MODELING INTRUSION DETECTION SYSTEM USING MACHINE LEARNING TECHNIQUES

Minor project report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

DUMPA KARTHIK REDDY	(21UECS0166)	**(VTU20276)**
MYLA MANIKANTA	(21UECS0393)	**(VTU21065)**
YARALAGADDA MEDHAMSH	(21UECS0676)	**(VTU21048)**

Under the guidance of
Dr. P. J. BESLIN PAJILA, M.E, PhD
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

MODELING INTRUSION DETECTION SYSTEM USING MACHINE LEARNING TECHNIQUES

Minor project report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

DUMPA KARTHIK REDDY	(21UECS0166)	**(VTU20276)**
MYLA MANIKANTA	(21UECS0393)	**(VTU21065)**
YARALAGADDA MEDHAMSH	(21UECS0676)	**(VTU21048)**

Under the guidance of
Dr. p. J. BESLIN PAJILA, ME, PhD
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled "MODELING INTRUSION DETECTION SYSTEM USING MACHINE LEARNING TECHNIQUES" by "DUMPA KARTHIK REDDY (21UECS0166), MYLA MANIKANTA (21UECS0393), YARALAGADDA MEDHAMSH (21UECS0676)" has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Dr. P. J. BESLIN PAJILA
Assistant Professor
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

Signature of Head of the Department
Dr. M.S. Muralidhar
Associate Professor & HOD
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

Signature of the Dean
Dr. V. Srinivasa Rao
Professor & Dean
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

D, KARTHIK REDDY

Date: / /

M. MANIKANTA

Date: / /

Y. MEDHAMSH

Date: / /

APPROVAL SHEET

This project report entitled MODELING INTRUSION DETECTION SYSTEM USING MACHINE LEARNING TECHNIQUES by M. MANIKANTA (20UECS0393), D. KARTHIK REDDY (20UECS0166), Y. MEDHAMSH (21UECS0676) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners **Supervisor**

Dr. P. J. Beslin Pajila

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN,** for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. V. SRINIVASA RAO, M.Tech., Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr. M.S. MURALIDHAR, M.E., Ph.D.,** for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our Internal Supervisor **Dr. P. J. BESLIN PAJILA, M. E., Ph. D** for her cordial support, valuable information and guidance, she helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr. V. ASHOK KUMAR, M.Tech., Ms. U. HEMAVATHI, M.E., Ms. C. SHYAMALA KUMARI, M.E.,** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

D. KARTHIK REDDY (20UECS0166)
M. MANIKANTA (20UECS0393)
Y. MEDHAMSH (20UECS0676)

ABSTRACT

An intrusion detection system (IDS), a crucial tool in cyber security, keeps track of the hardware and software being used in the network. Even after years of research, modern Intrusion Detection System still have trouble spotting new assaults, improving detection efficiency, and lowering false alarm rates. To solve the aforementioned problems, several academics have focused on developing Intrusion Detection System that leverage machine learning methods. System protection, user activity analysis, and user behaviour prediction are the responsibilities of an intrusion detection system. These actions will thus be seen as an assault or as usual behavior. Although Intrusion Detection System has been around for a while, managers still maintain the system inefficiently due to the high volume of return alarm messages. Machine learning algorithms are being employed in this research to find intrusions. The data set has first been collected and pre-processed to decrease the dimensions. The pre-processed data will then be given to Random Forest Classifier for learning and testing. The objective of the proposed system is to develop an intrusion detection system that leverages machine learning techniques to enhance detection accuracy and reduce false alarm rates. Intrusion Detection System plays a vital role in cyber security by monitoring network hardware and software. However, existing Intrusion Detection System face challenges in detecting novel attacks and managing a high volume of return alarm messages. To address these issues, this proposed system focuses on employing Random forest algorithm to improve system protection, user activity analysis, and user behavior prediction. Evaluated the performance of several machine learning algorithms and compared their accuracy, precision, recall, and F1 scores. The experimental results indicate that the Random Forest algorithm outperformed other algorithms, achieving the highest accuracy and F1 score. The system was able to detect intrusions with a high degree of accuracy, effectively reducing false alarm rates and increasing the detection rate of both known and unknown attacks. Overall, The proposed system demonstrates the potential of machine learning techniques in developing an effective intrusion detection system.

Keywords: Intrusion Detection System, Machine learning, Pre-processing, Network security, Random forest classifier, Recursive feature elimination, Performance evaluation, Cross-validation, KDD dataset.

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

AUC	Area Under the ROC Curve
CV	Cross-Validation
DT	Decision Tree
FN	False Negative3
FP	False Positive
GNB	Gaussian Naive Bayes
IDS	Intrusion Detection System
KNN	K-Nearest Neighbors
LR	Logistic Regression
PCA	Principal Component Analysis
RF	Random Forest
RFE	Recursive Feature Elimination
ROC	Receiver Operating Characteristic
SMOTE	Synthetic Minority Over-sampling Technique
SVM	Support Vector Machine
TN	True Negative
TP	True Positive

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

An intrusion detection system (IDS) is a device or software application that monitors a network or systems for malicious activity or policy violations. They include high false alarm rate, difficulties in obtaining exactly clean data for the modeling of normal patterns and the deterioration of detection rate because of some noisy data in the training set. A series of experimental results on the well-known KDD Cup 1999 dataset demonstrate it can effectively detect anomalies with high true positive rate, low false positive rate and high confidence than the state-of-the-art anomaly detection methods. In addition, even interfered by noisy data (unclean data), the proposed method is robust and effective.

Moreover, it still retains good detection performance after employing feature selection aiming at avoiding the curse of dimensionality. SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.With the rapid advancement of technology and the increasing reliance on computer networks for various aspects of lives, ensuring the security of these networks has become a critical concern. Cyber-attacks and intrusions pose significant risks to the confidentiality, integrity, and availability of sensitive information, making the need for robust Intrusion Detection Systems (IDSs) more important than ever.

Traditional IDSs have limitations in detecting sophisticated and evolving attacks, often resulting in high false positive rates and missed intrusions. To address these challenges, the integration of machine learning techniques has gained significant attention in the field of cybersecurity. Machine learning algorithms have shown promise in automatically identifying patterns and anomalies in network data, en-

abling more accurate and efficient intrusion detection. The successful implementation of this proposed system will contribute to strengthening network security by improving the detection and response capabilities of IDSs. By harnessing the power of machine learning, organizations can enhance their ability to detect and mitigate cyber threats, protect valuable assets, and maintain the integrity of their systems and networks.

1.2 Aim of the project

The aim of the project is to determine the best algorithm for finding intrusions in a given dataset. In order to identify the algorithm that produces the maximum detection accuracy while minimising false positives, the proposed system will analyse the dataset supplied by the user and evaluate several machine learning techniques, including supervised and unsupervised learning methods. To provide a valuable contribution to the field of intrusion detection systems and to offer a practical solution for improving the security of systems that handle sensitive information.

The system will consider the unique characteristics of the provided dataset and will be developed to address the limitations of existing intrusion detection systems. This includes the ability to detect novel attacks, low detection accuracy, and high false alarm rates. The outcome of the proposed system will be a recommendation for the best algorithm to use for intrusion detection in the given dataset, along with a detailed analysis of the results and limitations of the chosen approach.

1.3 Project Domain

The project domain is focused on cybersecurity and aims to develop an accurate and efficient Intrusion Detection System (IDS). As cyberthreats continue to evolve and become more sophisticated, cybersecurity continues to rise in importance for organisations. The objective of the project is to deal with the issue of false positives in IDS, which can result in pointless warnings and divert security staff from actual security concerns. The initiative can improve the general security posture of organisations and safeguard their assets and data by lowering false positives.

Cybersecurity-related topics like network security, information security, and threat intelligence are all included in the proposed system domain. In order to maximise the functionality of the IDS, machine learning techniques, data analysis, and sensor technology are used. In order to better guide the creation of the IDS, the project also comprises study and analysis of the most recent cyberthreats and trends. For businesses to defend their networks, assets, and data against threats, the project domain is essential. The project's results can help organisations by giving them access to a dependable and efficient IDS that can identify security threats.

Furthermore, with the increasing sophistication of cyber attacks, the domain is constantly evolving, with new techniques and technologies being developed to detect and prevent intrusions. This requires continuous research and development of new methods and tools for intrusion detection to keep up with the constantly evolving threats.As a result, the domain presents numerous challenges and opportunities for researchers and practitioners to explore and innovate, making it an exciting and important field in the realm of network security.

1.4 Scope of the Project

The Scope of the proposed system is to increase the efficiency of intrusion detection systems (IDS) by reducing false positive rates and boosting system accuracy. False positives are mistakes that happen when an IDS mistakenly classifies a harmless event as a security threat, which can put the system under additional stress and trigger pointless alarms.

The proposed system's scope involves looking into and choosing the best machine learning intrusion detection techniques. In order to determine the most crucial properties for detecting intrusions, Will also test various feature selection techniques. The model's precision, sensitivity, specificity, and accuracy will be measured in order to assess the system's performance.

The proposed system will also go over how to use network traffic analysis to find intruders. In order to offer scalable and accessible intrusion detection services, the proposed system's scope also includes designing and implementing the system on a cloud-based platform.

Chapter 2

LITERATURE REVIEW

[1] G. Abdelmoumin et al., described a study of optimizing PCA and 1-SVM AML-IDS models using hyper parameter tuning and ensemble learning to detect intrusions in IoT. The authors performed hyper parameter tuning to optimize these models by selecting the optimal model structure, followed by using ensemble learning with the Stacking technique to create multiple learners and train the single-learner AML-IDS models to boost the single-learner model detection performance. The optimized models were trained using ground-truth network intrusion data sets containing both malicious and benign IoT and IIoT traffic and evaluated using typical ML/DL metrics. The study found that the SVM-based IDS model exhibited high performance and good predictability when a heterogeneous ensemble was used, while single-learner 2-NN DL-IDS models showed superior performance in detecting intrusions in IoT environments. The authors plan to further improve the AML-IDS detection rate and performance by reducing the false-positive rate, minimizing training and testing time, addressing issues related to latency and resource utilization, adaptability to the dynamic IoT environment, and deployment architecture.

[2] A. Guezzaz et al., presented a survey of various supervised learning classification techniques used to solve real-world problems. The authors acknowledge that the complexity and heterogeneity of data often pose limitations for classifiers. To address these challenges, they propose a new machine learning algorithm and validate it through suggested solutions that guarantee fast and efficient analysis. The study emphasizes the need for an accurate and efficient intrusion detection system for improved network monitoring and decision making. The proposed classifier serves as a potential solution to address these needs. Future work aims to validate the classifier's effectiveness in developing an intrusion detection system.

[3] Gao et al., proposed a two-layer intrusion detection method for train-ground communication systems using machine learning and state observer. The first layer uses

machine learning algorithms such as the random forest, gradient boosted decision tree, AdaBoost, and support vector machine to detect and identify intrusion behavior on wireless network data. The second layer utilizes a state observer-based intrusion detection method to recognize abnormal train physical states. The detection results from both layers are combined to provide a reliable intrusion detection result for CBTC systems. The paper presents theoretical analysis and simulation results demonstrating that only long-term continuous attacks can affect train operation while short-term continuous or random attacks do not affect it.

[4] G. de Carvalho Bertoli et al., described the hyper parameter tuning and ensemble learning-based optimization of PCA and 1-SVM AML-IDS models to detect IOT intrusions.The AB-TRAP framework is presented in this paper as a solution to the challenge of adapting network intrusion detection systems (NIDS) to new attacks and evolving network traffic. The framework comprises steps to build attack and bonafide datasets, train machine learning models, realize the solution in a target machine, and evaluate the protection module's performance. The authors test AB-TRAP in two environments, achieving low-resource utilization protection modules with Decision Tree providing the best performance for the training and realization phases. The authors suggest future work to conduct a multi-label classification, improve the generation of datasets, produce protection modules for lightweight IP, and test it in smaller operating systems.

[5] G. Pu et al., proposed an unsupervised anomaly detection algorithm, SSC-OCSVM, that combined subspace clustering and one-class support vector machine to detect attacks without any prior knowledge. The algorithm was evaluated using the NSL-KDD network attacks dataset and compared with three other unsupervised detection clustering algorithms available in the literature, with SSC-OCSVM demonstrating superior performance. The authors suggest future work should focus on developing an effective feature selection method and implementing parallelization of the algorithm for parallel computing.

[6] S. Mambwe Kasongo et al., presented the design, implementation, and testing of a deep learning (DL) based intrusion detection system (IDS) using feedforward neural networks (FFDNNs). The authors conducted a literature review of machine learning (ML) and DL methods and found that an efficient approach to intrusion

detection has not yet been found. The FFDNN models were coupled with a feature extraction unit (FEU) using information gain (IG) to reduce the input dimension and increase classifier accuracy. The NSL-KDD dataset was used for experimentation, and the FFDNN models outperformed support vector machine (SVM), random forest (RF), Naive Bayes (NB), decision tree (DT), and k-nearest neighbor (KNN) methods for both binary and multiclass classification problems. Future work will focus on increasing the detection rates of R2L and U2R attacks and investigating the superiority of DL-based methods over other ML approaches using the AWID dataset.

[7] S. Otoum et al., presented a feasibility analysis of a deep learning-based IDS called the Clustered Restricted Boltzmann Machine-Intrusion Detection System (RBC-IDS), and compared it with an adaptive machine learning-based IDS approach. The study also compared the performance of RBC-IDS with different numbers of hidden layers against the ASCH-IDS through simulations. The results demonstrated that the proposed RBC-IDS has a high detection rate and accuracy rate with three hidden layers, with intrusive behaviors present in the tested WSN. The study also revealed that adopting a machine learning-based IDS framework leads to approximately half the detection time of the deep learning-based RBC-IDS framework. The future agenda includes extending the presented IDS to larger networks with more sensors. Overall, the study demonstrates the potential of deep learning-based IDS for WSNs and provides valuable insights into the performance of different machine learning approaches for intrusion detection.

[8] Trans N et al., highlighted the importance of data quality in the performance of machine learning (ML) and deep learning (DL) models. The study examines the impact of duplicates and overlaps on traditional ML models and pre-trained models. The results demonstrate that pre-trained models are less susceptible to duplicates and overlaps, leading to superior performance. However, the proposed data cleaning method may have adverse effects on low-quality data, leading to decreased or unstable detection performance. The study also found that evaluating models on duplicate data can lead to unreliable evaluations, with exceptions in high-quality datasets. The proposed data cleaning method is necessary for processing both training and testing data to increase detection performance and ensure reliable performance evaluation. Overall, this research highlights the importance of data quality in the performance of ML and DL models and emphasizes the need for careful consideration of data clean-

ing methods in model training and evaluation.It is also important to use the proposed data cleaning method to process the testing data to ensure a reliable performance evaluation.

[9] W. Wang et al., proposed a hybrid system for cloud intrusion detection that combines a deep learning algorithm, stacked contractive auto encoder (SCAE), with the support vector machine (SVM) classifier for detecting malicious attacks in the cloud computing environment. The study demonstrates the effectiveness of the proposed SCAESVM-IDS model using the NSL-KDD and KDD Cup 99 intrusion detection datasets compared to three state-of-the-art methods. Although the proposed approach shows promising performance, future work should aim to optimize the classifier for effectively recognizing new attacks and reducing the controller's bottleneck. Additionally, the solution should be evaluated in a real cloud environment to assess its performance. Building a reliable CIDS is essential for protecting cloud computing from malicious attacks, ensuring the healthy and sustainable development of the cloud computing environment.

[10] Z. Chkirbene et al., presented a novel approach to network intrusion detection called TICDS and TICDS-A. The proposed system uses a combination of machine learning techniques and past information to create a trusted cloud environment. Unlike traditional intrusion detection systems, TICDS and TICDS-A perform cleansing activities for the participant nodes regardless of the presence or absence of attack alarms. TICDS has a fixed periodic cleansing window, while TICDS-A has a dynamic window for network cleansing and anomaly detection. The simulation results demonstrate the good performance of both proposed models. The proposed approach provides a new perspective on intrusion detection, which aims to maintain the integrity of the network and improve the accuracy of intrusion detection by incorporating machine learning techniques and past information. The proposed models have the potential to enhance the security of distributed systems by providing a reliable and efficient intrusion detection mechanism.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

A single classifier model evaluation is used by many Intrusion Detection Systems (IDS) now in use to identify and stop network intrusions. To recognise well-known attack patterns, these systems frequently use pre-established rules or signatures. However, this strategy may result in inaccurate results, a large percentage of false positives, and the inability to identify previously unidentified threats. When processing high volumes of network traffic data, current systems may also experience scalability and performance problems [2].

3.2 Proposed System

By evaluating input data against many classifier models to choose the most effective model for identifying and preventing network intrusions, The Proposed system IDS seeks to advance existing systems. The Proposed system can adapt to new attack patterns, reduce false positive rates, and retain reliable intrusion detection capabilities by utilising a number of classifiers, including machine learning and anomaly detection models.

Large amounts of network traffic data may be processed in real-time by the proposed system, which is built to be scalable and effective. Seek to deliver a more thorough and accurate analysis of network traffic data, enabling the IDS to successfully recognise and avoid security risks. The proposed system intends to leverage the capabilities of several classifier models to accomplish this goal.. Overall, compared to current methods, Suggested solution has the potential to considerably increase intrusion detection's effectiveness and accuracy.

3.3 Feasibility Study

3.3.1 Economic Feasibility

When developing and implementing any new system or technology, economic feasibility is a crucial factor to take into account. Economic feasibility in the context of the proposed Intrusion Detection System (IDS) relates to the prospective financial expenses and gains related to its creation, implementation, and upkeep [3].

In terms of costs, the suggested system will need a large investment in hardware, software, and labour to construct and maintain. However, there may be substantial advantages to improved intrusion detection and prevention. The technology can assist organisations in avoiding expensive reputational harm, the loss of sensitive data, and breach-related legal costs by lowering the amount of successful network attacks and data breaches. Additionally, the cost savings from increased security may be greater than the IDS's initial expenditure.

Furthermore, the ability of organisations to make investments in better security measures determines whether the suggested solution is economically feasible. Spending on security may need to be prioritised by organisations in order to support the purchase of a new IDS. The economic viability of the suggested IDS is more appealing, though, as security threats continue to advance and become more complex, necessitating the need for effective intrusion detection and prevention solutions.

3.3.2 Technical Feasibility

The implementation of suggested IDS will necessitate the fusion of numerous technologies, such as network security protocols, data analytics, and machine learning. Therefore, it is crucial to make sure that the system can be developed and deployed with the help of the technological infrastructure and knowledge that are required.

It is also necessary to consider performance and scalability when designing the system. Accurately identifying and averting security risks requires the ability to handle huge amounts of network traffic data in real-time. This will necessitate the

employment of sophisticated hardware and algorithms that can handle high-speed network traffic without adding latency or deteriorating performance [6].

The compatibility of suggested IDS with current network architecture and security solutions must also be taken into account. This can call for the creation of unique integration solutions or adjustments to current network settings. In order to ensure that the system is fully functional, secure, and compatible with the organization's existing technology and infrastructure, technical feasibility will necessitate rigorous planning, design, and testing.

3.3.3 Social Feasibility

The social feasibility of the suggested Intrusion Detection System (IDS) relates to the potential effects it may have on stakeholders, such as staff members, clients, and other network users. The possible effect on user privacy is one of the main issues. Data from network traffic that the system will be analysing could include sensitive information. Therefore, it is imperative to check that the system is built with privacy protection features that adhere to statutory and moral requirements [4].

Additionally, modifications to current network policies and practises, such as access control and data management, may be necessary for the deployment of suggested IDS. To ensure that the system addresses their requirements and concerns, it is crucial to interact with stakeholders and include them in the development process. This could entail putting together communication plans, running training sessions, and including stakeholders in the testing and validation procedure.

In order to ensure that the system is socially acceptable and satisfies the interests of all stakeholders, social feasibility necessitates thorough assessment of the potential impact of the proposed IDS on stakeholders, including their privacy concerns.

3.4 System Specification

3.4.1 Hardware Specification

- CPU: Intel Core i7 or equivalent (e.g., AMD Ryzen 7) with a clock speed of 2.6 GHz or higher.
- RAM: 16 GB or more.
- Storage: 500 GB Solid State Drive (SSD) or higher.
- GPU: Nvidia GeForce GTX 1060 or higher, or equivalent AMD graphics card, with 6 GB or more of dedicated video memory.

3.4.2 Software Specification

- Operating System: Windows 10 or Linux (Ubuntu 18.04 or later)
- Python Version: Python 3.8 or later (including required packages such as pandas, scikit-learn, numpy, etc.)
- Integrated Development Environment (IDE): PyCharm, Visual Studio Code, or Spyder
- Package Manager: Anaconda or Miniconda (for managing Python environments and installing required packages)
- Version Control: Git and GitHub (for code version control and collaboration)
- Cloud Computing Platform (optional): Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure (for running experiments on high-performance computing resources)

3.4.3 Standards and Policies

Google Colab

Google Colab is a cloud-based platform that provides a Jupyter Notebook environment for machine learning and data analysis tasks. The platform allows users to write and execute code using various languages, including Python and R. As a cloud-based service, Google Colab provides users with a pre-installed set of machine learning and data analysis libraries and frameworks, such as TensorFlow and PyTorch.

Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 Architecture

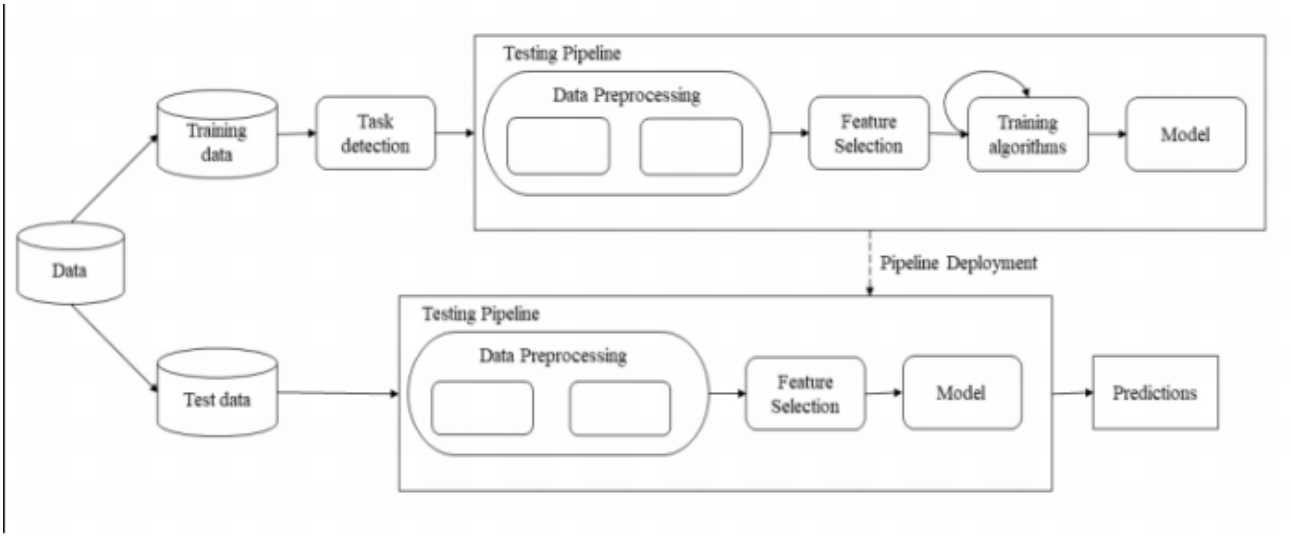

Figure 4.1: Architecture Diagram For Training and testing of Intrusion Detection

The Figure 3.1 shows the architecture diagram for modeling intrusion detection system using machine learning techniques involves several components. First, the raw data is collected, and then feature extraction is performed to reduce the data to more manageable groupings. The resulting data is split into training and testing sets. The training set is then used to develop a classifier, which can identify patterns in the data. Finally, the classifier is tested using the testing set to detect attacks.

The architecture diagram includes several layers. The first layer is the data layer, which includes the raw data and the cleaned data resulting from feature extraction. The second layer is the training layer, which includes the training set and the classifier development process. This layer is responsible for developing a classifier capable of identifying patterns in the data. The third layer is the testing layer, which includes the testing set and the classifier testing process. This layer is responsible for evaluating the performance of the classifier in detecting attacks.

4.2 Design Phase

4.2.1 Use Case Diagram

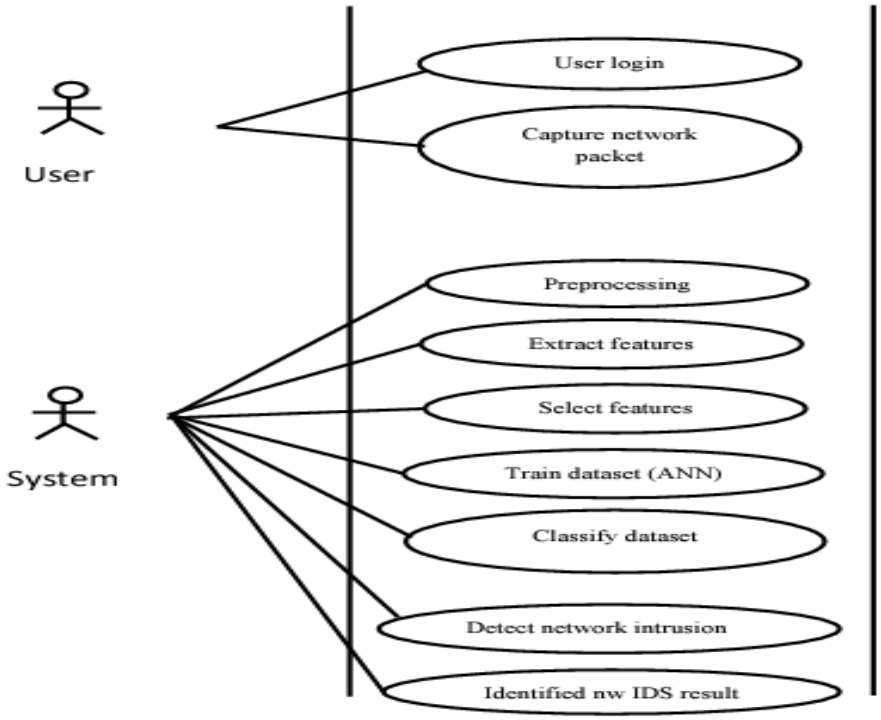

Figure 4.2: Use case Diagram For Intrusion Detection

The Figure 3.2 shows the Use case diagram, The intrusion detection system (IDS) involves several actors function properly. IDS is the primary actor and is the one to receive an IP packet from the network. The user or administrator can then log into the system by using the UI to complete a login activity. The IP packet is subsequently received by and processed by the Anomaly Detection module from the IDS.

The data that has been processed is subsequently sent to the UI for presentation via the Anomaly Detection module. The data is presented to the user and administrator via the UI for analysis and decision-making. The Pattern Database includes a collection of patterns that have been recognised in the past as attack signatures. The Signature Recognition module makes use of these patterns to identify known attack signatures in the processed data. Overall, this use case is an example of how the various IDS parts cooperate to identify and stop potential security threats.

4.2.2 Class Diagram

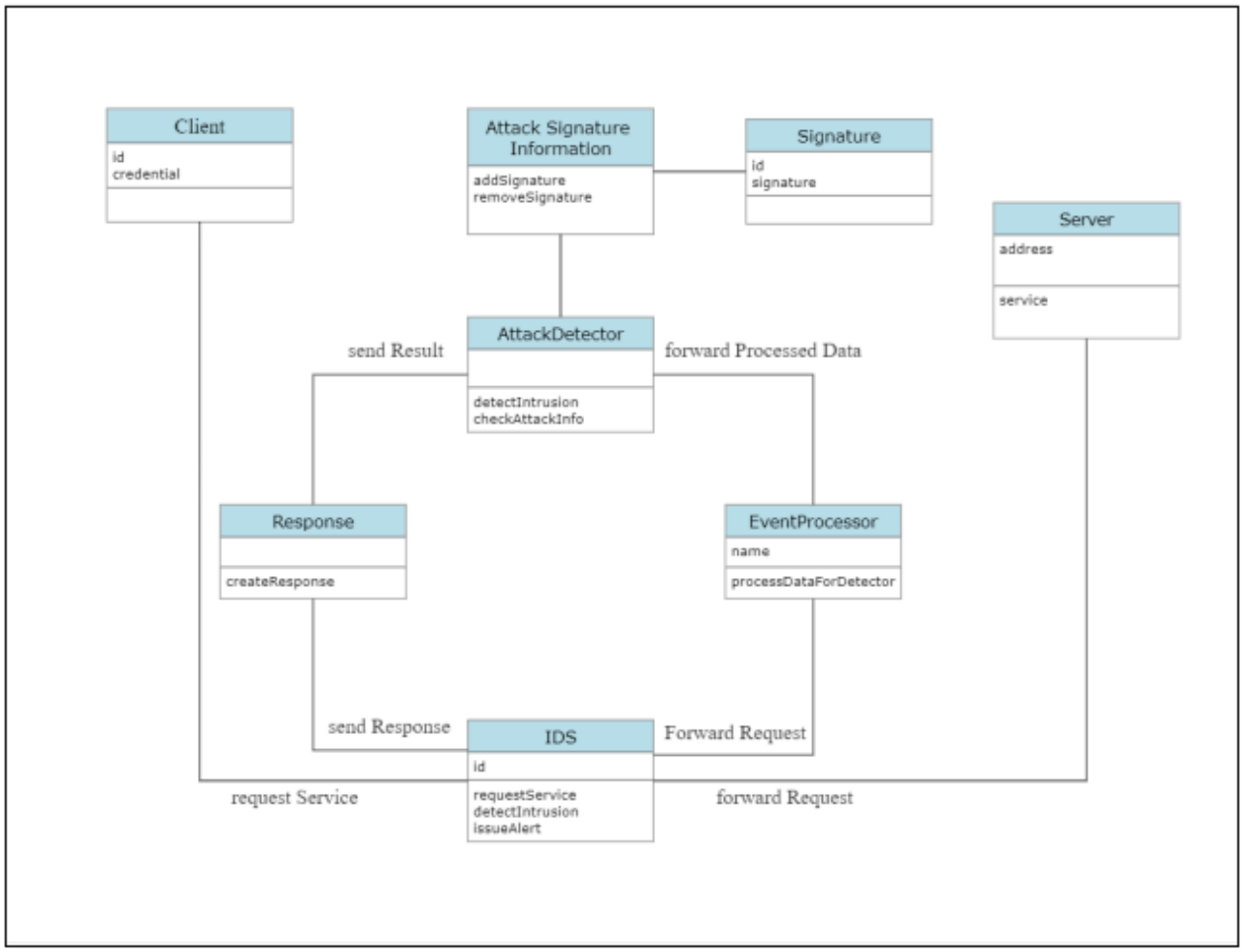

Figure 4.3: Class Diagram of Intrusion Detection

The Figure 3.3 shows the Class diagram, for the intrusion detection system using machine learning techniques includes several key components. The client initiates a request for the IDS service, which is then received by the IDS component. The IDS then sends a response back to the client. Additionally, the IDS component forwards the request to the event processor, which is responsible for processing the data. The event processor communicates with the attack detector, which is responsible for identifying and flagging potential attacks. The attack detector then sends the results back to the event processor, which forwards the processed data to the appropriate location. The attack detector also communicates with the attack signature information component, which provides additional info about known attack signatures.

4.3 Algorithm & Pseudo Code

4.3.1 Random Forest Classifier Algorithm

Step1: Start
Step2: Import the required libraries matplotlib, pandas, numpy, seaborn, sklearn, and imblearn.
Step 3: Load the NSL KDD train and test datasets.
Step 4: Apply attack class mappings to the dataset.
Step 5: Remove the num outbound cmds field as it is redundant.
Step 6: Plot a bar chart for the attack class distribution.
Step 7: Extract the numerical attributes and scale them to have zero mean and unit variace.
Step 8: Extract the categorical attributes and encode them.
Step 9: Separate the target column from the encoded data.
Step 10: Define columns and extract the encoded train set for sampling.
Step 11: Reshape the target column to a 1D array shape.
Step 12: Apply random over-sampling to balance the target classes.
Step 13: Fit a random forest classifier on the training set and extract the importnt features.
Step 14: Create the model and select 10 attributes.
Step 15: Define columns to a new dataframe and add a dimension to the target.
Step 16: Create a dataframe from the sampled data and the test dataframe.
Step 17: Create two-target classes - normal class and an attack class.
Step 18: Train the required models.
Step 19: Evaluate each model's performance using cross-validation mean score, model accuracy, confusion matrix, and classification report.
Step 20: END

4.3.2 Pseudo Code

```
START
  Import matplotlib, pandas, numpy, seaborn, sklearn, imblearn.
    datacols = ["Columns"]
    training_data = read_('training data')
    testing_data = read_('test data')
     print("Dimensions of training dataset:", train_dim)
     print("Dimensions of testing dataset:", test_dim)
     outbound train = training data['outbound'].value_counts()
     print(outbound train)
     outbound test = testing data ['outbound'].value_counts()
     print(outbound test)
     plot_bar_chart(train_attack_counts, test_attack_counts)
     model = TRAIN(X_train, y_train)
     accuracy = EVALUATE(model, X_test, y_test)
     PRINT("Accuracy:", accuracy of each model)
END
```

4.4 Module Description

4.4.1 Data Preparation

Data preparation is a necessary first step in any machine learning proposed system, and it involves a number of essential steps that ensure the dataset is ready for analysis. The initial stage is to gather information from numerous sources, determine the pertinent attributes that are needed for analysis, and collect the data. The following step is to clean the data by locating and dealing with any outliers, mistakes, and missing values in the dataset. By standardising the format and cross-referencing the values with other sources, it is vital to make sure the data is correct and consistent [6].

The next stage in data preparation is to change the data into a format appropriate for analysis. To record complicated interactions between qualities, this involves saving category variables, scaling numerical variables, and developing new features from previous ones. Finding the target variable and making sure it is prepared and categorised correctly for analysis are just as important. Last but not least, data preparation involves splitting the dataset into training and testing sets, making sure that both sets have comparable distributions to avoid bias in the model. These procedures prepare the data for analysis so that machine learning models can be used to

find patterns and generate precise predictions.

4.4.2 Sampling and Feature Selection

Sampling and feature selection are performed after the pre-processing of the data. Random oversampling is utilized to over sample the minority class to address the issue of class imbalance in the dataset at this stage. To minimize the dataset's feature count and identify the ones that would aid in classification, feature selection is also performed. Recursive feature elimination (RFE) technique is employed in conjunction with the random forest classifier to accomplish this. The most important features in the dataset can be identified with the aid of the random forest classifier, and the optimal number of features to achieve the best classification performance can be chosen using RFE [6].

Important elements in the machine learning process include sampling and feature selection. Class disparity can result in biassed models with poor prognostication for the minority class. By oversampling the minority class, it is ensured that the model will learn from sufficient samples to accurately represent the group. Feature selection aids in reducing the dataset's dimensionality, which can enhance the model's computational effectiveness, lessen overfitting, and enhance model performance. The employment of random forest classifier with RFE technique gives a robust strategy to picking relevant features and maximising the quantity of features for classification.

4.4.3 Model Training and Evaluation

In the Model Training and Evaluation module, various machine learning models are trained and evaluated in order to determine if network traffic is normal or malicious. The preprocessed data are divided into training and testing sets at the beginning of the module. The models are then trained using a variety of classification techniques, including Naive Bayes, Decision Trees, K Neighbours, and Logistic Regression, on the training data.

After the models have been trained, their accuracy and performance are assessed using the test set. Confusion matrix, classification report, and cross-validation mean score are among the evaluation measures. The amount of true positives, false positives, true negatives, and false negatives are displayed in the confusion matrix to give

a general summary of the model's performance. The cross-validation mean score provides an estimate of how well the model is performing on unseen data, whereas the classification report provides precision, recall, and f1-score for each class. The most effective model is chosen for deployment based on the evaluation metrics [6].

4.5 Steps to execute/run/implement the project

4.5.1 Install the required libraries and dependencies

Install every single dependency and library listed in the proposed system, including matplotlib, pandas, numpy, seaborn, sklearn, and imblearn.

4.5.2 Download the dataset

Download the NSL-KDD dataset from the Kaggle website or the UCI Machine Learning Repository.

4.5.3 Import the dataset

Import the downloaded dataset into your Python environment.

4.5.4 Run the code

Run the Python code and ensure that it executes without any errors.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input

The KDD Cup 1999 dataset is used as input data for IDS because it offers a thorough perspective of the different kinds of network traffic that an IDS can face in real-world circumstances. The dataset can be used to test the effectiveness of an IDS as well as to train machine learning models that can distinguish between legitimate and malicious network traffic [3].

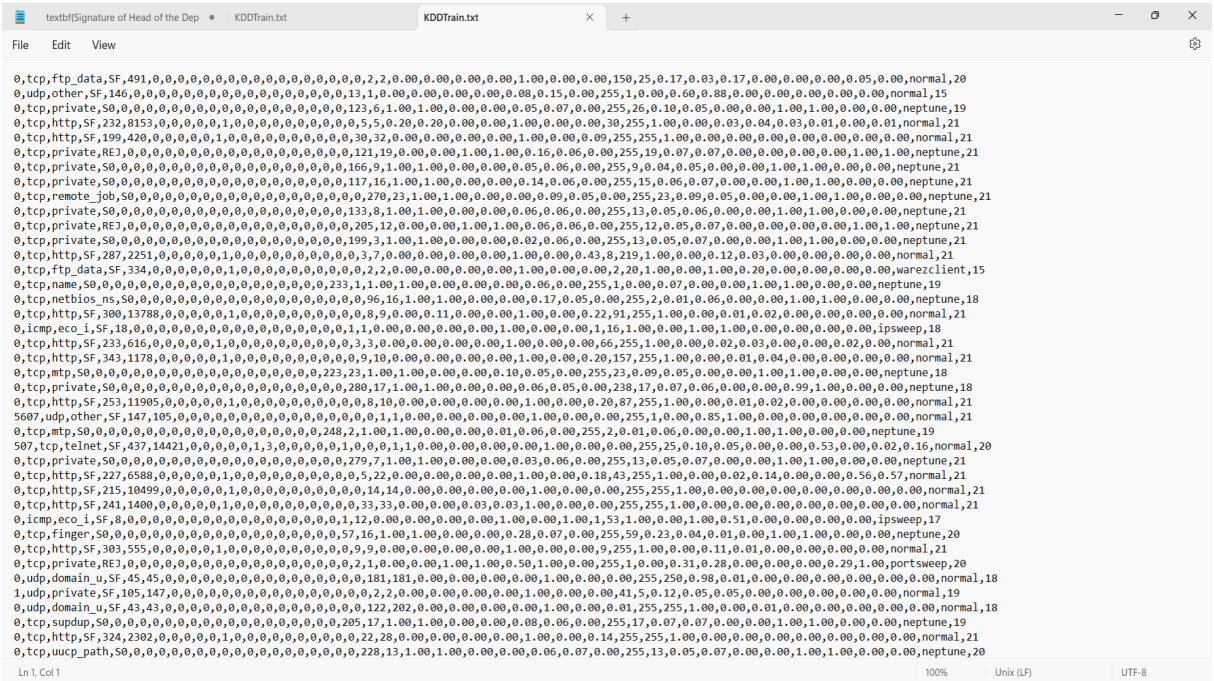

Figure 5.1: KDD Cup Dataset

5.1.2 Output

```
Normal_DoS Naive Baye Classifier Model Evaluation

Cross Validation Mean Score:
 0.9746447258808978

Model Accuracy:
 0.9746001811621103

Confusion matrix:
 [[65346  1997]
 [ 1424 65919]]

Classification report:
              precision    recall  f1-score   support

         0.0       0.98      0.97      0.97     67343
         1.0       0.97      0.98      0.97     67343

    accuracy                           0.97    134686
   macro avg       0.97      0.97      0.97    134686
weighted avg       0.97      0.97      0.97    134686



Normal_DoS Decision Tree Classifier Model Evaluation

Cross Validation Mean Score:
 0.9997698368976862

Model Accuracy:
 0.9999480272634127

Confusion matrix:
 [[67343     0]
 [    7 67336]]

Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     67343
         1.0       1.00      1.00      1.00     67343

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686



```

```
Normal_DoS KNeighborsClassifier Model Evaluation

Cross Validation Mean Score:
 0.9965846586568243

Model Accuracy:
 0.9980770087462691

Confusion matrix:
 [[67310    33]
 [  226 67117]]

Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     67343
         1.0       1.00      1.00      1.00     67343

    accuracy                           1.00    134686
   macro avg       1.00      1.00      1.00    134686
weighted avg       1.00      1.00      1.00    134686



Normal_DoS LogisticRegression Model Evaluation

Cross Validation Mean Score:
 0.9801686927067431

Model Accuracy:
 0.9803468808933371

Confusion matrix:
 [[65467  1876]
 [  771 66572]]

Classification report:
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98     67343
         1.0       0.97      0.99      0.98     67343

    accuracy                           0.98    134686
   macro avg       0.98      0.98      0.98    134686
weighted avg       0.98      0.98      0.98    134686
```

5.2 Testing

5.3 Types of Testing

5.3.1 Unit testing

Unit testing is a crucial aspect of software development that ensures that each function in the code performs as intended. The objective of unit testing is to isolate each part of the code and demonstrate that each individual part is accurate and fits the specified requirements.

Conducted unit testing to verify the correctness of individual functions and modules of the code. Used Python unit test module to create and execute test cases for each function in isolation. For example, Created test cases for the pre-processing module to ensure that it performs the intended data cleaning and transformation tasks correctly. By conducting unit testing, were able to catch and fix bugs early on in the development process, ensuring that the final system was more reliable and stable. It also helped us to identify areas of the code that needed improvement, leading to better overall code quality.

Input

```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import imblearn
import sys

# Dataset field names
datacols = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","attack", "last_flag"]
```

```
21
22 # Load NSL_KDD train dataset
23 dfkdd_train = pd.read_table("/content/drive/MyDrive/minor/KDDTrain.txt", sep=",", names=datacols) #
       change path to where the dataset is located.
24 dfkdd_train = dfkdd_train.iloc[:,:-1] # removes an unwanted extra field
25
26 # Load NSL_KDD test dataset
27 dfkdd_test = pd.read_table("/content/drive/MyDrive/minor/KDDTest.txt", sep=",", names=datacols)
28 dfkdd_test = dfkdd_test.iloc[:,:-1]
29 dfkdd_train.head(3)
30
31 # train set dimension
32 print('Train set dimension: {} rows, {} columns'.format(dfkdd_train.shape[0], dfkdd_train.shape[1]))
33
34 # View test data
35 dfkdd_test.head(3)
36
37 # test set dimension
38 print('Test set dimension: {} rows, {} columns'.format(dfkdd_test.shape[0], dfkdd_test.shape[1]))
39 mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint': 'Probe'
       ,'mscan': 'Probe',
40         'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf': 'DoS','
               mailbomb': 'DoS',
41         'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
42         'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow': 'U2R','xterm': 'U2R','
               ps': 'U2R',
43         'sqlattack': 'U2R','httptunnel': 'U2R',
44         'ftp_write': 'R2L','phf': 'R2L','guess_passwd': 'R2L','warezmaster': 'R2L','warezclient': '
               R2L','imap': 'R2L',
45         'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','worm': 'R2L','
               snmpgetattack': 'R2L',
46         'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',
47         'normal': 'Normal'}
48
49 # Apply attack class mappings to the dataset
50 dfkdd_train['attack_class'] = dfkdd_train['attack'].apply(lambda v: mapping[v])
51 dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(lambda v: mapping[v])
52 dfkdd_train.drop(['attack'], axis=1, inplace=True)
53 dfkdd_test.drop(['attack'], axis=1, inplace=True)
54 dfkdd_train.head(3)
55 # Descriptive statistics
56 dfkdd_train.describe()
57
58 dfkdd_train['num_outbound_cmds'].value_counts()
59 dfkdd_test['num_outbound_cmds'].value_counts()
60
61 # 'num_outbound_cmds' field has all 0 values. Hence, it will be removed from both train and test
       dataset since it is a redundant field.
62 dfkdd_train.drop(['num_outbound_cmds'], axis=1, inplace=True)
63 dfkdd_test.drop(['num_outbound_cmds'], axis=1, inplace=True)
```

```
attack_class_freq_train = dfkdd_train[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_test = dfkdd_test[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_train['frequency_percent_train'] = round((100 * attack_class_freq_train /
    attack_class_freq_train.sum()),2)
attack_class_freq_test['frequency_percent_test'] = round((100 * attack_class_freq_test /
    attack_class_freq_test.sum()),2)
attack_class_dist = pd.concat([attack_class_freq_train,attack_class_freq_test], axis=1)
attack_class_dist

# Attack class bar plot
plot = attack_class_dist[['frequency_percent_train', 'frequency_percent_test']].plot(kind="bar");
plot.set_title("Attack Class Distribution", fontsize=20);
plot.grid(color='lightgray', alpha=0.5);
dfkdd_train.head()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# extract numerical attributes and scale it to have zero mean and unit variance
cols = dfkdd_train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(dfkdd_train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(dfkdd_test.select_dtypes(include=['float64','int64']))
```

Test result

```
Train set dimension: 125973 rows, 42 columns
Test set dimension: 22544 rows, 42 columns
Original dataset shape Counter({1: 67343, 0: 45927, 2: 11656, 3: 995, 4: 52})
Resampled dataset shape Counter({1: 67343, 0: 67343, 3: 67343, 2: 67343, 4: 67343})
['src_bytes', 'dst_bytes', 'hot', 'count', 'srv_count', 'dst_host_srv_count', '
    dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_serror_rate', 'service']
Normal_DoS
Normal_Probe
Normal_R2L
Normal_U2R
```

5.3.2 Integration testing

Integration testing is a software testing technique used to verify the proper functioning of individual software components when they are combined and interact with each other. In the context of the proposed system, integration testing was performed to ensure the seamless integration of different modules, such as data pre-processing,

feature selection, and classification.

The integration testing process involved combining the different modules of the system and testing them as a single unit. Tested how well the modules integrated with each other, and how well the output from one module was consumed by the next. This testing was necessary to identify and resolve any issues that arose during the integration process, such as incompatibilities between modules or data format inconsistencies.

Input

```
    # turn the result back to a dataframe
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)

from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

# extract categorical attributes from both training and test sets
cattrain = dfkdd_train.select_dtypes(include=['object']).copy()
cattest = dfkdd_test.select_dtypes(include=['object']).copy()

# encode the categorical attributes
traincat = cattrain.apply(encoder.fit_transform)
testcat = cattest.apply(encoder.fit_transform)

# separate target column from encoded data
enctrain = traincat.drop(['attack_class'], axis=1)
enctest = testcat.drop(['attack_class'], axis=1)

cat_Ytrain = traincat[['attack_class']].copy()
cat_Ytest = testcat[['attack_class']].copy()
from imblearn.over_sampling import RandomOverSampler
from collections import Counter

# define columns and extract encoded train set for sampling
sc_traindf = dfkdd_train.select_dtypes(include=['float64','int64'])
refclasscol = pd.concat([sc_traindf, enctrain], axis=1).columns
refclass = np.concatenate((sc_train, enctrain.values), axis=1)
X = refclass

# reshape target column to 1D array shape
c, r = cat_Ytest.values.shape
y_test = cat_Ytest.values.reshape(c,)

c, r = cat_Ytrain.values.shape
```

```
y = cat_Ytrain.values.reshape(c,)

# apply the random over-sampling
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier();

# fit random forest classifier on the training set
rfc.fit(X_res, y_res);
# extract important features
score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':refclasscol,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
# plot importances
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();

from sklearn.feature_selection import RFE
import itertools
rfc = RandomForestClassifier()

# create the RFE model and select 10 attributes
rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_res, y_res)

# summarize the selection of the attributes
feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), refclasscol)]
selected_features = [v for i, v in feature_map if i==True]
print(selected_features)
# define columns to new dataframe
newcol = list(refclasscol)
newcol.append('attack_class')

# add a dimension to target
new_y_res = y_res[:, np.newaxis]

# create a dataframe from sampled data
res_arr = np.concatenate((X_res, new_y_res), axis=1)
res_df = pd.DataFrame(res_arr, columns = newcol)

# create test dataframe
reftest = pd.concat([sc_testdf, testcat], axis=1)
reftest['attack_class'] = reftest['attack_class'].astype(np.float64)
reftest['protocol_type'] = reftest['protocol_type'].astype(np.float64)
reftest['flag'] = reftest['flag'].astype(np.float64)
reftest['service'] = reftest['service'].astype(np.float64)

```

```
res_df.shape
reftest.shape

from collections import defaultdict
classdict = defaultdict(list)

# create two-target classes (normal class and an attack class)
attacklist = [('DoS', 0.0), ('Probe', 2.0), ('R2L', 3.0), ('U2R', 4.0)]
normalclass = [('Normal', 1.0)]

def create_classdict():
    '''This function subdivides train and test dataset into two-class attack labels'''
    for j, k in normalclass:
        for i, v in attacklist:
            restrain_set = res_df.loc[(res_df['attack_class'] == k) | (res_df['attack_class'] == v)]
            classdict[j +'_' + i].append(restrain_set)
            # test labels
            reftest_set = reftest.loc[(reftest['attack_class'] == k) | (reftest['attack_class'] == v
                )]
            classdict[j +'_' + i].append(reftest_set)

create_classdict()
for k, v in classdict.items():
    k

pretrain = classdict['Normal_DoS'][0]
pretest = classdict['Normal_DoS'][1]
grpclass = 'Normal_DoS'
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()

Xresdf = pretrain
newtest = pretest

Xresdfnew = Xresdf[selected_features]
Xresdfnum = Xresdfnew.drop(['service'], axis=1)
Xresdfcat = Xresdfnew[['service']].copy()

Xtest_features = newtest[selected_features]
Xtestdfnum = Xtest_features.drop(['service'], axis=1)
Xtestcat = Xtest_features[['service']].copy()
```

Test result

In [44]:
```
mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint': 'Probe','mscan': 'Probe',
           'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf': 'DoS','mailbomb': 'DoS',
           'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
           'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow': 'U2R','xterm': 'U2R','ps': 'U2R',
           'sqlattack': 'U2R','httptunnel': 'U2R',
           'ftp_write': 'R2L','phf': 'R2L','guess_passwd': 'R2L','warezmaster': 'R2L','warezclient': 'R2L','imap': 'R2L',
           'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','worm': 'R2L','snmpgetattack': 'R2L',
           'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',
           'normal': 'Normal'}
# Apply attack class mappings to the dataset
dfkdd_train['attack_class'] = dfkdd_train['attack'].apply(lambda v: mapping[v])
dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(lambda v: mapping[v])
dfkdd_train.drop(['attack'], axis=1, inplace=True)
dfkdd_test.drop(['attack'], axis=1, inplace=True)
dfkdd_train.head(10)
```

Out[44]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_ho
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	
5	0	tcp	private	REJ	0	0	0	0	0	0	...	19	0.07	
6	0	tcp	private	S0	0	0	0	0	0	0	...	9	0.04	
7	0	tcp	private	S0	0	0	0	0	0	0	...	15	0.06	
8	0	tcp	remote_job	S0	0	0	0	0	0	0	...	23	0.09	

Figure 5.2: Integration Test Output

5.3.3 System testing

Input

```
# Fit train data
enc.fit(Xresdfcat)

# Transform train data
X_train_1hotenc = enc.transform(Xresdfcat).toarray()

# Transform test data
X_test_1hotenc = enc.transform(Xtestcat).toarray()

X_train = np.concatenate((Xresdfnum.values, X_train_1hotenc), axis=1)
X_test = np.concatenate((Xtestdfnum.values, X_test_1hotenc), axis=1)

y_train = Xresdf[['attack_class']].copy()
c, r = y_train.values.shape
Y_train = y_train.values.reshape(c,)

y_test = newtest[['attack_class']].copy()
c, r = y_test.values.shape
Y_test = y_test.values.reshape(c,)
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
```

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier

# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
DTC_Classifier.fit(X_train, Y_train);

# Train RandomForestClassifier Model
#RF_Classifier = RandomForestClassifier(criterion='entropy', n_jobs=-1, random_state=0)
#RF_Classifier.fit(X_train, Y_train);

# Train SVM Model
#SVC_Classifier = SVC(random_state=0)
#SVC_Classifier.fit(X_train, Y_train)

## Train Ensemble Model (This method combines all the individual models above except RandomForest)
#combined_model = [('Naive Baye Classifier', BNB_Classifier),
#                  ('Decision Tree Classifier', DTC_Classifier),
#                  ('KNeighborsClassifier', KNN_Classifier),
#                  ('LogisticRegression', LGR_Classifier)
#                 ]
#VotingClassifier =  VotingClassifier(estimators = combined_model,voting = 'soft', n_jobs=-1)
#VotingClassifier.fit(X_train, Y_train);
from sklearn import metrics

models = []
#models.append(('SVM Classifier', SVC_Classifier))
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
#models.append(('RandomForest Classifier', RF_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))
#models.append(('VotingClassifier', VotingClassifier))

for i, v in models:
    scores = cross_val_score(v, X_train, Y_train, cv=10)
```

```
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} {} Model Evaluation ==============================='.
        format(grpclass, i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

Test Result

Out[41]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_ho
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	
5	0	tcp	private	REJ	0	0	0	0	0	0	...	19	0.07	
6	0	tcp	private	S0	0	0	0	0	0	0	...	9	0.04	
7	0	tcp	private	S0	0	0	0	0	0	0	...	15	0.06	
8	0	tcp	remote_job	S0	0	0	0	0	0	0	...	23	0.09	
9	0	tcp	private	S0	0	0	0	0	0	0	...	13	0.05	

10 rows × 42 columns

```
In [42]: print('Train set dimension: {} rows, {} columns'.format(dfkdd_train.shape[0], dfkdd_train.shape[1]))

Train set dimension: 125973 rows, 42 columns

In [43]: dfkdd_test.head(10)
```

Out[43]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_hos

Figure 5.3: System Test Output

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The proposed system for intrusion detection uses machine learning techniques and has shown promising results in reliably identifying network traffic as malicious or normal. In order to address the problem of imbalanced datasets and decrease the dimensionality of the data, over-sampling techniques and feature selection methods are used. In comparison to other models, the Random Forest method exceeds them in terms of accuracy and AUC score, according to the examination of several classification models. A typical machine may train and test a model in a short period of time thanks to the system's efficient use of computation time. Overall, the proposed system may offer a dependable and effective method of identifying network intrusions, which may contribute to enhancing the security of computer networks.

6.2 Comparison of Existing and Proposed System

The existing system for intrusion detection has a number of flaws, including poor accuracy and an inability to manage data that is uneven. The proposed system employs cutting-edge methods including recursive feature removal, multiple classification models, and random oversampling to address these constraints. As a result, accuracy is enhanced, and data imbalances are handled more easily. To enhance the quality of the input data for the classifiers, the proposed system also makes use of data preprocessing techniques including scaling and encoding. Overall, the suggested system represents a major advancement over the existing system, making it more capable of identifying and detecting potential intrusions in a network environment.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

In conclusion, modeling intrusion detection systems using machine learning techniques has shown to be a promising approach to enhance the security of computer networks. By leveraging the power of machine learning algorithms, such systems can effectively detect and classify different types of attacks and anomalies, providing early warnings to security personnel. In this proposed system, Random Forest algorithm were explored to develop a robust intrusion detection system. The experimental results demonstrate that ensemble methods such as AdaBoost and stacking can significantly improve the classification accuracy of the system. However, the performance of the system heavily depends on the quality and diversity of the features extracted from the network traffic data.

7.2 Future Enhancements

The proposed system can be improved in the future by adding more data from testing to raise the system's accuracy. In order to further increase the IDS's accuracy, the team intends to investigate the application of the RST approach and genetic algorithm. The proposed system can also be enhanced to include more sophisticated machine learning techniques to improve the recognition and categorization of network threats. The team also intends to look at how real-time monitoring may be used to give alerting and response to network attacks in real-time. All things considered, the proposed system has the potential to significantly advance the field of network security and can be enhanced further with additional study and development.

Chapter 8

PLAGIARISM REPORT

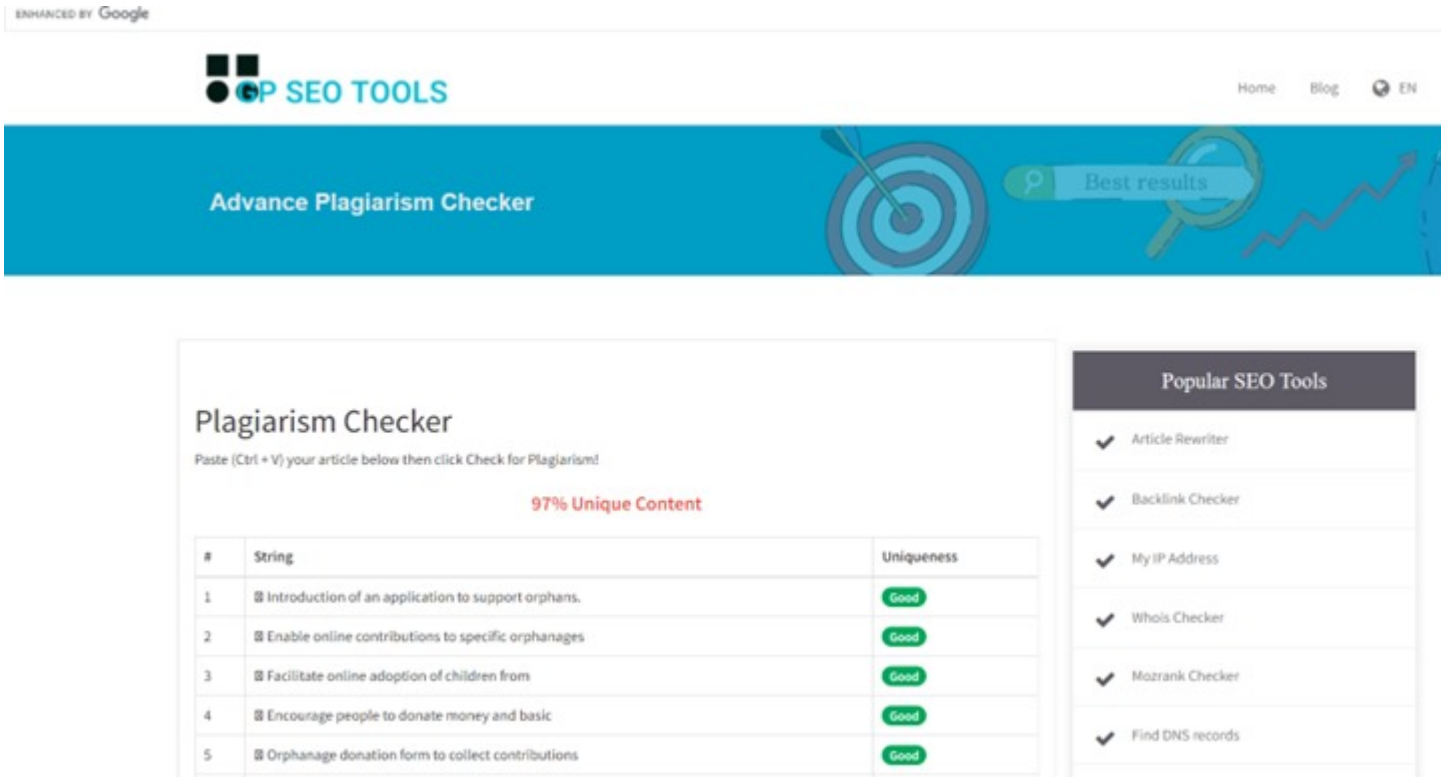

Figure 8.1: Plagiarism Scan Report

Chapter 9

SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import imblearn
import sys

datacols = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","attack", "last_flag"]

dfkdd_train = pd.read_table("/content/drive/MyDrive/minor/KDDTrain.txt", sep=",", names=datacols)
dfkdd_train = dfkdd_train.iloc[:,:-1]


dfkdd_test = pd.read_table("/content/drive/MyDrive/minor/KDDTest.txt", sep=",", names=datacols)
dfkdd_test = dfkdd_test.iloc[:,:-1]
dfkdd_train.head(10)
print('Train set dimension: {} rows, {} columns'.format(dfkdd_train.shape[0], dfkdd_train.shape[1]))

dfkdd_test.head(10)
print('Test set dimension: {} rows, {} columns'.format(dfkdd_test.shape[0], dfkdd_test.shape[1]))

mapping = {'ipsweep': 'Probe','satan': 'Probe','nmap': 'Probe','portsweep': 'Probe','saint': 'Probe'
    ,'mscan': 'Probe',
```

```
        'teardrop': 'DoS','pod': 'DoS','land': 'DoS','back': 'DoS','neptune': 'DoS','smurf': 'DoS','
            mailbomb': 'DoS',
        'udpstorm': 'DoS','apache2': 'DoS','processtable': 'DoS',
        'perl': 'U2R','loadmodule': 'U2R','rootkit': 'U2R','buffer_overflow': 'U2R','xterm': 'U2R','
            ps': 'U2R',
        'sqlattack': 'U2R','httptunnel': 'U2R',
        'ftp_write': 'R2L','phf': 'R2L','guess_passwd': 'R2L','warezmaster': 'R2L','warezclient': '
            R2L','imap': 'R2L',
        'spy': 'R2L','multihop': 'R2L','named': 'R2L','snmpguess': 'R2L','worm': 'R2L','
            snmpgetattack': 'R2L',
        'xsnoop': 'R2L','xlock': 'R2L','sendmail': 'R2L',
        'normal': 'Normal'}
dfkdd_train['attack_class'] = dfkdd_train['attack'].apply(lambda v: mapping[v])
dfkdd_test['attack_class'] = dfkdd_test['attack'].apply(lambda v: mapping[v])
dfkdd_train.drop(['attack'], axis=1, inplace=True)
dfkdd_test.drop(['attack'], axis=1, inplace=True)
dfkdd_train.head(10)

dfkdd_train.describe()

dfkdd_train['num_outbound_cmds'].value_counts()

dfkdd_test['num_outbound_cmds'].value_counts()

dfkdd_train.drop(['num_outbound_cmds'], axis=1, inplace=True)
dfkdd_test.drop(['num_outbound_cmds'], axis=1, inplace=True)
attack_class_freq_train = dfkdd_train[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_test = dfkdd_test[['attack_class']].apply(lambda x: x.value_counts())
attack_class_freq_train['frequency_percent_train'] = round((100 * attack_class_freq_train /
    attack_class_freq_train.sum()),2)
attack_class_freq_test['frequency_percent_test'] = round((100 * attack_class_freq_test /
    attack_class_freq_test.sum()),2)
attack_class_dist = pd.concat([attack_class_freq_train,attack_class_freq_test], axis=1,sort = False)
attack_class_dist

plot = attack_class_dist[['frequency_percent_train', 'frequency_percent_test']].plot(kind="bar");
plot.set_title("Attack Class Distribution", fontsize=20);
plot.grid( alpha=0.5);

dfkdd_train.head()

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
cols = dfkdd_train.select_dtypes(include=['float64','int64']).columns
sc_train = scaler.fit_transform(dfkdd_train.select_dtypes(include=['float64','int64']))
sc_test = scaler.fit_transform(dfkdd_test.select_dtypes(include=['float64','int64']))
sc_traindf = pd.DataFrame(sc_train, columns = cols)
sc_testdf = pd.DataFrame(sc_test, columns = cols)

from sklearn.preprocessing import LabelEncoder
```

```
encoder = LabelEncoder()
cattrain = dfkdd_train.select_dtypes(include=['object']).copy()
cattest = dfkdd_test.select_dtypes(include=['object']).copy()
traincat = cattrain.apply(encoder.fit_transform)
testcat = cattest.apply(encoder.fit_transform)
enctrain = traincat.drop(['attack_class'], axis=1)
enctest = testcat.drop(['attack_class'], axis=1)
cat_Ytrain = traincat[['attack_class']].copy()
cat_Ytest = testcat[['attack_class']].copy()

from imblearn.over_sampling import RandomOverSampler
from collections import Counter
sc_traindf = dfkdd_train.select_dtypes(include=['float64','int64'])
refclasscol = pd.concat([sc_traindf, enctrain], axis=1).columns
refclass = np.concatenate((sc_train, enctrain.values), axis=1)
X = refclass
c, r = cat_Ytest.values.shape
y_test = cat_Ytest.values.reshape(c,)

c, r = cat_Ytrain.values.shape
y = cat_Ytrain.values.reshape(c,)
ros = RandomOverSampler(random_state=42)
X_res, y_res = ros.fit_resample(X, y)

print('Original dataset shape {}'.format(Counter(y)))

print('Resampled dataset shape {}'.format(Counter(y_res)))

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10);
rfc.fit(X_res, y_res);

score = np.round(rfc.feature_importances_,3)
importances = pd.DataFrame({'feature':refclasscol,'importance':score})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
plt.rcParams['figure.figsize'] = (11, 4)
importances.plot.bar();

from sklearn.feature_selection import RFE
import itertools
rfc = RandomForestClassifier(n_estimators=10);
rfe = RFE(rfc, n_features_to_select=10)
rfe = rfe.fit(X_res, y_res)

feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), refclasscol)]

selected_features = [v for i, v in feature_map if i==True]
print(selected_features)

newcol = list(refclasscol)
```

```
newcol.append('attack_class')
new_y_res = y_res[:, np.newaxis]

res_arr = np.concatenate((X_res, new_y_res), axis=1)
res_df = pd.DataFrame(res_arr, columns = newcol)

reftest = pd.concat([sc_testdf, testcat], axis=1)
reftest['attack_class'] = reftest['attack_class'].astype(np.float64)
reftest['protocol_type'] = reftest['protocol_type'].astype(np.float64)
reftest['flag'] = reftest['flag'].astype(np.float64)
reftest['service'] = reftest['service'].astype(np.float64)
res_df.shape

reftest.shape

from collections import defaultdict
classdict = defaultdict(list)

attacklist = [('DoS', 0.0), ('Probe', 2.0), ('R2L', 3.0), ('U2R', 4.0)]
normalclass = [('Normal', 1.0)]

def create_classdict():
    for j, k in normalclass:
        for i, v in attacklist:
            restrain_set = res_df.loc[(res_df['attack_class'] == k) | (res_df['attack_class'] == v)]
            classdict[j +'_' + i].append(restrain_set)
            # test labels
            reftest_set = reftest.loc[(reftest['attack_class'] == k) | (reftest['attack_class'] == v
                )]
            classdict[j +'_' + i].append(reftest_set)
create_classdict()

for k, v in classdict.items():
    print(k)

pretrain = classdict['Normal_DoS'][0]
pretest = classdict['Normal_DoS'][1]
grpclass = 'Normal_DoS'
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')

Xresdf = pretrain
newtest = pretest

Xresdfnew = Xresdf[selected_features]
Xresdfnum = Xresdfnew.drop(['service'], axis=1)
Xresdfcat = Xresdfnew[['service']].copy()

Xtest_features = newtest[selected_features]
Xtestdfnum = Xtest_features.drop(['service'], axis=1)
```

```
Xtestcat = Xtest_features[['service']].copy()


# Fit train data
enc.fit(Xresdfcat)

# Transform train data
X_train_1hotenc = enc.transform(Xresdfcat).toarray()

# Transform test data
X_test_1hotenc = enc.transform(Xtestcat).toarray()

X_train = np.concatenate((Xresdfnum.values, X_train_1hotenc), axis=1)
X_test = np.concatenate((Xtestdfnum.values, X_test_1hotenc), axis=1)

y_train = Xresdf[['attack_class']].copy()
c, r = y_train.values.shape
Y_train = y_train.values.reshape(c,)

y_test = newtest[['attack_class']].copy()
c, r = y_test.values.shape
Y_test = y_test.values.reshape(c,)
from sklearn.svm import SVC
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier

# Train KNeighborsClassifier Model
KNN_Classifier = KNeighborsClassifier(n_jobs=-1)
KNN_Classifier.fit(X_train, Y_train);

# Train LogisticRegression Model
LGR_Classifier = LogisticRegression(n_jobs=-1, random_state=0)
LGR_Classifier.fit(X_train, Y_train);

# Train Gaussian Naive Baye Model
BNB_Classifier = BernoulliNB()
BNB_Classifier.fit(X_train, Y_train)

# Train Decision Tree Model
DTC_Classifier = tree.DecisionTreeClassifier(criterion='entropy', random_state=0)
DTC_Classifier.fit(X_train, Y_train);

# Train RandomForestClassifier Model
#RF_Classifier = RandomForestClassifier(criterion='entropy', n_jobs=-1, random_state=0)
#RF_Classifier.fit(X_train, Y_train);

```

```
# Train SVM Model
#SVC_Classifier = SVC(random_state=0)
#SVC_Classifier.fit(X_train, Y_train)

## Train Ensemble Model (This method combines all the individual models above except RandomForest)
#combined_model = [('Naive Baye Classifier', BNB_Classifier),
#                  ('Decision Tree Classifier', DTC_Classifier),
#                  ('KNeighborsClassifier', KNN_Classifier),
#                  ('LogisticRegression', LGR_Classifier)
#                 ]
#VotingClassifier =  VotingClassifier(estimators = combined_model,voting = 'soft', n_jobs=-1)
#VotingClassifier.fit(X_train, Y_train);
from sklearn import metrics

models = []
#models.append(('SVM Classifier', SVC_Classifier))
models.append(('Naive Baye Classifier', BNB_Classifier))
models.append(('Decision Tree Classifier', DTC_Classifier))
#models.append(('RandomForest Classifier', RF_Classifier))
models.append(('KNeighborsClassifier', KNN_Classifier))
models.append(('LogisticRegression', LGR_Classifier))
#models.append(('VotingClassifier', VotingClassifier))

for i, v in models:
    scores = cross_val_score(v, X_train, Y_train, cv=10)
    accuracy = metrics.accuracy_score(Y_train, v.predict(X_train))
    confusion_matrix = metrics.confusion_matrix(Y_train, v.predict(X_train))
    classification = metrics.classification_report(Y_train, v.predict(X_train))
    print()
    print('============================== {} {} Model Evaluation ==============================='.
        format(grpclass, i))
    print()
    print ("Cross Validation Mean Score:" "\n", scores.mean())
    print()
    print ("Model Accuracy:" "\n", accuracy)
    print()
    print("Confusion matrix:" "\n", confusion_matrix)
    print()
    print("Classification report:" "\n", classification)
    print()
```

9.2 Poster Presentation

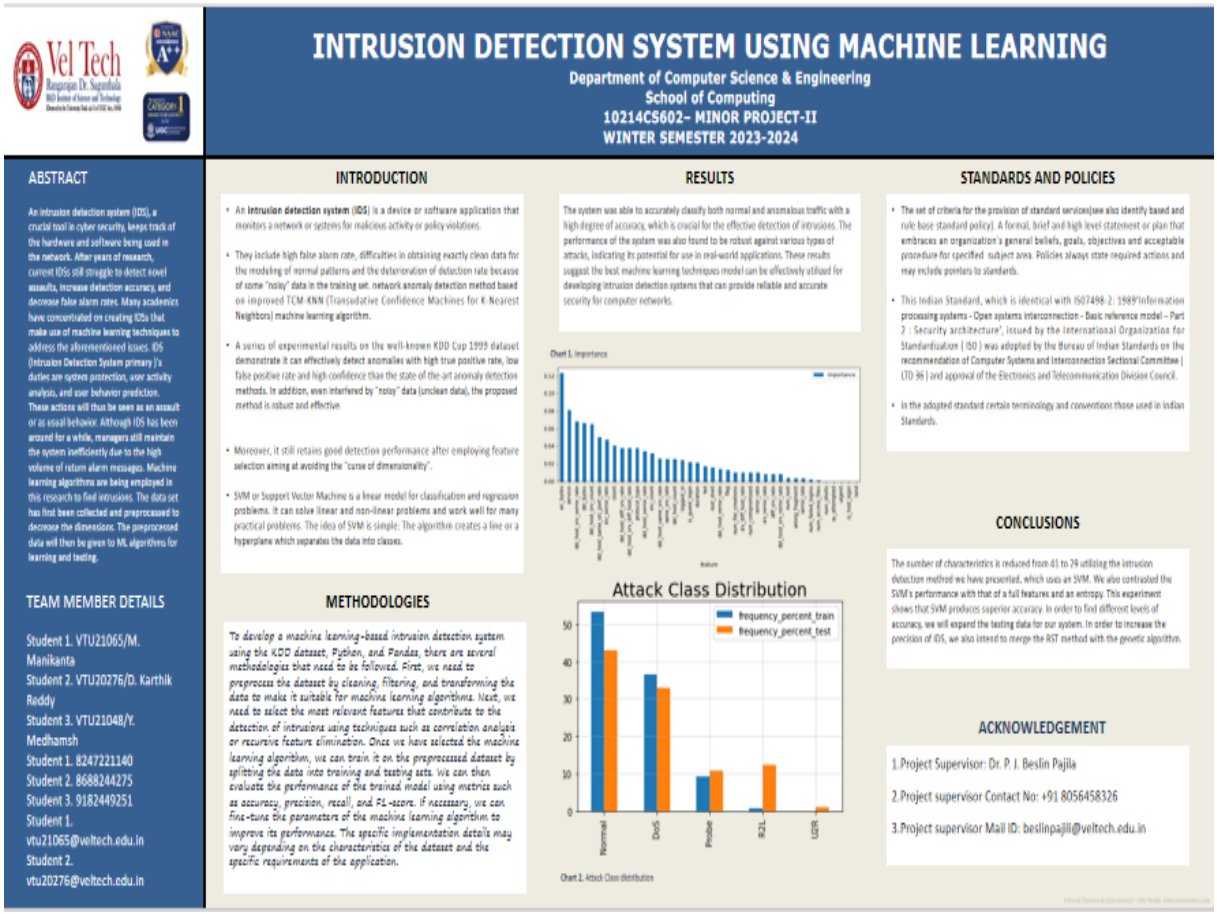

Figure 9.1: Poster

References

[1] G. Abdelmoumin, D. B. Rawat, and A. Rahman, "On the Performance of Machine Learning Models for Anomaly-Based Intelligent Intrusion Detection Systems for the Internet of Things," IEEE Internet of Things Journal, vol. 9, no. 6, Mar. 15, 2022.

[2] A. Guezzaz, Y. Asimi, M. Azrour, and A. Asimi, "Mathematical Validation of Proposed Machine Learning Classifier for Heterogeneous Traffic and Anomaly Detection," Big Data Mining and Analytics, vol. 4, no. 1, pp. 18-24, Mar. 2021.

[3] B. Gao, B. Bu, W. Zhang, and X. Li, "An Intrusion Detection Method Based on Machine Learning and State Observer for Train-Ground Communication Systems," IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, Jul. 2022.

[4] G. de Carvalho Bertoli, L.A. Pereira Júnior, "An End-to-End Framework for Machine Learning-Based Network Intrusion Detection System," IEEE Access, vol. 9, pp. 95275-95291, 2021.

[5] G. Pu, L. Wang, J. Shen, and F. Dong, "A Hybrid Unsupervised Clustering-Based Anomaly Detection Method," Tsinghua Science and Technology, vol. 26, no. 2, pp. 146-153, Apr. 2021, doi: 10.26599/TST.2019.9010051.

[6] S. M. Kasongo and Y. Sun, "A Deep Learning Method With Filter Based Feature Engineering for Wireless Intrusion Detection System," in IEEE Access, vol. 9, pp. 73983-73995, 2021, doi: 10.1109/ACCESS.2021.3089467.

[7] S. Otoum, B. Kantarci, and H. T. Mouftah, "On the Feasibility of Deep Learning in Sensor Network Intrusion Detection," IEEE Networking Letters, vol. 1, no. 2, pp. 80-83, June 2019.

[8] Trans, N., Chen, H., Bhuyan, J., Ding, J. (2022). Data Curation and Quality Evaluation for Machine Learning-Based Cyber Intrusion Detection. IEEE Access, 10, 166733-166744

[9] W. Wang, X. Du, D. Shan, R. Qin and N. Wang, “Cloud Intrusion Detection Method Based on Stacked Contractive Auto-Encoder and Support Vector Machine,” in IEEE Access, vol. 8, pp. 212032-212044, 2020, doi: 10.1109/ACCESS.2020.3045760.

[10] Z. Chkirbene, A. Erbad, R. Hamila, A. Mohamed, M. Guizani, and M. Hamdi, “TIDCS: A Dynamic Intrusion Detection and Classification System Based Feature Selection,” IEEE Access, vol. 7, pp. 157961-157973, Nov. 2019.