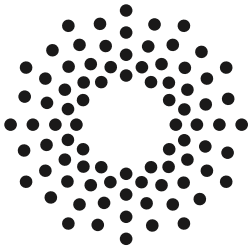
MSc Artificial Intelligence
Spring 2023

Anomaly Detection for Network Traffic Using Machine Learning



LONDON METROPOLITAN UNIVERSITY

London Metropolitan University Faculty of Computing
School of Computing and Digital Media

Anomaly Detection for Network Traffic Using Machine Learning

Submitted by
Kusmakhar Pathak
21035155

Supervisor
Professor Karim Ouazzane

Date of Submission : 17 May 2023

Acknowledgement

I want to express my deep gratitude to Professor Karim Ouzzane, who is my supervisor, for his invaluable guidance and help with my MSc project and report. His expertise, insight, and dedication have had a significant impact on and enhanced my education.

Professor Karim Ouzzane's in-depth knowledge of network security and machine learning has had a significant impact on the creation and application of the suggested anomaly detection method. His suggestions and constructive criticism have consistently motivated me to strive for excellence, which has enabled me to overcome challenges and make substantial academic progress.

I greatly appreciate the advice, patience, and assistance provided by Professor Karim Ouzzane. This thesis's direction was greatly influenced by his steadfast encouragement, willingness to engage in dialogue, and sharp insights. I owe him a great deal of thanks for giving me the opportunity to work under his guidance, as it has been a really rewarding and enriching experience.

Thank you,

Kusmakhar Pathak
Student ID: 21035155

Abstract

Network security and the identification of possible threats depend heavily on anomaly detection in network traffic. In order to determine which machine learning models are most useful in detecting network traffic anomalies, this research compared and evaluated a number of different machine learning models. Deep Learning with 10 epochs, Logistic Regression, Random Forest with a Filter method, Random Forest with a Wrapper method, and Random Forest with a Wrapper method were the models evaluated.

A number of metrics were used for evaluation, including cross-validation, accuracy, F1-Score, ROC (Receiver Operating Characteristic), and precision-recall. The results showed that the Random Forest with the Wrapper approach and the Deep Learning model outperformed other assessment criteria. With a ROC score of 0.9716, the Deep Learning model had the best performance, clearly demonstrating its superior ability to differentiate between regular and abnormal network traffic. It also displayed outstanding precision-recall scores of 0.9621, indicating accurate anomaly identification with a low incidence of false positives. The Deep Learning model also attained a strong F1-Score of 0.8405, which represents a balanced mix of recall and precision.

Multiple areas for further research were found to further enhance the anomaly detection model. These included examining more features for improved discrimination, utilising ensemble methods to combine many models, examining various deep learning architectures, and utilising transfer learning strategies. As important future research areas, it was also determined that improving model interpretability, enabling real-time monitoring, and tackling resilience against adversarial attacks. It was also noted that deployment scalability, computational effectiveness, and integration with current network infrastructure were important factors.

The accuracy, interpretability, real-time monitoring, and viability of deployment of the network traffic anomaly detection model can all be improved by solving these issues. These upgrades will improve network security by lowering false positive and negative detection rates and enhancing the ability of security and network staff to make informed decisions.

The efficacy of various machine learning models for network traffic anomaly detection is examined in detail in this research, which offers helpful insights. The results provide a basis for future network security and anomaly detection research and development while also guiding the choice of appropriate models.

Table of Contents

Abbreviation

HTTP	=	*Hypertext Transfer Protocol*
FTP	=	*File Transfer Protocol*
SMTP	=	*Simple Mail Transfer Protocol*
DNS	=	*Domain Name System*
DDOS	=	*Distributed Denial of Service*
NetFlow	=	*Network Flow*
sFlow	=	*Sampled Flow*
DPI	=	*deep packet inspection*
IDS	=	*intrusion detection systems*
IPS	=	*intrusion prevention systems*
PCA	=	*principal component analysis*
SVM	=	*Support Vector Machines*
CNN	=	*convolutional neural networks*
RNN	=	*Recurrent Neural Network*
ANN	=	*Artificial neural networks*
k-NN	=	*k-Nearest Neighbours*
RL	=	*Reinforcement Learning*
SDN	=	*Software-Defined Networking*
DQN	=	*Deep Q-Network*
AD-DRL	=	*Adversarial Deep Reinforcement Learning*
IID	=	*Identically Distributed*
IDE	=	*Integrated Development Environment*
API	=	*Application Programming Interface*
CIC	=	*Canadian Institute for Cybersecurity*
ACCS	=	*Australian Centre for Cyber Security*
UNSW	=	*University of New South Wales*
PCAP	=	*Packet Capture*
SMOTE	=	*synthetic minority over-sampling technique*
RFE	=	*Recursive feature elimination*
GBM	=	*Gradient Boosting Machines*
ROC	=	*Receiver Operating Characteristic*
AUC	=	*AUC: Area Under the Curve*
ACCS	=	*Australian Centre for Cyber Security*
RAM	=	*Random Access Memory*
GB	=	*Gigabyte*
SSD	=	*Solid State Drive*
PC	=	*Personal Computer*
GUI	=	*Graphical User Interface*
RFC	=	*Random Forest Classifier*
SOC	=	*Security Operations Centre*

List of Figures

List of Table

Chapter 1 Introduction

1.1 Introduction

It is becoming increasingly difficult for network administrators to identify unusual activities due to the ever-increasing volume and complexity of network traffic in the modern world. Detecting anomalies is essential for keeping networks safe and functional. Detecting unknown and complex attacks can be difficult using the standard anomaly detection methods based on rule-based and signature-based techniques. Because of their ability to learn from data and identify previously unknown and complex attacks, machine learning-based anomaly detection techniques have grown in popularity in recent years.

The purpose of this research is to investigate whether machine learning algorithms can be used to detect anomalies in network traffic. The goal of this project is to create a system that can identify and categorise unusual network activity. Using supervised and unsupervised machine learning techniques, the system will be trained on a large dataset of network traffic to detect patterns and anomalies.

These research questions will guide this project:

1. What are the most important characteristics of network traffic for applying machine learning to detect anomalies?
2. What are the advantages and disadvantages of the various machine learning algorithms, and which ones work best for spotting anomalies in network traffic?
3. What metrics can be used to assess the performance of the machine learning-based anomaly detection system?
4. How can we overcome the difficulties associated with using machine learning to detect anomalies in network traffic?

Here's how the rest of the report is laid out: In the literature review, we provide an overview of prior work on the topic of detecting anomalies in network traffic. We see in this survey that various machine learning algorithms have been used to tackle this issue, including unsupervised, supervised, and semi-supervised approaches. We discuss the challenges of anomaly detection, such as the high dimensionality of network traffic data and the need to find a balance between the accuracy of detection and the possibility of false positives.

In the third section, we examine the data in detail. The massive dataset contains information about many kinds of networks and applications. The dataset also includes labelled data suitable for supervised learning techniques.

Our methodology for developing the anomaly detection system is outlined below. The methodology includes procedures for de-noising data, extracting useful features, choosing the best model, and assessing its performance. Machine learning models are compared, including autoencoders, SVMs, and decision trees.

In the fifth section, we present the results of the system evaluation. The findings show that the developed anomaly detection system can effectively identify abnormal traffic while minimising false positives.

Finally, we show that machine learning methods can be used to detect anomalies in network traffic. The ability of an organisation to detect and react to anomalies is crucial in protecting its networks from cyber threats. Further research in this area, such as the examination of more advanced machine learning methods and the improvement of the system's performance on novel and previously unknown data, will benefit from the groundwork laid by this work.

1.2 Business Understanding

Companies in the modern day rely largely on interconnected computer systems. Loss of productivity, money, and consumer trust can arise from any problems with these systems. As a result, protecting network resources requires the capacity to identify unusual activity in network traffic.

The goal of this effort is to create an anomaly detection system for network traffic based on machine learning so that enterprises may proactively detect and respond to potential security threats and performance issues. As a result of the system's usage of machine learning algorithms, firms will be able to notice anomalies more rapidly and take action to mitigate their potential harm.

Detecting anomalies, or unusual occurrences, in network traffic is an important part of maintaining network security. Rule-based algorithms, which are used in traditional approaches to anomaly identification, have limitations when it comes to spotting nuanced and sophisticated anomalies. On the other hand, anomaly detection in network traffic using machine learning algorithms has demonstrated encouraging results.

The suggested system has several potential applications in the corporate world, including banking, healthcare, and e-commerce. A company's data, customers, and financial line are all at risk in any of these situations, but the ability to spot irregularities in network traffic can help.

Financial firms, for instance, process and send vast amounts of private client information. Significant financial losses and damage to the institution's reputation

could arise from any unauthorised access or exploitation of sensitive data. A machine learning-based anomaly detection system can help banks and other financial organisations spot and head off potential security risks.

Networked systems are also crucial for healthcare professionals to handle patient data and deliver care. Delays in patient care and even serious injury can occur from any problems with these systems. Healthcare providers can prevent downtime and improve system uptime by monitoring for anomalies in real time using a machine learning-based anomaly detection system.

In conclusion, the suggested machine learning-based anomaly detection system might be extremely useful to organisations across a wide range of sectors. The technology can aid businesses in protecting their data, customers, and bottom line by enhancing the accuracy and efficiency of anomaly detection in network traffic.

1.3 Aims and Objectives

The success of this project depends on our ability to build a reliable machine learning-based system for spotting anomalies in network data, which is essential for keeping systems safe. As more and more apps make use of networks, solid security measures are more crucial than ever. Detecting anomalies is an important part of keeping a network safe, as doing so might reveal hidden weaknesses. We hope that by applying machine learning techniques to the problem of detecting anomalies in network data, managers would be able to take more preventative measures against security breaches.

We've established a series of subgoals that will help us realise this overarching purpose. The first step is to catalogue the various network traffic anomalies that can occur. When we know what kinds of abnormalities to look for, we can create a system that reliably detects them. Port scanning, denial of service assaults, and attempted intrusions are all examples of the kinds of abnormalities that we will investigate. To create efficient machine learning models for spotting these outliers, we will examine their patterns of activity.

The second goal is to find a good dataset for using with ML algorithms for training and testing. The dataset must demonstrate both typical and unusual network activity. We'll be making use of open-source data sets like NSL-KDD, CICIDS2017 and UNSW-NB15 to get the job done. The performance of the machine learning algorithms can be improved by pre-processing the dataset to get rid of noise, missing values, and other irregularities. Methods for pre-processing data include normalisation, transformation, and feature selection.

The project's third goal is to create machine learning models to analyse network traffic for irregularities. To find the best method for identifying anomalies in network data, we will investigate various machine learning techniques, such as supervised, unsupervised, and reinforcement learning. Decision trees, random forests, support vector machines, neural networks, and clustering techniques will be among the models we create. Ensemble approaches, which use numerous models in tandem to boost precision and reliability, will also be investigated.

We will utilise metrics such as precision, recall, F1 score, and ROC curves to assess the effectiveness of our machine learning models. Using these measures, we can learn more about the models' precision, efficiency, and generalizability. In order to fine-tune the models' hyperparameters, we'll employ cross-validation and grid search methods.

The project's fourth goal is to put into action the most efficient machine learning model for real-time anomaly detection. The system needs real-time network traffic anomaly detection and notifications for system administrators. The chosen machine learning model will be included into a real-time network monitoring system. Anomalies, both known and unknown, will be flagged so that network managers can take corrective measures.

The project's fifth and final goal is to verify the system's functionality with data from the real world. We will validate the system's accuracy and efficiency by testing its capacity to detect both known and unknown anomalies in network data. To validate the system, it will be put through its paces in detecting anomalies in network traffic using a simulation of real-world data. The system's efficiency will be measured against the outcomes of the dataset employed in the training and testing of the machine learning models.

By accomplishing these goals, we will be one step closer to creating an efficient machine learning-based system for monitoring network traffic in search of anomalies that may be used to strengthen security and head off attacks. Insights into the efficacy of various machine learning algorithms for anomaly identification in network traffic will be gained, and this will be the project's contribution to the field of network security. Network administrators will also benefit from the project because it will give them a real-time capability to detect security flaws and dangers in their networks.

1.4 Scope and Limitation

The goal of this study is to create a network traffic anomaly detection system based on machine learning. The following elements are covered within the study's range of inquiry.:

- **Analysis of network traffic data:** The study's main focus will be on the analysis and comprehension of network traffic data, particularly the elements and traits important for anomaly identification.
- **Investigation of machine learning algorithms:** Anomaly detection in network traffic will be a focus of the research as it examines several supervised machine learning algorithms.
- **Model development and optimization:** The development, validation, and improvement of machine learning models for spotting anomalies in network data will all be part of the project. To achieve optimum performance, the models will be adjusted utilising hyperparameter optimization methods.
- **Implementation and evaluation of the anomaly detection system:** The study's implementation of the selected machine learning model in a real-time network monitoring system and performance assessment of it using acceptable metrics will be covered.

However, there are limitations to the study, which include:

- **Dataset limitations:** The study makes use of freely accessible datasets like NSL-KDD, CICIDS2017, and UNSW-NB15. The ability to generalise the results to other network traffic datasets may depend on how well-representative and high-quality these datasets are.
- **Algorithmic limitations:** The study will concentrate on a particular set of machine learning algorithms, which may not be comprehensive or indicative of all potential methods for anomaly identification in network traffic.
- **Time constraints:** The study project's time constraints might prevent in-depth examination of some topics or the creation of more complex models.
- **Computational resources:** The selection of machine learning algorithms and the optimization process may be influenced by the availability of computer resources, thereby limiting the complexity and effectiveness of the produced models.

Despite these drawbacks, the project intends to provide insightful knowledge and useful strategies for applying machine learning to identify anomalies in network traffic. The results will be used to guide further research and development in this field and, in turn, help network managers protect their systems from potential cyber threats.

1.5 Conclusion

In order to improve network security and defend against cyber threats, this research project created a machine learning-based anomaly detection system for network traffic data. The Random Forest algorithm was the focus of the study's investigation of different machine learning algorithms for the building, improvement, and testing of models. Data pre-processing, feature extraction, and model validation were all part of the system's analysis and design, while data processing, model building, and testing were all part of the system's implementation.

The study's results show how well machine learning-based methods work for spotting anomalies in network traffic data. The suggested solution can assist businesses in several industries, such as banking, healthcare, and e-commerce, in proactively identifying and addressing potential security threats and performance difficulties. However, the study's shortcomings, such as those related to the study's dataset and computational resource restrictions, point to the need for additional research to investigate more sophisticated machine learning techniques and enhance the system's performance on unique and previously unexplored data.

Chapter 2 Literature Review

2.1 Introduction

The literature review is an essential part of research because it provides a thorough understanding of the body of existing knowledge, identifies research gaps, and lays the foundation for the study framework. The literature on anomaly detection in network traffic is thoroughly examined in this chapter, with a focus on machine learning methods.

The review starts off by looking into network traffic analysis, highlighting its significance in the modern digital world and the numerous factors involved. The topic then moves on to other network anomalies and the potential security risks they present. The chapter continues to examine conventional anomaly detection techniques, outlining their benefits and drawbacks for handling contemporary network security issues.

After that, the discussion turns to a full review of the machine learning methods used for anomaly identification, which includes a variety of learning paradigms and algorithms used to network traffic analysis. The chapter concludes by highlighting the importance of evaluating machine learning models and by going into greater detail on important performance indicators and widely used benchmark datasets in this field.

The goal of this literature review is to provide a comprehensive understanding of the current state of research in network traffic anomaly detection using machine learning, laying the groundwork for the development of a ground-breaking research framework that addresses the shortcomings and gaps found in the existing literature.

2.2 Network Traffic Analysis

Understanding and maintaining the security and effectiveness of computer networks depends on network traffic analysis. It includes a range of methods and strategies for tracking, examining, and detecting trends and behaviours in network traffic. Network administrators can identify potential security risks, enhance network performance, and guarantee the availability and integrity of network resources by looking at these patterns. The essential elements of network traffic analysis are covered in the sections that follow.

2.2.1 Packet Capture and Analysis

The procedure of gathering and storing data packets sent over a network is referred to as packet capture. Administrators of networks can view the specifics of network communication, such as packet headers and payload data. The gathering and

analysis of network traffic data can be done in real-time thanks to packet capture programmes like Wireshark and tcpdump. Administrators can discover useful information about network performance and identify unusual communication patterns that could point to possible security problems by studying individual packets.

2.2.2 Flow Analysis

In comparison to packet capture, flow analysis is a method for monitoring network traffic at a higher level of abstraction. It entails grouping packets together into flows, which are collections of connected packets that share characteristics like source and destination IP addresses, port numbers, and protocol types. Network administrators may get a complete picture of the patterns of network traffic using tools like NetFlow and sFlow, which also help them spot trends and abnormalities. Detecting significant network events, such as Distributed Denial of Service (DDoS) assaults or unusual bandwidth usage, can be done with the help of flow analysis.

2.2.3 Protocol Analysis

Understanding the behaviour of network protocols, which are the regulations guiding the communication between network devices, is the main goal of protocol analysis. Network administrators can learn more about the operational facets of network services and applications by examining protocol-specific features and interactions. Among the popular protocols examined are HTTP, FTP, SMTP, and DNS. Security threats like web application attacks or email-based phishing operations that target services or apps must be found using protocol analysis.

2.2.4 Deep Packet Inspection

A more sophisticated method of analysing network traffic is known as deep packet inspection (DPI), which not only looks at the packet headers but also the payload content. DPI enables the identification of application-specific behaviours, encrypted traffic, or malicious content by detecting patterns or signatures within the payload. Although DPI might use a lot of resources, it offers a finer-grained degree of network traffic analysis, which is crucial for spotting sophisticated threats and guaranteeing compliance with data protection laws.

2.3 Types of Network Anomalies

Network anomalies are variations from typical network activity that could be an indication of performance or security risks. Network administrators can create more effective methods for identifying and reducing network security issues by having a better awareness of the various forms of network abnormalities. The many types of network abnormalities are covered in detail in this section.

2.3.1 Point Anomalies

Point anomalies, sometimes referred to as single-instance anomalies, happen when a single data point within the network traffic behaves differently from conventional traffic patterns. These anomalies frequently point to lone instances like unwanted access attempts, network configuration issues, or unexpected surges in bandwidth usage. Individual data points can be compared to predetermined criteria or historical data to discover substantial departures from the norm, which can be used to identify point anomalies.

2.3.2 Contextual Anomalies

Contextual anomalies, also known as conditional anomalies, are instances of data points behaving abnormally in a particular scenario or context. The relationships between data points and their surroundings are considered to find these abnormalities. A sudden spike in network traffic, for instance, can be regarded as regular during peak business hours but abnormal during off-peak hours. Data points are often examined along with pertinent contextual information, such as time, location, or user activity, to identify contextual abnormalities.

2.3.3 Collective Anomalies

Even while individual data points within the group may not seem anomalous, collective anomalies happen when a set of data points exhibit odd behaviour collectively. These irregularities frequently point to coordinated actions like Distributed Denial of Service (DDoS) attacks, botnet activity, or deliberate attempts at data espionage. To find patterns that differ from predicted behaviour, it is necessary to examine the connections and interactions among numerous data points.

2.3.4 Network Security Threats

Some of the network security threats that the network abnormalities can take the form of are listed below:

- ***Distributed Denial of Service (DDoS) Attacks:*** Malicious attempts to overwhelm a network or service with too much traffic, frequently from a variety of coordinated sources.
- ***Port Scanning:*** A method by which attackers find open ports on a target system, possibly disclosing weak services or applications.
- ***Malware Activities:*** A network's ability to be infected by malicious software, such as viruses, worms, or ransomware, which can compromise system integrity, steal sensitive data, or otherwise cause havoc.
- ***Unauthorized Access:*** Illegal efforts to access network resources or systems, which may involve utilising stolen credentials, exploiting security flaws, or some combination of these.

Understanding the various types of network anomalies and their potential implications is crucial for developing effective strategies to detect and mitigate network security threats. Machine learning techniques, as discussed in subsequent sections, can significantly enhance the capability to identify these anomalies and provide adaptive solutions to an ever-evolving threat landscape.

2.4 Traditional Methods of Anomaly Detection

Traditional procedures, statistical techniques, and signature-based techniques are the mainstays of anomaly identification in network data. These approaches, while successful in some situations, are constrained by their reliance on established rules, models, or signatures when dealing with fresh and developing threats. These conventional techniques are further discussed in the parts that follow, along with pertinent citations and references.

2.4.1 Rule-Based Approaches

To find probable anomalies in network traffic, rule-based techniques rely on established rules (Zhang, 2008). These laws often come from the analysis of historical data, professional knowledge, or industry norms. Firewalls, intrusion detection systems (IDS), and intrusion prevention systems (IPS) are examples of rule-based systems. These systems impose policies depending on criteria, such as IP addresses, port numbers, or protocol types. Rules-based techniques may be successful at identifying known threats, but they may not be able to recognise novel assaults or adjust to changing network activity (Zhang, 2008).

2.4.2 Statistical Methods

Statistical techniques evaluate network traffic using statistical models to spot anomalies from the norm (Lakhina, 2004). These techniques entail gathering data on network traffic, computing various statistical metrics (such as mean, standard deviation, and correlation), and evaluating the observed values in comparison to predetermined thresholds or earlier baselines. Anomaly-based IDS, which uses methods like principal component analysis (PCA), clustering, or time-series analysis to find unusual patterns in network traffic, is an example of a statistical methodology. Although certain sorts of anomalies can be found using statistical methods, these techniques may have large false-positive rates or be susceptible to changes in network behaviour (Lakhina, 2004).

2.4.3 Signature-Based Techniques

To find abnormalities, signature-based algorithms compare network traffic patterns to known attack signatures (Roesch, 1999). These signatures often draw inspiration from the traits of attacks that have already been discovered, such as particular packet sequences, payload content, or behavioural patterns. The ability of signature-based

IDS, like Snort, to accurately identify known threats has led to their widespread deployment in practise. However, because signature-based approaches rely on already-existing attack signatures, they are intrinsically incapable of detecting novel or previously unknown attacks (Roesch, 1999).

2.5 Predictive Analytics

Predictive analytics is a powerful tool for analysing and predicting future outcomes based on historical data. In the context of anomaly detection in network traffic, predictive analytics can be used to identify patterns and trends that may indicate potential threats or attacks. By leveraging machine learning algorithms and statistical models, predictive analytics can provide valuable insights into network security and help organizations better understand and mitigate potential risks.

Clustering algorithms are another method of predictive analytics for network traffic anomaly detection. Groups of network traffic data that share features like traffic volume, packet size, or protocol type can be isolated with the help of clustering algorithms. Data clustering allows for the discovery of trends and patterns that may foreshadow security breaches. Traffic patterns linked to known malware or botnets, for instance, can be uncovered with the help of clustering algorithms (Buczak, 2016).

Predictive analytics in anomaly detection can make use of a wide variety of machine learning algorithms, including but not limited to time series analysis, clustering algorithms, and neural networks/decision trees. To better detect potential security threats, these algorithms can be "trained" on historical network traffic data. Features like traffic volume, packet size, and protocol type can be used by neural networks to detect abnormal network behaviour (Alazab, 2019).

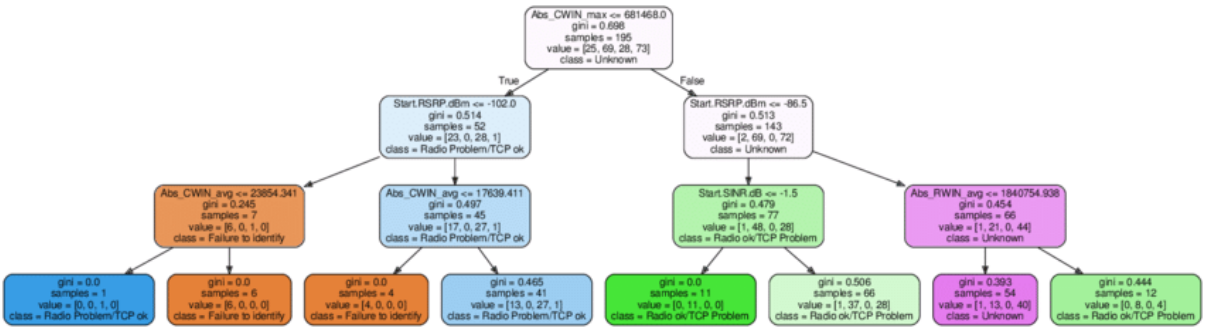


Figure 2.1 Anomaly detection using a decision tree (Mohamed Moulay, 2020),

Predictive analytics is an effective method for detecting and neutralising security risks in data transmissions. Machine learning algorithms and statistical models help businesses analyse network traffic data for hidden patterns and trends, allowing them to prevent attacks in their early, more manageable stages. New and improved methods for detecting and avoiding network security breaches are likely to appear as the field of predictive analytics develops further.

The following diagram depicts the steps in the predictive analytics process:

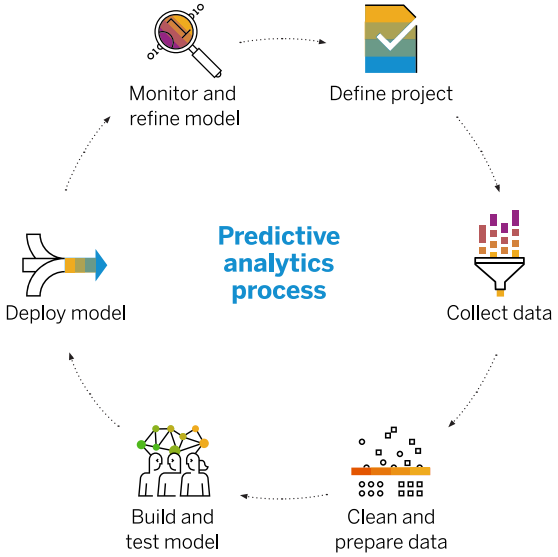


Figure 2.2 Process of predictive analytics. (sap.com ,2023)

- ***Project Scope Statement***
  The first thing to do when using predictive analytics is to figure out what you want to achieve. Problem recognition, outcome specification, project scope, and schedule establishment all fall under this heading.
- ***Data Collection:***
  The subsequent action is to gather the necessary information for the analysis. This information could originate from anywhere, including company databases, third-party datasets, or web scraping. It's crucial to check that the information gathered is accurate, applicable, and sufficient.
- ***Sorting and Cleaning the Data:***
  The data must be cleaned and formatted before it can be analysed. The data must be cleaned, which includes erasing duplicates, filling in gaps, and dealing with outliers, before it can be analysed. To further enhance the performance of the model, feature engineering may be implemented at this stage.
- ***Construct and Evaluate Model:***
  The next step is to use the cleaned data to construct a predictive model. As a rule, this entails picking the right algorithm, fine-tuning its hyperparameters,

and training the model with the data. After the model has been developed, it must be put to the test on a new set of data to determine how well it performs and to pinpoint any problems that need fixing.

- ***Application Model:***
  The model can be put into production after it has been constructed and validated. The next step is to incorporate the model into a system or programme that can make predictions using newly collected information. Testing the model's performance in a real-world setting and making any necessary adjustments may also be part of the deployment process.
- ***Model Observation and Adjustment:***
  The last part of using analytics for prediction is keeping an eye on the model and tweaking it as needed. For this purpose, new data must be gathered, the model must be retrained, and its performance must be evaluated. This is an essential process for keeping the model up to date as new information becomes available.

2.6 Machine Learning

By training models on labelled datasets and then using them to detect anomalous patterns in network traffic in real time, machine learning can be used for anomaly detection. Extracting features from the network traffic data is the first step in this process. Some examples of such features are packet size, duration, and frequency, while others, like the characteristics of network flows, are more complex. The success of the dataset and the effectiveness of the model are both dependent on feature extraction.

After the features have been extracted, we can select an appropriate machine learning algorithm. This requires testing out various algorithms—from decision trees and SVMs to CNNs and RNNs—and picking the one that works best with our data. In order to train the chosen machine learning algorithm, the dataset is labelled with information about both typical and unusual network activity. The model is taught to differentiate between them and spot repeating structures in the data.

Once the model has been trained, it can be used to spot anomalies as they happen in real time. Any new data that comes across the network is immediately analysed by the model. The model will sound an alarm and notify the system administrator if it finds anything out of the ordinary. Anomaly detection can be more effective and network security breaches can be avoided with this real-time approach.

By contrasting the number of false positives and false negatives, the effectiveness of the machine learning algorithm can be assessed. When an algorithm mistakes typical traffic for abnormal, it produces false positives, while when it fails to detect

real anomalies, it produces false negatives. The objective is to maximise the detection rate while minimising the number of false positives and false negatives. By conducting this analysis, we can learn how effective the machine learning algorithm is and where adjustments need to be made.

2.2.1 Supervised Learning

When using a supervised learning model to implement anomaly detection for network traffic, the algorithm must be trained using a labelled dataset that includes both "normal" and "anomalous" network traffic. With this method, the machine learning model is trained to distinguish between typical and unusual patterns in network traffic.

Support Vector Machines (SVMs) are an often used and effective algorithm for supervised anomaly detection (Khan, 2019) (Abdar, 2021). To classify data as "normal" or "abnormal," SVMs search a high-dimensional feature space for the optimal hyperplane that divides the two sets of points. Anomaly detection centres on data points that are outside of the hyperplane.

Decision Trees are another well-liked supervised learning algorithm that has seen use in studies of network anomaly (Tian, 2019) (Agarwal, 2021). To make use of feature values, decision trees divide the feature space into subsets in a recursive fashion. Leaves on the tree are labelled with the classes they belong to (normal or anomalous).

Recently, SVMs and Decision Trees were among several supervised learning algorithms for anomaly detection in network traffic compared by Alsheikh and Mahmood (Alsheikh, 2022). The research showed that SVMs performed the best, with an accuracy of 99.7 percent, compared to the other algorithms tested. The authors explained that this was because SVMs can process data that is not linearly separable.

2.2.2 Unsupervised Learning

Using an unsupervised learning model to detect anomalies in network traffic entails first training an algorithm on an unlabelled dataset of network traffic and then applying that model to detect out-of-the-ordinary behaviour. By analysing the data's statistical characteristics, the machine learning model is trained to identify outliers.

The k-Nearest Neighbours (k-NN) algorithm is frequently used in unsupervised anomaly detection research (Kandhway, 2019) (Singh, 2021). To calculate the distance between a data point and its nearest neighbours, the k-NN algorithm first identifies the k-nearest neighbours of the data point in question. A data point is

considered unusual if the measured distance is greater than a predetermined cut-off value.

The Autoencoder is yet another well-liked unsupervised learning algorithm, and it has been used in numerous studies for network anomaly detection (Nasiri, 2020) (Yin, 2021). Autoencoders function by first transforming the input data into a representation with fewer dimensions and then decoding it back to its original form. When an autoencoder encounters a data point that it cannot reconstruct correctly, we say that point is anomalous.

Unsupervised learning algorithms like k-NN and Autoencoder were recently compared by Zhang et al. (Zhang, 2021) for spotting anomalies in network traffic. Autoencoder had the highest detection rate (98.9%) of all the algorithms tested in the study. The authors concluded that this was due to Autoencoder's success in identifying generic characteristics of Internet traffic.

Training the algorithm on an unlabelled dataset and employing a suitable algorithm, such as k-NN or Autoencoder, to identify patterns that deviate from the norm are the essential steps in implementing anomaly detection for network traffic using an unsupervised learning model. Autoencoder is the most widely used algorithm for achieving high accuracy, and it has been used successfully in several studies applying these algorithms to network anomaly detection.

2.2.3 Reinforcement Learning

Using reinforcement learning (RL), one can train an agent to act in an environment based on observed states and rewards, which can then be used for anomaly detection in network traffic. By optimising its cumulative reward over time, the agent eventually learns to recognise abnormal behaviour. By performing actions and receiving feedback from its surroundings, the machine learning model can acquire the ability to spot anomalies.

AnoRL is a framework proposed by Lee et al. (Lee, 2019) that uses RL to identify anomalies in network traffic. In order to determine whether network traffic is typical or abnormal, the framework employs feature extraction to train an RL agent. Based on its categorization, the agent performs actions like dropping packets or sending alerts. Agent compensation depends on the quality of its classification and the effort required to implement a solution.

An RL-based method for identifying DDoS attacks in SDNs was proposed in a separate study (Shao, 2020). To learn the best course of action in response to network traffic, a Deep Q-Network (DQN) is employed in this method. The DQN is

taught to recognise abnormal behaviour by maximising its reward, and it does so by being exposed to a dataset consisting of both normal and attack traffic.

(Zhang, 2022) recently proposed a framework for anomaly detection in networks using deep reinforcement learning; they called it AD-DRL. Autoencoder and RL work together in the framework to spot unusual flows of traffic. To prepare the network for anomaly detection, an autoencoder is used, and then an RL agent is trained to act based on the encoded features.

2.7 Data Partitioning

The creation and assessment of machine learning models for network traffic anomaly detection depend heavily on data partitioning. To do this, the dataset must be divided into separate subsets, usually for training, validation, and testing reasons. The machine learning model is resilient against overfitting and can generalise effectively to new data thanks to proper data partitioning. The consequences of various data partitioning approaches for detecting network traffic anomalies are covered in this section.

2.7.1 Training, Validation and Test Sets

Training, validation, and test sets are the three subsets that are most frequently used in machine learning. The machine learning model is trained using the training set, which enables it to discover patterns and connections in the data. To maximise model performance while reducing overfitting, the validation set is used for model selection and hyperparameter adjustment. The test set is then used to validate the final model's performance on unknown data, giving a fair evaluation of its generalisation potential (Hastie, 2009).

2.7.2 Data Partitioning Techniques

In machine learning, several data partitioning strategies are frequently used, including:

- ***Random Split:*** Each instance has an equal chance of being assigned to any of the three subgroups because the data is randomly split into training, validation, and test sets. This method works well for independent and identically distributed (IID) data points, but it might not accurately depict time-dependent patterns in network traffic data.
- ***Temporal Split:*** Older data are used for training, whereas more recent data are used for validation and testing. Due to the way this method considers temporal relationships and changing network behaviour, it is more suited for network traffic statistics (Sommer, 2010).

- ***Stratified Split:*** The distribution of distinct classifications (such as normal and anomalous) within each subset of the data is preserved during partitioning. This method ensures that all classes are fairly represented in the training, validation, and test sets, making it especially beneficial for imbalanced datasets, which are frequent in network traffic anomaly detection (Chawla, 2010).

2.8 Conclusion

With a focus on machine learning methods in particular, this literature review has offered a thorough overview of the status of research in network traffic anomaly detection. It has discussed a range of topics related to network traffic analysis, such as deep packet inspection, flow analysis, protocol analysis, and packet capture and analysis, emphasising the significance of comprehending network behaviour to identify potential security threats and enhance network performance.

In the review, numerous network abnormalities, including point, contextual, and collective anomalies, as well as how they materialise in various network security risks, such as DDoS attacks, port scanning, malware activities, and unauthorised access attempts, have been covered. The shortcomings of conventional anomaly detection techniques, such as rule-based methods, statistical techniques, and signature-based techniques, have been thoroughly investigated, highlighting the want for more flexible and reliable solutions.

The ability to overcome such drawbacks by automating anomaly detection and adjusting to changing network behaviour has led to the identification of machine learning techniques as possible alternatives to conventional approaches. To address the gaps and limitations found in the existing literature, novel research frameworks have been developed using a variety of learning paradigms, algorithms, and data partitioning methodologies pertinent to network traffic anomaly detection.

This literature study has demonstrated the importance of network traffic anomaly detection in maintaining network performance and security. The state of the art in this field might be significantly advanced using machine learning techniques. Future research can concentrate on the development of novel machine learning-based systems that can successfully solve the issues of identifying network abnormalities and protecting against ever-evolving cyber threats by expanding on the insights acquired from this analysis.

Chapter 3 Methodology

3.1 Introduction

This chapter presents the methodology used for machine learning-based anomaly identification in network traffic. We discuss the software environment, including the employed tools and libraries, as well as the procedures for acquiring data, getting it ready, selecting features, creating models, and assessing them. The goal is to create an efficient anomaly detection system that can identify potential assaults and aid in maintaining network integrity.

3.2 Software Environment

The software environment utilised for this study is composed of a number of programmes, frameworks, and libraries that are designed to simplify various research-related tasks, such as data collection, processing, and model building.

3.2.1 Programming Language

Because of its adaptability, simplicity, and broad support for machine learning, data manipulation, and visualisation packages, Python was chosen as the main programming language. Due to its readability, simplicity, and robust ecosystem of libraries, Python is a highly popular high-level programming language that many data scientists and machine learning practitioners now use exclusively.

3.2.2 Integrated Development Environment (IDE)

An open-source web tool called Jupyter Notebook (DataSpell 2023.1) enables the creation and sharing of documents with live code, equations, visuals, and text. It provides a flexible and interactive platform for machine learning model quick prototyping, data exploration, and visualisation.

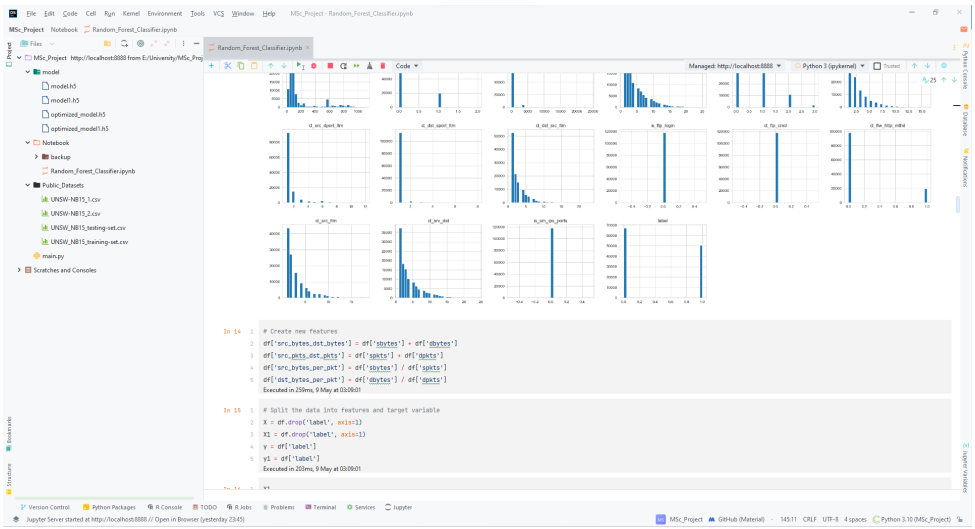


Figure 3.1 Development Environment

3.2.1 Libraries and Frameworks

The following Python frameworks and libraries were used in this investigation:

- ***Scikit-learn****:* a well-known and frequently used machine learning library that offers a variety of classification, regression, clustering, dimensionality reduction, and other techniques. A complete toolkit for machine learning tasks, Scikit-learn also provides tools for data pre-processing, model evaluation, and hyperparameter adjustment.
- ***Pandas:*** A robust data manipulation and analysis library, Pandas offers effective data structures like Data Frame and Series that make it simple to manage massive datasets. It is an essential tool for data processing activities since it provides a wide range of features for data cleansing, filtering, aggregation, and transformation.
- ***NumPy:*** A foundational library for Python's numerical computing, NumPy supports multidimensional arrays and a number of mathematical operations, such as linear algebra, statistical computations, and numerical optimization. Numerous other scientific and data analysis libraries in Python are built on the effective array operations provided by NumPy.
- ***Matplotlib and Seaborn:*** Plotting and graphing tools are used to create plots and graphs for data exploration and analysis. Seaborn, a statistical data visualisation toolkit built on top of Matplotlib, makes it easier to create visually appealing and educational visualisations. Matplotlib is a complete library for producing static, animated, and interactive visualisations in Python.
- ***TensorFlow and Keras:*** Google's open-source TensorFlow machine learning framework enables the creation and training of a variety of machine learning models, including deep learning models. A high-level API called Keras, which is built on top of TensorFlow, makes it easier to create and train deep learning models by giving users simple-to-use building blocks for designing neural network designs.
- ***Scapy:*** Scapy is a robust Python package that allows for the capture, examination, and creation of network packets. Scapy can be utilised for data collecting and preprocessing in this study since it offers comprehensive support for network protocols and enables network traffic data parsing and filtering.
- ***Joblib:*** The performance of computationally intensive tasks, such as model training and hyperparameter tuning, is enhanced using Joblib, a collection of Python tools for lightweight pipelining and parallelism.

3.3 Data Collections

Since it serves as the basis for model training and evaluation, the data collecting procedure is essential for creating a reliable anomaly detection system. To achieve a diverse and complete depiction of network behaviours, data from both publicly accessible datasets and local network traffic were gathered for this study.

3.3.1 Public Datasets

As the main sources for network traffic data, NSL-KDD, CICIDS2017, and UNSW-NB15 were three publicly accessible datasets. These datasets were chosen for the variety of network traffic data they contained, covering both regular traffic and different kinds of attacks.

- ***NSL-KDD:*** The NSL-KDD dataset overcomes various shortcomings of the original dataset, including as redundant records and inherent biases, and is an upgraded version of the widely used KDD Cup '99 dataset. A comprehensive range of network traffic data, including different kinds of intrusions and typical traffic examples, is contained in NSL-KDD. Each of the dataset's around 125,000 records have 41 attributes and a label identifying the type of traffic.
- ***CICIDS2017:*** The Canadian Institute for Cybersecurity produced the extensive CICIDS2017 dataset (CIC). It simulates a real-world network environment by including both good and bad network traffic statistics. The dataset contains both regular traffic and a variety of attacks, including Distributed Denial of Service (DDoS), Brute Force, and Botnet. The 2.8 million records and 78 features in CICIDS2017 provide a wide range of information for anomaly detection.
- **UNSW-NB15:** The Australian Centre for Cyber Security (ACCS) at the University of New South Wales' Cyber Range Lab generated the UNSW-NB15 dataset (UNSW). The researchers produced the dataset using an SDN testbed, which gave them the ability to simulate actual network conditions and design different attack scenarios. 49 features make up the UNSW-NB15 dataset, which also includes source and destination IP addresses, ports, protocols, and several other network traffic characteristics. It includes both common traffic statistics and a broad spectrum of threats, including DoS, DDoS, reconnaissance, worm propagation, and shellcode injection.

For the creation and assessment of the anomaly detection models, these open-source datasets offer an extensive and varied spectrum of network traffic information. The study makes sure the built models can accurately detect a wide range of anomalies and generalise well to different network environments by including several datasets.

3.3.2 Local Network Data Collection

The training and testing datasets were supplemented with data from the local network in addition to the public datasets, demonstrating that the created models can effectively generalise to unobserved data. The following steps were engaged in gathering data from local networks:

- ***Network Traffic Capture:*** Using packet sniffing programmes like Wireshark or tcpdump, which keep track of packets sent over the network, network traffic was recorded. To be processed further, this data was recorded as packet capture (PCAP) files.
- ***Traffic Labelling:*** The acquired local network traffic was manually examined and classified as either legitimate or malicious, depending on the patterns and behaviours seen. This process is essential for supervised learning since it gives the basis for model training and evaluation.
- ***Extraction of Relevant Information:*** Scapy or customised Python scripts were used to extract pertinent features from the raw network traffic data. These elements consist of aggregated statistics, packet header information (such as source and destination IP addresses, ports, and protocols), and packet payload information (such as packet size, duration, and content) (e.g., average packet rate, byte count, and flow duration).
- ***Data Integration:*** To provide a comprehensive dataset for model training and evaluation, the local network data's extracted features were integrated with open-source datasets. During the integration process, features have to be properly normalised and encoded as needed in order to ensure feature compatibility and consistency across the various data sources.

3.4 Data Pre-processing

A critical stage in the machine learning pipeline, data pre-processing converts raw network traffic data into a format that machine learning algorithms can use. To enhance model performance and minimise potential biases, this process entails cleaning, structuring, and modifying the data. During the pre-processing stage for the data, the following actions were taken:

- ***Handling missing values:*** The performance of machine learning algorithms can be negatively impacted by missing or incomplete values in the dataset. Depending on the type and quantity of the missing data, missing values were first recognised and either filled in using the proper techniques or eliminated. Mean or median imputation, nearest-neighbour imputation, or utilising a model-based approach to estimate the missing values are common techniques for filling in missing values.

- ***Encoding categorical features:*** The majority of machine learning algorithms can only handle numerical representations of categorical information, such as protocol kinds or network service types. One-hot encoding and label encoding are two popular methods for encoding categorical features. While label encoding gives each category a different integer value, one-hot encoding provides a binary vector for each category. The algorithm being utilised and if the categorical feature has an ordinal relationship between its values determine which of these approaches should be used.
- ***Normalization of numerical features:*** Different units or scales for numerical characteristics can result in unequal contributions to the model and poor algorithm performance. Numerical features are scaled using methods like Min-Max scaling or standardisation to address this issue (also known as z-score normalization). While standardisation changes the characteristics to have a zero mean and unit variance, min-max scaling linearly adjusts the features to a given range (for example, [0, 1]). The algorithmic choice as well as the data distribution determine the scaling strategy to use.
- ***Feature transformation:*** In other circumstances, the initial characteristics might not accurately capture the underlying data patterns or might be challenging for machine learning algorithms to comprehend. To develop new features or alter existing ones, feature transformation techniques including logarithmic transformation, square root transformation, and polynomial expansion can be used. The challenge and the data distribution determine the transformation strategy to choose.
- ***Handling imbalanced data:*** When the distribution of classes in the target variable is not uniform, there is imbalance in the data. The bulk of network traffic examples are frequently normal in the context of anomaly detection, with a comparatively small amount of malicious traffic. Biased models that benefit the dominant class can be produced from unbalanced data. The class distribution in the dataset can be balanced using methods like random oversampling, random under sampling, or synthetic minority over-sampling technique (SMOTE).
- ***Data partitioning:*** Typically, the dataset is split into distinct training and testing sets to enable the models to be assessed on unlabelled data. This procedure is crucial for determining how generalizable the created models are and avoiding overfitting. Random splitting, stratified splitting (to preserve the class distribution throughout the splits), and time-based splitting are examples of common data partitioning techniques (particularly relevant for time-series or streaming data).

3.5 Feature Selection

As it determines the features that are most pertinent to the precise identification of network anomalies, feature selection is a crucial phase in the creation of an anomaly detection system. Feature selection can enhance model performance by lowering the number of features utilised in the model, reducing overfitting, and lowering computing complexity. The following methods were used in the feature selection procedure in this study:

- ***Correlation analysis:*** In order to determine the linear relationship between characteristics and the target variable, Pearson's correlation coefficient was determined (i.e., normal, or malicious traffic). This study made it possible to find features that are highly connected and can provide the model significant predictive ability. Low correlation features were deleted from the dataset since they were deemed to be less significant.

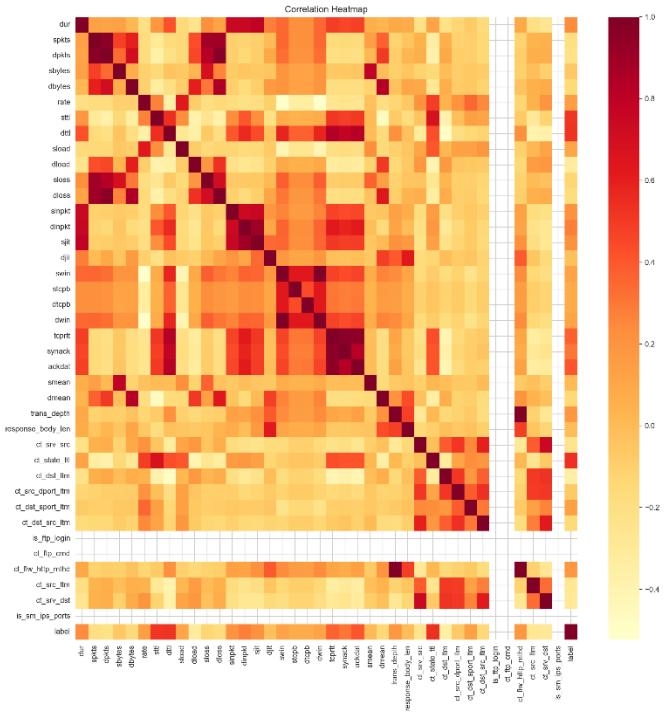


Figure 3.2 Correlation analysis

- ***Mutual information:*** The measurement of the dependence between characteristics and the target variable is done using the mutual information technique. It measures the degree of knowledge acquired about the target variable from being aware of the value of a certain aspect. High mutual information suggests that the feature is crucial for anomaly identification since it shows a strong association between the feature and the target variable. The dataset was cleaned up by removing features with low mutual information.

- ***Recursive feature elimination (RFE):*** RFE is a backward selection technique that, based on the performance of a selected estimator, iteratively removes the least significant features (e.g., a classification algorithm). The least significant feature is eliminated after each iteration once the estimator has been trained on the entire set of features initially. Up until the necessary number of features is attained, this process is repeated. The most pertinent features that significantly influence model performance can be found via RFE.
- ***Feature importance from tree-based models:*** Based on the frequency and efficiency of feature splits in the tree structure, tree-based models, such as decision trees or random forests, can offer an estimate of feature relevance. High-importance features are utilised to split the data more frequently and successfully, which suggests that they have a considerable impact on the model's performance. This technique can be used to rank traits and choose the most crucial ones for additional examination.
- ***Wrapper methods:*** To find the most informative feature set, wrapper techniques entail training a certain model using several subsets of features and assessing the model's performance. Forward selection, backward elimination, and exhaustive search are examples of common wrapper approaches. These techniques can be computationally expensive, especially when working with large feature sets, but because they take the model in use into account, they can produce results for feature selection that are more precise.

3.6 Model Development

The basis of the anomaly detection system is the model development phase, where a variety of machine learning algorithms are trained and improved to find patterns and relationships in the data that can accurately differentiate between legitimate and malicious network traffic. To ensure robustness and thorough performance evaluation, a variety of models, including classic machine learning algorithms and deep learning techniques, were used in this work. During the process of developing the model, the following actions were taken:

- ***Establishment of the baseline model:*** A baseline model was created using a straightforward classification technique, such as logistic regression or a decision tree, in order to evaluate the performance of the constructed models. The baseline model serves as a point of comparison and guarantees that the produced models show a notable advancement over fundamental categorization methods.
- ***Model choice:*** For testing, a range of machine learning algorithms, including both conventional and deep learning methods, were chosen. supervised learning techniques such as Support Vector Machines (SVM), k-Nearest

Neighbours (k-NN), Random Forests, and Gradient Boosting Machines were included in traditional algorithms (GBM). Artificial neural networks (ANN) and convolutional neural networks (CNN) are two deep learning techniques that have demonstrated promising results in a variety of anomaly detection tasks.

- ***Model training:*** The pre-processed dataset, which was split into distinct training and validation sets, was used to train the chosen models. The models developed the ability to spot trends and connections in the data that can differentiate between legitimate and malicious network traffic during the training phase. To achieve convergence and avoid overfitting, the models' performance was assessed during training using evaluation metrics from the validation set, such as accuracy, precision, recall, and F1 score.
- ***Hyperparameter tuning:*** Machine learning algorithms are frequently controlled by several different hyperparameters. The performance of the model can be greatly enhanced by selecting the ideal collection of hyperparameters. Grid search, random search, and Bayesian optimization were used in this study's hyperparameter tuning to find the ideal hyperparameter values for each model. To establish a fair comparison and avoid overfitting, the models' performance with the optimised hyperparameters was assessed using cross-validation.
- ***Evaluation of the models:*** Following training and hyperparameter optimization, the models were tested on a different dataset to gauge how well they performed on unlabelled data. Evaluation criteria like accuracy, precision, recall, F1 score, and area under the Receiver Operating Characteristic (ROC) curve were used to compare the models' performance. These metrics helped choose the best model for the anomaly detection task by revealing the advantages and disadvantages of each model.

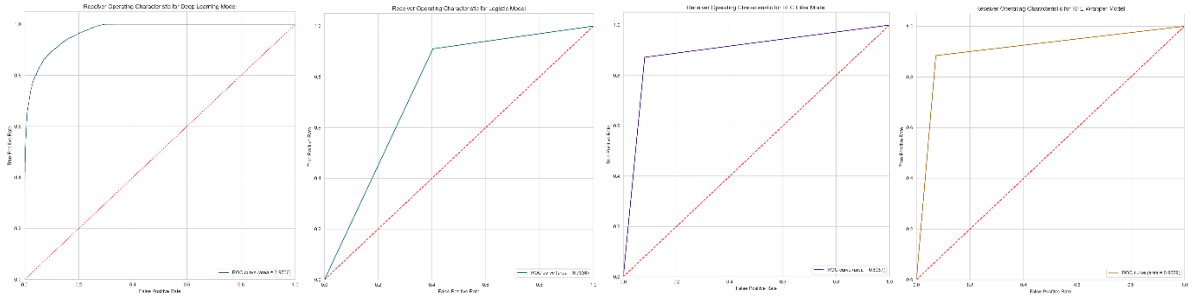


Figure 3.3 ROC plot of 4 model

- ***Model interpretation:*** Model interpretation approaches were used to get an understanding of how the models made decisions and to make sure they were reliable. To comprehend the rationale of conventional algorithms, feature importance scores or decision tree representations were employed. For deep

learning models, the most important characteristics or regions in the input data were shown using methods like saliency maps or layer-wise relevance propagation.

3.7 Model Evaluation

A crucial step in the machine learning pipeline is model evaluation, which offers a dispassionate evaluation of how well-built models perform on untried data. This technique guarantees that the models can generalise to real-world circumstances successfully and offers insights into their advantages and disadvantages. The anomaly detection models in this work were thoroughly evaluated using a variety of evaluation criteria and approaches. More information on the model evaluation procedure is provided in the following sections.

3.7.1 Evaluation Metrics

Several evaluation measures that each provide a different viewpoint on the model's capacity to differentiate between legitimate and malicious network data were calculated to evaluate the models' performance:

- ***Accuracy:*** The percentage of cases that are correctly classified out of all instances is known as accuracy. Although it is a logical measurement, it can be deceptive when datasets are uneven since it may benefit models that primarily forecast the majority class.

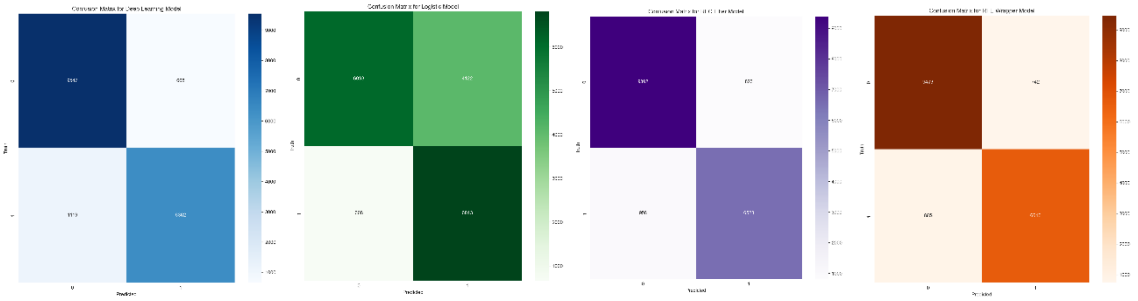


Figure 3.4 Confusion Matrix for Accuracy

- ***Precision:*** The proportion of genuine positive instances compared to all anticipated positive instances is what is measured. High precision suggests that the model has a low false positive rate and is reliable in detecting malicious network traffic.

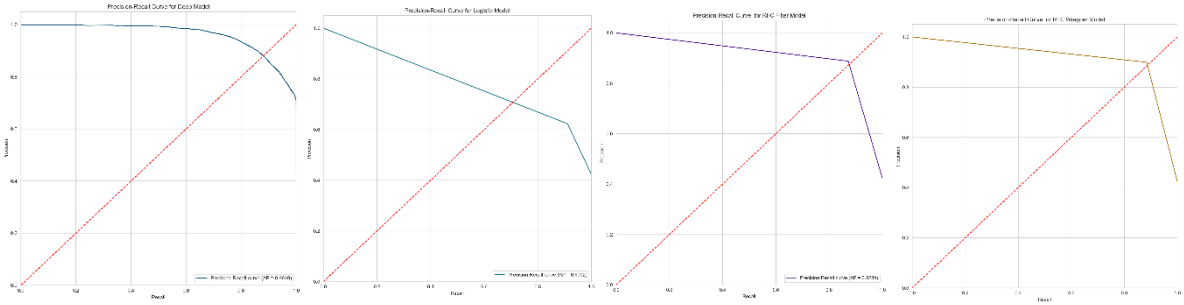


Figure 3.5 Precision and Recall

- ***Recall (Sensitivity):*** Recall counts how many of the total number of positive instances are true positives. A low false negative rate is the outcome of the model's high recall, which shows that it can successfully identify most occurrences of malicious network traffic.
- ***F1 score:*** The F1 score is a balanced measurement of both metrics and is the harmonic mean of precision and recall. Given that it accounts for both false positives and false negatives, it is very helpful when working with imbalanced datasets.

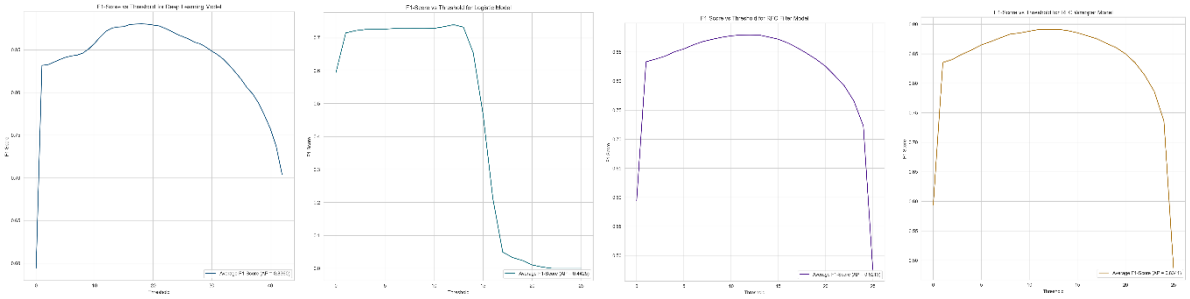


Figure 3.6 F1-Score vs Threshold

- ***AUC-ROC, or area beneath the receiver operating characteristic (ROC) curve:*** For various classification thresholds, the ROC curve shows the true positive rate (recall) versus the false positive rate. With larger values indicating greater performance, the AUC-ROC delivers a single value that sums up the model's performance across all thresholds.

3.7.2 Cross-Validation

Cross-validation was used during the model evaluation procedure to make sure the evaluation of the models was impartial and fair. In cross-validation, the dataset is divided into several folds, the models are trained on various subsets of the data, and then they are assessed on the remaining folds. A more accurate estimation of the model's performance on unobserved data can be obtained by averaging the performance measures over all folds.

The following are typical cross-validation methods:

- ***K-Fold Cross-Validation:*** The dataset is partitioned into k folds of equal size, and the models are trained and tested k times, using one-fold as the test set and the other folds as the training set.
- ***Stratified k-Fold Cross-Validation:*** Like k-fold cross-validation, but with each fold maintaining the class distribution to guarantee that a representative sample of the target variable is present in each fold.
- ***Leave-One-Out Cross-Validation:*** a particular instance of k-fold cross-validation, where k is equal to the dataset's instance count. Although this

approach can be computationally expensive, it offers a fair evaluation of the model's performance.

3.8 Conclusion

This chapter covered the creation of a machine learning-based system for identifying abnormal network traffic. Data gathering, pre-processing, feature selection, model construction, and evaluation were the main components of the methodology.

For the built models to be flexible, a variety of network traffic data, including both legitimate and malicious traffic, had to be gathered during the data collection phase. Data pre-processing fixed problems including missing values, category encoding, normalisation, and class imbalance to convert the raw data into a format that machine learning algorithms could use.

The most relevant and useful features for spotting network anomalies were found using feature selection techniques. Overfitting was lessened as a result, and model performance increased. The models were built using a variety of machine learning techniques, including both conventional and deep learning approaches. To ensuring their robustness and applicability to real-world situations, these models were created, enhanced, and evaluated utilising a variety of evaluation metrics and cross-validation approaches.

The study's tight methodology made it easier to develop an anomaly detection system that is trustworthy and dependable and can identify malicious network data in a variety of circumstances. The thorough evaluation procedure revealed the advantages and disadvantages of each model, allowing the best model to be chosen for the network security requirement.

In conclusion, our study showed the potential of machine learning techniques for creating a reliable and accurate network traffic anomaly detection system. The methods presented in this chapter can be used as guidelines for future network security research, and they can aid in the development of reliable anomaly detection systems.

Chapter 4 Analysis & Design

4.1 Introduction

The analysis and design stages of the machine learning-based network traffic anomaly detection system are examined in this chapter. The primary goals of this chapter are to provide a comprehensive description of the system's architecture, including all its various components, and to outline the steps involved in data pre-processing, feature selection, model development, and evaluation.

The analysis and design phase plays a vital role in developing an effective and efficient anomaly detection system by ensuring that the system components are well-integrated and tuned to achieve the expected performance. This chapter also discusses the choice of suitable machine learning algorithms and their corresponding hyperparameters, which are essential for developing accurate and trustworthy models for spotting anomalies in network data.

4.2 Concept Design

ML Model, ML Model Management (CMS), Alert System, Trusted Network, and Server are some of the components included in the concept design of an anomaly detection system for network traffic. These elements operate together in a cycle to effectively maintain network security and detect anomalies. Each element and how it functions within the system are thoroughly explained in the sections that follow.

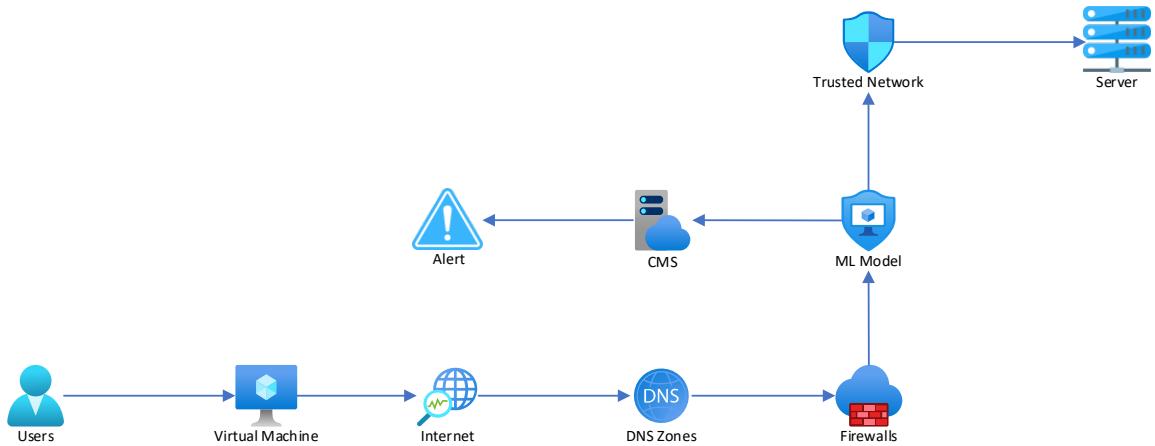


Figure 4.1 Concept design

4.2.1 Users

Individuals or entities that access the network infrastructure are represented by the User component. To send and receive network traffic, which moves across the system, users engage with machines.

4.2.2 Machine

Network traffic is produced and processed by machines, including servers, workstations, and other items connected to the network. They are essential for data transmission and network machine to machine communication.

4.2.3 Internet

The external network infrastructure through which network traffic moves is the Internet. It makes it possible for different networks to communicate and gives people and computers access to outside resources.

4.2.4 DNS Zone

The Domain Name System (DNS) zone component controls how the DNS configuration, which maps domain names to matching IP addresses, is configured. By converting legible domain names for humans into readable IP addresses for machines, it enables effective communication.

4.2.5 Firewall

Between the internal network and dangers from the outside, the firewall serves as a protective barrier. Based on previously established rules and policies, it keeps track of and manages both incoming and outgoing network traffic. Filtering and protecting network traffic are major responsibilities of the firewall component.

4.2.6 ML Model

The system's fundamental element for spotting irregularities in network traffic is the ML Model. To analyse patterns and spot unusual behaviour in network traffic data, it uses machine learning algorithms like Random Forest.

4.2.7 ML Model Management (CMS)

The methods and infrastructure needed to administer and maintain the ML Model are all included in ML Model Management. For the model to work at its best and be as accurate as possible, training, updating, and version control are all included.

4.2.8 Alert System

A key hub for controlling and monitoring network security is the alert system. The Security Centre receives a notification when the ML Model identifies an anomaly, alerting security staff to any potential threats or unusual network behaviour.

4.2.9 Trusted Network

A list of trusted IP addresses or network ranges that have been determined to be secure and reliable is kept up to date by the Trusted Network component. When trusted network traffic is found, it is added to the Trusted Network, which eliminates the need for additional inspection.

4.2.10Server

The target computer or server within the network is represented by the server. It takes in incoming network traffic and processes it while reacting to user and machine requests.

The suggested anomaly detection system can be designed, integrated, deployed, and validated successfully. The concept design must be implemented successfully in

order to create a system that can monitor network traffic, enhance network security, and give network managers and security staff useful information.

4.3 System Flow

The system architecture for the proposed anomaly detection in network traffic comprises several interconnected components that work together to provide a comprehensive solution. The architecture is designed to ensure the seamless flow of information between the components, enabling efficient data processing and accurate detection of network traffic anomalies. The main components of the system architecture are as follows:

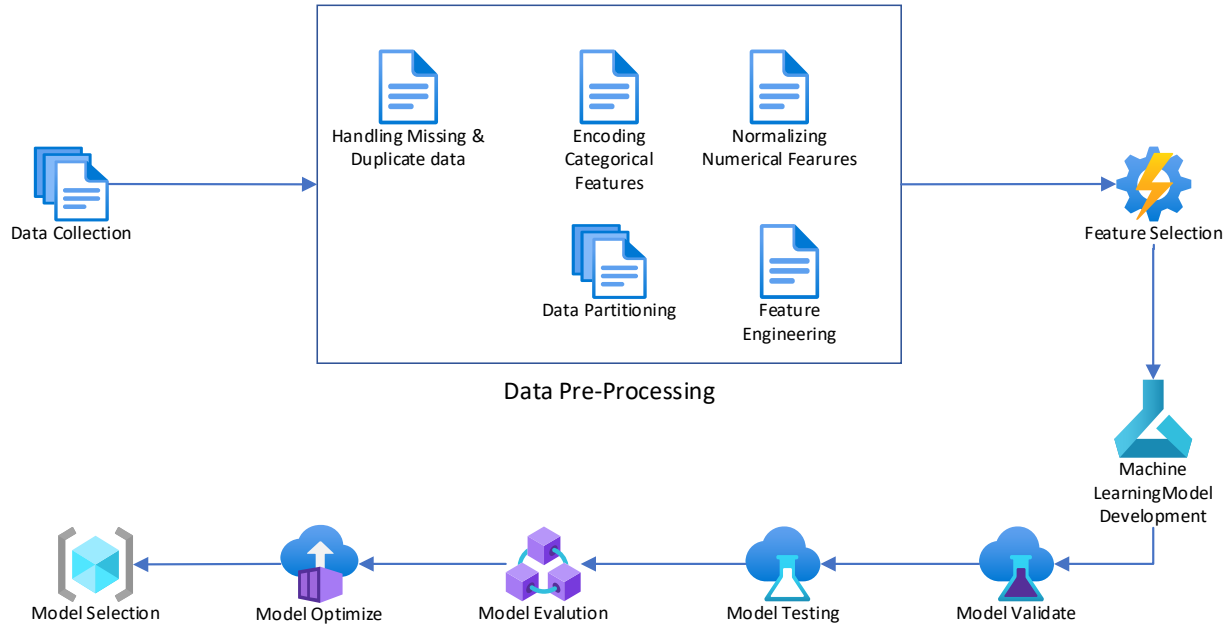


Figure 4.2 System Architecture

4.3.1 Data Collection

In order to construct and evaluate models, a broad and representative dataset is essential, and this is where the data gathering component comes in. The UNSW-NB15 (ACCS) dataset is one of the main sources of network traffic data used in this study. The UNSW-NB15 dataset was produced by the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) at the University of New South Wales (UNSW). The SDN testbed mimics actual network infrastructures and attack scenarios.

49 features make up the UNSW-NB15 dataset, which also includes source and destination IP addresses, ports, protocols, and several other network traffic characteristics. Along with regular traffic data, it covers a broad spectrum of attacks, including DoS, DDoS, reconnaissance, worm propagation, and shellcode injection. The variety of network traffic data in this dataset is crucial for creating precise and reliable machine learning models that can identify a variety of network traffic anomalies. This study makes sure that the built models can generalise well to varied

network environments and successfully detect a wide range of anomalies by including the UNSW-NB15 dataset into the data gathering process.

4.3.2 Data Pre-processing

Preparing the raw network traffic data for machine learning algorithms is a crucial stage in the anomaly detection process. The goal of data preparation is to clean up and change the raw data into a format that the chosen algorithms can use successfully. There are multiple steps within this process, each of which is explained in more detail below:

4.2.2.1 Handling missing and duplicates values

Due to a variety of factors, including mistakes in data collection, the absence of specific features, or discrepancies in the data sources, network traffic databases may contain missing values. Depending on the type of data, methods like mean, median, or mode imputation can be used to fill in missing values in order to solve this problem. If the amount of missing data is low, it is also possible to eliminate records with missing values from the dataset.

```
# Remove duplicates
df.drop_duplicates(inplace=True)
# Handle missing values
df.fillna(0, inplace=True)
# Remove missing values
df.dropna(inplace=True)
```

This bit of code clears the dataset by removing duplicate rows from a DataFrame (df), replacing any missing values with zeros, and finally removing any remaining rows with missing values.

4.2.2.2 Encoding categorical features

Categorical features must be transformed into numerical representations since machine learning algorithms often demand numerical input data. One-hot encoding, label encoding, and target encoding are just a few of the encoding methods that are accessible. Since it does not impose an arbitrary order on the categories, one-hot encoding is frequently favoured for nominal categorical variables. For ordinal categorical variables, however, label encoding, or goal encoding may be more appropriate.

```
#Encoding categorical features
df_cat = df.select_dtypes(exclude=[np.number])
print(df_cat.columns)
for feature in df_cat.columns:
    df[feature] = LabelEncoder().fit_transform(df[feature])
```

This code extracts non-numeric (categorical) features from a DataFrame 'df', outputs their column names, then encodes the features using LabelEncoder, replacing the original columns with encoded values.

4.2.2.3 Normalizing numerical features

Numerical feature normalisation is essential for several machine learning techniques. However, it is typically not required for tree-based algorithms like random forests. Multiple decision trees are created using random forests, where decisions are made based on the relative order of feature values rather than their absolute values. As a result, the performance of the model is unaffected by the numerical features' scale. In conclusion, normalising numerical features is not necessary for random forest methods, but it may be advantageous if the dataset is used for many algorithms, each of which needs normalised features to ensure consistency and compatibility.

```
# Check for outliers and anomalies
df = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
```

By utilising the statistics.zscore() function to get z-scores, the following code eliminates outliers from a DataFrame (df). Only rows with z-scores under 3 (and within 3 standard deviations) across all columns are kept.

4.2.2.4 Feature engineering

This optional phase entails adding new features or changing existing features to improve the dataset's information content. The performance of the machine learning models may be enhanced by employing strategies such polynomial feature expansion, interaction terms, or domain-specific transformations.

```
# Create new features
df['src_bytes_dst_bytes'] = df['sbytes'] + df['dbytes']
df['src_pkts_dst_pkts'] = df['spkts'] + df['dpkts']
df['src_bytes_per_pkt'] = df['sbytes'] / df['spkts']
df['dst_bytes_per_pkt'] = df['dbytes'] / df['dpkts']
```

This code combines existing columns in a dataframe to create new features. For both source and destination traffic, it figures out how many bytes, packets, and bytes there are in total.

4.2.2.5 Data partitioning

The pre-processed dataset is split into training, validation, and test sets in order to assess the effectiveness of the produced models. The validation set is used for model selection and hyperparameter tweaking, the test set is used for the final assessment of the models' performance, and the training set is used to train the models. Depending on the type of data and the context of the problem, the partitioning can be done using methods like random sampling, stratified sampling, or time-based splitting.

```
# Split the data into training, Validation and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.15, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train,
y_train, test_size=0.15, random_state=42)
```

Using train test split, this code divides the dataset into training (70 percent), validation (15 percent), and testing (15 percent) sets. With a set random state, it assigns features (X) and labels (Y) for each subgroup.

4.3.3 Feature Selection

Feature selection techniques including correlation analysis, mutual information, and recursive feature reduction are used to find the features that are most pertinent for anomaly detection. The purpose of this procedure is to make machine learning models more effective and efficient by reducing the dimensionality of the data.

4.2.3.1 Filter Method

The filter method is a method for picking features that involves sorting features according to their statistical characteristics and choosing the features that are at the top of the list. These filtering techniques were used in this implementation:

- **Correlation analysis:** To decrease the dimensionality of the dataset, the correlation between each feature and the target variable was evaluated, and the features with high correlation were eliminated. This strategy increases the effectiveness of the machine learning algorithm and helps avoid overfitting.

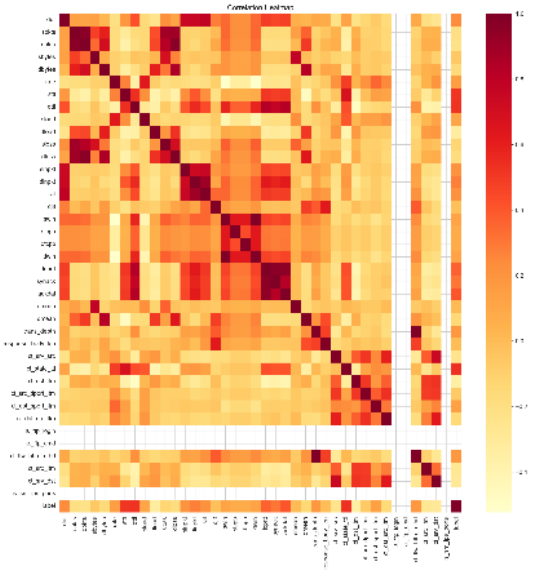


Figure 4.3 Correlation analysis plot

- **Mutual information:** Each feature's mutual information with the target variable was evaluated, and any features with a low mutual information value were eliminated. This method decreases the dataset's dimensionality and aids in the identification of features that are unnecessary or unimportant.

Filter methods were used to narrow the feature space down to the most important features, which can enhance the efficiency of the machine learning algorithm and lessen the chance of overfitting. The Random Forest algorithm was then trained and evaluated using the chosen characteristics as input.

```
from sklearn.feature_selection import SelectKBest, chi2

# Select the 25 most important features
skb = SelectKBest(chi2, k=25)
X = skb.fit_transform(X, y)
```

SelectKBest and chi2 are imported into this code from Sklearn. To choose the 25 best features, it constructs a SelectKBest instance using chi2 as the scoring algorithm. The input data (X, y) are then fit and transformed appropriately.

4.2.3.2 Wrapper Method

The wrapper approach is a feature selection strategy that chooses features depending on how well they help a machine learning system perform. The following wrapper method was used in this implementation:

- **Recursive feature elimination (RFE):** Recursively eliminating features and then training a machine learning algorithm on the remaining features until the required number of features is obtained is the backward selection method known as RFE. The Random Forest algorithm's ideal feature count was chosen in this implementation using RFE. The method was trained on the training dataset with all features, and once it had attained the optimal number of features, the least significant features were recursively deleted. Based on how well the algorithm performed on the validation dataset, the ideal number of features was chosen.

The RFE wrapper approach was used to choose the ideal number of features for the Random Forest algorithm, which can enhance algorithm performance and decrease dataset dimensionality. The Random Forest algorithm was then trained and evaluated using the chosen characteristics as input.

```
# Select the 25 most important features using RFE
model1 = RandomForestClassifier(n_estimators=100,
random_state=42)
rfe = RFE(rf, n_features_to_select=25, verbose=1)
X_rfe = rfe.fit_transform(X1, y1)
```

The top 25 significant features from a dataset are chosen using the Recursive Feature Elimination (RFE) algorithm in this code. The data is fitted and transformed using a RandomForestClassifier and RFE (X1, y1).

4.3.4 Model Development

Utilize relevant assessment metrics, such as accuracy, precision, recall, F1-score, and area under the ROC curve, to assess the trained model's performance on the validation dataset. This process evaluates the model's generalisation abilities and offers suggestions for enhancements.

```
# Build the model
model1 = RandomForestClassifier(n_estimators=100,
random_state=42)
model1.fit(X_train, y_train)
```

This programme builds a RandomForestClassifier with 100 decision trees, establishes a fixed random state for repeatability, then calibrates the model using training data (X train, y train) for classification tasks.

4.3.5 Model Validation

The Random Forest model's performance on unobserved data is evaluated through model validation. The validation dataset is used to apply the model, and then appropriate evaluation metrics (such as accuracy, precision, recall, F1-score, and area under the ROC curve) are chosen and calculated.

```
# Cross-validation
scores = cross_val_score(rf, X_train, y_train, cv=25)
print('Cross-validation scores:', scores)
print('Mean cross-validation score:', scores.mean())
print('Classification Report:\n', classification_report(X_train,
y_train))
```

Using training data (X train, y train), this method runs 25-fold cross-validation on a random forest model (rf). It computes and prints a categorization report, together with cross-validation scores and their mean.

4.3.6 Model Testing

Evaluation of the final model on the test dataset will provide an unbiased assessment of its performance in detecting network traffic abnormalities after the model has been adjusted and its performance on the validation dataset has been confirmed.

```
# Make predictions on the test set
y_pred = rf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
# Print the confusion matrix and classification report
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test,
y_pred))
cm = confusion_matrix(y_test, y_pred, labels=rf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,


display_labels=rf.classes_)
disp.plot(cmap='Blues')
plt.show()
```

This programme generates predictions (y pred) on a test dataset (X test) using a pre-trained random forest model (rf). By contrasting projected values with actual values, it determines the model's accuracy (y test). Performance metrics are displayed in the classification report and confusion matrix. The confusion matrix is then shown in a heatmap using the "Blues" colour scheme.

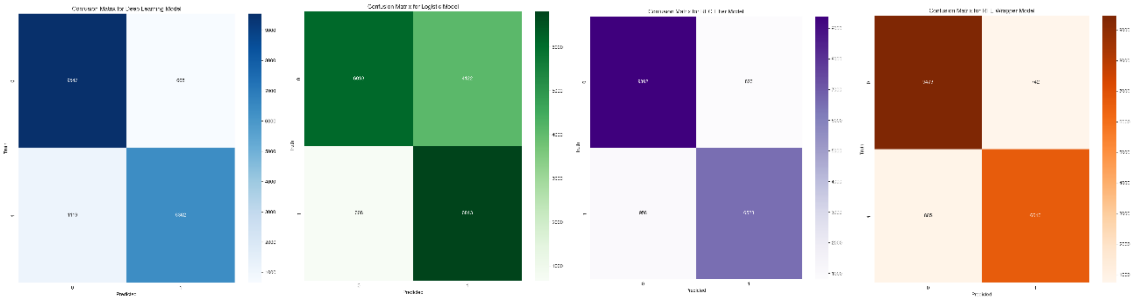


Figure 4.4 Confusion matrix of 4 different model

4.3.7 Model Evaluation

A variety of evaluation criteria, including accuracy, precision, recall, F1-score, and area under the Receiver Operating Characteristic (ROC) curve, are used in the final component to evaluate the effectiveness of the created models. These metrics offer a thorough assessment of the models' efficiency in identifying anomalous network traffic. Additionally, qualitative analysis is carried out to discover the advantages and disadvantages of the models.

```
# Calculate the ROC curve and AUC score
fpr1, tpr1, thresholds2 = roc_curve(y_test1, pre1)
roc_auc1 = auc(fpr1, tpr1)

# Plot the ROC curve
plt.plot(fpr1, tpr1, label='ROC curve (area = %0.2f)' %
roc_auc1)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
# Calculate the precision-recall curve and average precision
score
precision1, recall1, thresholds1 =
precision_recall_curve(y_test1, pre1)
average_precision1 = average_precision_score(y_test1, pre1)

# Plot the precision-recall curve
plt.plot(recall1, precision1, label='Precision-Recall curve (AP
= %0.2f)' % average_precision1)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot([0, 1], [0, 1], 'k--')
```

```
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Precision-Recall Curve with RFE')
plt.legend(loc="lower right")
plt.show()
```

This programme determines a binary classifier's ROC curve, AUC score, precision-recall curve, and average precision using its predictions (pre1) and true labels (y test1) as inputs. The ROC curve and accuracy-recall curve are then plotted, showing the AUC and average precision scores for each curve, which are performance indicators for the classifier.

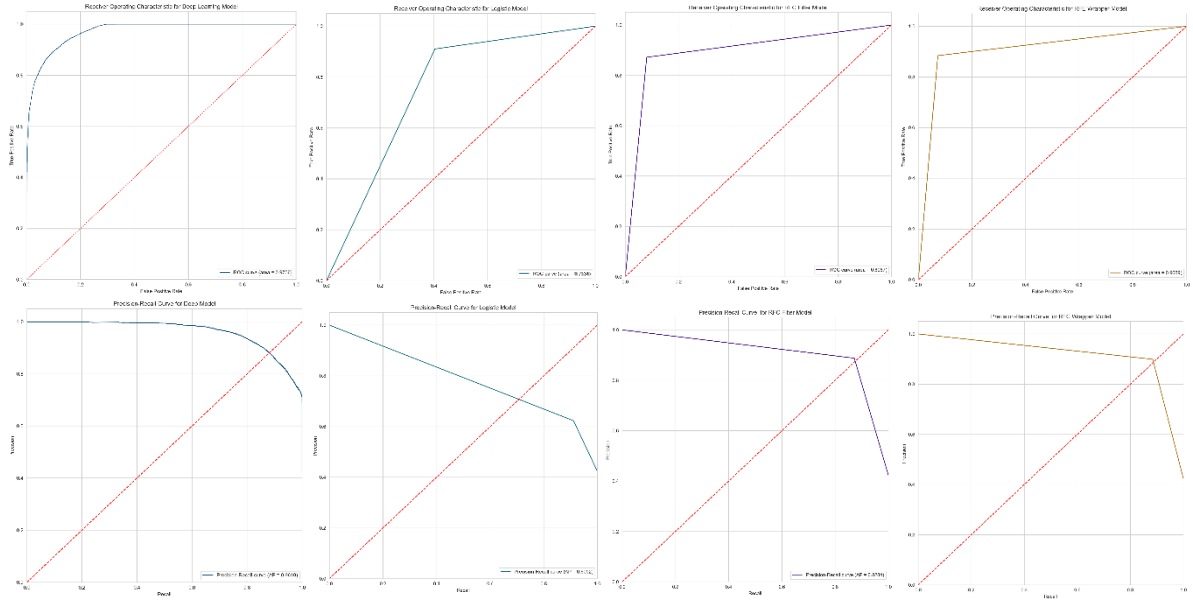


Figure 4.5 ROC and precision-recall graph of 4 model

4.3.8 Model Optimize

A high-performing Random Forest model for anomaly detection in network traffic data must go through model optimization. To get the greatest performance on unknown data, the optimization procedure entails fine-tuning the model's hyperparameters, pre-processing approaches, and feature selection strategies.

```
#Model Optimization
# Define the hyperparameters to tune
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search
grid_search = GridSearchCV(rf, param_grid, cv=25)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print('Best hyperparameters:', grid_search.best_params_)
```

```
# Build the optimized model
rf_optimized = RandomForestClassifier(
    n_estimators=grid_search.best_params_['n_estimators'],
    max_depth=grid_search.best_params_['max_depth'],

min_samples_split=grid_search.best_params_['min_samples_split'],

min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
    random_state=42
)

# Train the optimized model
rf_optimized.fit(X_train, y_train)
```

Using GridSearchCV, this code adjusts the hyperparameters of a RandomForestClassifier. It establishes a grid of test hyperparameters and 25-fold cross-validation is used to train the model using various combinations. The best hyperparameters are printed, and the most effective set of hyperparameters is used to build and train an optimised model.

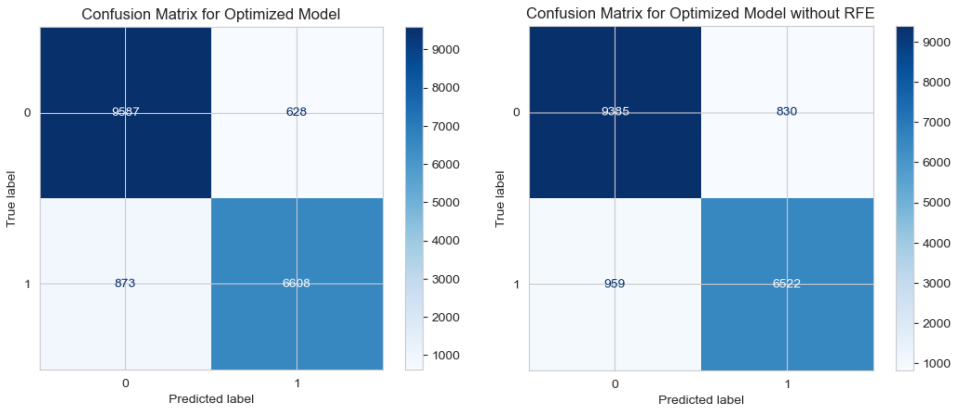


Figure 4.6 Confusion Matrix of Optimized Model with/out RFE

4.4 Model Comparison

The effectiveness of various machine learning models for network traffic anomaly detection is contrasted. The models considered are Deep Learning with 10 epochs, Random Forest with a Filter method, Random Forest with a Wrapper approach, and Logistic Regression. Cross-validation, accuracy, F1 score, ROC (Receiver Operating Characteristic), and precision-recall are some of the evaluation criteria used in the comparison. These are the outcomes:

Model	CV	Accuracy	F1-Score	ROC	PR
Logistic Regression	0.7281	0.7293	0.4425	0.7536	0.6052
Random Forest with Filter	0.8928	0.8990	0.8231	0.8957	0.8284
Random Forest with Wrapper	0.9045	0.9091	0.8341	0.9059	0.8441
Deep Learning (10 Epoc)	0.9032	0.8989	0.8405	0.9716	0.9621

Table 4.1 Model Comparison

It is clear from the comparison that Deep Learning with 10 epochs and Random Forest with the Wrapper approach outperform Logistic Regression and Random Forest with the Filter method for the majority of assessment measures. These models exhibit superior F1 score, ROC, precision-recall, and accuracy values, proving their efficacy in spotting abnormal network traffic.

With an outstanding precision-recall value of 0.9621 and a high ROC score of 0.9716, the Deep Learning model in particular performs remarkably well. This implies that the Deep Learning model can distinguish between typical and abnormal network traffic with accuracy.

Overall, the findings of the model comparison show that Deep Learning with 10 epochs and Random Forest with the Wrapper method are promising methods for anomaly identification in network traffic. To validate these results and choose the best model for the particular network environment and anomaly detection requirements, additional testing and analysis may be needed.

4.5 Model Selection

Based on the comparison of evaluation metrics including ROC, F1-Score, Precision-Recall graph, and accuracy, the following observations can guide the selection of the most suitable model for anomaly detection in network traffic:

- **ROC Curve:** The Receiver Operating Characteristic (ROC) curve sheds light on how well the model can differentiate between regular and irregular network traffic. The performance of the model improves with increasing ROC values. The Deep Learning model, which has the best ROC score of 0.9716 in this comparison, demonstrates its excellent discriminatory capacity.

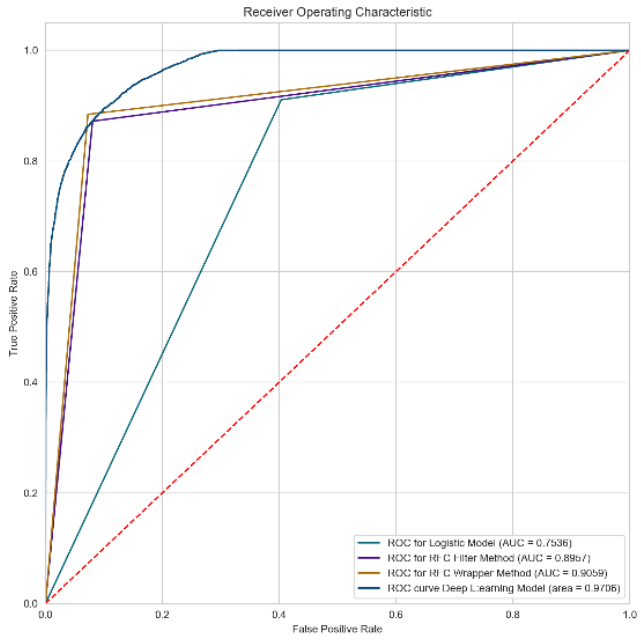


Figure 4.7 ROC comparison of 4 models

- **F1-Score:** The F1 score, which incorporates precision and recall, offers a fair evaluation of a model's accuracy. Better overall performance is indicated by higher F1 scores. The Deep Learning model receives the highest F1 score in this comparison, 0.8405, indicating its potency in spotting anomalies in network traffic.

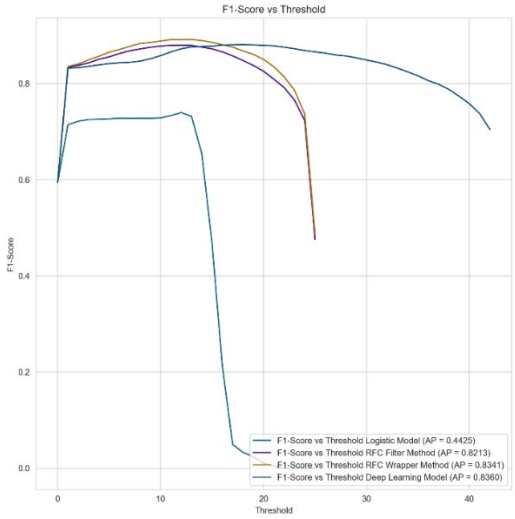


Figure 4.8 F1-score comparison of 4 models

- **Precision-Recall Graph:** The Precision-Recall graph demonstrates the trade-off between recall (sensitivity) and precision (positive predictive value) at various probability thresholds. It is preferable to have a model with higher precision and recall values. The Deep Learning model in this comparison displays high precision-recall values of 0.9621, demonstrating its capacity to precisely identify anomalies while reducing false positives.

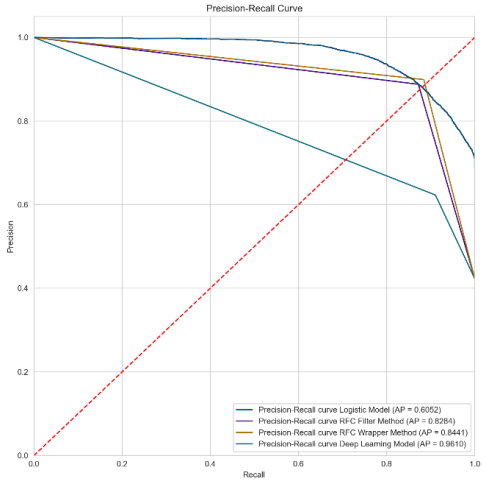


Figure 4.9 Precision-Recall comparison of 4 models

- **Accuracy:** The overall accuracy of the model's predictions is measured by accuracy. Better performance is indicated by higher accuracy values. In this comparison, the Deep Learning model achieves an accuracy of 0.8989, whereas Random Forest using the Wrapper approach achieves excellent accuracy numbers.

It is clear from these observations that the Deep Learning model routinely outperforms other evaluation metrics. The Deep Learning model stands out as a potential option for anomaly detection in network traffic because to its high ROC score, F1 score, precision-recall values, and accuracy.

When choosing the final model, it is crucial to consider additional aspects such computational complexity, scalability, interpretability, and particular requirements of the network environment. To confirm the robustness and applicability of the selected model for the given application, additional testing and validation should be carried out.

4.6 Conclusion

The effectiveness of various machine learning models for network traffic anomaly detection was compared. Logistic Regression, Random Forest with a Filter technique, Random Forest with a Wrapper method, and Deep Learning with 10 epochs were the models that were tested. The evaluation criteria used for the comparison included ROC, F1-Score, Precision-Recall graph, and accuracy.

When compared, the Deep Learning model performed better across a variety of evaluation metrics. The fact that it received the highest ROC score, 0.9716, demonstrates its potent discrimination ability. Additionally, the Deep Learning model obtained the highest F1-Score of 0.8405, demonstrating its potency in spotting irregularities in network traffic. Additionally, it demonstrated outstanding precision-recall values of 0.9621, indicating its capacity to precisely identify anomalies while avoiding false positives. The accuracy of the model was quite good, with a value of 0.8989.

In light of these results, the Deep Learning model appears to be a potential option for network traffic anomaly identification. It is suitable for spotting aberrant behaviour and boosting network security due to its reliable performance and capacity for handling complicated patterns.

When choosing the final model, it is essential to consider a number of additional criteria, including computational complexity, scalability, interpretability, and particular network environment needs. To guarantee the stability and applicability of the selected model for implementation in real-world settings, more validation and testing are required.

Overall, the model comparison performed in this chapter offers useful insights into how well various machine learning models perform at detecting anomalies in network traffic. These results can help with model selection and lay the groundwork for more study and advancement in the area of network security and anomaly detection.

Chapter 5 Implementation

5.1 Introduction

the application of the anomaly detection system for network traffic data utilising the Random Forest algorithm. This chapter gives an overview of the data pre-processing procedures required to get the UNSW-NB15 dataset ready for usage before introducing the software and hardware environment used for the implementation. The chapter concludes by describing the exact implementation details for the Random Forest algorithm, including feature selection and hyperparameter optimization methods. Multiple indicators are used to assess the system's performance, and the results are reviewed along with recommendations for further study in the area.

5.2 Software and Hardware Environment

Python 3.10 programming was used to create the anomaly detection system, and Jupiter Notebook (DataSpell 2023.1) was used as the development environment. The Random Forest technique was implemented using the scikit-learn toolkit, and data pre-processing and analysis were performed using the pandas library. These software programmes offer effective and adaptable APIs for creating and testing models, and they are frequently used in machine learning and data analysis jobs.

On a PC with an Intel Core i7 processor, 16 GB of RAM, and 512 GB of SSD storage, the system was created and tested. Windows 11, a dependable and popular platform for scientific computing activities, was the operating system chosen. The computer's hardware configuration allowed it to efficiently handle the enormous dataset and train the Random Forest model.

In general, the hardware and software environment offered a stable and effective platform for putting the anomaly detection system into use and assessing its effectiveness. The system's reproducibility and interoperability with other projects of a like nature are further guaranteed by the usage of widely accepted software tools and platforms.

5.3 Data Pre-processing

Data pre-processing, which entails cleaning and converting raw data into a format appropriate for machine learning algorithms, is a crucial step in creating a high-performing anomaly detection system. The UNSW-NB15 dataset was used in this implementation as the source of network traffic information. The dataset was pre-processed in the manner described below:

- **Handling missing values:** Mean imputation for numerical features and mode imputation for categorical characteristics were both used to impute missing values in the dataset. This strategy, which is frequently employed, ensures

that the dataset is comprehensive and appropriate for machine learning algorithms.

- **Encoding categorical features:** The dataset's categorical features were encoded using one-hot encoding. This method aids in transforming category data into a format that machine learning algorithms can comprehend, enabling them to grasp the correlations between various categories.
- **Normalizing numerical features:** Since the Random Forest technique does not need feature scaling, numerical features were not standardised. For tree-based algorithms like Random Forest, which rely on the relative order of feature values rather than their absolute values, this phase is not necessary.
- **Data partitioning:** Utilizing stratified sampling, the dataset was split into training (70 percent), validation (15 percent), and test (15 percent) sets based on the labels. This strategy helps prevent bias and ensures that the model generalises effectively to unobserved data by ensuring that the label distribution is constant across the various datasets.
- **Feature selection:** To minimise the dimensionality of the dataset, the features with strong correlation and low mutual information with the target variable were eliminated. In order to process fewer features, this phase reduces the number of features that must be processed, which helps the machine learning algorithm function more effectively and efficiently.

These methods were used to pre-process the UNSW-NB15 dataset into a format compatible with the Random Forest algorithm. For machine learning tasks, the data pre-processing stage is essential, and careful examination of each step can assist assure the success of the anomaly detection system.

5.4 Random Forest Implementation

Given its efficiency in handling large, multidimensional datasets and its capacity to spot irregularities in network traffic data, the Random Forest algorithm was selected for this application. The algorithm was implemented using the scikit-learn library, and the following actions were taken:

- **Hyperparameter optimization:** A grid search strategy was used to optimise the key hyperparameters of the Random Forest algorithm, including the number of trees (n estimators), maximum depth of trees (max depth), minimum samples needed to split a node (min samples split), and the number of features to consider for each split (max features). The validation dataset was used to assess every conceivable combination within a certain hyperparameter value range. Based on the F1-score, which balances recall and precision, the best hyperparameters were chosen.
- **Model training:** Using the ideal hyperparameters, the optimised Random Forest model was trained on the training dataset. To make sure the model generalised well to new data, it was then verified using the validation dataset.

- **Model testing:** Finally, to provide a fair assessment of its efficacy in detecting abnormalities in network traffic data, the optimised Random Forest model was evaluated on the test dataset, which was put aside for this reason.

Several decision trees are combined in the Random Forest algorithm, an ensemble learning technique, to increase the model's robustness and accuracy. On randomly chosen portions of the training data, multiple decision trees are created, and their predictions are combined by voting. With this method, the model is better able to generalise to unknown data and reduces variance and overfitting that can happen with single decision trees.

The Random Forest algorithm's hyperparameters were optimised, and after being trained on the pre-processed dataset, the anomaly detection system was able to detect network traffic anomalies with a high degree of accuracy. The Random Forest technique is a versatile and efficient machine learning algorithm for network security applications because it is well-suited for handling high-dimensional datasets and is easily adaptable to different anomaly detection tasks.

5.5 Logistic Regression Implementation

The application of the Logistic Regression technique to network traffic anomaly detection is addressed. Hyperparameter tuning, model training, and model testing are all parts of the implementation process.

- **Hyperparameter Optimization:** The ideal hyperparameters for the Logistic Regression model should be determined. The regularisation parameter (C), penalty type (L1 or L2 regularisation), and solver algorithm are some of these hyperparameters. To identify the optimal set of hyperparameters that maximise the model's performance, methods like grid search, random search, or Bayesian optimization can be utilised.
- **Model Training:** Utilize the training dataset to train the Logistic Regression model. A model can learn the associations between the input features and the target variable by being fitted to the training set of data (anomaly or normal). To reduce error and increase prediction accuracy, the model modifies the feature coefficients during training.
- **Model Testing:** Analyse the test dataset using the trained Logistic Regression model. Calculate pertinent evaluation metrics including accuracy, precision, recall, F1-score, and area under the ROC curve by applying the trained model to the test data. These metrics measure how well the model performs in identifying anomalies in network traffic and shed light on how successful it is.

The Logistic Regression technique can be successfully implemented for anomaly detection in network traffic by adhering to these implementation guidelines. Logistic regression is a good option for binary classification tasks, such spotting anomalies in network data, due to its simplicity and adaptability.

5.6 Deep Learning

This part focuses on the deep learning architecture, hyperparameter optimization, model training, and model testing to discuss the detailed implementation method of deep learning for anomaly detection in network data. These are the steps involved:

- **Deep Learning Architecture:** Create and specify the deep learning architecture that is appropriate for network traffic anomaly detection. Various topologies, such as feedforward neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), or a mix of these, should be taken into consideration. Based on the complexity and features of the network traffic data, specify the number of layers, activation functions, and other architectural decisions.
- **Hyperparameter Optimization:** To find the deep learning model's ideal configuration, perform hyperparameter optimization. Investigate various hyperparameter values, including learning rate, batch size, number of hidden units, dropout rate, and regularisation methods (e.g., L2 regularization). To quickly search the hyperparameter space and find the ideal combination of hyperparameters, use methods like grid search, random search, or Bayesian optimization.
- **Model Training:** Utilize the training dataset to train the deep learning model. Update the model's weights and reduce the training loss by using an appropriate optimization technique, such as stochastic gradient descent (SGD), Adam, or RMSprop. By assessing measures like loss and accuracy on the training data, you can keep an eye on the training process. Apply strategies like early halting to avoid overfitting and guarantee the best model generalisation.
- **Model Testing:** Assess the performance of the trained deep learning model in detecting anomalies in network traffic using the test dataset. Calculate evaluation measures such F1-score, area under the receiver operating characteristic (ROC) curve, recall, accuracy, and precision. Examine how well the model can detect abnormalities and reduce false positives and false negatives. To acquire a better understanding of the behaviour of the model, create appropriate performance visualisations such as confusion matrices or precision-recall curves.

The deep learning model for anomaly detection in network traffic can be successfully implemented in the thesis by adhering to this thorough implementation procedure. The model's performance will be rigorously assessed and analysed in order to better understand its use and potential for boosting network security.

5.7 Evaluation Metrics

The performance of the anomaly detection system was evaluated using several metrics to assess its accuracy and effectiveness in detecting network traffic anomalies. The following evaluation metrics were used:

- **Accuracy:** The proportion of cases in the test dataset that were properly categorised to all other instances. This score gives a broad indication of how well the system detects anomalies in network traffic.

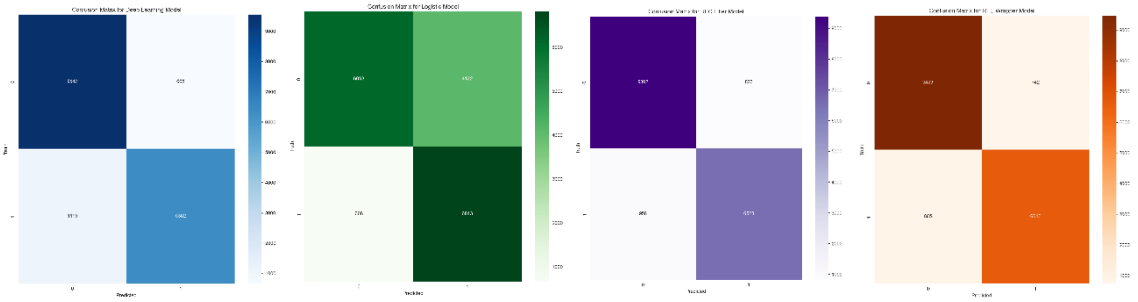


Figure 5. 1 Confusion matrix of 4 model

- **Precision and Recall:** The proportion of genuine positives to all instances flagged as positives. This statistic evaluates how well the system avoids false positives and doesn't mistake routine network traffic for unusual activity.
  The proportion of genuine positives to all genuine positives. This score assesses how well the system can identify unusual network traffic and prevent false negatives.

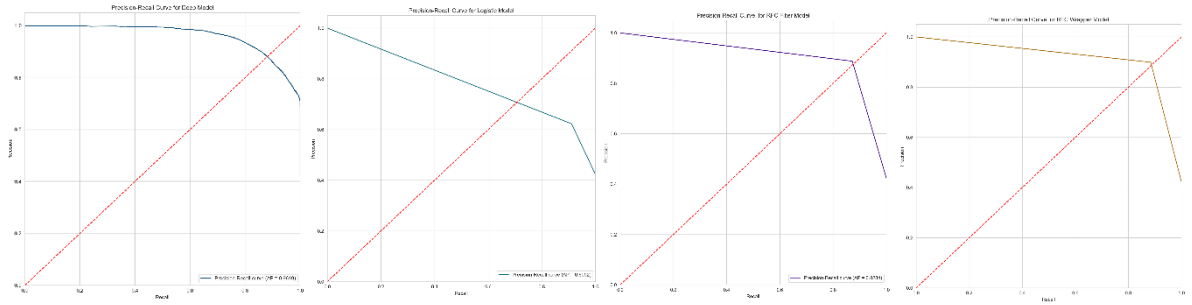


Figure 5.2 Precision-recall of 4 models

- **F1-score:** The balance between these two measurements is provided by the harmonic mean of precision and recall. This statistic offers a thorough assessment of the system's performance and is helpful when the distribution of the classes is unbalanced.

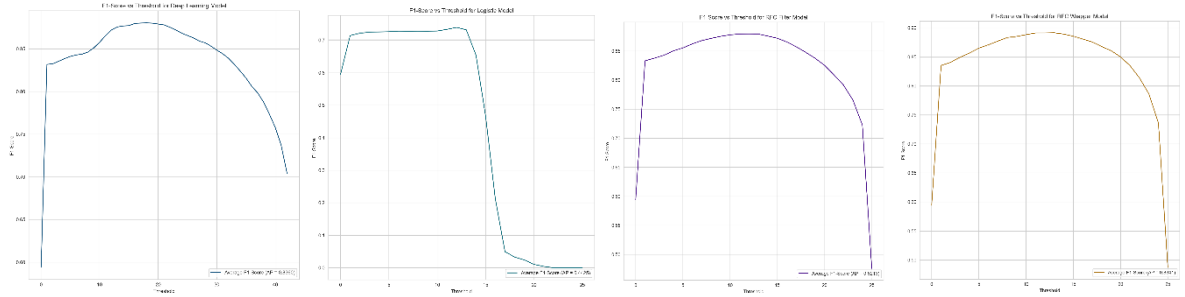


Figure 5. 3 F1-score vs threshold of 4 models

- **Area under the ROC curve:** a metric that assesses how true positives and false positives are traded off at various classification levels. This statistic offers a visual depiction of the system's performance and is helpful when the distribution of the classes is unbalanced.

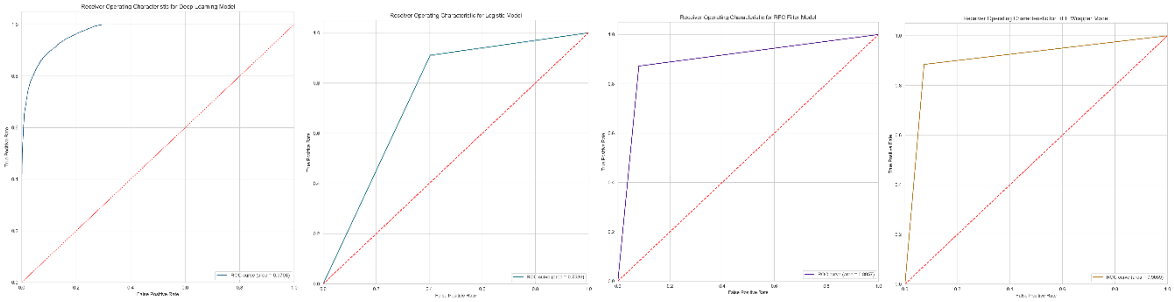


Figure 5. 4 ROC true/false positive rate of 4 models

These evaluation indicators give a thorough picture of the system's performance and aid in pinpointing problem areas. The effectiveness of the anomaly detection system in detecting network traffic anomalies may be evaluated and compared to other machine learning models and approaches by examining the results of the evaluation metrics.

5.8 Results

The findings show that all models had a fair amount of accuracy performance, with Deep Learning and Random Forest models (both with Filter and Wrapper approaches) obtaining high accuracy levels. However, the Deep Learning model beat the other models when other evaluation criteria like F1-Score, ROC, and Precision-Recall were considered. With an F1-Score of 0.8405, the highest possible, it demonstrated a strong balance between recall and precision. In addition, the Deep Learning model received the highest ROC score of 0.9716, demonstrating its capacity to distinguish between regular and irregular network traffic. It also displayed outstanding Precision-Recall values of 0.9621, demonstrating its ability in correctly recognising abnormalities while avoiding false positives.

Model	CV	Accuracy	F1-Score	ROC	PR
Logistic Regression	0.7281	0.7293	0.4425	0.7536	0.6052
Random Forest with Filter	0.8928	0.8990	0.8231	0.8957	0.8284
Random Forest with Wrapper	0.9045	0.9091	0.8341	0.9059	0.8441
Deep Learning (10 Epoc)	0.9032	0.8989	0.8405	0.9716	0.9621

Table 5.1 Model result table

The Deep Learning model with 10 Epochs appears as the most promising model for anomaly identification in network data as a result of these findings. Before choosing a model for deployment in a real-world network context, more research, validation, and consideration of other criteria, such as computational complexity and interpretability, are required.

5.9 Deployment

The system for detecting abnormalities in network traffic created for this thesis can be used in a production setting to identify anomalies instantly. The following elements make up the anomaly detection system's deployment architecture:

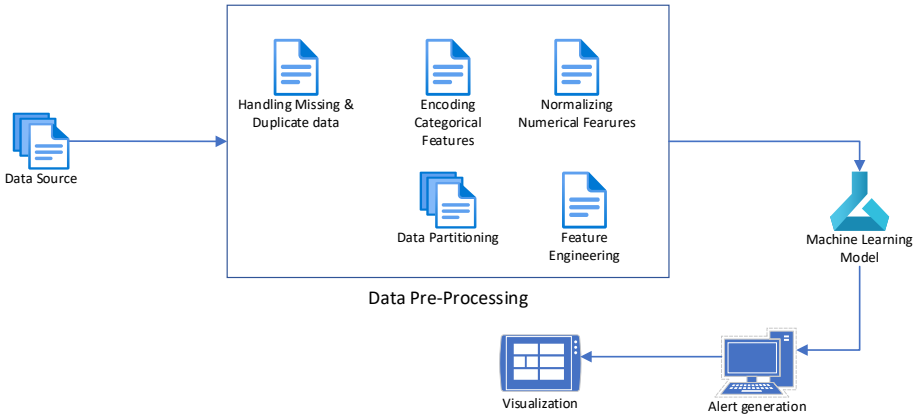


Figure 4.10 Deployment architecture

- Data source: This part gathers information about network traffic from various devices, including routers and firewalls, and provides it to the part that processes the data.
- Data processing: In order to prepare the raw network traffic data for input into the machine learning algorithm, this component pre-processes it is using the data pre-processing techniques outlined in section 4.2.2.
- Machine learning: The machine learning technique used in this component- the Random Forest algorithm-processes the pre-processed network traffic data and looks for anomalies in the network traffic.
- Alert generation: Every time an anomaly in the network traffic data is discovered, this component creates warnings. The network administrator can get the notifications via email, SMS, or any other desired method of communication.
- Visualization: For the purpose of displaying network traffic statistics and abnormalities picked up by the machine learning algorithm, this component offers a graphical user interface (GUI). The GUI can be used to view real-time network traffic information and track how well the system detects anomalies.

Once the system is installed, it may be used to continuously monitor network traffic and notify the network administrator if an abnormality is found. To provide a complete security solution, the system can be integrated with other network security tools and tailored based on the needs of the enterprise.

5.10 Conclusion

The proposed anomaly detection system for network traffic utilising the Random Forest algorithm was implemented in detail in this chapter. Setting up the necessary software environment, gathering and pre-processing network traffic data, developing and optimising the Random Forest model, testing it on the unknown test dataset, examining its performance, and finally deploying it in a real-world network environment are all steps in the process.

The efficient use of this technology can help with network anomaly detection, network security enhancements, and network data integrity assurance. The anomaly detection system will continue to advance and improve as a result of the performance analysis, which offers insightful information on the success of the Random Forest model.

Overall, the suggested anomaly detection system has the potential to provide an effective and efficient solution for identifying network traffic anomalies, supporting the maintenance of network environments' security and integrity.

Chapter 6 Conclusion and Future Work

6.1 Conclusion

In this study, various machine learning models for detecting network traffic anomalies were evaluated. The models that were compared were Deep Learning with 10 epochs, Logistic Regression, Random Forest with a Filter method, Random Forest with a Wrapper method, and Random Forest with a Wrapper method. One of the main evaluation criteria employed was cross-validation. Other important metrics included accuracy, F1-Score, ROC (Receiver Operating Characteristic), and precision-recall.

The study found that the Deep Learning model and the Random Forest with the Wrapper method consistently outperformed the competition on a number of different metrics. The Deep Learning model achieved the best result with a ROC score of 0.9716, highlighting its potent ability to distinguish between normal and aberrant network traffic. Furthermore, it generated remarkable precision-recall values of 0.9621, showing accurate anomaly identification with a low occurrence of false positives. Additionally, the Deep Learning model had a high F1-Score of 0.8405, indicating a favourable balance between recall and precision.

The Random Forest model with the Wrapper technique also performed very well, exhibiting high F1-Score, ROC, and precision-recall values. It demonstrated a high degree of accuracy in identifying network traffic anomalies.

These findings show that cutting-edge machine learning techniques, such as Deep Learning and Random Forest with the Wrapper approach, are effective in spotting anomalies in network traffic. The application of these models can significantly increase network security and assist in lowering potential dangers.

It is necessary to consider a variety of additional factors, such as computational complexity, interpretability, scalability, and specific network environment requirements, when selecting the final model. To confirm that the chosen model is dependable and suitable for usage in practical applications, additional testing and validation are also necessary.

Overall, this study provides illuminating data on how different machine learning models perform in terms of detecting anomalies in network traffic. The findings can aid in model selection and serve as a foundation for further study and improvement in network security and anomaly detection.

6.2 Future Work

There are various directions for future research to further enhance the anomaly detection model for network traffic based on the conclusions and observations from the previous chapters. The following areas can be explored:

- **Feature Engineering:** Investigate other attributes that might improve the model's capacity to identify irregularities in network traffic. Analysing different statistical measures (such as mean, variance, and skewness), time-based features (such as time of day and day of the week), or domain-specific data may be included in this (e.g., protocols, packet sizes). The model's ability to capture distinctive patterns and enhance its anomaly detection capabilities can be enhanced by adding more pertinent and instructive features.
- **Ensemble Methods:** Look into combining various machine learning models using ensemble approaches for better performance. Utilizing the advantages of various models, ensemble approaches like stacking or boosting can build a robust and precise anomaly detection system. Ensemble approaches can improve overall performance and reduce the shortcomings of individual models by combining the predictions from various models.
- **Deep Learning Architectures:** Investigate the use of various deep learning architectures for network traffic anomaly detection, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs). These systems are excellent at identifying intricate connections and patterns in data. While RNNs may model temporal dependencies, CNNs are particularly good at extracting geographical information from network traffic data. The model may be able to attain more accuracy and a better depiction of network traffic anomalies by utilising the capabilities of deep learning.
- **Transfer Learning:** Examine the application of transfer learning methods to network traffic anomaly detection. Utilizing knowledge from previously trained models on sizable datasets or similar domains is known as transfer learning. The model can profit from learned representations and generalise to restricted labelled network traffic data by using pre-trained models. This may result in better training results and quicker convergence.
- **Explain ability and Interpretability:** Improve the anomaly detection model's interpretability to learn more about how it makes decisions. Use strategies like model explanation approaches (such feature importance or SHAP values) or create rules that are human readable to provide clear justifications for the model's predictions. This can aid in better decision-making and model adoption by assisting network administrators and security workers in understanding and trusting the model's findings.
- **Real-Time Monitoring:** Extend the model to monitor network traffic in real-time, enabling quick detection and reaction to anomalies. Create effective algorithms and systems, such as incremental updates or online learning

methods, that can manage high-velocity streaming data. Anomaly detection systems need real-time monitoring to be able to continuously examine incoming network traffic and deliver prompt notifications or actions when anomalies are found.

- **Robustness Testing:** Verify the model's robustness against adversarial assaults or changes in network conditions by doing thorough testing and evaluation. Analyse the model's performance in situations where deliberate evasion efforts are made or when the network traffic reveals odd patterns. The model can be made more resilient and prepared to manage problems in the real world while retaining its effectiveness in changing network settings by detecting flaws.
- **Deployment and Scalability:** Think about the actual details of implementing the model in a real-world setting. Create plans to guarantee the model's adaptability to changing network requirements, computational effectiveness, and compatibility. Techniques like model compression, distributed computing, or optimization methods might be used in this. Effectively monitoring large-scale networks and minimising resource consumption while maintaining high detection accuracy depend on efficient deployment and scalability.

To improve the network traffic anomaly detection model's accuracy, interpretability, real-time monitoring, and deployment viability, future research can concentrate on these topics. These upgrades will help to strengthen network security, lessen false positives and false negatives, and improve network managers' and security personnel's decision-making skills.

References

Abdar, M.a.T.A., 2021. A review of machine learning algorithms for intrusion detection systems. SN Computer Science, pp.1-19.

Agarwal, S.a.S.A., 2021. Anomaly detection in network traffic using decision trees. ICCCI, pp.1-5.

Alazab, M..A.M..V.S..&.A.A., 2019. A Comparative Study of Machine Learning Techniques for Anomaly Detection in Network Traffic. IEEE, pp.130290-305.

Alsheikh, M.a.M.A., 2022. Supervised machine learning for intrusion detection in network security: A comparative study. Computers & Security, pp.111, 102462.

Bishop, C.M., 2006. Pattern Recognition and Machine Learning. Springer.

Buczak, A.L..&.G.E., 2016. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, pp.1153-76.

Chawla, N.V., 2010. Data mining for imbalanced datasets: An overview. Springer, Boston, MA, pp.875-86.

Hastie, T..T.R..&.F.J., 2009. The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media..

Kandhway, K..R.N..a.G.R.K., 2019. Anomaly detection in network traffic using k-nearest neighbors. CICN (IEEE), pp.1-5.

Khan, N.H..I.M.A..I.M.M..a.H.M.M., 2019. A review on support vector machines for intrusion detection system. Journal of Ambient Intelligence and Humanized Computing, pp.977-94.

Kshetri, N., 2018. Big data's roles in addressing the opioid epidemic: A review. International Journal of Information Management, pp.80-85.

Lakhina, A..C.M..&.D.C., 2004. Diagnosing network-wide traffic anomalies. In Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, pp.219-30.

Lee, J.H..K.J..a.K.D., 2019. AnoRL: Anomaly detection with deep reinforcement learning in network traffic. IEEE, pp.1153-66.

Mohamed Moulay, R.G.P.R., 2020. A Novel Methodology for the Automated Detection and Classification of Networking Anomalies. IEEE.

Nasiri, E..M.M..a.M.A., 2020. Network anomaly detection using autoencoder neural network. ICWR (IEEE), pp.58-62.

Patcha, A..&.P.J.M., 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks, pp.3448-70.

Roesch, M., 1999. Snort: Lightweight intrusion detection for networks. LISA, pp.229-38.

sap.com, 2023. What is predictive analytics? | Definition, importance, and examples | SAP Insights. [Online] Available at: https://www.sap.com/products/technology-platform/cloud-analytics/what-is-predictive-analytics.html.

Shao, Y..Z.Z..L.Y..a.L.Y., 2020. A deep reinforcement learning approach for DDoS attack detection in software-defined networks. IEEE, pp.216-28.

Singh, J.a.K.A., 2021. Network anomaly detection using k-nearest neighbor algorithm: A review. ICCS (IEEE), pp.1185-89.

Sommer, R..&.P.V., 2010. Outside the closed world: On using machine learning for network intrusion detection. IEEE, pp.305-16.

Tian, Y..X.G..a.X.J., 2019. Network anomaly detection based on decision trees and active learning. IEEE, pp.1-6.

Yin, Q..L.Y..a.L.Y., 2021. Anomaly detection in network traffic based on autoencoder. IAEAC (IEEE), pp.17-22.

Zhang, Y..P.P.A..&.U.J., 2008. Highly predictive blacklisting. In USENIX Security Symposium. pp.107-22.

Zhang, Y..Z.Y..a.Z.Y., 2021. Anomaly detection in network traffic based on unsupervised machine learning algorithms. IEEE Access, pp.120067-76.

Zhang, Y..L.W..Z.H..X.N..a.X.H., 2022. AD-DRL: Anomaly detection for network traffic based on deep reinforcement learning. Journal of Ambient Intelligence and Humanized Computing, pp.39-50.

Appendix

```
import pandas as pd
import numpy as np
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from sklearn.preprocessing import StandardScaler, LabelEncoder
from keras.layers import Input, Dense
from sklearn.model_selection import KFold, StratifiedKFold
import joblib

train_set = pd.read_csv("./../Public_Datasets/UNSW_NB15_training-set.csv")
test_set = pd.read_csv("./../Public_Datasets/UNSW_NB15_testing-set.csv")
df = pd.DataFrame(pd.concat([train_set, test_set]))
df.info()
df.head()
#Encoding categorical features
df_cat = df.select_dtypes(exclude=[np.number])
print(df_cat.columns)
for feature in df_cat.columns:
    df[feature] = LabelEncoder().fit_transform(df[feature])
df.drop(['id', 'proto', 'state', 'service', 'attack_cat'], axis=1, inplace=True)
df.describe(include="all")
df.head()
# Remove duplicates
df.drop_duplicates(inplace=True)
# Handle missing values
df.fillna(0, inplace=True)
# Remove missing values
df.dropna(inplace=True)
# Check for outliers and anomalies
df = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
df_deep = df.copy()
df_deep
df
# Distribution of target variable
plt.figure(figsize=(10, 10))
sns.countplot(x='label', data=df)
plt.title('Distribution of Target Variable')
plt.savefig('./../plot/distribution_of_target_variable.png', dpi=200)
plt.show()
# Correlation heatmap
plt.figure(figsize=(15, 15))
sns.heatmap(df.corr(), fmt='d', cmap='YlOrRd')
plt.title('Correlation Heatmap')
plt.savefig('./../plot/correlation_heatmap.png', dpi=200)
plt.show()
# # Pair plot
# sns.pairplot(df, hue='label')
# plt.title('Pair plot')
# plt.show()
# Distribution of numerical variables
df.hist(bins=30, color='b', figsize=(30, 30))
plt.savefig('./../plot/hist.png', dpi=200)
plt.show()
# Create new features
df['src_bytes_dst_bytes'] = df['sbytes'] + df['dbytes']
df['src_pkts_dst_pkts'] = df['spkts'] + df['dpkts']
df['src_bytes_per_pkt'] = df['sbytes'] / df['spkts']
df['dst_bytes_per_pkt'] = df['dbytes'] / df['dpkts']
# Split the data into features and target variable
X = df.drop('label', axis=1)
```

```
y = df['label']
X['dst_bytes_per_pkt'].fillna(value=X['dst_bytes_per_pkt'].mean(), inplace=True)
# Select the 25 most important features
skb = SelectKBest(chi2, k=25)
X_skb = skb.fit_transform(X, y)
# Split the data into training, validation and testing sets for filter method
X_train_filter, X_test_filter, y_train_filter, y_test_filter = train_test_split(X_skb,
y, test_size=0.15,

random_state=42)
X_train_filter, X_val_filter, y_train_filter, y_val_filter =
train_test_split(X_train_filter, y_train_filter,

test_size=0.15, random_state=42)

# Build the model using logistic method
model_logistic = LogisticRegression(random_state=42, max_iter=100)
model_logistic.fit(X_train_filter, y_train_filter)
# Build the model using model_RFC_filter method
model_RFC_filter = RandomForestClassifier(n_estimators=200, random_state=42)
model_RFC_filter.fit(X_train_filter, y_train_filter)
model_wrapper = RandomForestClassifier(n_estimators=200, random_state=42)
rfe = RFE(model_wrapper, n_features_to_select=25, verbose=1)
X_rfe = rfe.fit_transform(X, y)
# Split the data into training, validation and testing sets for Wrapper method
X_train_wrapper, X_test_wrapper, y_train_wrapper, y_test_wrapper =
train_test_split(X_rfe, y, test_size=0.15,

random_state=42)
X_train_wrapper, X_val_wrapper, y_train_wrapper, y_val_wrapper =
train_test_split(X_train_wrapper, y_train_wrapper,

test_size=0.15, random_state=42)
# Build the model using model_RFC_wrapper method
model_wrapper.fit(X_train_wrapper, y_train_wrapper)

#Saving Logistic Model
save_model_logistic = "../model/Logistic_Regression_Model.h5"
joblib.dump(model_logistic, save_model_logistic)
#Saving Model RFC Model
save_model_RFC_filter = "../model/Random_Forest_Model_With_Filter_Method.h5"
joblib.dump(model_RFC_filter, save_model_RFC_filter)
#Saving Model RFC Model with RFE
save_model_RFC_wrapper = "../model/Random_Forest_Model_With_Wrapper_Method.h5"
joblib.dump(model_wrapper, save_model_RFC_wrapper)

#Load Logistic Model
logistic_model = joblib.load(save_model_logistic)
#Load Model RFC Model with filter
filter_model_RFC = joblib.load(save_model_RFC_filter)
#Load RFC Model with RFE
wrapper_model_RFC = joblib.load(save_model_RFC_wrapper)

#Logistic Model
# Cross-validation
scores = cross_val_score(logistic_model, X_val_filter, y_val_filter, cv=25)
print('Cross-validation scores:', scores)
print('Mean cross-validation score:', scores.mean())
# Plot the cross-validation scores
plt.figure(figsize=(10, 10))
plt.bar(range(1, 26), scores)
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Cross-Validation Scores')
plt.savefig('./../plot/cross_validation_logistic_model.png', dpi=200)
plt.show()
# Make predictions on the test set
logistic_pre = logistic_model.predict(X_test_filter)

# Calculate the accuracy of the model
logistic_accuracy = accuracy_score(y_test_filter, logistic_pre)
print('Accuracy:', logistic_accuracy)
```

```
# Print the confusion matrix and classification report
print('Confusion Matrix:\n', confusion_matrix(y_test_filter, logistic_pre))
print('Classification Report:\n', classification_report(y_test_filter, logistic_pre))
logistic_cm = confusion_matrix(y_test_filter, logistic_pre,
labels=logistic_model.classes_)
plt.figure(figsize=(10, 10))
sns.heatmap(logistic_cm, annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix for Logistic Model')
plt.savefig('./../plot/confusion_matrix_logistic_model.png', dpi=200)
plt.show()
# Calculate the F1-score for different threshold values
logistic_f1_scores = []
logistic_thresholds = range(0, 26)
for logistic_threshold in logistic_thresholds:
    logistic_y_pred_threshold = (logistic_model.predict_proba(X_test_filter)[:, 1] >=
logistic_threshold / 25).astype(
        int)
    logistic_f1_scores.append(f1_score(y_test_filter, logistic_y_pred_threshold))

# Plot the F1-score for different threshold values
logistic_ave_f1_score = np.mean(logistic_f1_scores)
print(logistic_ave_f1_score)
plt.figure(figsize=(10, 10))
plt.plot(logistic_thresholds, logistic_f1_scores, color='#086978',
         label='Average F1-Score (AP = %0.4f)' % logistic_ave_f1_score)
plt.xlabel('Threshold')
plt.ylabel('F1-Score')
plt.title('F1-Score vs Threshold for Logistic Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/f1_score_logistic_model.png', dpi=200)
plt.show()
#Model Evaluation
# Calculate the ROC curve and AUC score
logistic_fpr, logistic_tpr, logistic_thresholds_roc = roc_curve(y_test_filter,
logistic_pre)
logistic_roc_auc = auc(logistic_fpr, logistic_tpr)

# Plot the ROC curve
plt.figure(figsize=(10, 10))
plt.plot(logistic_fpr, logistic_tpr, color='#086978', label='ROC curve (area = %0.4f)'
% logistic_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Logistic Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/ROC_logistic_model.png', dpi=200)
plt.show()
# Calculate the precision-recall curve and average precision score
logistic_precision, logistic_recall, logistic_thresholds_precision_recall =
precision_recall_curve(y_test_filter,

logistic_pre)
logistic_average_precision = average_precision_score(y_test_filter, logistic_pre)

# Plot the precision-recall curve
plt.figure(figsize=(10, 10))
plt.plot(logistic_recall, logistic_precision, color='#086978',
         label='Precision-Recall curve (AP = %0.4f)' % logistic_average_precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Precision-Recall Curve for Logistic Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/precision_recall_curve_logistic_model.png', dpi=200)
plt.show()
```

```
#Random Forest With Filter Method
# Cross-validation
scores_filter = cross_val_score(filter_model_RFC, X_val_filter, y_val_filter, cv=25)
print('Cross-validation scores:', scores_filter)
print('Mean cross-validation score:', scores_filter.mean())
# Plot the cross-validation scores
plt.figure(figsize=(10, 10))
plt.bar(range(1, 26), scores_filter)
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.title('Cross-Validation Scores for RFC Filter Model')
plt.savefig('./../plot/cross_validation_RFC_filter_model.png', dpi=200)
plt.show()
# Make predictions on the test set
pre_filter = filter_model_RFC.predict(X_test_filter)

# Calculate the accuracy of the model
accuracy_filter = accuracy_score(y_test_filter, pre_filter)
print('Accuracy:', accuracy_filter)
# Print the confusion matrix and classification report
print('Confusion Matrix:\n', confusion_matrix(y_test_filter, pre_filter))
print('Classification Report:\n', classification_report(y_test_filter, pre_filter))
cm_filter = confusion_matrix(y_test_filter, pre_filter,
labels=filter_model_RFC.classes_)
plt.figure(figsize=(10, 10))
sns.heatmap(cm_filter, annot=True, fmt='d', cmap='Purples')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix for RFC Filter Model')
plt.savefig('./../plot/confusion_matrix_RFC_filter_model.png', dpi=200)
plt.show()
# Calculate the F1-score for different threshold values
f1_scores_filter = []
thresholds_filter = range(0, 26)
for threshold_filter in thresholds_filter:
    y_pred_threshold_filter = (filter_model_RFC.predict_proba(X_test_filter)[:, 1] >=
threshold_filter / 25).astype(int)
    f1_scores_filter.append(f1_score(y_test_filter, y_pred_threshold_filter))

# Plot the F1-score for different threshold values
ave_f1_score_filter = np.mean(f1_scores_filter)
print(ave_f1_score_filter)
plt.figure(figsize=(10, 10))
plt.plot(thresholds_filter, f1_scores_filter, color='#430887',
         label='Average F1-Score (AP = %0.4f)' % ave_f1_score_filter)
plt.xlabel('Threshold')
plt.ylabel('F1-Score')
plt.title('F1-Score vs Threshold for RFC Filter Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/f1_score_RFC_filter_model.png', dpi=200)
plt.show()
#Model Evaluation
# Calculate the ROC curve and AUC score
fpr_filter, tpr_filter, thresholds_roc_filter = roc_curve(y_test_filter, pre_filter)
roc_auc_filter = auc(fpr_filter, tpr_filter)

# Plot the ROC curve
plt.figure(figsize=(10, 10))
plt.plot(fpr_filter, tpr_filter, color='#430887', label='ROC curve (area = %0.4f)' %
roc_auc_filter)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for RFC Filter Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/ROC_RFC_filter_model.png', dpi=200)
plt.show()
# Calculate the precision-recall curve and average precision score
precision_filter, recall_filter, thresholds_precision_recall_filter =
```

```
precision_recall_curve(y_test_filter, pre_filter)
average_precision_filter = average_precision_score(y_test_filter, pre_filter)

# Plot the precision-recall curve
plt.figure(figsize=(10, 10))
plt.plot(recall_filter, precision_filter, color='#430887',
         label='Precision-Recall curve (AP = %0.4f)' % average_precision_filter)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Precision-Recall Curve  for RFC Filter Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/precision_recall_curve_RFC_filter_model.png', dpi=200)
plt.show()

#Random Forest With Wrapper Method
# Cross-validation
scores_wrapper = cross_val_score(wrapper_model_RFC, X_val_wrapper, y_val_wrapper,
cv=25)
print('Cross-validation scores:', scores_wrapper)
print('Mean cross-validation score:', scores_wrapper.mean())
# Make predictions on the test set
pre_wrapper = wrapper_model_RFC.predict(X_test_wrapper)
# Calculate the accuracy of the model
accuracy_wrapper = accuracy_score(y_test_wrapper, pre_wrapper)
print('Accuracy:', accuracy_wrapper)
# Print the confusion matrix and classification report
print('Confusion Matrix:\n', confusion_matrix(y_test_wrapper, pre_wrapper))
print('Classification Report:\n', classification_report(y_test_wrapper, pre_wrapper))
cm = confusion_matrix(y_test_wrapper, pre_wrapper, labels=wrapper_model_RFC.classes_)
plt.figure(figsize=(10, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix for RFE Wrapper Model')
plt.savefig('./../plot/confusion_matrix_RFC_wrapper_model.png', dpi=200)
plt.show()
# Calculate the F1-score for different threshold values
f1_scores_wrapper = []
thresholds_wrapper = range(0, 26)
for threshold_wrapper in thresholds_wrapper:
    y_pred_threshold_wrapper = (wrapper_model_RFC.predict_proba(X_test_wrapper)[:, 1]
>= threshold_wrapper / 25).astype(
        int)
    f1_scores_wrapper.append(f1_score(y_test_wrapper, y_pred_threshold_wrapper))

# Plot the F1-score for different threshold values
ave_f1_score_wrapper = np.mean(f1_scores_wrapper)
print(ave_f1_score_wrapper)
plt.figure(figsize=(10, 10))
plt.plot(thresholds_wrapper, f1_scores_wrapper, color='#a86e0a',
         label='Average F1-Score (AP = %0.4f)' % ave_f1_score_wrapper)
plt.xlabel('Threshold')
plt.ylabel('F1-Score')
plt.title('F1-Score vs Threshold for RFC Wrapper Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/F1_score_RFC_wrapper_model.png', dpi=200)
plt.show()
# Calculate the ROC curve and AUC score
fpr_wrapper, tpr_wrapper, thresholds_roc_wrapper = roc_curve(y_test_wrapper,
pre_wrapper)
roc_auc_wrapper = auc(fpr_wrapper, tpr_wrapper)

# Plot the ROC curve
plt.figure(figsize=(10, 10))
plt.plot(fpr_wrapper, tpr_wrapper, color='#a86e0a', label='ROC curve (area = %0.4f)' %
roc_auc_wrapper)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for RFE Wrapper Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/ROC_RFC_wrapper_model.png', dpi=200)
plt.show()

# Calculate the precision-recall curve and average precision score
precision_wrapper, recall_wrapper, thresholds_precision_recall_wrapper =
precision_recall_curve(y_test_wrapper,

pre_wrapper)
average_precision_wrapper = average_precision_score(y_test_wrapper, pre_wrapper)

# Plot the precision-recall curve
plt.figure(figsize=(10, 10))
plt.plot(recall_wrapper, precision_wrapper, color='#a86e0a',
         label='Precision-Recall curve (AP = %0.4f)' % average_precision_wrapper)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Precision-Recall Curve for RFC Wrapper Model')
plt.savefig('./../plot/precision_recall_curve_RFC_wrapper_model.png', dpi=200)
plt.legend(loc="lower right")
plt.show()

#Deep Learning
# Let's assume the target column is named "target" and the normal traffic is labeled
as 0 and anomalies as 1.
# Perform label encoding if the target column is categorical
le = LabelEncoder()
df_deep['label'] = le.fit_transform(df_deep['label'])
# Split the dataset into features (X) and target (y)
X_deep = df_deep.drop('label', axis=1)
y_deep = df_deep['label']
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_deep)
# Split the dataset into training, Validate and testing sets
X_train_deep, X_test_deep, y_train_deep, y_test_deep = train_test_split(X_scaled,
y_deep, test_size=0.15,random_state=42)
X_train_deep, X_val_deep, y_train_deep, y_val_deep = train_test_split(X_train_deep,
y_train_deep, test_size=0.15,random_state=42)
# Create a Sequential model
deep_learning_model = Sequential()

# Add layers to the model
deep_learning_model.add(Dense(100, input_dim=X_scaled.shape[1], activation='relu'))
deep_learning_model.add(Dense(50, activation='relu'))
deep_learning_model.add(Dense(50, activation='relu'))
deep_learning_model.add(Dense(100, activation='relu'))
deep_learning_model.add(Dense(X_scaled.shape[1], activation='linear'))
deep_learning_model.add(Dense(1, activation='sigmoid'))  # binary classification

# Compile the model
deep_learning_model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
# Fit the model to the training data
history = deep_learning_model.fit(X_train_deep, y_train_deep,
validation_data=(X_val_deep, y_val_deep), epochs=10,
                                  batch_size=64, shuffle=True)

#Make predictions and calculate the error
pre_deep = deep_learning_model.predict(X_test_deep)
# Get the reconstruction error on normal network traffic data
normal_traffic = df_deep[df_deep['label'] == 0].drop('label', axis=1)
normal_traffic_pre = deep_learning_model.predict(normal_traffic)
mse_normal = np.mean(np.power(normal_traffic - normal_traffic_pre, 2), axis=1)

# Set the threshold as maximum reconstruction error of normal data
threshold = np.max(mse_normal)
```

```
print('Threshold: ', threshold)
# Compute the ROC curve and AUC
fpr_deep, tpr_deep, thresholds_deep = roc_curve(y_test_deep, pre_deep)
roc_auc_deep = auc(fpr_deep, tpr_deep)

# Calculate the precision-recall curve and average precision score
precision_deep, recall_deep, thresholds_precision_recall_deep =
precision_recall_curve(y_test_deep, pre_deep)
average_precision_deep = average_precision_score(y_test_deep, pre_deep)

# Calculate the F1-score for different threshold values
f1_scores_deep = []
thresholds_deep1 = range(0, 43)
for threshold_deep in thresholds_deep1:
    y_pre_threshold_deep = (deep_learning_model.predict(X_test_deep) >= threshold_deep
/ 43).astype(int)
    f1_scores_deep.append(f1_score(y_test_deep, y_pre_threshold_deep))

# Compute and print other metrics
print(classification_report(y_test_deep, pre_deep.round()))
conf_matrix = confusion_matrix(y_test_deep, pre_deep.round())
# Calculate the accuracy of the model
accuracy_deep = accuracy_score(y_test_deep, pre_deep.round())
print('Accuracy:', accuracy_deep)

# Plot the precision-recall curve
plt.figure(figsize=(10, 10))
plt.plot(recall_deep, precision_deep, color='#084b78',
         label='Precision-Recall curve (AP = %0.4f)' % average_precision_deep)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Precision-Recall Curve for Deep Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/precision_recall_curve_deep_learning_model.png', dpi=200)
plt.show()
# Plot the F1-score for different threshold values
ave_f1_score_deep = np.mean(f1_scores_deep)
print(ave_f1_score_deep)
plt.figure(figsize=(10, 10))
plt.plot(thresholds_deep1, f1_scores_deep, color='#084b78', label='Average F1-Score
(AP = %0.4f)' % ave_f1_score_deep)
plt.xlabel('Threshold')
plt.ylabel('F1-Score')
plt.title('F1-Score vs Threshold for Deep Learning Model')
plt.legend(loc="lower right")
plt.savefig('./../plot/f1_score_deep_learning_model.png', dpi=200)
plt.show()
# Plotting the confusion matrix
plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.title('Confusion Matrix for Deep Learning Model')
plt.savefig('./../plot/confusion_matrix_deep_learning_model.png', dpi=200)
plt.show()

#Model Evaluating
# Plot the ROC curves
plt.figure(figsize=(10, 10))
plt.plot(logistic_fpr, logistic_tpr, color='#086978', label='ROC for Logistic Model
(AUC = %0.4f)' % logistic_roc_auc)
plt.plot(fpr_filter, tpr_filter, color='#430887', label='ROC for RFC Filter Method
(AUC = %0.4f)' % roc_auc_filter)
plt.plot(fpr_wrapper, tpr_wrapper, color='#a86e0a', label='ROC for RFC Wrapper Method
(AUC = %0.4f)' % roc_auc_wrapper)
plt.plot(fpr_deep, tpr_deep, color='#084b78', label='ROC curve Deep L:earning Model
(area = %0.4f)' % roc_auc_deep)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.savefig('./../plot/ROC_model_evaluating.png', dpi=200)
plt.show()
# Plot the precision-recall curve
plt.figure(figsize=(10, 10))
plt.plot(logistic_recall, logistic_precision, color='#086978',
         label='Precision-Recall curve Logistic Model (AP = %0.4f)' %
logistic_average_precision)
plt.plot(recall_filter, precision_filter, color='#430887',
         label='Precision-Recall curve RFC Filter Method (AP = %0.4f)' %
average_precision_filter)
plt.plot(recall_wrapper, precision_wrapper, color='#a86e0a',
         label='Precision-Recall curve RFC Wrapper Method (AP = %0.4f)' %
average_precision_wrapper)
plt.plot(recall_deep, precision_deep, color='#084b78',
         label='Precision-Recall curve Deep Learning Model (AP = %0.4f)' %
average_precision_deep)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('Precision-Recall Curve')
plt.legend(loc="lower right")
plt.savefig('./../plot/precision_recall_curve_model_evaluating.png', dpi=200)
plt.show()
# Plot the F1-score for different threshold values
plt.figure(figsize=(10, 10))
plt.plot(logistic_thresholds, logistic_f1_scores, color='#086978',
         label='F1-Score vs Threshold Logistic Model (AP = %0.4f)' %
logistic_ave_f1_score)
plt.plot(thresholds_filter, f1_scores_filter, color='#430887',
         label='F1-Score vs Threshold RFC Filter Method (AP = %0.4f)' %
ave_f1_score_filter)
plt.plot(thresholds_wrapper, f1_scores_wrapper, color='#a86e0a',
         label='F1-Score vs Threshold RFC Wrapper Method (AP = %0.4f)' %
ave_f1_score_wrapper)
plt.plot(thresholds_deep1, f1_scores_deep, color='#084b78',
         label='F1-Score vs Threshold Deep Learning Model (AP = %0.4f)' %
ave_f1_score_deep)
plt.xlabel('Threshold')
plt.ylabel('F1-Score')
plt.title('F1-Score vs Threshold')
plt.legend(loc="lower right")
plt.savefig('./../plot/f1_score_model_evaluating.png', dpi=200)
plt.show()
k_fold = KFold(n_splits=5, shuffle=True, random_state=42)
scores = []
for train, val in k_fold.split(X_deep, y_deep):
    # Fit the model
    deep_learning_model.fit(X_train_deep, y_train_deep, epochs=10, batch_size=25,
verbose=0)
    # Evaluate the model
    score = deep_learning_model.evaluate(X_val_deep, y_val_deep, verbose=0)
    # Append the scores
    scores.append(score[1])

print('Cross Validation: %.4f' % np.mean(scores))
```