-KDD, and Ton-IoT. To collect deeper insights and locate the maximum influential homes throughout exceptional datasets, we constructed a correlation matrix. This matrix lets us look at the establishments between variables and perceive which of them have the exceptional correlations with every difference. The variables we checked out included beauty label, same service price, vacation spot host company rely, destination host equal company charge, logged-in reputation, host server charge, error fee, carrier errors price, and consider. By analysing those correlations, we

are able to come across patterns and dependencies which can imply cyber infiltration or one-of-a-kind community irregularities. This targeted evaluation permits us to gain a deeper understanding of the dataset's underlying homes and select the most vital features for our prediction fashions. Finally, our technique improves our intrusion detection system's accuracy and efficacy with the aid of the usage of focusing at the most informative and discriminative functions across several datasets, allowing us to better find out and mitigate feasible protection risks. We used a bar graph to decide the maximum critical functions in the datasets, which helped us prioritise variables that had the most important impact on our have a look at.

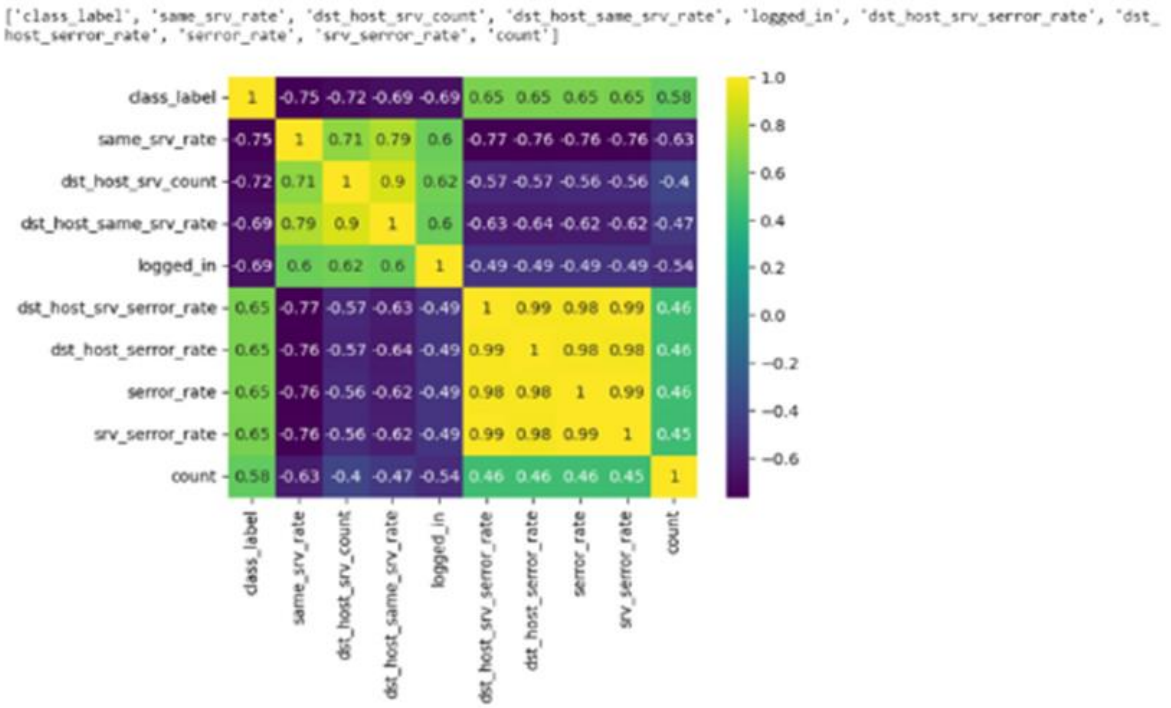


Fig 8.2.1. Correlation Matrix for Feature Selection

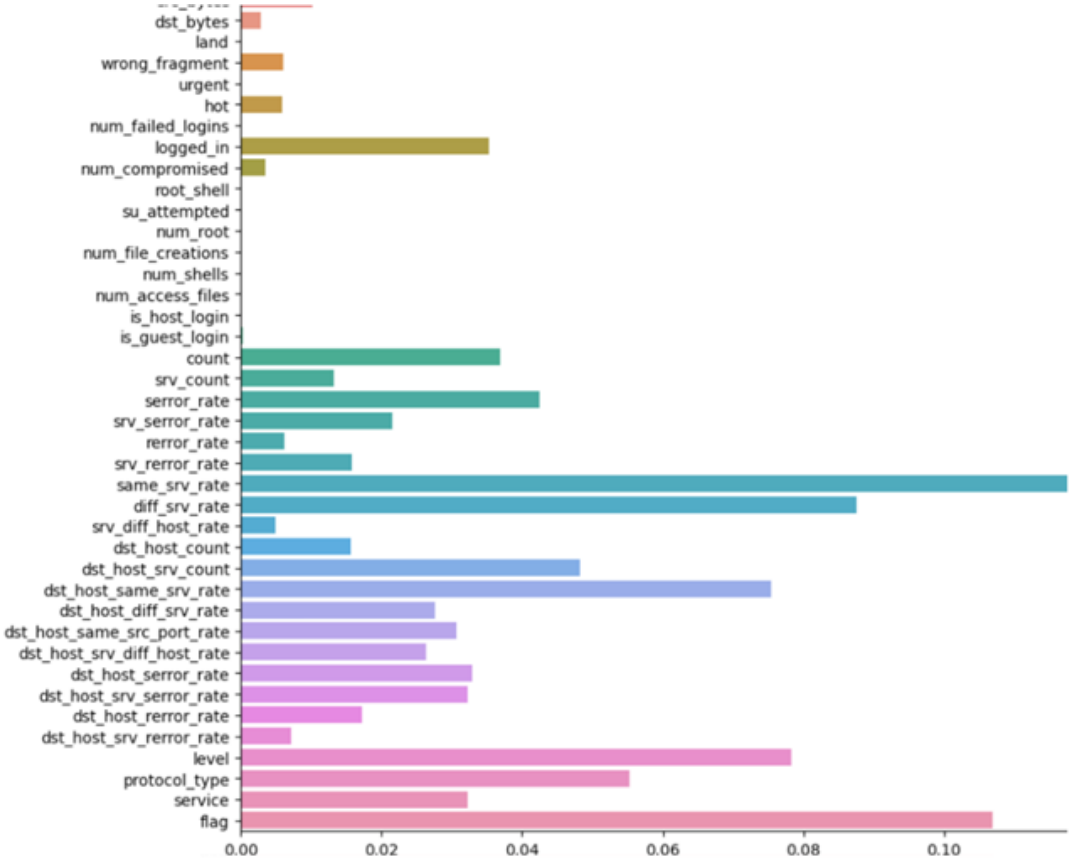


Fig 8.2.2. Most Significant Features of the Datasets

We used a lot of strategies to enhance our model accuracy and find massive elements in our undertaking. These strategies blanketed Perceptual Pigeon Galvanised Optimization (PPGO), correlation matrix analysis, Random Forest, Gradient, Principal Component Analysis (PCA), and Pearson correlation coefficient analysis. Each method tried to pick out vital dataset residences and improve anticipated accuracy. After evaluating their overall performance, we discovered that PPGO had the best accuracy of about 97%. As a result, we chose PPGO for feature choice for the duration of version training, which allowed us to identify relevant dataset capabilities and growth model accuracy even further. This strategic technique allowed us to develop our fashions and consciousness on key features, resulting in greater accurate predictions of cyber-assaults or network anomalies. We progressed our intrusion detection system by employing complex algorithms which includes PPGO and carefully selected capabilities, supplying community managers with beneficial insights into the way to boom safety in opposition to potential threats. Overall, this system progressed our model's accuracy and self belief, therefore increasing community safety.

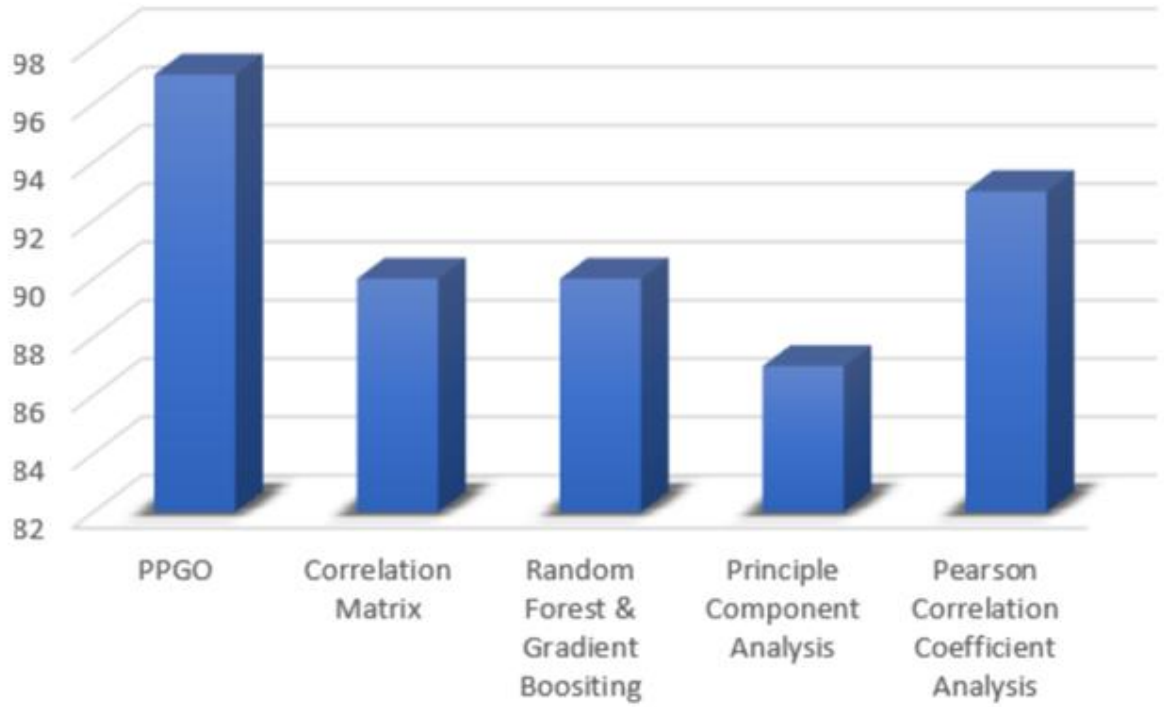


Fig 8.2.3. Accuracy Achieved for Feature Selection Models

In our endeavour to build a great version for the United states dataset, we experimented with many laptop programs inclusive of Decision Tree, Random Forest, KNN, Logistic Regression, LSTM, and Multi-Layer Perceptron (MLP). We tested how efficiently every of these programs understood and expected the data. Some outperformed others, with accuracy starting from 84.Five% to ninety one%. Surprisingly, the Multi-Layer Perceptron (MLP) executed exceptionally well at information in the dataset, with an accuracy of 91%. This proves that MLP is pretty appropriate at identifying data, together with the US dataset. To enhance our model even in addition, we employed a smart approach referred to as Perceptual Pigeon Galvanised Optimization (PPGO) to pick the most enormous bits of the dataset. We completely trained and tested those vital additives of our model, hoping to improve its performance. This allowed us to further refine our MLP model, making it extremely effective at anticipating numerous sorts of attacks. We hired smart pc tools and methodologies to make certain that our version was correct and will aid in our expertise and prevention of cyber dangers. This is a great development in network security.

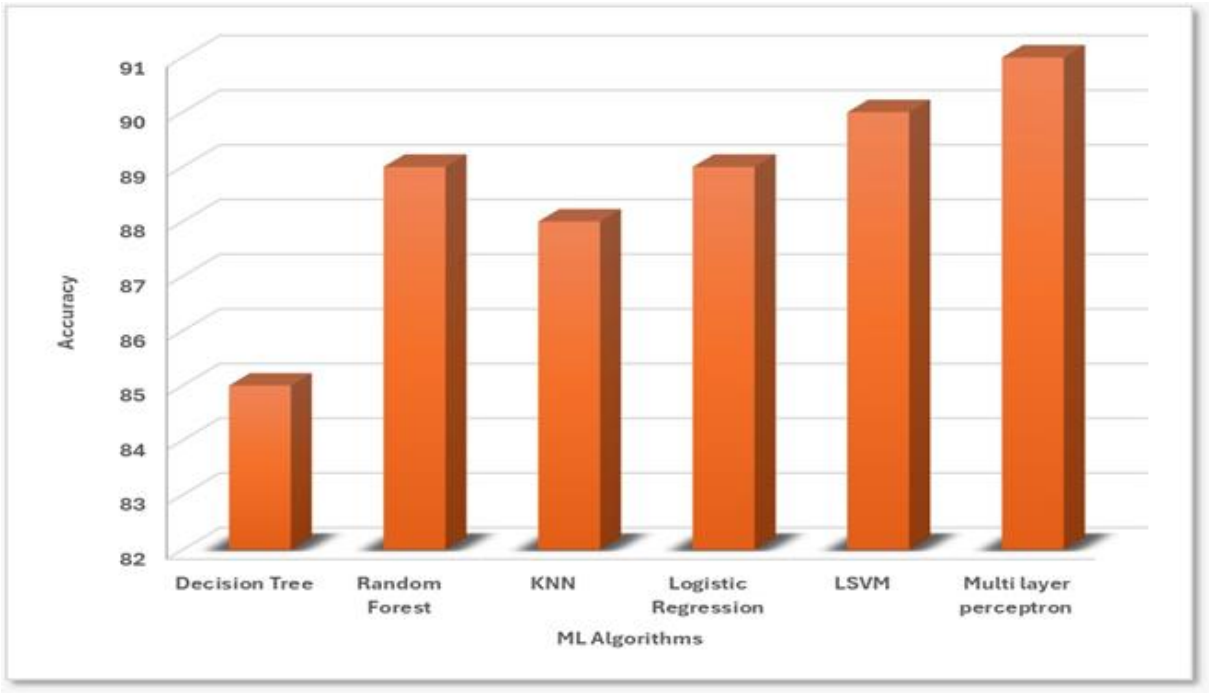


Fig 8.2.4. Accuracy Achieved while testing UNSW dataset ML models.

In our work to optimise the model for the Ton IoT dataset, we tried various computer algorithms such as Decision Tree, Random Forest, Gradient Boosting Classifier (GBC), KNN, Logistic Regression, Naive Bayes, Linear Discriminant Analysis (LDA)), see. Quadratic Discrimination Analysis (QDA).), and short-term memory. We rigorously tested each system to ensure how well it understands and predicts on the Ton-IoT dataset. Some programs performed better than others, with accuracies ranging from 77% to 90%.The LSTM software performed well, with an impressive 90% accuracy indicating that LSTM performs reasonably well with data such as the Ton-IoT dataset. To improve the effectiveness of our model, we used the Perceptual Pigeon Galvanised Optimization (PPGO) method to identify the most important regions in the dataset. We focused on these important features when training and testing our model, with the hope of improving its performance. This improved our LSTM algorithm, which performed consistently well across the predicted attack types. We used smart computational tools and techniques to ensure that our model is accurate and can help us better understand and prevent cybersecurity threats. This is an important step to protect IoT networks from problems.

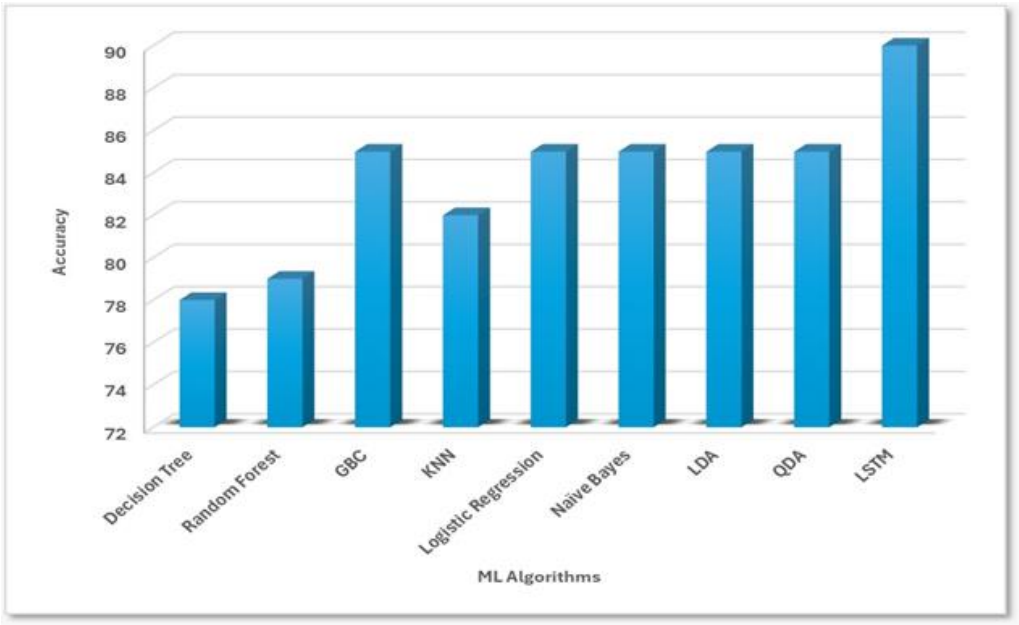


Fig 8.2.5. Accuracy Achieved while testing Ton-IoT dataset ML models.

As a result of our studies, we determined that Perceptual Pigeon Galvanised Optimization (PPGO) is a brilliant characteristic selection technique. PPGO can assist us discover the most vital capabilities in a dataset. Based on these statistics, we will pick a machine learning version that is closely related to the dataset's properties. In addition, we will pick the model that performs the exceptional on that dataset. This technique permits us to modify our version selection technique to every dataset's unique attributes, thereby boosting the overall performance and dependability of our prediction models.

CHAPTER 9

CONCLUSIONS AND FURTHER WORK

In summary, the counselled technique, which makes use of characteristic selection and system gaining knowledge of algorithms to provide a methodical and all-encompassing method to thwarting cybersecurity threats, is a noteworthy leap forward within the area of intrusion detection. A wealth of records about the effectiveness of numerous approaches for precisely identifying and categorising community intrusions has been accrued through the thorough evaluation of numerous datasets and algorithms.

The study's findings spotlight the effectiveness of ensemble techniques like gradient boosting and random forest in coping with challenging intrusion detection assignments. These strategies show robust effectiveness in figuring out harmful hobbies from legitimate activity and recording a whole lot of community traffic styles. Ensemble procedures are quite useful for offering particular and dependable intrusion detection competencies because they make use of the mixed know-how of several selection bushes.

The research additionally demonstrates how nicely deep getting to know algorithms, such Long Short-Term Memory (LSTM) and Multilayer Perceptron (MLP), capture complex styles and subtly apprehend abnormalities in community facts. Sophisticated cyber threats may be efficiently treated via those algorithms due to their top notch ability to pick out aberrant activity and study from sequential information.

To enhance model overall performance and accuracy, function choice techniques like Perpetual Pigeon Galvanized Optimization (PPGO) and Pearson correlation coefficient evaluation are critical in addition to algorithmic strategies. These methods aid inside the discount of model complexity and decorate the effectiveness of intrusion detection algorithms by way of figuring out the most informative subset of attributes.

With the help of function choice techniques and gadget studying algorithms, the cautioned answer offers a sturdy basis for intrusion detection that efficiently counters cybersecurity risks. Through ongoing refinement and optimization of intrusion detection models, that are informed by means of the insights received from this study, groups may also augment their ability to pick out and alleviate community intrusions, therefore defending their critical assets and infrastructure against imagined cyber risks.

There are numerous possibilities for greater examination and development in the intrusion detection sector within the destiny. Integrating anomaly detection methods with conventional signature-primarily based strategies to improve detection capabilities is one feasible research topic. A fascinating problem for lecturers is also the creation of actual-time intrusion detection structures which can manage streaming records in excessive-pace network settings. Additionally, investigating how new technology like federated mastering and blockchain might be carried out to intrusion detection may lead to new opportunities for enhancing network machine security and privateness. To preserve in advance of growing cyber threats and ensure the resilience of network structures in an increasingly linked global environment, it's imperative that research and innovation in intrusion detection preserve.

References

[1] S. Shitharth; Pravin R. Kshirsagar; Praveen Kumar Balachandran; Khaled H. Alyoubi; An Innovative Perceptual Pigeon Galvanised Optimization (PPGO) Based Likelihood Naïve Bayes (LNB) Classification Approach for Network Intrusion Detection System. Publisher: IEEE,**2022,**10,46424-46441,10.1109/ACCESS.2022.3171660

[2] Miloud Bagaa; Tarik Taleb; Jorge Bernal Bernabe; Antonio Skarmeta. A Machine Learning Security Framework for Iot Systems. Publisher: IEEE,**2020**,8, 114066-114077,10.1109/ACCESS.2020.2996214

[3] AM Banaamah, I Ahmad;Intrusion Detection in IoT Using Deep Learning. Sensors, **2022**,https://doi.org/10.3390/s22218417

[4] In Lee;Internet of Things (IoT) cybersecurity: Literature review and IoT cyber risk management. Future internet, **2020**,https://doi.org/10.3390/fi12090157

[5] J Zhang, L Pan, QL Han, C Chen, S Wen, Y Xiang;Deep learning based attack detection for cyber-physical system cybersecurity: A survey. IEEE/CAA Journal of Automatica Sinica, **2021**,9(3),377-391, 10.1109/JAS.2021.1004261

[6] Y Li, Y Zuo, H Song, Z Lv; Deep learning in security of internet of things. IEEE Internet of Things Journal, **2021**,9(22),22133-22146, 10.1109/JIOT.2021.3106898

[7] M Bhavsar, K Roy, J Kelly, O Olusola;Anomaly-based intrusion detection system for IoT application. Discover Internet of Things, Springer, **2023**,5

[8] A El-Ghamry, A Darwish, AE Hassanien; An optimised CNN-based intrusion detection system for reducing risks in smart farming. Elsevier- Internet of Things, **2023**,22,100709,https://doi.org/10.1016/j.iot.2023.100709

[9] C Hazman, A Guezzaz, S Benkirane, M Azrour; lIDS-SIoEL: intrusion detection framework for IoT-based smart environments security using ensemble learning. Springer-Cluster Computing, **2023**,26,4069-4083

[10] EM Onyema, S Dalal, CAT Romero, B Seth, P Young, MA Wajid; Design of intrusion detection system based on cyborg intelligence for security of cloud network traffic of smart cities. Springer-Journal of Cloud Computing, **2022**,11(26)

[11] JB Awotunde, S Misra; Feature extraction and artificial intelligence-based intrusion detection model for a secure internet of things networks. Springer- Illumination of artificial intelligence in cybersecurity and forensics, **2022**

[12] T Xu, JB Wendt, M Potkonjak; Security of IoT systems: Design challenges and opportunities. IEEE/ACM International Conference on Computer-Aided Design, **2014**, 10.1109/ICCAD.2014.7001385

[13] P. K. Gupta, N. K. Singh, V. Mahajan; Intrusion Detection in Cyber-Physical Layer of Smart Grid Using Intelligent Loop Based Artificial Neural Network Technique, IJE, [Volume 34, Issue 5, 2021, Pages 1250-1256], 10.5829/IJE.2021.34.05B.18

[14] H. R. Hematia, M. Ghasemzadeh*a, C. Meinelb; A Hybrid Machine Learning Method for Intrusion Detection, IJE, [Volume 29, Issue 9, 2016, Pages 1242-1246]

[15] F. Zare, P. Mahmoudi-Nasr*; A Hybrid Machine Learning Method for Intrusion Detection [Volume 29, Issue 9, 2016, Pages 1242-1246]

[16] J. Ghasemi*a, J. Esmailyb; A Novel Intrusion Detection Systems based on Genetic Algorithms-suggested Features by the Means of Different Permutations of Labels' Orders,IJE [Volume 30, Issue 10, 2017, Pages 1494-1502]

[17] G. Meena and R. R. Choudhary, "A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA," 2017 International Conference on Computer, Communications and Electronics (Comptelix), Jaipur, India, 2017, pp. 553-558, doi: 10.1109/COMPTELIX.2017.8004032. keywords: {Entropy; Bayes methods; Classification algorithms; Usability; Databases; Graphical user interfaces;IDS;Classification;Data Mining;WEKA;KDD 99;NSL KDD;J48 Graft; NAÏVE BAYES}

[18] G. Guo, X. Pan, H. Liu, F. Li, L. Pei and K. Hu, "An IoT Intrusion Detection System Based on TON IoT Network Dataset," *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2023, pp. 0333-0338, doi:10.1109/CCWC57344.2023.10099144. keywords: {Performance evaluation; Learning systems; Correlation coefficient; Conferences; Computational modeling;Intrusion detection;Machine learning; Machine learning; Intrusion Detection Systems; Feature selection;TON_IoT}

[19] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 2015, pp. 1-6, doi:

10.1109/MilCIS.2015.7348942. keywords: {Telecommunication traffic; Feature extraction;Servers;Training;Data models; IP networks; Benchmark testing;UNSW-NB15 data set;NIDS;low footprint attacks;pcap files; testbed}

[20] E. Hodo et al., "Threat analysis of IoT networks using artificial neural network intrusion detection system," 2016 International Symposium on Networks, Computers and Communications (ISNCC), Yasmine Hammamet, Tunisia, 2016, pp. 1-6, doi: 10.1109/ISNCC.2016.7746067. keywords: {Internet of things;Artificial Neural Network;Denial of Service;Intrusion detection System and Multi-Level Perceptron.

PLAGIARISM REPORT

Doc

ORIGINALITY REPORT

13%	9%	5%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to VNR Vignana Jyothi Institute of Engineering and Technology Student Paper	4%
2	www.coursehero.com Internet Source	1%
3	Submitted to Griffith College Dublin Student Paper	<1%
4	fastercapital.com Internet Source	<1%
5	mdpi-res.com Internet Source	<1%
6	Submitted to Midlands State University Student Paper	<1%
7	www.ijraset.com Internet Source	<1%
8	www.researchgate.net Internet Source	<1%
9	Submitted to Institute of Technology, Nirma University	<1%

AI DETECTION REPORT

How much of this submission has been generated by AI?

***16%**

of qualifying text in this submission has been determined to be generated by AI.

* Low scores have a higher likelihood of false positives.

Caution: Percentage may not indicate academic misconduct. Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Frequently Asked Questions

What does the percentage mean?
The percentage shown in the AI writing detection indicator and in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was generated by AI.

Our testing has found that there is a higher incidence of false positives when the percentage is less than 20. In order to reduce the likelihood of misinterpretation, the AI indicator will display an asterisk for percentages less than 20 to call attention to the fact that the score is less reliable.

However, the final decision on whether any misconduct has occurred rests with the reviewer/instructor. They should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in greater detail according to their school's policies.

How does Turnitin's indicator address false positives?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be AI-generated will be highlighted blue on the submission text.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

What does 'qualifying text' mean?
Sometimes false positives (incorrectly flagging human-written text as AI-generated), can include lists without a lot of structural variation, text that literally repeats itself, or text that has been paraphrased without developing new ideas. If our indicator shows a higher amount of AI writing in such text, we advise you to take that into consideration when looking at the percentage indicated.

In a longer document with a mix of authentic writing and AI generated text, it can be difficult to exactly determine where the AI writing begins and original writing ends, but our model should give you a reliable guide to start conversations with the submitting student.

Disclaimer
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify both human and AI-generated text) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

SHOW AND TELL