University of Reading

NETWORK INTRUSION DETECTION SYSTEM FOR DETECTING UNKNOWN NETWORK ATTACKS USING MACHINE LEARNING METHODS

Thesis submitted for the Degree of Doctor of Philosophy

Department of Computer Science

School of Mathematical, Physical and Computational Sciences

Saif Mohammad Yousef Alzubi

September 2022

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Saif Mohammad Yousef Alzubi

Acknowledgements

The pursuit of a PhD is a long and challenging journey which requires years of hard work, commitment and dedication to succeed. First and foremost, thanks to Allah (God), the Almighty, for giving me the strength, health, patience and guidance to complete this journey.

I would like to express my gratitude and appreciation to my supervisor, Dr. Fredric Stahl, for his direction, continual support and encouragement throughout my PhD journey. I am honoured and fortunate to be one of his students. Also, a special thanks goes to my external supervisor, Professor Mohamed Medhat Gaber, for his valuable comments and feedback.

I also want to express my sincere gratitude to my wonderful family for their endless support. To my father, my inspiration and source of strength, Mohammad—for all the sacrifices he made to make my dream come true, and for his unlimited support, financially and emotionally. To my mom, Nabeela, the most precious person in my life—for her prayers, encouragement, motivation and unconditional love. To my beloved wife and soul-mate, Bayan, who believed in me, encouraged me, and stood by me through thick and thin; without her, I wouldn't have been able to complete this journey. To my children Murad, Ahmad and Yasmeen, who always surrounded me with love, joy and happiness. To my brothers Samer and Yazan, my sister Heba and my brother-in-law Abdallah for their constant support and encouragement. And a big 'thank you' to my aunt, Wafaa, for all the advice and for always being there for me during this journey.

Abstract

Since the beginning of the internet age, the number of internet users has been rapidly increasing. Accordingly, the number of network attacks and their associated complexity is likewise rising. This increase in network attacks has triggered an alarm for governments and organisations, which have begun to invest millions in cybersecurity to mitigate the risk of cyberattacks. One effective, practical tool to defend against cyberattacks is the Intrusion Detection System (IDS) [1]. IDSs have been brought to the attention of researchers, who have begun incorporating Machine Learning (ML) methods into these systems. For this purpose, different IDSs using supervised and unsupervised ML methods have been proposed.

An IDS based on supervised learning methods can detect known network attacks that the system has previously encountered and been trained on. However, they often fail to detect network attacks that are unfamiliar to the supervised model. Unsupervised learning methods can overcome this limitation and detect new, unfamiliar attack types that the system has never encountered. Nevertheless, unsupervised learning methods can produce many false positives [2], low precision and recall results.

For this thesis, four research aims were developed and investigated. The first regards the possibility of developing a network IDS that offers high detection performance. The second aim considers the ability of the developed system to detect new network attacks introduced to the system. The third aim investigates the possibility of improving the overall results by implementing supervised ML models in the system. The fourth aim focuses on the feasibility of including explainable methods to help domain experts assess the threat level and understand the model's decisions.

To achieve these goals, this thesis presents a novel Network Intrusion Detection System framework that utilises the power of both unsupervised and supervised learning methods for network intrusion detection. The proposed framework consists of three components. The first component is a novel heterogeneous unsupervised bagging ensemble, called the Unknown Network Attack Detector (UNAD). A set of anomaly detection algorithms were

evaluated for their potential utility as base learners for UNAD. Among these algorithms, the Local Outlier Factor (LOF) and Isolation Forest (iForest) algorithms were selected as UNAD's base learners, as they produced the best results. Further, the weighted majority voting method is used as a results combiner for UNAD's base learners.

The second component of this framework is the supervised algorithm, trained on UNAD's detected benign/no-rmal and attack flows, that improves the overall detection results. The Random Forest (RF) classifier was selected for this component because it produced the strongest results, as measured empirically. The third component in this framework is the explainable component, which explains the decision made by the model in a human-understandable way. Two types of explainability are implemented and illustrated in this thesis: local and global. For local explainability, Local Interpretable Model-agnostic Explanations (LIME) [3] was used, and for global explainability, the surrogate method based on the Decision Tree (DT) was used.

The framework proposed in this thesis was evaluated using two publicly available datasets: CICIDS2017 [4] and NSL-KDD [5]. Empirical results revealed that UNAD—the first component—can detect completely new attack types with high detection rates for most attack types, and the RF classifier—the second component—can boost the detection rate for most attack types. The overall F1-scores for the CICIDS2017 and the NSL-KDD datasets were 98.31% and 98.25%, respectively. These experimental results showed that the explainable methods used in the system—the third component—can help domain experts assess threat levels and understand how the model made its decisions.

Contents

List of Figures

List of Tables

Abbreviations

AdaBoost	Adaptive Boosting
AUC	Area Under the Curve
Bagging	Bootstrap aggregating
C1	First Component
C2	Second Component
C3	Third Component
CSV	Comma-Separated Values
CV	Cross-validation
DDoS	Distributed Denial of Service
DoS	Denial of Service
DT	Decision Tree
EE	Elliptic Envelope
FI	Features Importance
FN	False Negative
FP	False Positive
HIDS	Host-based Intrusion Detection System
HTTPS	Hypertext Transfer Protocol Secure
ICA	Independent Component Analysis
IDS	Intrusion Detection System
IG	Information Gain
IoT	Internet of Things
iForest	Isolation Forest
KNN	K-nearest neighbors
LIME	Local Interpretable Model-agnostic Explanations
LOF	Local Outlier Factor
ML	Machine Learning
MV	Majority Voting

NB	Naive Bayes
NIDS	Network-based Intrusion Detection System
OCSVM	One-Class Support Vector Machine
PDP	Partial Dependence Plot
PC	Principal Component
PCA	Principal Component Analysis
RF	Random Forest
ROC	Receiver Operating Characteristics
SDN	Software Defined Networking
SSL	Secure Sockets Layer
SSC	Sub-Space Clustering
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
R2L	Remote to Local Attack
U2R	User to Root Attack
UNAD	Unsupervised Network Anomaly Detector
WMV	Weighted Majority Voting
XSS	Cross-Site Scripting

Chapter 1

Introduction

1.1 Background

The internet and its services have become an essential part of daily life, with billions of online users every month using mobile, desktops, tablets and Internet of Things (IoT) devices. According to Statista—the Statistics portal website—nearly 4.66 billion people were active on the internet as of January 2021 (Figure 1.1) [10].

Furthermore, the COVID-19 pandemic pushed more activity on to the internet, with many people shifting to remote work, creating new challenges for the public and private sectors [11]. This rapid increase in the number of internet users attracted hackers, hacktivists (a person or a group of people who hack into a computer or misuse a network for a social or politically motivated cause) and organisations with political agendas to develop sophisticated new types of network attacks aimed at leaking sensitive government data, exploiting victims' data to steal bank account details, using ransomware to extort money from victims, etc., all in spite of strict cybercrime legislation in most of the countries. These attacks are difficult to distinguish from benign/normal flow. As a result, cybersecurity has become integral to a country's national security, and businesses and governments are investing in cybersecurity to protect against these attacks. For instance, in the UK government's 2021 autumn budget and spending review report, there are plans to invest £2.6 billion in cyber

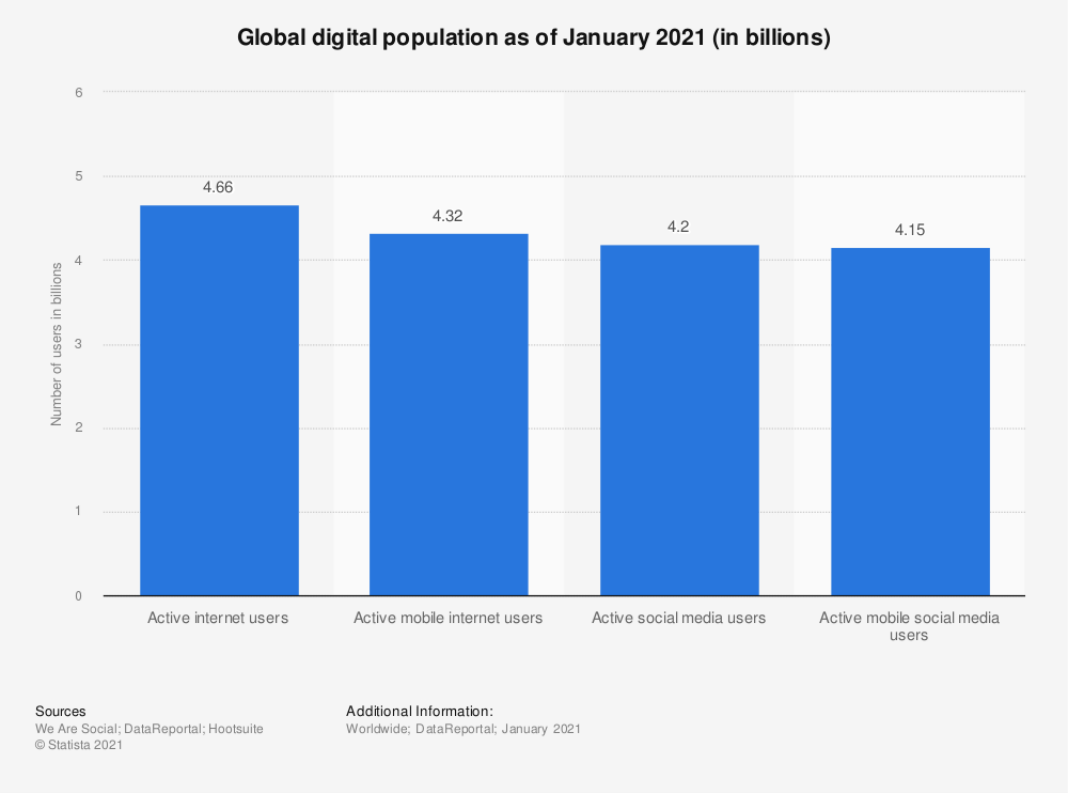


Figure 1.1: Global Digital Population as of January 2021

and legacy IT systems for the next three years (2022–2025) to improve the government's cybersecurity [12].

In general, network attacks have an enormous impact on an economy, and their annual cost keeps increasing; different sectors (e.g., government, health and education) are targeted with different types of attacks. According to Cybersecurity Ventures—the world's leading cybersecurity researcher and publisher—cybercrimes are expected to cost $10.5 (£7.9) trillion annually by 2025 [13].

Network attacks on businesses can be catastrophic, as they can affect reputation and cause financial losses [14]. For instance, Ticketmaster, the American ticket sales and distribution, was fined £1.25m over a payment data breach because the payment information for millions of European customers had been stolen due to a vulnerability in a third-party chatbot installed on Ticketmaster's online payments portal that allowed the hacker to gain access [15].

Similarly, Flightradar24, the popular real-time flight-tracking website, was struck by three cyberattacks on two consecutive days in September 2020 [16]. India's national airline, Air India, was hit by a cyberattack on its data servers in February 2021; customer details, including passport, ticket information and credit card data, were disclosed for around 4.5 million customers [17]. A cyberattack hit Sunderland University in the UK in October 2021. This major attack left the university's telephone, website and IT services offline and inaccessible for several days [18]. In 2021, attackers targeted multiple UK Voice-over-Internet Protocol service providers with DDoS attacks to render their servers down and inaccessible [19]. A more recent cyberattack focused on the Colonial Pipeline, the largest fuel pipeline in the US, forcing its operators to shut it down. This incident caused distribution problems that led to consumer panic-buying and an increase in fuel prices [20].

1.2 Research Motivation

With the increased number and complexity of the newly created network attacks, it became challenging to detect them. One tool that can be used to defend against network attacks is an Intrusion Detection System (IDS) which monitors and analyses the incoming network traffic [1].

Currently, there is considerable research interest in developing an IDS using ML algorithms. Many of the proposed systems are based on supervised learning methods [21]; however, these are effective only in detecting previously known attacks, as a supervised models must be trained on an attack beforehand. This research explores the use of unsupervised algorithms to address this limitation on the utility of IDSs.

Unsupervised intrusion detection aims to detect network attacks by being trained on an unlabeled dataset, thus assuming no previous knowledge about an attack. However, because unsupervised intrusion detection generally incurs many false positives [2], low precision and recall results, there is a need for more investigation of these methods. Supervised learning methods, as previously pointed out, effectively detect previously known attacks;

therefore, incorporating this capability into the proposed system may improve the overall results.

Using these ML models in the system will lead to effective detection of network attacks, but the reasoning behind the predictions of these models will not be transparent. This disadvantage will prevent domain experts from verifying, interpreting, and understanding the system's logic [22]. Therefore, in order for domain experts to rely on such systems, it is critical to overcome the transparency limitation of the black-box ML model [23] by incorporating ML explainability into the system.

Accordingly, this research first evaluates the performance of several unsupervised algorithms with respect to precision, recall, and F1-score and their limitations. Then, it examines ensemble methods as a possible way to mitigate the limitations of existing anomaly detection methods and derive a new, more accurate, reliable technique to work as the system's first line of defence, while deploying a supervised model to serve as the second line. This supervised model will be trained on the detected attacks from the unsupervised ensemble, so as more attacks are encountered, the system learns them. Finally, this research explores ML explainability methods to help domain experts assess threat levels by providing them with interpretations of the model's decision.

1.3 Problem Statement

The research problem statement can be summarised as follows:

1. An increase in network sizes, speeds and complexity is causing significant challenges in detecting network attacks. Therefore, a model capable of handling these changes is needed.

2. Most IDSs are supervised, meaning they have been trained on attacks previously; unsupervised IDSs usually exhibit poor detection performance, putting more pressure on the domain expert/system administrator to fine-tune the system.

3. Current unsupervised IDSs do not provide insights into network attack detections.

An explanation of the detected attack would help domain experts assess the threat.

1.4 Research Questions

Based on the research problem statement, this thesis covers the following research questions:

1. Is it possible to develop an unsupervised Network Intrusion Detection System that can exhibit a high detection performance in terms of precision, recall and F1-score while maintaining good performance over time with the current complexity in network attacks?
2. To what extent can the developed unsupervised Network Intrusion Detection System accurately detect attacks that have not been encountered before?
3. Is it possible to improve the system's detection accuracy after the initial discovery of a new type of attack using supervised methods?
4. Can a mechanism within the IDS that explains attack detections help a domain expert to assess the threat level and understand how the model's decisions are made?

1.5 Research Aims and Objectives

The research ***aims*** to develop a novel ML system to successfully detect unknown network attacks or attacks that have never been previously introduced to the system while maintaining a low false positive rate. To achieve this aim, the following set of objectives have been identified:

- **Objective 1**: Review and evaluate the current state-of-the-art literature on unsupervised, supervised and ML explainability in the intrusion detection domain to understand the challenges and limitations.
- **Objective 2**: Empirically investigate several unsupervised algorithms to identify their capabilities and limitations to design and develop an unsupervised ensemble

for detecting unknown network attacks.

- **Objective 3**: Examine different supervised classifiers to extend the developed unsupervised ensemble by adding a supervised component as a second stage to improve the overall detection performance of the system.
- **Objective 4**: Investigate ML explainability techniques and use them as the final stage of the model to explain the decision made by the system in a way that is understandable to the domain expert.
- **Objective 5**: Evaluate and discuss the final model performance and determine the model's effectiveness in detecting and minimising the damage of network attacks.

1.6 Research Methodology

To answer the research questions, the research will examine different unsupervised anomaly detection methods that will be evaluated on intrusion detection datasets, which will be assessed with respect to precision, recall and F1-score. Then, based on preliminary results, the models that are effective in dealing with high network flows with high performance will be implemented into an ensemble model as base learners, and an ensemble model consisting of the selected models will be developed and evaluated on the same dataset. For this, the following hypothesis is proposed:

- **Hypothesis 1:** *"Anomaly detection methods can be adapted to detect new and previously unknown network attacks as new attacks are expected to be an anomaly to the normal network flow pattern. Moreover, the detection performance can be improved by constructing an ensemble-based model consisting of anomaly detection techniques."*

Next, the study will investigate and assess the capability of supervised models to detect attacks that the system has encountered before, to serve as a 'second line of defence', thus improving the system over time. Hence, the following hypothesis is proposed:

- **Hypothesis 2:** *"Having a supervised model will assist in detecting attacks that have*

been encountered before, since these attacks become known to the system, thus improving the overall detection results."

Finally, model explainability methods will be applied to help domain experts understand the black-box ML model behaviour and its decision locally (in terms of any single prediction) and globally (in terms of the whole model). Consequently, the following hypothesis is proposed:

- **Hypothesis 3:** *"It is possible to obtain some explanation from the developed model to support domain experts in evaluating the level of threats and understanding the decisions made by the model."*

1.7 Contribution

Using ML is crucial to defend against cybercrime. This thesis empirically investigates, tests and evaluates supervised and unsupervised ML models in terms of their performance and explainability to help the fight against network attacks. Accordingly, this thesis makes the following contributions:

1. The main contribution of this thesis is a novel Network Intrusion Detection System framework, evaluated on both older (NSL-KDD) and newer (CICIDS2017) intrusion detection datasets, for detecting a wide type of network attacks.

2. Analysis of the performance of different commonly used anomaly detection algorithms such as Local Outlier Factor, Isolation Forest, One-Class SVM and Elliptic Envelope for their suitability in detecting unknown (zero-day) network attacks.

3. A novel heterogeneous unsupervised bagging ensemble which acts as the first component of the framework called **U**nknown **N**etwork **A**ttack **D**etector (**UNAD**), capable of detecting new previously unseen and unknown network attack types. UNAD comprises two well-known anomaly detection algorithms, the Local Outlier Factor and Isolation Forest, which are used as UNAD's base learners. Further, UNAD uses the weighted majority voting as a results combiner method for its base learners.

4. Analysis of the performance of different supervised algorithms, namely Random Forest, KNN, Naive Bayes and AdaBoost, as a second component in the framework to improve the detection performance of the network attacks. The empirical results showed that Random Forest outperformed the other three algorithms in boosting the detection of the attacks.

5. A utilisation of explainable Machine Learning as a third component of the framework which includes:

   (a) An adaptation of LIME as a local explainable method which aims to provide domain experts with explanations for any data instances, thus helping to understand the prediction made by the ML model for any data instance in the dataset.

   (b) An adaptation of the surrogate method as an explainable global method to provide a comprehensive explanation of the model and, within this global explainable method, a rule extractor is developed, which extracts the rules of the explainable model and generates a CSV report for domain experts in a readable and understandable way.

1.8 Publication

Part of Chapter 3 and Chapter 4 of this thesis appears in the following publication:

1. Alzubi, Saif [1], Stahl, Frederic [2] and Gaber, Mohamed Medhat [3]. 'Towards intrusion detection of previously unknown network attacks' In 35th ECMS INTERNATIONAL CONFERENCE ON MODELLING AND SIMULATION, pp. 35-41 , 2021 [24].

[1] Lead author: Conducted the experiments and wrote the manuscript.
[2] Revised and edited the manuscript.
[3] Paper final revision.

1.9 Thesis Experimental Data and Code

Thesis experimental data and research code are available at **https://github.com/salzoubi/PhD_experiments**

1.10 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2: This chapter discusses using ML in the intrusion detection field. First, it provides an overview of the IDSs types and methods. Then, it reviews anomaly detection techniques and describes some of the supervised and unsupervised learning algorithms used for classification in anomaly detection tasks. Next, it presents the measures used to evaluate IDS models' performance which will help assess the thesis's experimental results. Then, it explains the ensemble techniques and the methods used to combine the ensemble results. In addition, this chapter discusses model explainability in ML in general and presents standard preprocessing methods. Finally, this chapter discusses work related to unsupervised intrusion detection.

Chapter 3: This chapter describes the research methodology workflow and the system's three components. Further, it describes the experimental setup, the datasets used in the experiments and the preprocessing workflow for each dataset. Then, it evaluates the selected anomaly detection algorithms and presents a discussion of the preliminary experiments' results. Finally, the initial experiment summary appears at the end of this chapter.

Chapter 4: This chapter answers ***RQ1*** and ***RQ2*** of this thesis. First, it describes the proposed unsupervised ensemble learner UNAD's architecture and workflow, which acts as the first component of the system. Next, it compares UNAD's results with its standalone algorithms. It also provides an experimental evaluation of the UNAD ensemble using two results combiner methods, Majority Voting and Weighted Majority Voting. Then, this chapter compares and summarises the results of these two methods. Finally, it highlights the best methods to adopt for UNAD as a results combiner based on the given results.

Chapter 5: This chapter answers ***RQ3*** and ***RQ4*** of this thesis by presenting the second and third components of the system. First, it describes the supervised component's workflow in detail, which acts as the second component of the system. Next, it introduces the proposed classifiers used for the second component and compares and summarises their results. It also provides a further investigation of the second component's selected classifier. This chapter highlights the importance of explainability in ML—the third component of the system—and the benefits of incorporating it. Finally, it explains the two types of explainability considered in this thesis and provides examples for each type.

Chapter 6: This chapter presents the overall boosted results after implementing the second component and combining its results with UNAD. It begins by revisiting UNAD and the second component classifier results and then comparing them with the overall combined results. A further investigation of the detection rate for benign/normal and all attack types after combining the second component classifier results with UNAD results is also discussed.

Chapter 7: This chapter provides a summary and conclusion of the project. It also presents the investigated research questions and hypotheses and their outcomes. Finally, it discusses the limitations and the future direction of this thesis.

Chapter 2

Background and Literature Review

This chapter presents a comprehensive overview of the use of ML in the intrusion detection domain. It begins by explaining the taxonomy of IDS in general and presents anomaly detection methods and techniques. It also describes some supervised and unsupervised learning algorithms used for classification in the anomaly detection field. Furthermore, it reviews the standard measures used to assess ML models' performance within the intrusion detection domain to evaluate the research experimental results. Hence, they will help answer the research questions and address research hypotheses.

Additionally, it discusses ensemble techniques and their combination methods, which will be the starting point in building the unsupervised ensemble model. It also provides an overview of model explainability in ML. Furthermore, a section on dataset preprocessing methods for ML models is also included in this chapter. Finally, this chapter provides a critical analysis of some related works relevant to unsupervised intrusion detection and highlights some of their limitations.

2.1 Intrusion Detection System

Intrusion is defined as *'An unauthorised penetration of your enterprise's network, or an individual machine address in your assigned domain'* [25]. It is believed that many unreported or unnoticed intrusion cases may exist, as systems can never be completely secured [26]. Furthermore, these intrusions can originate from someone either inside or outside the network system seeking to gain unauthorised access [27]. Hence, networks must be constantly monitored to avoid intrusions, which can be achieved by implementing Intrusion Detection Systems [26, 27].

Anderson introduced the IDS concept in the 1980s [28]. Intrusion detection is a cybersecurity mechanism that aims to detect abnormal behaviour (i.e., attacks) in the host and/or network environments, requiring a quick response to stop the behaviour once detected [29]. IDSs is categorised by the type (deployment method) and the detection method used. Figure 2.1 shows the IDSs categories.

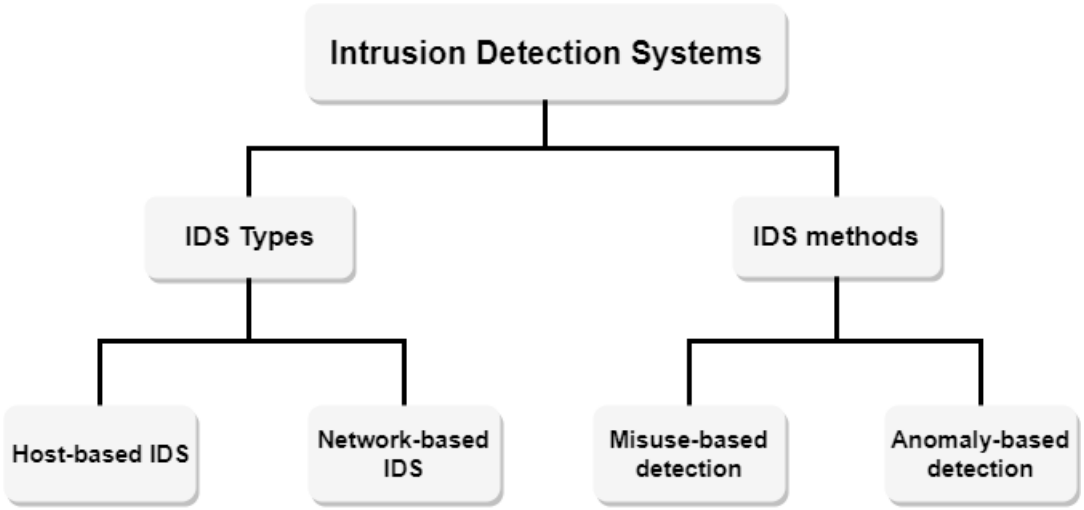


Figure 2.1: Intrusion Detection System Categories

2.1.1 Types of Intrusion Detection System

IDSs can be categorised into two types: host-based IDSs (HIDS) and network-based IDSs (NIDS), which are described as follows:

Host-Based Intrusion Detection System (HIDS)

A HIDS depends entirely on the host system itself [30]. It monitors the events running on a single host machine to detect internal intrusions (attacks) [31]. This can be done by collecting and analysing information about the host machine's activities, such as integrity of the file system, host access, system registry and system log files [31, 32]. Hence, once an abnormal behaviour or an attack incident occurs, the HIDS system should detect it [33] and promptly report it to the system administrators. However, the limitations of HIDS are that they only monitor the host that the system is installed on and consume the host machine's resources, which affects its overall performance [34].

Network-Based Intrusion Detection System (NIDS)

NIDS are intended to detect and identify potentially malicious activities such as DoS, DDoS and portscan attacks by monitoring the network streams [35]. Furthermore, NIDS monitor, capture and analyse network traffic to detect possible intrusions throughout the network [36]. The crucial factors for effective NIDS are processing and analysing the traffic promptly, with high accuracy [35], to find suspicious patterns. Nevertheless, NIDS cannot detect attacks in encrypted traffic and cannot provide full detection support under high network traffic [36].

2.1.2 Intrusion Detection System Methods

There are two intrusion detection system methods: misuse-based IDSs and anomaly-based IDSs.

Misuse-Based Intrusion Detection System Method

Misuse-based Intrusion Detection Systems use a set of predefined rules that are stored in the IDS database; each rule represents a signature of a known attack pattern [37, 38]. The IDS matches these rules with the network traffic packets to detect malicious traffic, and if a pattern matches any of the rules stored in the IDS database, the system detects it and triggers an alarm [37, 38, 39]. An example of a misuse-based IDS is the Snort Intrusion Detection System [40]. Misuse-based IDS are known for achieving low false alarm rates [41]. However, they fail to detect novel or unknown attacks since no pattern is stored about them in the system's database [42, 43]. Hence, once a new attack occurs, the domain expert/network administrator should update the system's database by adding the new attack pattern to be able to identify the attack and detect it in the future, which can be a tedious and time-consuming process [42, 43].

Anomaly-Based Intrusion Detection System Method

An anomaly in network security context is an event that is unusual [44]. Hence, anomaly detection is the identification of patterns in data that do not match normal or known behaviour [45]. This method assumes that abnormal behaviour is uncommon and different from normal behaviour [41]. In contrast to the misuse-based IDS method, the anomaly-based IDS method can detect unknown and novel attacks [46]. In the anomaly-based method, a baseline profile for the expected behaviour is created by monitoring normal and regular activities from different sources, such as network connections, hosts or users, over a period of time [36]. Then, the system compares the patterns with the created profiles [41]. If a divergence from the expected behaviour occurs, the system will detect it and trigger an alarm [41]. The drawback of this method is that it suffers from high false positive rates [47].

Based on the descriptions provided for the IDS and its types and methods in this section and aligned with the thesis research questions and objectives, this thesis will implement a Network-Based Intrusion Detection System (NIDS) because it is concerned with examining and detecting network attacks. Furthermore, this thesis will examine anomaly-based IDS methods because they include ML methods, which will be reviewed in the following section.

2.2 Anomaly Detection Methods and Techniques

Many anomaly detection techniques have been employed to detect network attacks. The most-used techniques are statistical-based anomaly detection, classification-based anomaly detection and clustering-based anomaly detection. These methods are described as follows:

1. ***Statistical-based anomaly detection:*** According to Anscombe and Guttman [48], a statistical anomaly is defined as *'an observation which is suspected of being partially or wholly irrelevant because it is not generated by the stochastic model assumed'* [48]. Therefore, statistical-based anomaly detection works by applying a statistical model to the dataset in which a statistical inference test is used to determine whether the unseen data instances are relevant to this model [45]. Hence, data instances with a low probability of being relevant to the model are deemed anomalies [45]. The test is adjusted for each distribution based on the number of expected outliers and the space in which to expect an outlier [45]. A commonly used test is the three-sigma rule ($3\sigma - rule$), in which data instances that diverge more than three times the standard deviation from the mean of the normal distribution can be counted as outliers [49].

2. ***Classification-based anomaly detection:*** This technique consists of two steps—the training step, which trains the classifier using the training data in a feature space, and the testing step, which classifies and distinguishes unseen data instances into two categories as normal or anomalous using the trained classifier [45]. The training step depends on the normal traffic activity profile, which establishes the knowledge

base and considers activities that differ from the normal traffic activity profile as anomalous [50]. The advantage of classification-based anomaly detection is that it can detect new and unseen attacks in an unsupervised way, assuming that they present divergences from the normal profile [45]. An example of supervised classification-based anomaly detection algorithms are K-Nearest neighbour (KNN) and Decision Tree (DT); examples of unsupervised algorithms include Isolation Forest (iForest) and Local Outlier Factor (LOF).

3. ***Clustering-based anomaly detection:***

   This unsupervised method places similar data instances into clusters [45]. Hence, data instances that do not belong to any clusters are considered outliers [49]. In order to detect outliers, clusters are assigned using a threshold to distinguish between inliers and outliers [45, 49]. Hence, data instances below that defined threshold are recognised as outliers [45, 49].

   As explained by Chandola *et al.* [45], clustering-based anomaly detection depends on one of three assumptions. The first is that that normal data instances fit in the cluster, while anomalies do not fit in any cluster. An example of an algorithm based on this category of assumption is DBSCAN. The second assumption is that normal data instances are located near the centre of the closest cluster, while anomalies are distant from the centre of their closest cluster. An example of an algorithm based on this category is K-Means Clustering. The third assumption is that normal data instances reside in large clusters, while anomalies reside in small or scattered clusters. An example of an algorithm based on this category is the Cluster-Based Local Outlier Factor [45].

2.3 Network Anomaly Detection Using Machine Learning Algorithms

Many of the ML algorithms used for classification tasks have been applied to Intrusion Detection Systems. This section presents the unsupervised and supervised classification ML models explored and implemented for this thesis. These models, which have been used widely in the literature, are expected to perform well in detecting network attacks. The use of unsupervised ML algorithms will help detect new and unknown network attacks which have not been introduced to the system before. Among the reviewed algorithms are LOF, iForest, Elliptic Envelope (EE) and One-Class Support Vector Machine (OCSVM). Supervised learning methods can detect previously known attacks that have been introduced to the system. Therefore, implementing the supervised algorithm will help boost the system's ability to detect network attacks once it becomes known to the system. Among the considered supervised algorithms are Decision Tree (DT), Random Forest (RF), Adaptive Boosting (AdaBoost), Naive Bayes (NB) and K-Nearest Neighbor (KNN).

2.3.1 Unsupervised Machine Learning Algorithms

1- Local Outlier Factor (LOF)

LOF detects local outliers by comparing the local density of an object to its neighbours [51]. LOF considers an object an outlier if the average of the local reachability density of that object is lower than the local reachability density of its neighbours [51]. LOF's main advantage is detecting local and neighbouring outliers to data instances in very large datasets with heterogeneous densities [52, 53].

2- Isolation Forest (iForest)

According to Liu *et al.* [54], an iForest is a tree-structured algorithm which partitions all data instances until they are fully separated. Moreover, iForest assumes that anomalies are expected to be split in early partitioning; therefore, instances with short path lengths are

expected to be anomalies. iForest provides low linear time complexity with a low memory requirement, making it ideal for detecting network attacks quickly. Furthermore, iForest can deal with high-dimensional data with unrelated attributes [54].

3- Elliptic Envelope (EE)

EE detects outliers on multivariate Gaussian distributed datasets [55]. EE creates and fits an ellipse around the centre of a group of data instances using the Minimum Covariance Determinant (MCD) [55]. Hence, any data instance outside the ellipse is considered an outlier [55].

4- One-Class Support Vector Machine (OCSVM)

OCSVM, introduced by Schölkopf *et al.*, extends the SVM algorithms. OCSVM creates a function f that returns $+1$ for normal data points and -1 for outliers [56]. The OCSVM uses kernels to map the data into a nonlinear high-dimensional feature space, thus separating them from the origin with a maximum margin boundary [56]. Furthermore, it creates a hyperplane that works as that boundary to separate normal data points $(+1)$ from anomalies (-1) [56]. OCSVM can detect outliers, but it requires a high computational complexity when dealing with large and high-dimensional datasets [57, 58, 59, 60], which makes it unsuitable for dealing with a large amount of network flows.

2.3.2 Supervised Machine Learning Algorithms

1- Decision Tree (DT)

DT is a simple but effective ML method which can be applied to classification and regression tasks [61]. DT is a tree-structured model formed of internal and leaf nodes [61]. The internal nodes represent the tests on features, while the leaf nodes represent a class label [61]. Hence, each branch of the tree depicts a possible outcome. Furthermore, the classification rules are derived from the the leaf node to the root node [61]. The most common DT algorithms are ID3, C.45 and CART. DT uses different splitting criteria, such as Information Gain, Gain Ratio, and Gini Index, which aim to achieve the best separation for a

given data in order to group similar data into smaller partitions based on the outcomes of the splitting criterion [62]. The advantages of DT are that they can deal with both nominal and numeric input features and the possibility of turning the decision trees into a set of rules, hence providing simplicity in interpreting the rules [63].

2- Random Forest (RF)

RF is an algorithm developed by Breiman [64]. According to Breiman [64], RF can be applied to classification and regression tasks. RF consists of an ensemble of the CART decision tree in which each tree is trained on a bootstrap sample of the training set with replacement using the resampling method [64]. Furthermore, on each bootstrap sample, about one-third of the training set is retained as a testing set (out-of-bag sample), which estimates the prediction error for the DT [64]. Concerning the prediction results, RF uses Majority Voting in the classification task where each tree in the ensemble represents one vote, and the class with the highest number of votes is selected as the final prediction [64]. The main advantage of RF is that it has a low probability of overfitting [64].

3- Adaptive Boosting (AdaBoost)

AdaBoost is a boosting ensemble algorithm developed by Freund and Schapire. AdaBoost uses a sequential approach to create highly accurate predictions by combining many subsequent weak learners [65]. First, it creates a set of base learners that assign an equal weight for the training data instance [66]. Next, if a training data instance is misclassified, the weight of that training data instance is increased, and the following base learner is then trained on the updated weight [65, 67]. Finally, the final prediction is determined by computing the weighted majority vote of all base learners in the ensemble [65].

4- Naive Bayes (NB)

NB is a probabilistic ML classifier based on the Bayes' theorem; it works on the assumption that the probability of one feature does not influence the probability of the others and that all features are mutually independent in a given class [68, 69]. The main advantage of NB is that it can deal with many features and datasets [70].

5- K-Nearest Neighbor (KNN)

The KNN algorithm was first introduced in the early 1950s [62]. KNN is known for being an easy-to-apply classification method [71]. The KNN selects a set of k data instances in the training set, which are the nearest to the unclassified/unlabelled data instance in the testing set [71]. The closeness of the k data instances is calculated using distance or similarity metrics such as Euclidean distance [71]. Furthermore, the optimum number of the KNN is determined experimentally, where the experiments are repeated by incrementing the value of k to add more neighbours [62]. Hence, the k value, which provides the highest results, is selected [62]. For the final results, KNN uses majority voting, where the class label with the highest number of votes between the k data instances is selected [71].

2.4 Intrusion Detection Systems Evaluation Measures

This section describes the evaluation measures used to evaluate the performance of the Network Intrusion Detection Systems (NIDS). Typically, the main performance measures used to evaluate a NIDS are the confusion matrix, accuracy, recall, precision, F1-score and Receiver Operating Characteristics (ROC) [27].

1. **Confusion matrix:** An $n \times n$ matrix used to evaluate the performance of a classifier [72]. For example, in a binary classification task, a 2×2 confusion matrix is produced (figure 2.2) to define the dispositions of the set of instances [72]. Moreover, several evaluation metrics, such as accuracy, recall, precision, F1-score and ROC-AUC, can be obtained from the confusion matrix [72].

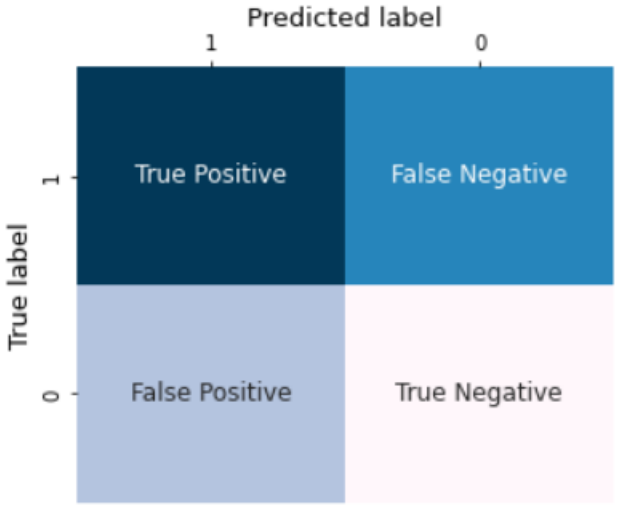


Figure 2.2: Sample Binary Confusion Matrix

A confusion matrix consists of the following measures [62] in a Network IDS evaluation:

- **True Positive (TP)**: The number of attack instances (malicious traffic) predicted (detected) correctly.
- **True Negative (TN)**: The number of normal instances predicted (detected) correctly.
- **False Positive (FP)**: The number of normal instances predicted (detected) incorrectly.
- **False Negative (FN)**: The number of attack instances (malicious traffic) predicted (detected) incorrectly.

2. **Accuracy:** Measures the percentage of data samples that are correctly classified [62]. However, the accuracy measure is unreliable when used to evaluate models based on imbalanced datasets and provides inaccurate and misleading results [73]. The accuracy metric is defined by the following equation [62]:

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

3. **Precision:** Denotes the proportion of predicted positive data instances that are indeed positive data instances [74]. The precision metric is defined by the following equation [62]:

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

4. **Recall (Sensitivity/Detection Rate):** Denotes the proportion of positive data instances that are predicted correctly [74, 75]. The recall metric is defined by the following equation [62]:

$$recall = \frac{TP}{TP + FN} \tag{2.3}$$

   Note: in this thesis, the term Detection Rate will be used to indicate the number of correctly detected attacks (*TP*) to the total number of that attack type (*TP + FN*).

5. **F1-score:**: Denotes the harmonic mean of ***recall*** and ***precision*** [62]. The F1-score metric is defined by the following equation [62]:

$$F_1 = 2 \times \frac{recall \times precision}{recall + precision} = \frac{2TP}{2TP + FP + FN} \tag{2.4}$$

6. **Receiver Operating Characteristic (ROC) Curve:** A two-dimensional graph used to plot and choose classifiers based on their performance [72]. The ROC curve plots the trade-off between the false alarm rates (False Positive Rate) on the x-axis and the True Positive Rate (Recall) on the y-axis of the classifier at a different threshold values [72]. A perfect ROC curve has a 0% false alarm rate while having a 100% True Positive Rate [76]. However, this is hard to achieve in a real-world scenario; thus, literature tends to calculate the True Positive Rate for several false alarm rates and plot the results on the ROC curve graph [76].

7. **Area Under the Curve (AUC):** A classification performance measure [77] commonly used with the ROC measure and referred to as the Area Under the ROC Curve (ROC-AUC) [72, 78]. AUC calculates the percentage of accurately ranked pairs of

positive and negative data samples [79]. AUC is interpreted as the higher the AUC, the more effective the classifier [80]. For instance, a classifier with an AUC of 1 depicts a perfect and correct classification, while an AUC near 0.5 indicates a classifier with a random classification [80].

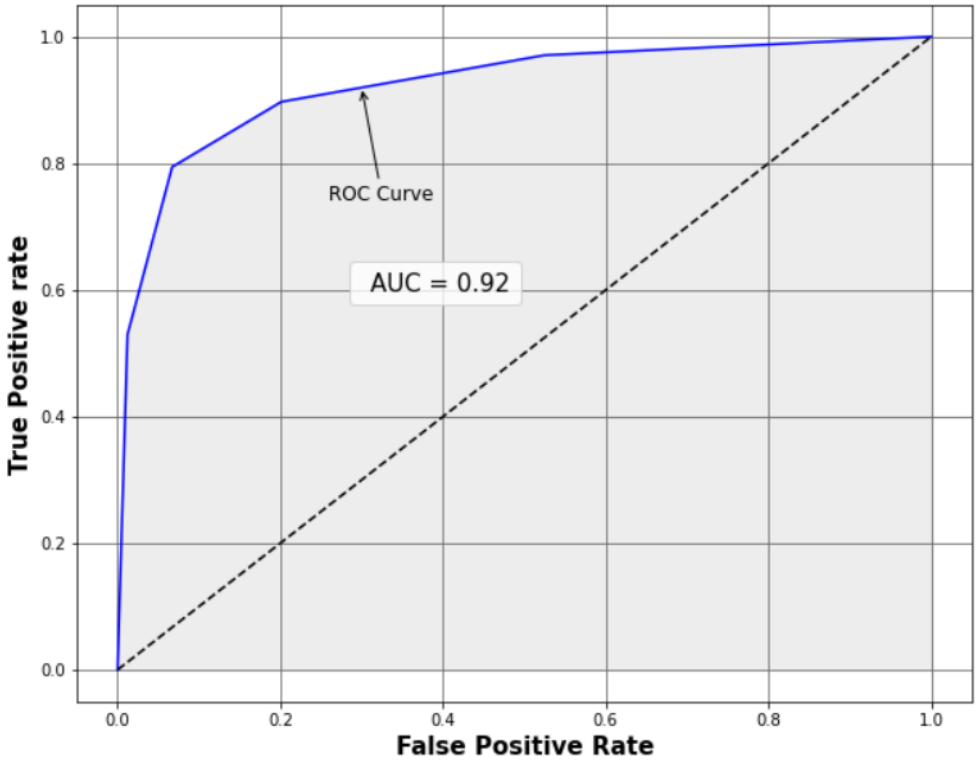


Figure 2.3: ROC-AUC Example on Toy Dataset

Figure 2.3 illustrates ROC-AUC results on a toy dataset. In this figure, the x-axis represents the FPR and the y-axis represents the TPR. The blue line represents the ROC curve, while AUC is the entire area underneath the ROC line shaded in light grey. The result shows that the classifier has reached an ROC-AUC of 92%, meaning that the classifier is nearly perfect in distinguishing between positive and negative data instances.

2.5 Ensemble Methods

Ensemble methods combine a set of multiple weak classifiers, known as base learners, to improve the performance and reach better results over the ensemble's base learners [81]. Therefore, applying the ensemble methods in the system is expected to help overcome any shortcomings in the anomaly detection algorithms and provide stronger results than these algorithms, once they are implemented as base learners for the ensemble.

Ensemble methods are categorised into two main types: homogeneous ensemble, which is using the same algorithm to produce different base learners, and heterogeneous ensemble, which is using different base learners from different algorithms [81]. The most popular ensemble methods are bagging, boosting and stacked generalisation.

2.5.1 Bagging

The bagging method, first introduced by Breiman [82], is an abbreviation of 'bootstrap aggregating'. According to Breiman [82], the bagging method produces multiple base learners where each base learner is trained on a random sample of the training set with replacement 'bootstrap sampling'. Hence, the probability that any data instance in the training set will occur in the sampling at least once is $1 - (1/e) \simeq 0.632$ for each base learner [82]. However, some samples may not appear in the sample [82]. The main advantages of bagging are the reduction of error generalisation and the improvement of performance while reducing unstable learners' variance [81].

2.5.2 Boosting

Boosting is a sequential ensemble method that aims to transform weak learners into strong learners [81]. Weak base learners are created and trained sequentially; the subsequent base learners focus more on the mistakes of the previous base learners and correct them by giving the mistakes a higher weight [81, 83]. Therefore, the base learners become dependent on each other. The main advantage of boosting is reducing the bias of the weak learners [84]. An example of a boosting algorithm is AdaBoost [83].

2.5.3 Stacked Generalisation (Stacking)

Wolpert proposed stacking in 1992. Stacking is commonly made up of two levels; the first level comprises different algorithms, trained on the entire training set, in which the output (prediction) of the first level algorithms serves as input features for the second level [81, 85]. The second level, commonly known as the meta-learner, is a combiner algorithm that makes the final prediction based on the first level's predictions [81, 85]. The main advantage of stacking is reducing error generalisation [81].

2.6 Ensemble Results Combiner Methods

Once the base learners make their predictions, they are aggregated for the final results. The aggregation method depends on the type of problem. For example, voting methods are used for classification problems, while averaging methods are used for regression problems [81]. These methods are described by Zhou [81] as follows:

2.6.1 Voting

1. **Majority Voting:** Every base learner provides an equal vote for one class label, and the final prediction is made for the class label with more than half of the votes [81].

2. **Plurality Voting:** Each base learner in the ensemble provides one vote for the class label; unlike majority voting, which requires the class label to have more than half of the votes to be selected as the final prediction, plurality voting selects the class label with the most votes as the final ensemble prediction. [81].

3. **Weighted Voting:** In this method, a higher weight value is given to the base learners that provide better performance; thus, these will have more weight in the voting [81].

4. **Weighted Majority Voting:** In this method, a weight is assigned to each base learner, obtained from the prediction performance for each base learner in the ensemble [86]. Then, the total weight of the base learners is combined and calculated for each class, and the class with the highest weight is selected as the final result

[86, 87].

5. **Soft Voting:** This method calculates the probabilities of the class labels; the final prediction is made for the class with the highest sum probability [81].

2.6.2 Averaging

1. **Simple Averaging:** This method calculates the average of the predictions from individual learners. Therefore, the final prediction is made for the class with the highest sum average [81].

2. **Weighted Averaging:** This method calculates the weighted average of the predictions from individual learners, where each base learner has different weights based on their performance. Hence, base learners with better performance contribute a higher weight proportion [81].

2.7 Model Explainability in Machine Learning

Explainability and interpretability are often used interchangeably [88, 89, 90]. In the context of ML, explainability is defined thusly: *'Given a certain audience, explainability refers to the details and reasons a model gives to make its functioning clear or easy to understand'* [91]. Hence, explainability aims to answer questions in the form of what-questions, how-questions and why-questions [92]. Consequently, to be considered useful, models are expected to provide the reasoning process, results and recommendations behind their decisions. Therefore, an explainable model can support domain experts and system developers in understanding its decisions and reasoning process, allowing them to adjust the model accordingly [93]. Explainability goals are described by Arrieta *et al.* [91] as follows:

1. **Trustworthiness:** Confidence that the model will perform as expected when dealing with a problem.

2. **Interactivity:** Provide the end-user with the capability to adjust and interact with the models to ensure success.

3. **Accessibility:** Allow end-users to get more engaged in improving and building the model, making it easier for non-experts to deal with models.

4. **Informativeness:** Explainable models provide information about the problem being addressed to link the user's decision to the answer provided by the model.

2.7.1 Machine Learning Explainability Criteria

Explainability in ML can be categorised by model, method and scope [88].

1. **Explainabilty by model:**

   - **Intrinsic Explainabilty**: A self-explanatory ML model that can provide explainability as part of its structure [88, 94].

   - **Post-hoc Explainabilty**: Helps explain models that are not explainable by design by applying explanation methods after the training of the model [91, 95].

2. **Explainabilty by method:**

   - **Model-agnostic**: A post-hoc method that can be used on any model after training. However, the model-agnostic method cannot access model internals such as weights or structural information [88].

   - **Model-specific**: An intrinsic method that is restricted to a certain ML model and cannot be applied to any other models. For instance, the explanation of the weighted linear regression model is model-specific and cannot be performed on any other model [88].

3. **Explainabilty by scope:**

   - **Local Explainability**: Explains individual/single prediction. Thus, it helps understand how the model made the decision (prediction) for any single prediction [96].

   - **Global Explainability**: Explains the entire model, therefore, it helps understanding of the overall reasoning of the model [96].

2.7.2 Machine Learning Explainability Techniques

The most common techniques used to explain a ML model are as follows:

- **Decision Trees (DT)**: A tree-structured model formed of internal and leaf nodes [61]. The internal nodes represent the tests on features, while the leaf nodes represent a class label [61]. Hence, each branch of the tree depicts a possible outcome [61]. Furthermore, the classification rules are derived from the leaf node to the root node. The most common techniques used in DT are C4.5 and ID3 [61].
- **Rule-based**: Creates rules to identify data that the model will learn from. In rule-based, the knowledge is stored as simple rules in the IF-THEN form or as a set of more complicated simple rules [91]. Common rule-based techniques are RIPPER and CN2 [97].
- **Partial Dependence Plot (PDP)**: Helps visualise the relationship between the reduced input features space and the model's predicted outcome [96].
- **Features Importance (FI)**: A simple method that returns the weight and importance of the features used in the model as an explanation [96]. The feature importance is calculated using the weights of the coefficients, which define the association between the features and the target of linear models used as explainable models [96].
- **Linear and Logistic Regression classifiers**: The linear regression model is used to understand the relation between independent and dependent features, in which the dependent variables are considered a continuous outcome [98]. The logistic regression model, by contrast, predicts a binary dependent feature; hence, the result is only one outcome variable [91].
- **Global Surrogate method**: A model-agnostic method that does not require any knowledge of how the model works internally [99]. It explains the prediction of a complex model's 'black-box' using uncomplicated self-explanatory models such as decision trees [99, 100].

2.8 Dataset Preparation (Preprocessing)

Data preprocessing is a vital phase to be performed before feeding the data into models and has a significant effect on the accuracy and performance of the ML model [101]. The dataset preprocessing methods considered in this research are as follows:

1. **Data Cleaning:** It consists of handling missing, *NaN* and redundant values by either dropping the instances or using statistical methods to maintain data quality and avoid biased results [62, 102].

2. **Categorical Encoding:** It consists of converting categorical columns to numeric form so the model can understand and interpret these inputs [102].

3. **Training, Validation, and Test Split step:** To evaluate the performance of the models and how they perform on new data, performance assessment methods should be completed [103]. However, these methods can also ensure that the models are not overfitting, which occurs when the classifier is very customised to the training set and fails to generalise to new data from the same environment [103]. Moreover, these methods help in the model selection by tuning the model's hyperparameters, which is a set of parameters that need to be tuned before running [104]. The most common performance assessment methods are Leave-one-out Cross-Validation, k-Fold Cross-Validation and Hold-out Cross-Validation.

   (a) *Leave One Out Cross-validation*: Every data instance in the dataset works as a validation set ($k = 1$) [105]. Hence, the first validation set is the first data instance and the second validation set is the second data instance, and so on [105].

   (b) *k-fold Cross-validation*: A dataset is first randomly split into k disjoint sets with nearly the same size of data instances [106]. Then, it is trained using $k-1$ and evaluated on the remaining set, which is known as the validation set. This method is repeated k times until each set has worked as a validation set [106]. The final model performance is calculated by taking the overall mean for all the

k validation sets [105, 106].

(c) *Hold-out validation*: A simple cross-validation method with a single split of data (2-fold cross-validation) [107]. The dataset is split into two sets, the training set and the testing set (the hold-out set) [107]. A popular method in the hold-out validation is dividing the original dataset into training, validation (commonly known as development set) and test sets [108]. The validation set will be used to assess the generalisation performance of the ML algorithm and ensure that the model is not overfitting on the test set [108].

4. **Data Transformation (feature scaling):** In this step, data normalisation and data standardisation are standard methods which are described as follows:

   (a) *Data normalisation (MinMax scaling)*: Features are transformed between a range of [0,1] where the minimum feature value is 0 and the maximum feature value is 1 [109]. MinMax scaling is defined by the following equation [109]:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2.5}$$

   (b) *Data standardisation*: Features are transformed to have a mean of zero and a standard deviation of 1; therefore, features will have a standard normal distribution (Gaussian distribution). Standardisation is defined by the following equation [109]:

$$X_{scaled} = \frac{X - X_{mean}}{X_{sd}} \tag{2.6}$$

5. **Dimensionality Reduction:** A technique used to lower the number of input variables in a dataset [110]. Hence, help mitigate the curse of dimensionality problem, which occurs when the ML model is not performing well due to dealing with a high number of features (high dimensional spaces) [111].The most popular unsupervised methods are Principal Component Analysis (PCA) and Independent Component Analysis

(ICA), where the input feature space is converted into a lower-dimensional subspace that keeps most of the relevant information [110].

(a) ***Principal Component Analysis (PCA):*** An unsupervised multivariate statistical technique that transforms large feature spaces into a smaller unrelated subspace by discovering a few orthogonal linear groups of the original features with the greatest variance [112, 113]. One of PCA's main advantages is that it increases interpretability, yet it minimises the information loss simultaneously by generating new uncorrelated features [112, 113, 114].

PCA works by converting n correlated random features into $d \leq n$ uncorrelated features [114]. Therefore, the new generated features are a linear reduced form of the original features [114]. The first transformed component is the reduced form of the original feature with the greatest variance [113]. Therefore the first principal component is projected in the direction where the projection's variance is maximised, and the second principal component is the reduced form of the original feature with the second-largest variance, and orthogonal to the first principal component [113].

(b) ***Independent Component Analysis (ICA):*** This method aims to obtain a linear representation from non-normally distributed data to reduce the statistical dependence between components [115, 116]. Hence, ICA assumes that the data are linearly mixed by a group of isolated independent sources; therefore, breaking up (unmixing) these sources is based on their statistical independency calculated by the sources' shared information [117]. ICA model is defined by the following equation 2.7:

$$x = As \tag{2.7}$$

Where:

- x : Denotes the observed features

- A : Denotes the mixing matrix
- s : Denotes the independent components

6. **Feature Selection:** A method used to reduce the number of features by choosing a subset of relevant features that can efficiently represent the input data while minimising the impact of noise and redundant and irrelevant features to improve the ML model performance [118, 119]. In addition, feature selection helps create a more generalisable prediction ML model, lowers computational costs, and reduces the required storage [120]. Feature selection is categorised under three main methods: *Filter*, *Wrapper* and *Embedded* methods [121].

   (a) *Filter method*: This method selects a subset of features independently from the classification model by ordering the features using some criteria such as distance, dependency, information and correlation [120]. Once the features are ranked, a threshold is used to filter out less relevant features that do not help determine the classes, while leaving only features that hold helpful information in predicting these classes [119]. Examples of the filter methods algorithms are Fisher score, correlation coefficient and Information Gain based methods [118, 119, 120].

   (b) *Wrapper method*: In this method, the selection of the features depends on the performance of the chosen ML algorithms [121]. The selected ML model is wrapped around a search algorithm (hence the name) to find the best subset of features that produces the highest performance results [119]. A wrapper algorithm will first create a set of features and then evaluate those features using the chosen ML model [119, 120, 121]. This process is repeated until either the optimal performance is obtained from the model or the predefined number of features is achieved [119, 120, 121]. Examples of the wrapper search algorithm are forward selection and backward elimination [118].

   (c) *Embedded*: The embedded method combines the filter and wrapper methods. It aims to include feature selection as part of the ML model training [121].

According to Tang *et al.* [120] embedded methods have three different types [120]:

i. The pruning method: This method begins by training the model on all features and then prunes the least important features returned by the model by assigning the corresponding coefficients to zero. This method works recursively so that less important features are eliminated in each iteration until finally reaching the number of desired features. An example of this method is Recursive Feature Elimination (RFE).

ii. ML algorithms with their own feature selection mechanisms, such as ID3 and C4.5 decision trees.

iii. Regularisation ML algorithms such as Ridge and Lasso which aim to minimise the model's fitting errors and force the feature coefficients to be small or exactly zero.

7. **Handling Imbalanced Datasets:** A class imbalance occurs when the number of instances in one class is significantly more than in other classes [122]. Imbalanced datasets are a common issue in the intrusion detection domain [123], where in this case, the number of normal classes significantly overcomes the number of benign/attack classes. Class imbalance can cause the classification model to be biased in favour of the majority class [124]. Hence, some evaluation metrics, such as accuracy, will become impractical for evaluating the performance of the model [124]. To overcome this challenge, the data instances in the training set should be resampled. This can be done using two techniques: *oversampling* the minority class by increasing the number of data instances that belong to the minority class to correspond to the number of data instances that belong to the majority class, or *undersampling* the majority class by reducing the number of instances that belong to the majority class to correspond to the number of data instances that belong to the minority class [125].

2.9 Related Work

Supervised ML methods for IDS commonly achieve high accuracy, recall and precision, such as [126, 127, 128]. Nevertheless, these methods are not suitable for detecting unknown attack types. Accordingly, unsupervised ML methods are examined.

This section discusses some of the relevant work on detecting intrusion using unsupervised ML by presenting the methods and techniques used to detect these intrusions and providing a critical analysis of these works. Furthermore, a summary table of the reviewed literature is provided at the end of this section.

2.9.1 Unsupervised Anomaly Algorithms for Intrusion Detection

Labonne *et al.* [129] propose an IDS framework that consists of five neural networks: deep autoencoders, deep MLPs, LSTMs, BiLSTMs, and GANs. After the preprocessing stage, the algorithms were trained on the protocol headers in an unsupervised way. The method was evaluated on 11 attack types on the CICIDS2017 Dataset. Furthermore, the proposed IDS had the capability of adding or removing protocols based on the monitored network. However, the authors mentioned that the training time for the algorithms was very high, taking up to 14 hours to train a single protocol. Furthermore, the method failed to detect four of the attack types.

Lee *et al.* [130] propose a method that analyses traffic to identify whether the network traffic is benign or attack. Their approach used Deep Sparse AutoEncoder (DSAE) as an unsupervised dimensionality reduction method, assuming that since DSAE compresses images without reducing the number of features, it can reduce the number of features without any loss. Once the features were reduced, the RF algorithm was used to classify the network flow. The method was evaluated on the CICIDS2017 dataset. The authors claimed that their approach improved the training time and classification by 99%. However, the recall and F1-score for the rare classes, such as infiltration and heartbleed attacks, were low.

Mhamdi *et al.* [131] propose a hybrid unsupervised method to detect DDoS attacks in the Software Defined Networking (SDN) paradigm. For their approach, they used stack autoencoder and OCSVM. Like Lee *et al.* [130], they used SAE as a feature extraction method to reduce the number of features. Afterwards, they trained the OCSVM to detect DDoS attacks. Their approach was evaluated on the CICIDS2017, for which they were able to obtain good results. However, OCSVM has a high computational complexity [57, 58, 59]; therefore, it requires a long training time. Furthermore, the method proposed was evaluated on only one network attack type. In addition, the authors mention that they had a high false-positive rate, but no number was provided in the paper.

Nguyen *et al.* [132] propose a nested OCSVM model to detect network attacks. As a part of their method, the authors trained their model only on normal flow, where they calculated the nearest and farthest neighbour's distances from each data sample in the training set. The method was evaluated on the KDD99 dataset, where it was able to detect attacks with a low false-positive rate. However, the KDD99 suffers from a significant number of redundant records; around 78% and 75% of the records are duplicated in the training and testing set [5]. In addition, Nguyen *et al.* did not mention any preprocessing step for the dataset. Moreover, only 4601 data instances were used in the experiment, which does not represent real-world traffic reflecting today's networks and attacks.

Chou and Wang [133] propose an adaptive IDS for the cloud environment formed of three components: the preprocessor, which converts raw packets into connection records including features; the analyzer, which implements and trains the spectral clustering algorithm to adjust the IDS to the current environment by clustering the collected connection records; and the detector, which makes the decisions and reports based on a decision tree. The adaptive IDS was evaluated on the DARPA 2000 and the KDDCup 1999 data set. The authors state that the adaptation method gave the IDS a high detection rate while keeping the false positive rate low. However, the method assumes that attacks consist of relatively little traffic. Hence, DoS and DDoS attacks cannot be detected as they comprise significant traffic. Furthermore, network connection data are generated in enormous volume, making it time and space-intensive for the spectral clustering algorithm to compute the eigenvalues

and eigenvectors.

Zhang and Zulkernine [134] propose an unsupervised anomaly IDS framework that uses the RF's proximities to detect outliers by calculating the closeness of each network service in the dataset. Hence, if a service activity is categorised as another service, it will be detected as an outlier. The protocols used in the experiment are FTP, HTTP, POP, SMTP, and telnet from the KDD99 dataset. The results showed a high detection rate with a low false positive rate. However, the authors mention that they reduced the number of normal attacks in the training set from 4,898,431 to 47,426 without explanation for using this number. Furthermore, the performance of the system decreases if the number of attacks increases in the flow. Hence, like Chou and Wang *et al.* [133], this method will fail in detecting attacks with high number of flows, such as DDoS attacks.

Leung and Leckie [135] propose a hybrid unsupervised anomaly detection framework consisting of a density-based and grid-based high dimensional clustering algorithm for large datasets; the proposed clustering method was derived from an algorithm called CLIQUE [136]. To evaluate the algorithm, the authors conducted a complexity analysis, and the results showed that it scales linearly with the number of data instances in the dataset. Furthermore, the KDD99 dataset was used to evaluate the performance of the method. The results showed that the method achieved a good detection rate while maintaining a low false positive rate. However, the authors state that their approach suffered from a high false positive rate compared to other clustering methods.

Pu *et al.* [137] propose an unsupervised anomaly detection method consisting of Sub-Space Clustering (SSC) and OCSVM. Their approach begins by setting an empty vector, dividing the feature into subspaces, then applying the OCSVM to each subspace to produce a partition. The vector in this approach is used to store the distance between the different outliers detected in the sub-space, after which the dissimilarity vector is updated depending on each partition. Finally, the vector will order the detected outliers. Hence, the data sample is considered an anomaly if the dissimilarity value is higher than the predefined threshold value. To evaluate their approach, they used the NSL-KDD99 dataset; the authors claim they obtained a high detection rate and low false alarm rate. However, the model was

trained and optimised on data that contains attack data instances, but OCSVM should be trained only on normal data instances.

Zhang *et al.* [138] propose an IDS based on OCSVM to detect network intrusions. Their model consisted of two modules: the first extracted the features and split the training and testing dataset, and the second trained and tested the OCSVM algorithm. The authors used the KDD99 dataset to evaluate their model, and the results showed a high detection rate for some of the attacks. However, the authors did not provide any details about the feature extraction method used in the model; moreover, they used stratified random sampling to reduce the dataset's size without justification for reducing the size.

Bezerra *et al.* [139] propose a host-based approach, called IoTDS, to detect Botnets in IoT devices using anomaly detection algorithms, namely EE, Isolation Forest, LOF and OCSVM. Furthermore, an HTTPS-based agent manager is used to prevent the device from being overwhelmed by the training activities. To analyse the device's behaviour, the authors extract several features from the device's such as CPU utilisation and temperature, memory consumption and the number of running tasks. If the device's behaviour displays an irregularity, a notification of a Botnet attack is sent to the central server. This approach was evaluated on a dataset that comprises data obtained from infected IoT devices. The results showed a good overall prediction performance, in which the LOF algorithm had the best performance among all the other three algorithms. Nevertheless, this approach is limited to only one attack type (botnet). In addition, a host-based system is difficult to manage and must be installed on every IoT device; thus, it can monitor only the device on which it is installed on.

Guven *et al.* [140] propose a multiple classification cyber attack framework. First, the authors compared LOF, iForest and OCSVM in detecting outliers as part of the data cleaning, followed by feature normalisation and feature selection. Next, four supervised classifiers, Random Forest, Naive Bayes, Logistic Regression and Decision Tree, were evaluated on the CICIDS2017 after removing the outliers. The results showed that using LOF for detecting outliers with Random Forest provided the highest results.

2.9.2 Unsupervised Anomaly Algorithms for Outlier Detection

Sun *et al.* [141] propose a method to detect abnormal user behaviour on payroll access logs using iForest. In their workflow, a parser preprocesses all log files and stores records by users. Next, the system extracts the features for each user, and a baseline user model is produced by generating a set of an extended iForest tree. Hence, once a new user is added to the system, it is mapped into every iForest tree, and the anomaly score is computed. If the score falls below the designated threshold, the user will be considered normal; otherwise, the user is counted as an anomaly. The proposed system was evaluated on a real payroll access log, on which they achieved a high recall. However, precision and accuracy were relatively low for all experiments.

Xu *et al.* [142] propose a method to automatically tune the LOF hyperparameters. They aimed to apply this method to anomaly detection applications in general. Thus, the authors evaluated the method on various datasets from the anomaly detection domain. Concerning the intrusion detection dataset used in the experiments, the authors chose the KDD99 dataset, focusing only on the SMTP and HTTP protocols. The experimental results showed that both protocols achieved a relatively low F1-score. Moreover, the authors state that for the method to be useful, it should be used with the assumption that there be enough normal data instances in the training set and the anomalous data instances can be distinguished from the normal data instances based on their relative local density.

Liu *et al.* [143] propose a two-layer ensemble method for outlier detection comprising LOF and iForest. The ensemble works in sequential order and includes three primary steps. First, iForest is used to calculate the anomaly score for the data points. Then, prune data points based on the pruning threshold to acquire the outlier candidate set. Finally, calculate the LOF value for the acquired outlier candidate set and select the points with high LOF values as the target outliers. The ensemble was evaluated on six synthetics and six real-world datasets, in which the results showed that the system is effective in detecting outliers.

Sahu *et al.* [144] propose an ensemble-based outlier detection comprising LOF, iForest, and OCSVM methods and the majority voting method as a results combiner. The purpose of this ensemble was to be used as a preprocessing step to remove outliers from datasets before training the supervised classifiers. The proposed ensemble was evaluated on the NSL-KDD dataset, in which the results of the supervised classifiers improved after removing the outliers from the dataset.

This section provided a critical analysis of some of the literature related to unsupervised IDS and evaluated their limitations. For example, some researchers used an algorithm known for having a high computational complexity, while others assessed their method on a low number of data instances; the reason for this could be to speed up the experiments or having limited computational resources that can not handle training a high number of data instances. However, the low proportion of data instances does not represent today's network standard. Furthermore, some literature included a limited number of attack types in their experiment, achieved low precision, recall and/or F1-scores, or used a dataset with high duplicate records, which can affect the results. Table 2.1 summarises the literature discussed in this section including the dataset used for the evaluation such as CICIDS2017 which is described in section 3.3.1 and NSL-KDD which is described in section 3.3.2.

Table 2.1: Summary of Literature on IDS

No.	Authors	Year	Technique(s)	Dataset	Problem\Limitation	Attack type(s)
Unsupervised Anomaly Algorithms for Intrusion Detection						
1	Labonne *et al.* [129]	2020	• Deep Autoencoders • Deep MLP • LSTM • BiLSTM • GAN	CICIDS2017	• Very high training time. • Failed to detect 4 attacks. • Explainability was not included.	11 Attacks
2	Lee *et al.* [130]	2020	• Deep Sparse Autoencoders (DSAE) • Random Forest	CICIDS2017	• Low Recall and F1-Score for rare class attacks. • Explainability was not included.	12 Attacks
3	Mhamdi *et al.* [131]	2020	• Stack Auto Autoencoders • OCSVM	CICIDS2017	• OCSVM has a high computational complexity. • High False positive rate • Explainability was not included.	DDoS
4	Nguyen *et al.* [132]	2018	• OCSVM	KDD99	• Dataset with high duplicate records • OCSVM has a high computational complexity. • Preprocessing steps not mentioned. • Tested on very low data instances. • Explainability was not included.	4 Attacks
5	Chou and Wang [133]	2015	• Spectral Clustering • Decision Tree	DARPA2000 KDD99	• DoS and DDoS attack are not detected. • Time and space consuming to compute the eigenvalues and eigenvectors. • Explainability was not included.	4 Attacks
6	Zhang and Zulkernine [134]	2006	• Random forests' proximities	KDD99	• Dataset with high duplicate records • The performance decreases if the number of attacks increases in the flow. • Explainability was not included.	4 Attacks
7	Leung and Leckie [135]	2005	• Clustering Algorithm	KDD99	• Dataset with high duplicate records • High False positive rate • Explainability was not included.	4 Attacks

8	Pu *et al.* [137]	2021	• Sub-Space Clustering • OCSVM	NSL-KDD99	• Trained on data that contains attacks. Though OCSVM should be trained on normal data only. • OCSVM has a high computational complexity. • Explainability was not included.	4 Attacks
9	Zhang *et al.* [138]	2015	• OCSVM	KDD99	• Dataset with high duplicate records • OCSVM has a high computational complexity. • Explainability was not included.	4 Attacks
10	Bezerra *et al.* [139]	2019	• Elliptic Envelope • LOF • iForest • OCSVM	Infected IoT devices	• Limited to one attack type only • OCSVM has a high computational complexity. • Host based system which need to be installed on every machine. • Explainability was not included.	Botnets
11	Guven *et al.* [140]	2022	• LOF • iForest • OCSVM	CICIDS2017	• The anomaly detection methods were used to remove outliers from a dataset and then used supervised learning to detect network attacks. • Explainability was not included.	14 attacks
			Unsupervised Anomaly Algorithms for Outlier Detection			
12	Sun *et al.* [141]	2016	• iForest	Payroll access logs	• Low precision and accuracy • Explainability was not included.	detecting anomalies in user behaviour
13	Xu *et al.* [142]	2019	• LOF	KDD99	• Dataset with high duplicate records • Only on SMTP and HTTP protocol • Works on the assumption that data are well sampled in the training set. • Explainability was not included.	4 Attacks
14	Liu *et al.* [143]	2019	• Ensemble in a sequential order (iForest then LOF)	six synthetics and six real-world datasets	• The ensemble was used as a preprocessing method to remove outliers from a dataset. • The SMTP dataset is a subset of the KDD99 dataset and is known for a high number of duplicates • Explainability was not included.	SMTP dataset (related to Intrusion detection)
15	Sahu *et al.* [144]	2019	• Ensemble (LOF, iForest and OCSVM)	NSL-KDD	• The ensemble was used as a preprocessing method to remove outliers from a dataset, and then supervised classifiers were applied. • Explainability was not included.	4 attacks

2.10 Chapter Summary

This chapter has presented an overview of the use of ML in the intrusion detection domain. First, it reviewed the taxonomy of IDS, then it explained some of the supervised and unsupervised learning algorithms used for classification tasks. It also reviewed the standard measures used to evaluate ML models' performance, which will be used to evaluate the research experimental results. This chapter also addressed ensemble techniques and their combination methods, as well as ML explainability and dataset preprocessing methods used in this research.

Finally, this chapter addressed several limitations in some of the relevant literature related to IDS, such as using an algorithm known for having a high computational complexity, using a low number of data instances in the experiment, testing the approach on a limited number of attack types or models that produce low precision, recall or F1-scores. Understanding these limitations will provide the basis for designing the research methodology workflow, which will help answer this thesis's research questions and address the hypotheses.

Chapter 3

Preliminaries: Experiments and Evaluation

This chapter introduces the research methodology workflow at both a high and low level. First, it explains the preliminary experimental setup of the anomaly detection algorithms. It also describes the datasets used in the experiments and the preprocessing workflow applied to each dataset. Then, it examines and explains the unsupervised algorithms used in this chapter. Next, this chapter discusses and evaluates the preliminary experiments' results for the algorithms and presents the best-performing algorithms. Finally, this chapter summarises the initial experiment, including the best hyperparameter values and principal components used in each dataset.

3.1 Research Methodology Workflow

To answer the research questions and address the thesis's hypotheses, a workflow has been designed. Figure 3.1 shows the workflow steps that were followed to design the system.

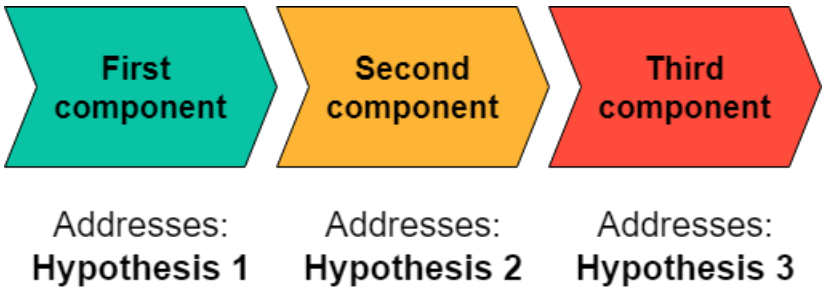


Figure 3.1: Research Methodology Workflow

The first component, termed *'C1'*, addresses Hypothesis 1 of the research. First, an unsupervised ensemble learner component, UNAD, is developed and tested to detect unknown network attacks. The second component, termed *'C2'*, which addresses Hypothesis 2, consists of a supervised component trained on the correctly detected data instances from the first component, aiming to improve the overall detection of the system. Lastly, the third component, termed *'C3'*, which addresses Hypothesis 3, explains the detected attacks to help domain experts assess the threat and understand the decision made by the system. Figure 3.2 explains the research methodology workflow in more detail.

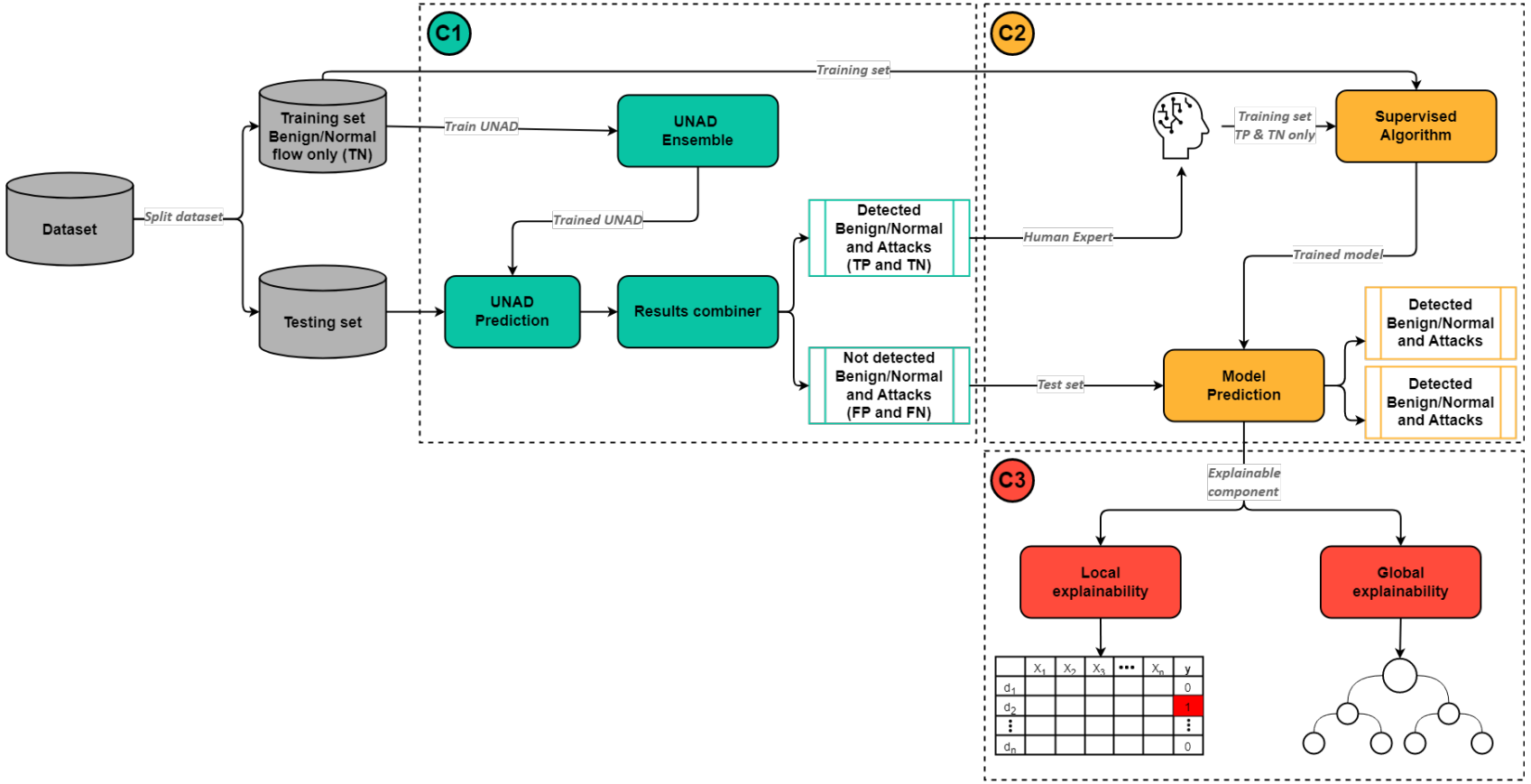


Figure 3.2: Taxonomy of the System's Framework

$C1$ represents the first component of the system, which consists of the UNAD ensemble. UNAD is a bagging ensemble, built using anomaly detection algorithms, that works in an unsupervised manner to detect previously unknown network attacks. The anomaly detection algorithms are selected as base learners for UNAD based on their F1-score results, since the F1-score measure combines the results (harmonic mean) of both the precision and recall measures. In addition, the F1-score is effective in the case of data imbalance [145]. Furthermore, UNAD will be trained on benign/normal flow only and evaluated using the Majority Voting method as a results combiner.

$C2$, the second component of the system, is the supervised classifier. The goal of this component is to boost the overall results and improve the detection rate. For this, the second component will be trained on UNAD's detected benign/normal and attack flow (*TP* and *TN*) in addition to the training set used to train UNAD, which has only benign/normal flow. Furthermore, before feeding the *TP* and *TN* of UNAD to the supervised model, a domain expert with knowledge of networks and ML techniques will act as 'Human-in-the-Loop' to check and evaluate a subsample of UNAD's results, ensuring that the supervised algorithm is fed with accurate data and therefore reducing the possibility of error. As with UNAD, the selection of the supervised model will be based on evaluating a set of supervised algorithms in which the model with the highest F1-score will be implemented in the system.

The last component of the system is the explainable component, $C3$. This component aims to explain the decision made by the model in a human-understandable way. For this, two explainable methods—local and global explainability—are adopted. The local explainability feature will provide an explanation for the domain expert for any single prediction made by the system, and the global explainability feature will explain the entire model.

3.2 Experimental Setup

In this thesis, all experiments were implemented in Python 3.6 using Google Colaboratory [146]. Furthermore, the scikit-learn library [147] was used to carry out the research

preprocessing steps and to implement the ML algorithms.

This research first evaluates several unsupervised algorithms for their suitability to be selected as base learners for UNAD. Four different anomaly detection algorithms that previously applied to network attack detection literature were considered: OCSVM [148], iForest [149], LOF [51] and EE [55]. The OCSVM algorithm was excluded from the selection process early, as it is unsuitable for fast network flows due to its high computational complexity [57, 58, 59].

The remaining three algorithms were experimentally optimised on the CICIDS2017 [4] and NSL-KDD [5] datasets and subsequently evaluated for their inclusion in the UNAD ensemble. The reason for choosing an ensemble approach here is that, as pointed out in **Section 2.5**, ensemble approaches tend to improve the average classification accuracy over any ensemble member and reduce overfitting [150].

As mentioned in **Section 3.1**, the evaluation metrics used to evaluate the anomaly detection algorithms are precision, recall and F1-score. In UNAD, precision denotes the proportion of attacks correctly classified as an attack by UNAD; recall denotes the proportion of actual attacks detected by UNAD in the network flow. Therefore, high precision is as important high recall, since false positive alarms may activate expensive actions to address a non-existent attack. Because both measures are equally important, the models have been selected based on their F1-score, which combines the results of both the precision and recall measures.

3.3 Datasets Used in this Research

A real dataset cannot be used to evaluate ML models due to privacy issues, but publicly available synthetic benchmarking datasets can be used. The CICIDS2017 [4] and NSL-KDD [5] datasets are used to evaluate the research workflow.

3.3.1 CICIDS2017 Dataset

CICIDS2017 [4] is a publicly available benchmarking dataset generated by the Canadian Institute for Cybersecurity over five days; it consists of about 3 million data instances. Sharafaldin Sharafaldin *et al.* [4] created a complete network topology to generate the dataset, including modems, firewalls, switches and routers, various operating systems such as Windows, Ubuntu and Mac, and commonly available protocols like HTTP, HTTPS, FTP, SSH and email protocols. In the network architecture, Sharafaldin *et al.* [4] created two networks, victim and attack, with their associated public and private IPs.

The attack network comprises one router, one switch and four PCs [4]. The victim network includes three servers, one firewall, two switches and ten PCs connected by a domain controller and active directory [4]. The CICIDS2017 outperforms the most commonly used datasets because it is more recent (2017), more realistic and covers 14 network attacks.

CICIDS2017 Attack Description

CICIDS2017 consists of 14 network attacks that belong to seven of the most up-to-date attack categories. The CICIDS2017 attack types are described as follows:

1. ***Brute Force Attack***: This attack uses the hit-and-trial method to crack passwords by trying out different password combinations until it finds the correct one [151]. This method can take a few seconds or longer, depending on password complexity. In addition, the brute force attack can be used to find hidden content and pages within a web application [151]. Attacks included in the dataset under the Brute Force category are as follows:
    - FTP-Patator
    - SSH-Patator
2. ***Heartbleed Attack***: This is an implementation flaw (bug) in the OpenSSL cryptography library, which implements the Secure Sockets Layer (SSL) and Transport Layer

Security (TLS) protocols [152]. It is usually exploited by sending a malicious heartbeat message with a small payload and a greater length field than the client's actual payload to obtain sensitive information and content stored in the client's memory, such as cryptographic keys and login credentials [4, 152].

3. ***Botnets***: These are a group of infected machines that are remotely controlled by the attacker (botmaster) [153]. Botnet attacks are used to perform various malicious activities, such as stealing data and crashing servers [153].

4. ***Denial-of-Service (DoS) Attack***: This attack attempts to render services temporarily unavailable to legitimate users by flooding the targeted machine or its surrounding infrastructure with spurious requests until it cannot process any requests and thus becomes unavailable [154]. Attacks included in the dataset under the DoS category include the following:

    - DoS Slowloris
    - DoS Slowhttptest
    - DoS Hulk
    - DoS GoldenEye

5. ***Distributed Denial-of-Service (DDoS) Attack***: These attacks utilise multiple compromised computer systems to flood the targeted system with overwhelming internet traffic that leaves the target services temporarily unavailable [155].

6. ***Web Attacks***: Three different web attacks were included in the dataset: a SQL Injection, where an attacker creates a SQL query and executes it into an entry field in the web application to make the database retrieve sensitive information or obtain unauthorised access to the database [156]; Cross-Site Scripting (XSS), in which the attacker inserts malicious scripts into a legitimate web page or browser, and once the end-user visits the the web page or lunch the browser, the malicious code is executed, granting the attacker access to sensitive information maintained by the browser or used by the web page [157]. The third one is Brute Force over HTTP, which attempts

to obtain the administrator's password and gain unauthorised access by trying a list of passwords [4].

7. ***Infiltration Attack***: This attack is usually performed from the internal network by exploiting vulnerable application software [4]. Once the attack is successful, a backdoor will be implemented on the target's machine, allowing for various attacks on the network, such as IP sweep and portscan [4].

Table 3.1 summarises CICIDS2017 attack types and provides number and percentage of instances for each attack.

Table 3.1: CICIDS2017 Attack Distribution

Type	**Count**	**Percentage (%)**
BENIGN	2,358,036	83.3
DoS Hulk	231,073	8.2
portscan	158,930	5.6
DDoS	41,835	1.5
DoS GoldenEye	10,293	0.4
FTP Patator	7,938	0.3
SSH Patator	5,897	0.2
DoS SlowLoris	5,796	0.2
DoS SlowHTTPTest	5,499	0.2
Botnet	1,966	0.07
Web Attack: Brute Force	1,507	0.05
Web Attack: XSS	625	0.02
Infiltration	36	0.001
Web Attack: SQL Injection	21	0.0007
HeartBleed	11	0.0004
Total	**2,829,463**	**100**

CICIDS2017 Feature Description

CICFlowMeter [6, 7] was used to extract the network traffic features from the generated PCAP file, which is an API for capturing packets from the network. CICFlowMeter is a Java application flow-based feature extractor and analyser which reads PCAP files and generates 84 network traffic features, in addition to a CSV file that consists of those generated features [6, 7]. Table 3.2 shows the CICIDS2017 dataset extracted features. The full feature description is available in **Appendix A.1**.

Table 3.2: CICIDS2017 Features

No.	Feature	No.	Feature	No.	Feature	No.	Feature
1	Flow ID	22	Flow Packets/s	43	Fwd Packets/s	64	Fwd Avg Bulk Rate
2	Source IP	23	Flow IAT Mean	44	Bwd Packets/s	65	Bwd Avg Bytes/Bulk
3	Source Port	24	Flow IAT Std	45	Min Packet Length	66	Bwd Avg Packets/Bulk
4	Destination IP	25	Flow IAT Max	46	Max Packet Length	67	Bwd Avg Bulk Rate
5	Destination Port	26	Flow IAT Min	47	Packet Length Mean	68	Subflow Fwd Packets
6	Protocol	27	Fwd IAT Total	48	Packet Length Std	69	Subflow Fwd Bytes
4	Time stamp	28	Fwd IAT Mean	49	Packet Len. Variance	70	Subflow Bwd Packets
8	Flow Duration	29	Fwd IAT Std	50	FIN Flag Count	71	Subflow Bwd Bytes
9	Total Fwd Packets	30	Fwd IAT Max	51	SYN Flag Count	72	Init_Win_bytes_fwd
10	Total Backward Packets	31	Fwd IAT Min	52	RST Flag Count	73	Act_data_pkt_fwd
11	Total Length of Fwd Pck	32	Bwd IAT Total	53	PSH Flag Count	74	Min_seg_size_fwd
12	Total Length of Bwd Pck	33	Bwd IAT Mean	54	ACK Flag Count	75	Active Mean
13	Fwd Packet Length Max	34	Bwd IAT Std	55	URG Flag Count	76	Active Std
14	Fwd Packet Length Min	35	Bwd IAT Max	56	CWE Flag Count	77	Active Max
15	Fwd Pck Length Mean	36	Bwd IAT Min	57	ECE Flag Count	78	Active Min
16	Fwd Packet Length Std	37	Fwd PSH Flags	58	Down/Up Ratio	79	Idle Mean
17	Bwd Packet Length Max	38	Bwd PSH Flags	59	Average Packet Size	80	Idle Packet
18	Bwd Packet Length Min	39	Fwd URG Flags	60	Avg Fwd Segment Size	81	Idle Std
19	Bwd Packet Length Mean	40	Bwd URG Flags	61	Avg Bwd Segment Size	82	Idle Max
20	Bwd Packet Length Std	41	Fwd Header Length	62	Fwd Avg Bytes/Bulk	83	Idle Min
21	Flow Bytes/s	42	Bwd Header Length	63	Fwd Avg Packets/Bulk	84	Label

3.3.2 NSL-KDD Dataset

The NSL-KDD dataset [5] is an updated version of the KDD99 dataset [158], based originally on the DARPA98 dataset. The data in The DARPA98 dataset is raw TCPdump data collected with a packet sniffer placed on the network segment outer of the router [159]. The NSL-KDD [5] that overcomes the issues related to the latter, such as duplicate records, that cause the model to be biased towards these records [5]. NSL-KDD dataset consists of two files, KDDTrain+, which has a 125,973 record dataset, and KDDTest+, which has a 22,544 record dataset. The advantage of NSL-KDD is that the entire dataset can be used for evaluation without the need to split it randomly and select a small proportion of it [5].

NSL-KDD Attacks Description

NSL-KDD consists of four main attack categories. These attacks are described by Tavallaee *et al.* [5] as follows:

1. ***Denial of Service Attack (DoS)***: Attacker overwhelms a machine or network resources with requests to render services temporarily unavailable to legitimate users.

2. ***User to Root Attack (U2R)***: Attacker obtains access to a normal user account to

exploit system vulnerabilities, aiming to achieve root access to that system.

3. ***Remote to Local Attack (R2L)***: Intruder sends packets over the network to another machine that does not have permission in order to exploit vulnerabilities and achieve unauthorised access to that machine.

4. ***Probing Attack***: Intruder collects information about the structure of the network to overcome the network security controls [5] and eventually access the system.

Table 3.3 summarises NSL-KDD attack types and the number of instances for each attack [5].

Table 3.3: NSL-KDD Attack Distribution

	KDDTrain+ dataset		**KDDTest+ dataset**	
Type	**Count**	**Percentage (%)**	**Count**	**Percentage (%)**
Normal	67,343	53.5	9,711	43
DoS	11,656	9.3	7,458	33.1
U2R	52	0.04	200	0.9
R2L	995	0.8	2,754	12.2
Probe	45,927	36.6	2,421	10.7
Total	**125973**	**100**	**22544**	**100**

NSL-KDD Features Description

NSL-KDD has 43 features classified into three feature groups [5]:

1. ***Basic features***: Includes all the features produced from a TCP/IP connection [5].

2. ***Traffic features***: Contains features calculated for a window interval having two groups, 'same host' and 'same service' features [5].

3. ***Content features***: Contains features that help find suspicious behaviour in the data flow (e.g., the number of failed logins) for some attack types, such as R2L and U2R attacks [5].

Table 3.4 shows the NSL-KDD dataset features; a full feature description is available in **Appendix B.1**.

Table 3.4: NSL-KDD Features

No.	Feature	No.	Feature	No.	Feature	No.	Feature
1	Duration	12	Logged In	23	Count	34	Dst Host Same Srv Rate
2	Protocol Type	13	Num Compromised	24	Srv Count	35	Dst Host Diff Srv Rate
3	Service	14	Root Shell	25	Serror Rate	36	Dst Host Same Src Port Rate
4	Flag	15	Su Attempted	26	Srv Serror Rate	37	Dst Host Srv Diff Host Rate
5	Src Bytes	16	Num Root	27	Rerror Rate	38	Dst Host Serror Rate
6	Dst Bytes	17	Num File Creations	28	Srv Rerror Rate	39	Dst Host Srv Serror Rate
7	Land	18	Num Shells	29	Same Srv Rate	40	Dst Host Rerror Rate
8	Wrong Fragment	19	Num Access Files	30	Diff Srv Rate	41	Dst Host Srv Rerror Rate
9	Urgent	20	Num Outbound Cmds	31	Srv Diff Host Rate	42	Class
10	Hot	21	Is Hot Logins	32	Dst Host Count	43	Difficulty Level
11	Num Failed Logins	22	Is Guest Login	33	Dst Host Srv Count		

3.4 Research Dataset Preprocessing Steps

3.4.1 CICIDS2017 Dataset Preprocessing

Preprocessing methods were applied to the CICIDS2017 dataset to make it suitable for training the anomaly detection algorithms. Figure 3.3 shows the initial experiment workflow, including the preprocessing steps applied to the CICIDS2017 dataset.

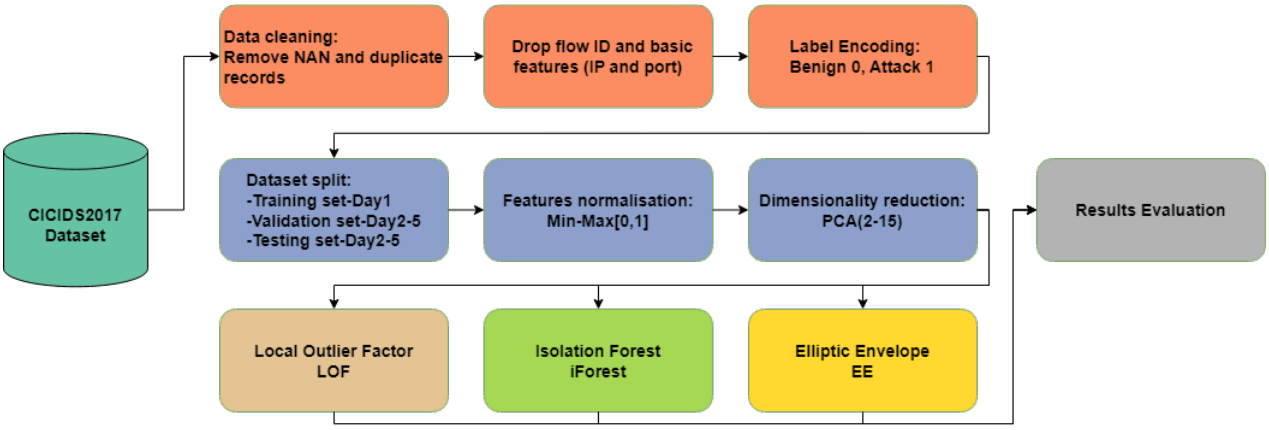


Figure 3.3: Initial Experiment Workflow

The first step was to clean the dataset by dropping missing and *NaN* values, which can cause the ML model to make biased predictions and reduce its accuracy. Duplicated records were also dropped. This step was followed by dropping out some features that could affect the model's performance; for instance, ID features were removed, as they do not have discriminatory value for attacks. Next, features containing IP addresses were also removed because attackers often spoof their IP addresses to avoid IP filtering systems [127]. Finally,

features representing port information were removed, since they can cause models to overfit towards socket information [160]. Next, the categorical text in the Label feature was converted to numeric form, as the scikit-learn library requires all data to be in numerical form [161]. Therefore, the label for all attack types was converted to '1' and '0' for benign instances. Next, the dataset was divided into training, validation, and test sets (Figure 3.4). Assuming that the system has no prior knowledge about the network attacks and following the network flow scenario used to generate the CICIDS2017 dataset, the data from the first day (Monday), which comprises 529,445 benign flows (about 19% of the entire dataset), was used to train the model. The remaining four-day dataset, which contains 2,298,225 of both attacks and benign flow, were split for validation and testing (50% each). Figure 3.4 shows the CICIDS2017 dataset split workflow, and Table 3.5 illustrates the dataset split distribution.

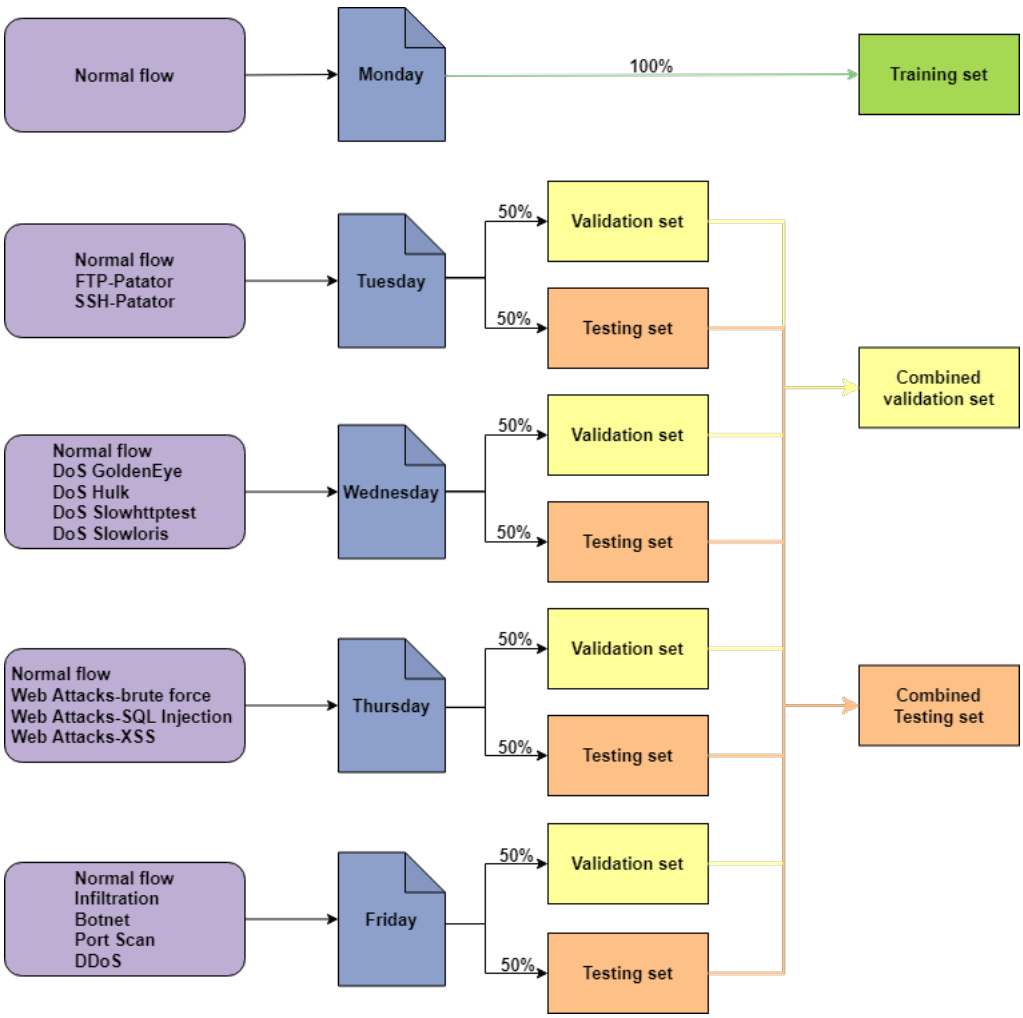


Figure 3.4: Dataset Split Workflow

Table 3.5: Dataset Split Distribution

Set	Benign '0'	Attack '1'	Total
Training	529,445	0	529,445
Validation	870,834	278277	1,149,111
Test	870,838	278,278	1,149,116

Next, the data were normalised between [0-1] using MinMaxScaler [147]. This step gives the data equal importance while keeping the shape of the original data distribution. Following this, PCA was applied as a dimensionality reduction method. PCA has been applied widely in the intrusion detection area, such as in [162], [163] and [164], as it only requires

a few parameters of the principal components to be managed for future detection and, most importantly, the statistics can be estimated quickly during the detection stage, which makes PCA feasible for real-time use [165, 166]. A preliminary exploratory analysis was performed on the dataset to gain insight into the explained variance ratio for every principal component. Figure 3.5 illustrates the explained variance ratio preserved for each principal component.

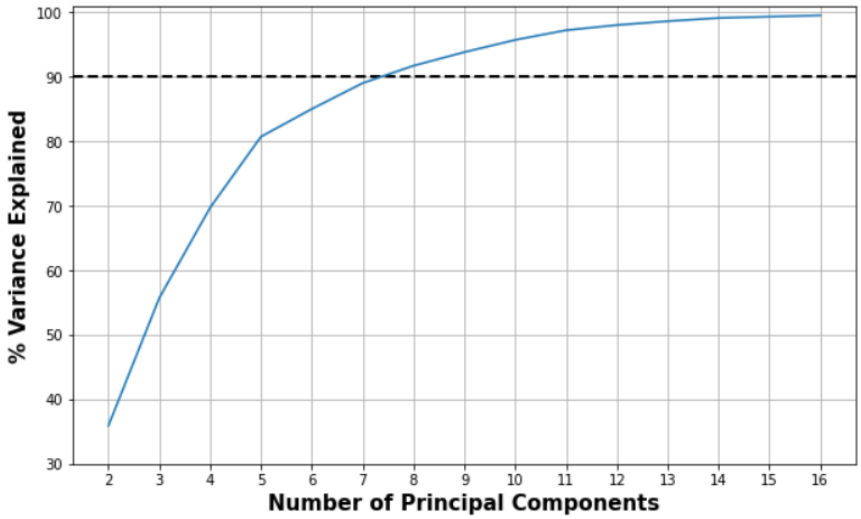


Figure 3.5: PCA Method to the CICIDS2017 Dataset

Figure 3.5 shows that 90% of the explained variance can be preserved using seven or more principal components. However, the number of principal components which are considered in the initial experiments 2–15 PCs.

3.4.2 NSL-KDD Dataset Preprocessing

As datasets usually differ from one another (e.g., number and the type of features), dataset preprocessing steps can differ too. Figure 3.6 illustrates the initial experiment workflow with the preprocessing steps applied to the NSL-KDD dataset. Like the CICIDS2017 dataset, preprocessing began by dropping missing and *NaN* records. Next, duplicate records were identified and removed. Then, protocol type, service, flag and class features were converted from text to numeric. Finally, the label for all attack types was converted to '1' and '0' for normal instances. The next step was to split the dataset into training, validation, and

test sets.

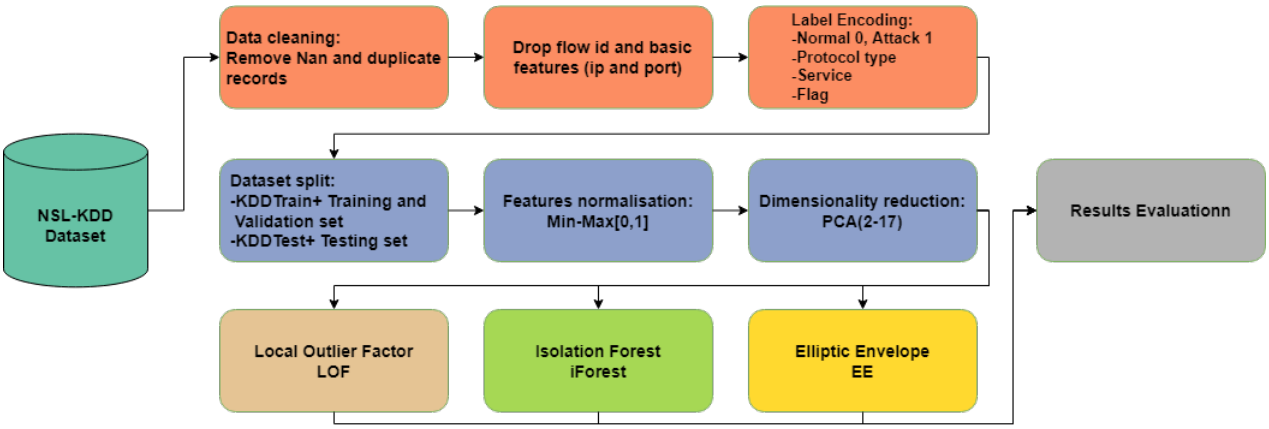


Figure 3.6: Initial Experiment workflow

As previously mentioned, the NSL-KDD dataset is made of two files: training and testing. The training file was split into 60% training, which is ideal and sufficient to train UNAD, and 40% for validation. Then, the validation set was combined with the testing set to ensure the same type of attacks in both sets, as some of the network attacks in the KDDTrain+ file were not available in the KDDTest+ file. Once combined, the dataset was split again into validation and testing sets with a $\approx 50\%$ proportion each, using a stratified random sampling method, which helps eliminate bias and provides a representative population of all attack types in both sets (Figure 3.7). Therefore, the total number of data instances for each set to this stage was 75,583 for the training set, 37,791 for the validation set and 35,140 for the testing set. In the final step of data splitting, all attack flows in the training set were dropped, retaining just the normal flow, since the models will be trained only on normal flow. The remainder of the training set comprised 40,405 normal data instances. Figure 3.7 shows the NSL-KDD dataset split workflow, and Table 3.6 illustrates the dataset split distribution.

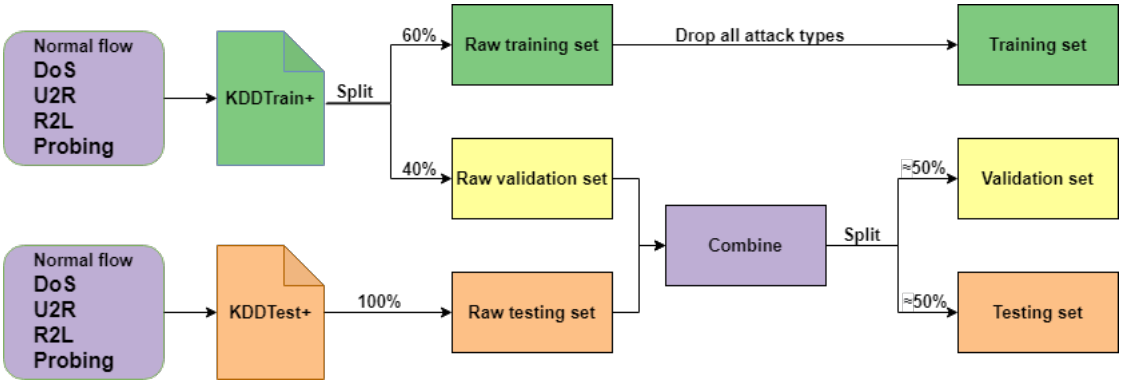


Figure 3.7: NSL-KDD Dataset Split Workflow

Table 3.6: NSL-KDD Dataset Split Distribution

Set	Benign '0'	Attack '1'	Total
Training	40,405	0	40,405
Validation	18,990	18,801	37,791
Test	17,658	17,482	35,140

The next step was to normalise data between [0-1] using MinMaxScaler [147]. Following this, the PCA method was used to reduce the dimensionality of the features. Figure 3.8 depicts the explained variance ratio preserved for every principal component.

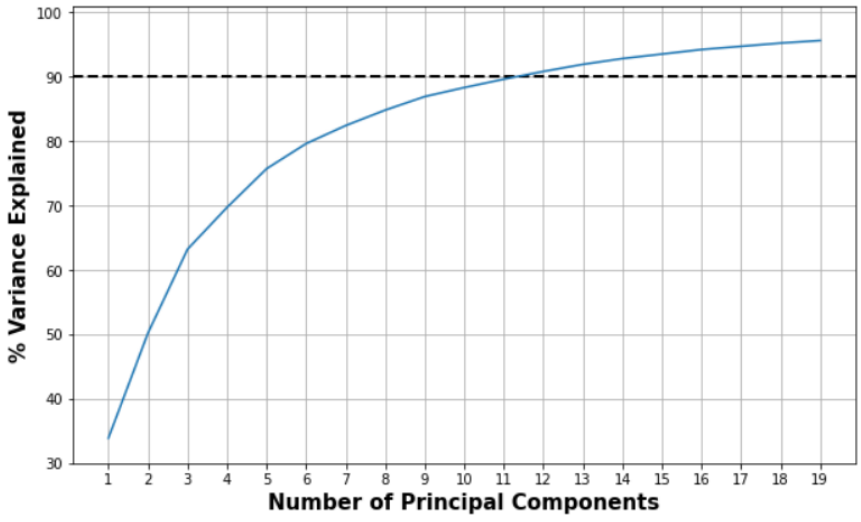


Figure 3.8: PCA Method on the NSL-KDD Dataset

Figure 3.8 shows that 90% of the explained variance can be preserved using 11 or more

principal components. However, the number of principal components which are considered in the initial experiments 2–17 PCs.

3.5 Evaluation of Anomaly Detection Algorithms as Base Learners for UNAD

As mentioned in **Section 3.2**, LOF, iForest and EE algorithms will be evaluated for selection as base learners for UNAD. For this, the algorithms will be trained using the training dataset (comprising only benign/normal network flow) and optimised using the validation set (including all types of attacks) to find the best combinations of hyperparameters in terms of the F1-score results. Furthermore, various Principal Components (PCs) are considered: 2–15 PCs for the CICIDS2017 dataset and 2–17 PCs for the NSL-KDD dataset, to reduce the data's dimensionality. A summary of the experiments and final results will be discussed in **Section 3.6**.

3.5.1 Local Outlier Factor (LOF)

With the current massive network traffic, LOF is expected to play a significant role in detecting attacks. Accordingly, it is evaluated here as a potential part of the proposed ensemble-based UNAD. The LOF module from scikit-learn [147] was used. The hyperparameters are *contamination* and *n_neighbours*. *Contamination* is the proportion of the outliers expected in the dataset ranging from 0 to 0.5, and *n_neighbours* is the number of nearest neighbours needed to classify a data sample [147]. No knowledge about the proportion of outliers (attacks) in the training data was assumed. The hyperparameters were optimised using various combinations of values. The *contamination* parameter was optimised from 0.01 to 0.5 in steps of 0.01. The *n_neighbours* parameter value was optimised from 5 to 50 in steps of 5. Once the hyperparameters were optimised and the best combination was determined in terms of the F1 score, they were applied to the test set for every number of PCs ranging between 2–15 for the CICIDS2017 dataset and 2-17 for the NSL-KDD dataset.

CICIDS2017 Results

Table 3.7 shows LOF overall experimental results for the CICIDS2017 dataset.

Table 3.7: CICIDS2017 LOF Overall Experimental Results (in %)

PCA	n_neighbors	Contamination	Accuracy	Precision	Recall	F1-score	ROC-AUC score
2	25	0.10	79.68	56.71	67.93	61.82	75.68
3	10	0.12	80.54	56.73	82.74	67.31	81.29
4	10	0.13	79.55	55.10	84.06	66.57	81.09
5	15	0.10	82.78	60.16	85.57	70.65	83.73
6	10	0.09	83.68	61.87	84.98	71.61	84.12
7	30	0.07	85.65	**66.19**	83.27	**73.76**	**84.84**
8	30	0.07	84.59	63.71	84.47	72.64	84.55
9	35	0.07	84.16	63.15	83.06	71.75	83.79
10	35	0.08	83.50	61.43	**85.62**	71.54	84.22
11	35	0.08	82.69	60.54	81.96	69.64	82.44
12	35	0.07	83.45	62.23	80.58	70.22	82.47
13	35	0.07	83.48	62.03	81.98	70.62	82.97
14	35	0.07	82.56	60.50	80.56	69.10	81.88
15	35	0.07	82.95	60.90	82.72	70.15	82.88

Table 3.7 shows that the highest precision, F1-score and ROC-AUC score was for 7 PCs (with contamination value of 0.07 and 30 neighbours), followed by 8 and 9 PCs (with contamination values of 0.07 for both and 30 and 35 neighbours, respectively). On the other hand, 2, 3, and 4 PCs had the lowest precision, F1-score and ROC-AUC score. Concerning the recall results, the highest recall was observed for 10 PCs (with contamination values of 0.08 and 35 neighbours), followed by 5 and 6 PCs (with contamination values of 0.10 and 0.09 and 15 and 10 neighbours, respectively). Moreover, 12, 14 and 2 PCs had the lowest recall. Therefore, based on the F1-score results, the optimal number of PCs for the LOF algorithms for the CICIDS2017 dataset is 7.

Figures 3.14 – 3.17 show the best results for each number of PCs used; the red bar represents the highest results obtained for each measure. Figure 3.13 depicts the results of precision, recall, F1-score and ROC-AUC measures.

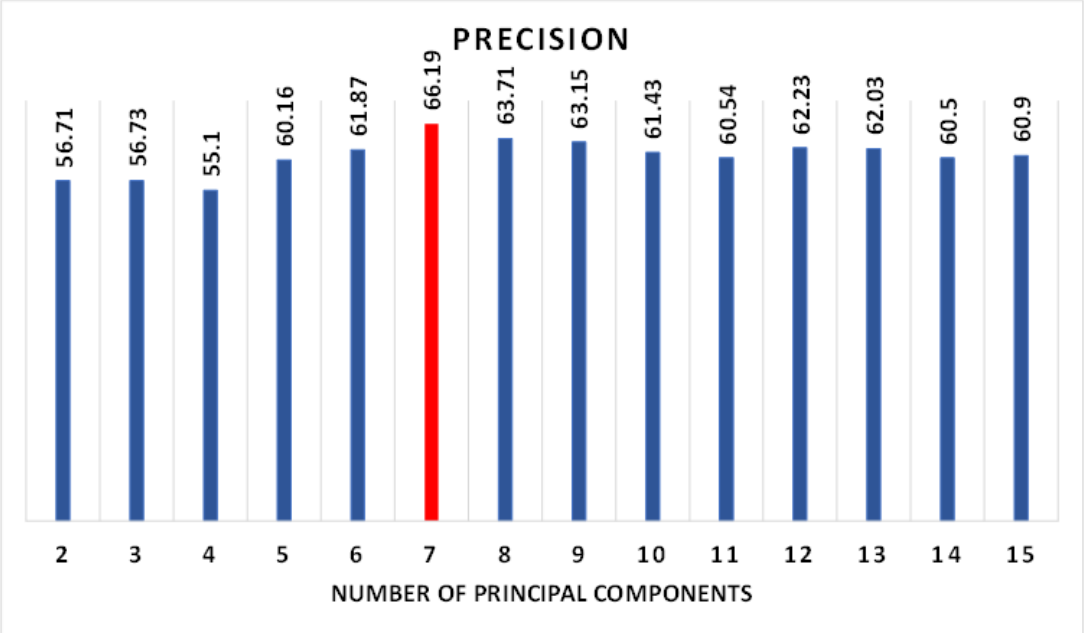


Figure 3.9: CICIDS2017 Precision Results for LOF-Based Workflow

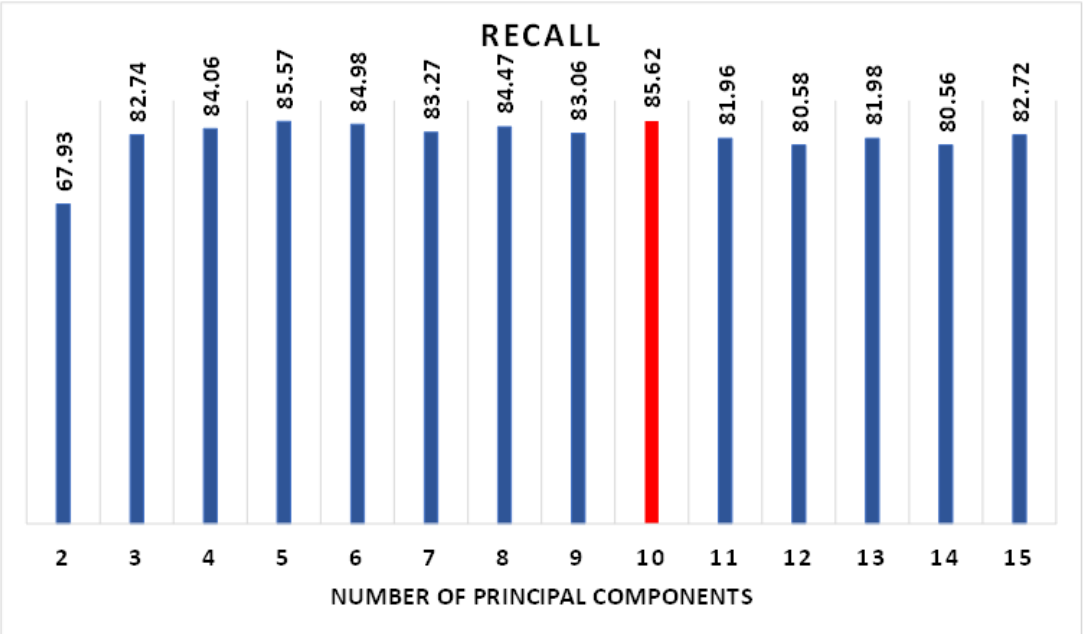


Figure 3.10: CICIDS2017 Recall Results for LOF-Based Workflow

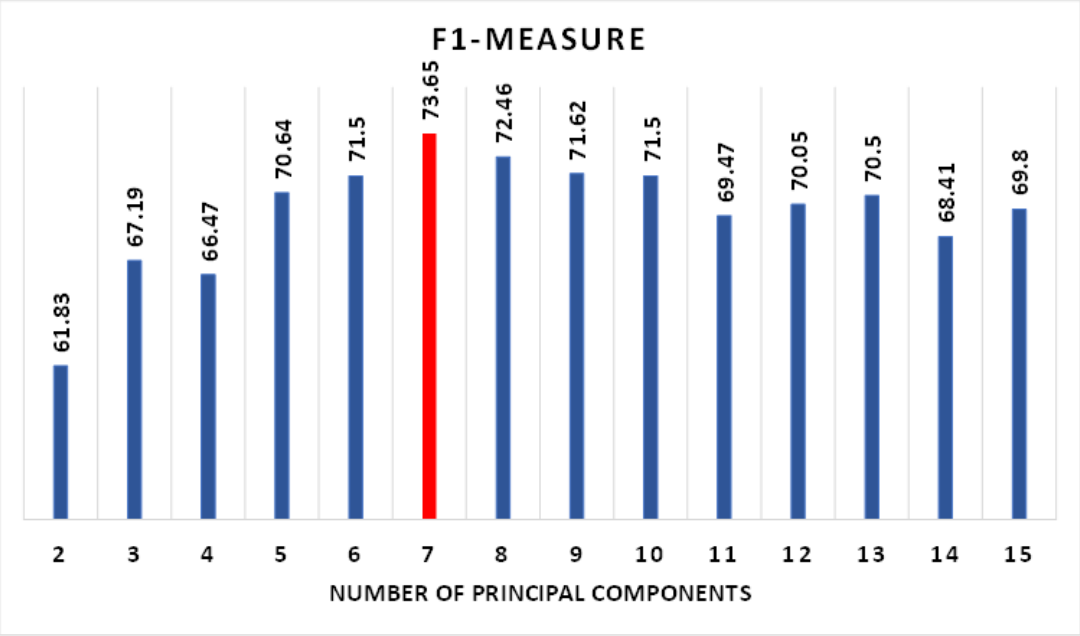


Figure 3.11: CICIDS2017 F1-score Results for LOF-Based Workflow

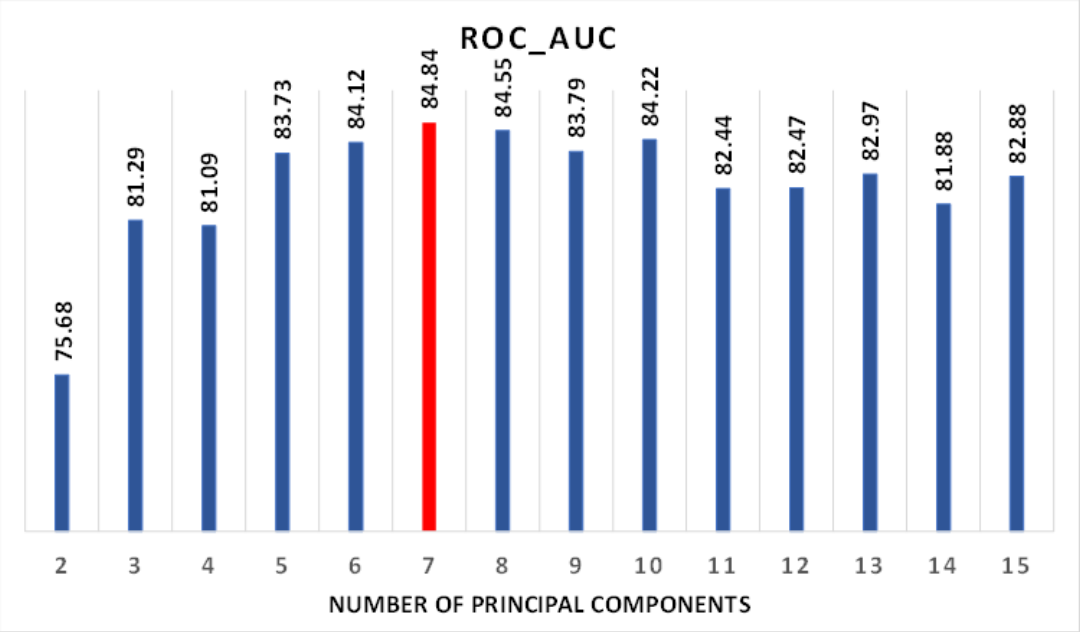


Figure 3.12: CICIDS2017 ROC-AUC Results for LOF-Based Workflow

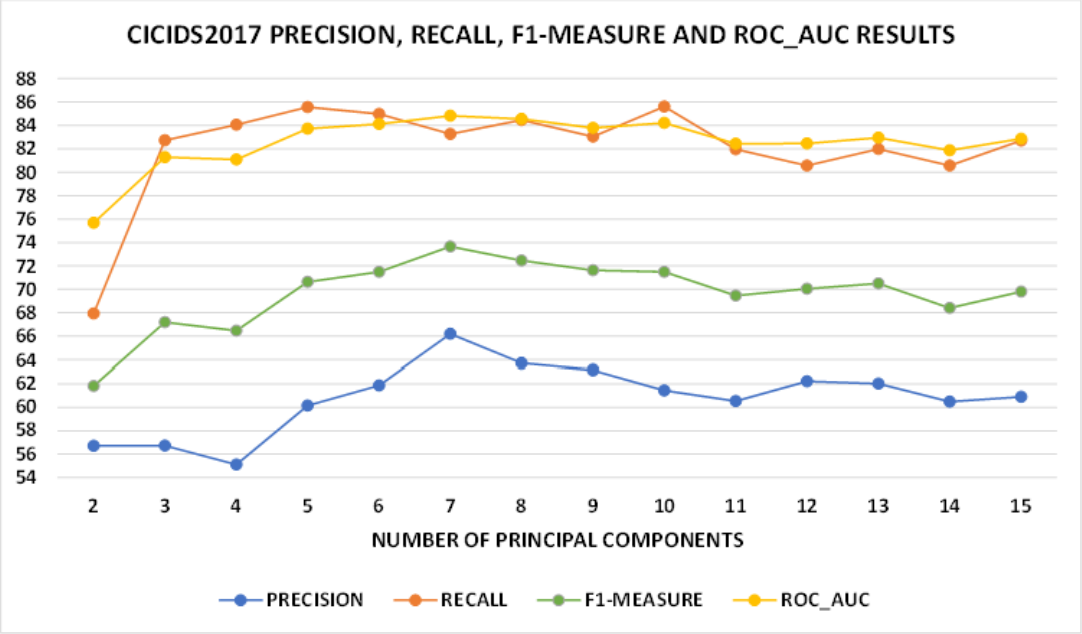


Figure 3.13: CICIDS2017 Precision, Recall, F1-score AND ROC-AUC Results For LOF

NSL-KDD Results

Table 3.8 shows LOF overall experimental results for the NSL-KDD dataset.

Table 3.8: NSL-KDD LOF Overall Experimental Results (in %)

PCA	n_neighbors	Contamination	Accuracy	Precision	Recall	F1-score	ROC-AUC score
2	45	0.33	76.06	70.96	87.83	78.50	76.12
3	5	0.21	82.23	77.18	91.27	83.63	82.28
4	5	0.12	84.31	**82.56**	86.79	84.62	84.32
5	5	0.16	83.92	79.51	91.16	84.94	83.95
6	5	0.15	85.48	81.47	91.65	86.26	85.51
7	5	0.14	85.53	81.05	92.55	**86.42**	**85.57**
8	5	0.14	84.87	80.08	92.65	85.91	84.91
9	5	0.13	84.99	80.44	92.27	85.95	85.03
10	35	0.24	81.99	77.09	90.78	83.38	82.04
11	35	0.24	80.77	75.40	91.07	82.50	80.83
12	10	0.17	84.92	80.53	91.09	85.84	84.95
13	5	0.15	81.46	75.42	93.06	83.31	81.51
14	5	0.13	82.36	76.55	93.04	83.99	82.41
15	5	0.14	82.19	76.14	93.50	83.93	82.25
16	5	0.13	83.35	77.21	94.39	83.94	83.41
17	5	0.13	83.04	76.71	**94.64**	83.74	83.10

Table 3.8 shows that the highest precision was achieved using 4 PCs (with a contamination value of 0.12 and 5 neighbours). Furthermore, using 6 and 7 PCs also produced high

precision results (with 5 neighbours for each and a contamination value of 0.15 and 0.14, respectively). Concerning the recall results, using 17 PCs produced the highest results (with a contamination value of 0.13 and 5 neighbours), followed by 16 PCs (with a contamination value of 0.13 and 5 neighbours) and 15 PCs (with a contamination value of 0.14 and 5 neighbours). For F1-score and ROC-AUC score results, the highest results were obtained using 7 PCs (with a contamination value of 0.14 and 5 neighbours) followed by 6 and 9 PCs. On the other hand, using 2 PCs had the least precision, F1-score and ROC-AUC score, while using 4 PCs had the least recall. Thus, based on the F1-score results, the optimal number of PCs for the LOF algorithms for the NSL-KDD dataset is 7.

Figures 3.14 – 3.17 show the best results for each number of PCs used; the red bar represents the highest results obtained for each measure. Figure 3.18 depicts the results of precision, recall, F1-score and ROC-AUC measures.

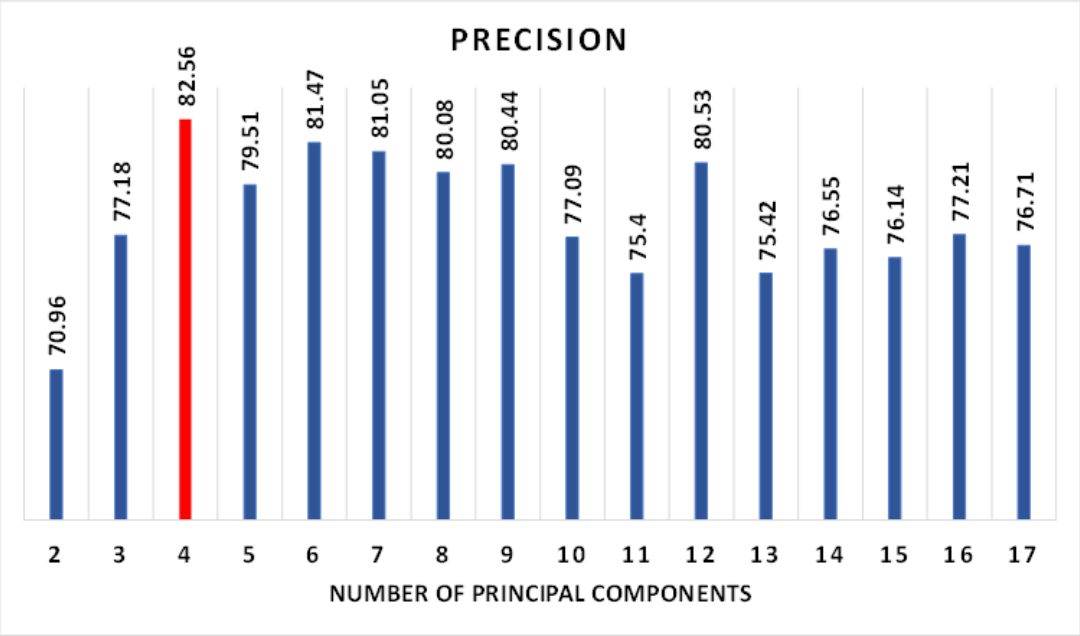


Figure 3.14: NSL-KDD Precision Results for LOF-Based Workflow

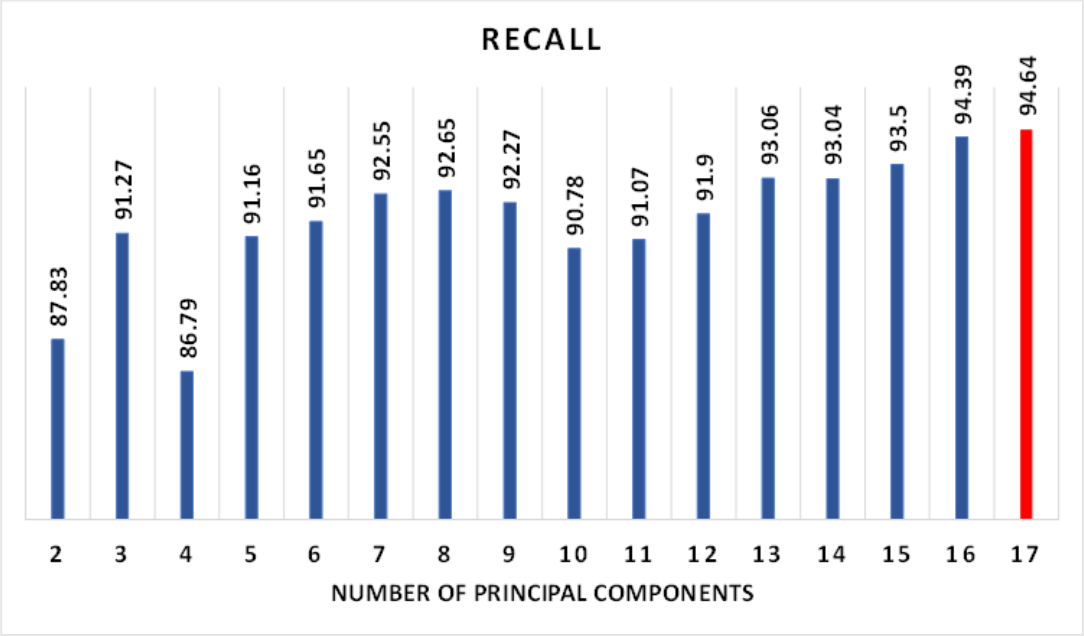


Figure 3.15: NSL-KDD Recall Results for LOF-Based Workflow

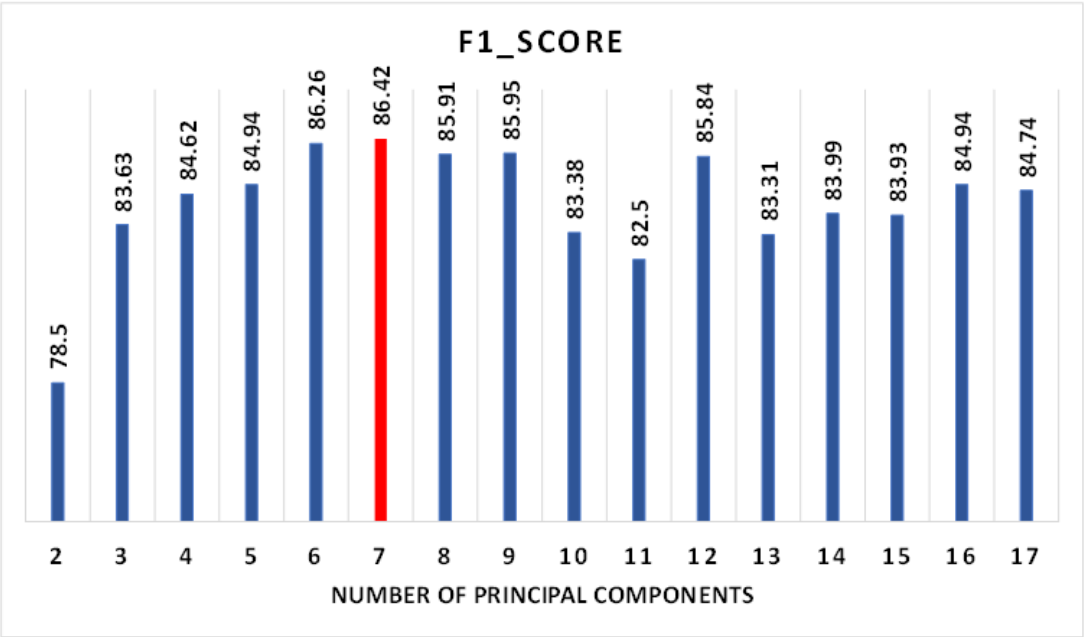


Figure 3.16: NSL-KDD F1-Score Results for LOF-Based Workflow

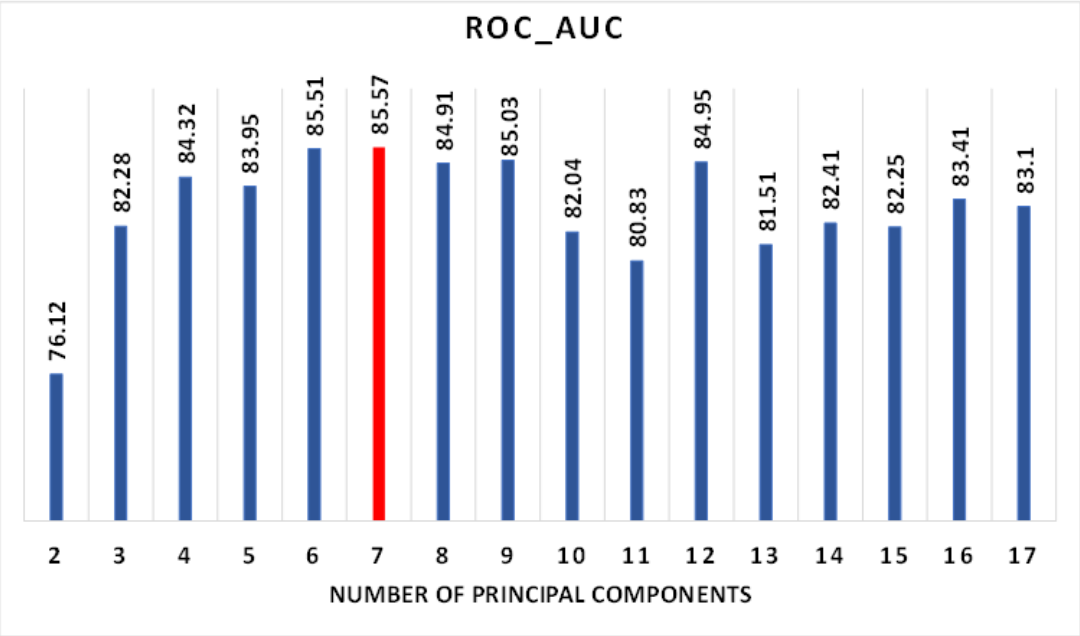


Figure 3.17: NSL-KDD ROC-AUC Results for LOF-Based Workflow

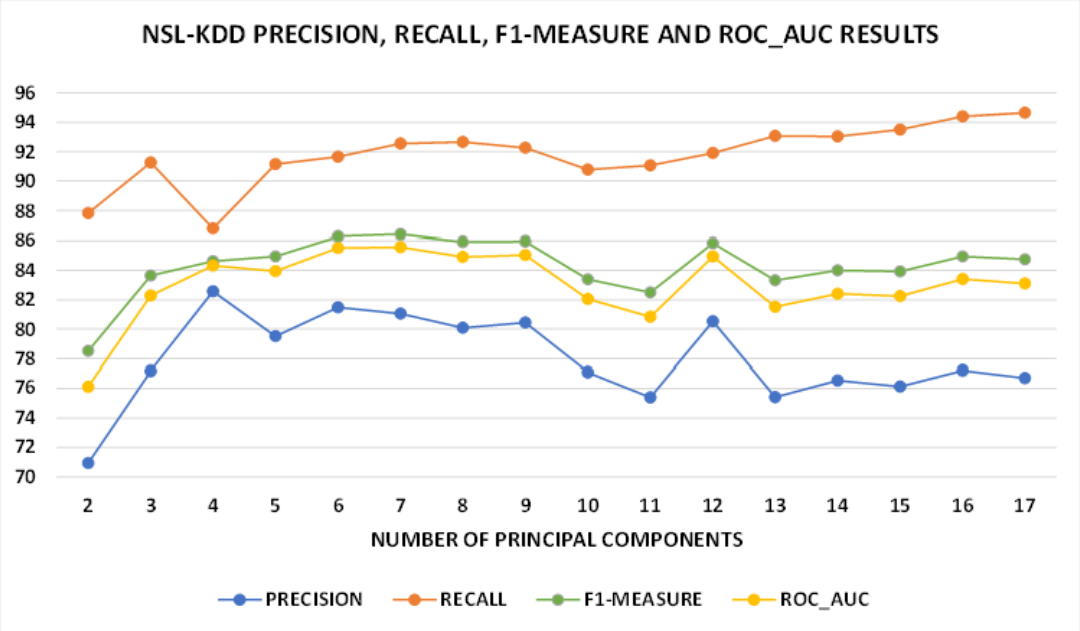


Figure 3.18: NSL-KDD Precision, Recall, F1-score AND ROC-AUC Results For LOF

3.5.2 Isolation Forest (iForest)

iForest provides low linear time complexity with a low memory requirement, suiting it well for detecting network attacks quickly [54]. Furthermore, iForest can deal with high-

dimensional data with unrelated attributes [54]. These attributes make it ideal for integration into the proposed UNAD ensemble. iForest module from scikit-learn [147] was used.

The hyperparameters are *contamination* factor, *n_estimators* (number of trees) and *max_samples*. The *contamination* parameter is the same as for LOF, and *n_estimators* is the number of trees to be built in the forest [147]. No knowledge about the proportion of outliers in the training data was assumed. The hyperparameters were optimised using various combinations of values. For the *contamination* parameter, it was optimised from 0.01 to 0.5 in steps of 0.01. The number of *n_estimators* was selected from 50 to 450 in steps of 50.

Regarding the *max_samples* parameter, which selects the portion of the training data for each base estimator [147], proportion settings of 25%, 50%, 75% and 100% were used in addition to the default setting of 256 samples, which corresponds to 0.05% and 0.63% of the training set for the CICIDS2017 and NSL-KDD datasets, respectively. The *max_features* parameter, which controls the number of features to be extracted from the dataset to train each estimator [147], was set to its default value (1.0) to use all the features to train the estimators, and the *random_state* parameter was set to a fixed number (42) for results reproducibility. Once the hyperparameters were optimised and the best combination was determined, they were applied to the test set for every number of PCs ranging between 2–15 for the CICIDS2017 dataset and 2–17 for the NSL-KDD dataset.

CICIDS2017 Results

Table 3.9 shows the iForest overall experimental results for the CICIDS2017 dataset.

Table 3.9: CICIDS2017 iForest Overall Experimental Results (in %)

PCA	n_estimators	Contamination	max_samples	Accuracy	Precision	Recall	F1-score	ROC-AUC score
2	150	0.28	0.5	71.02	44.96	87.73	59.45	76.71
3	150	0.32	auto (256)	71.45	45.07	81.83	58.12	74.98
4	50	0.31	0.25	70.06	44.04	87.37	58.56	75.95
5	50	0.29	1	70.98	44.80	85.35	58.76	75.87
6	200	0.43	auto (256)	66.9	41.64	**91.39**	57.21	75.23
7	200	0.35	auto (256)	71.47	45.11	82.04	58.21	75.07
8	350	0.39	auto (256)	69.62	43.47	84.76	57.47	74.77
9	350	0.41	auto (256)	68.08	42.19	85.89	56.58	74.14
10	50	0.34	auto (256)	71.21	44.76	80.77	57.60	74.46
11	400	0.24	0.25	74.44	**48.42**	84.84	**61.65**	**77.98**
12	100	0.33	auto (256)	72.60	46.37	84.02	59.76	76.48
13	350	0.41	auto (256)	69.97	44.12	90.08	59.23	76.81
14	100	0.38	auto (256)	70.79	44.70	87.04	59.07	76.31
15	100	0.38	auto (256)	70.04	44.04	87.72	58.64	76.05

Table 3.9 shows that the highest combined precision, F1-score and ROC-AUC score was for 11 PCs (with a contamination value of 0.24, 400 estimators and 25% max samples), followed by 12 PCs (with a contamination value of 0.33, 100 estimators and the default settings for the max samples). On the other hand, 9, 8 and, 6 PCs had the lowest precision and F1-score, and 9, 10, and 8 PCs had the lowest ROC-AUC score.

Furthermore, the highest recall was observed for 6 PCs (with contamination value of 0.43, 200 estimators and default setting for max_samples), followed by 13 and 2 PCs (with contamination values of 0.41 and 0.28, 350 and 150 estimators and the default settings (256) and 50% max_samples, respectively). Moreover, 10, 3 and 7 PCs had the lowest recall. Hence, based on the F1-score results, the optimal number of PCs for the iForest algorithms for the CICIDS2017 dataset is 11 PCs.

Figures 3.19 – 3.22 show the best results for each number of PCs used; the red bar represents the highest results obtained for each measure. Figure 3.23 depicts the results of precision, recall, F1-score and ROC-AUC measures.

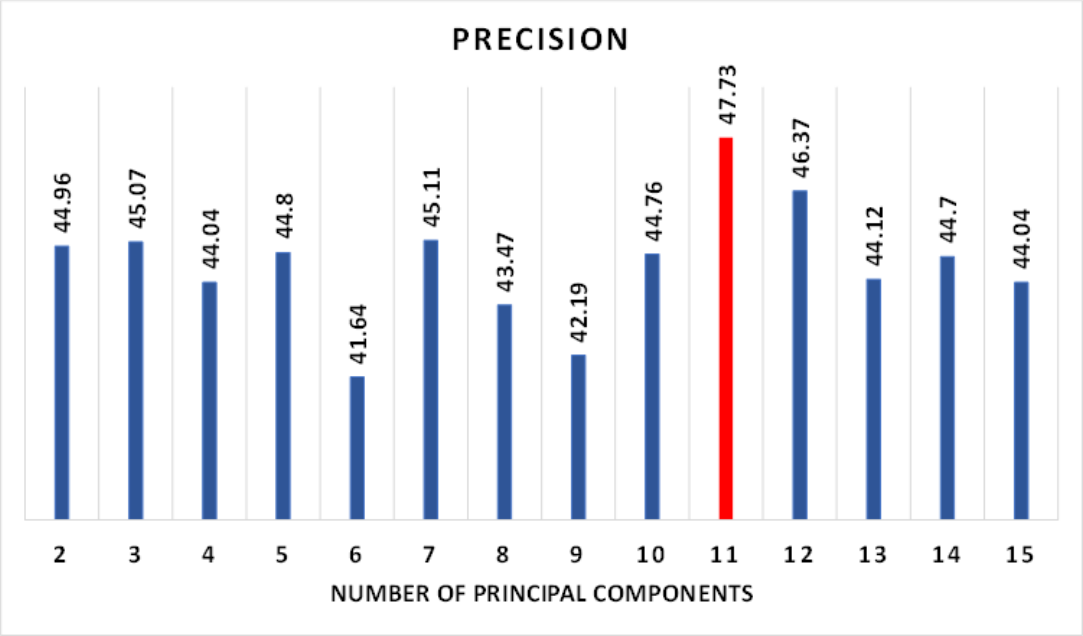


Figure 3.19: CICIDS2017 Precision Results for iForest-Based Workflow

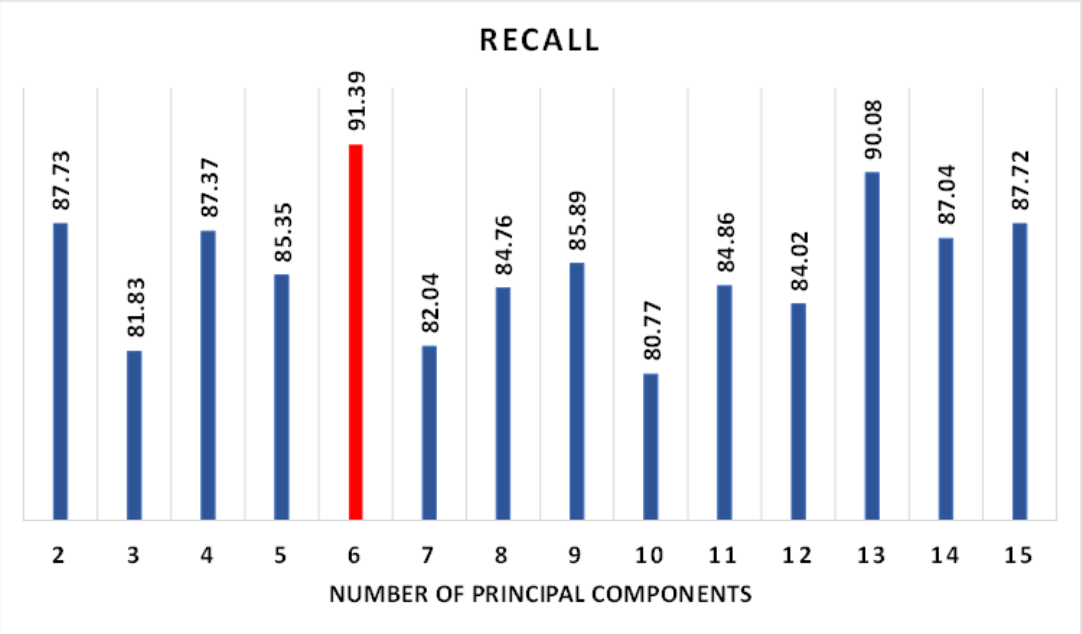


Figure 3.20: CICIDS2017 Recall Results for iForest-Based Workflow

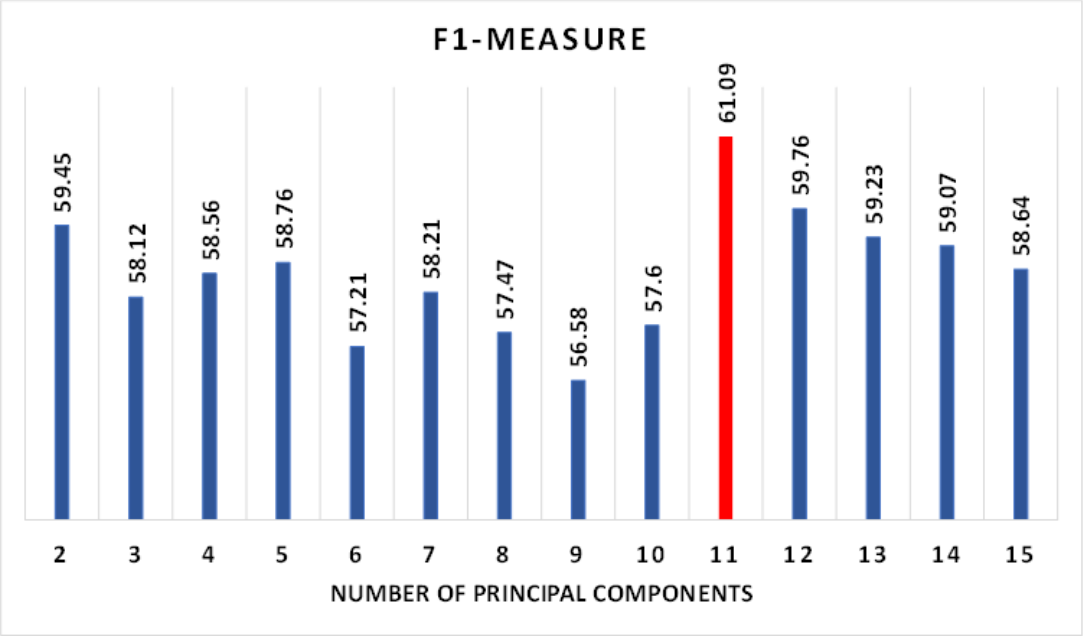


Figure 3.21: CICIDS2017 F1-score Results for iForest-Based Workflow

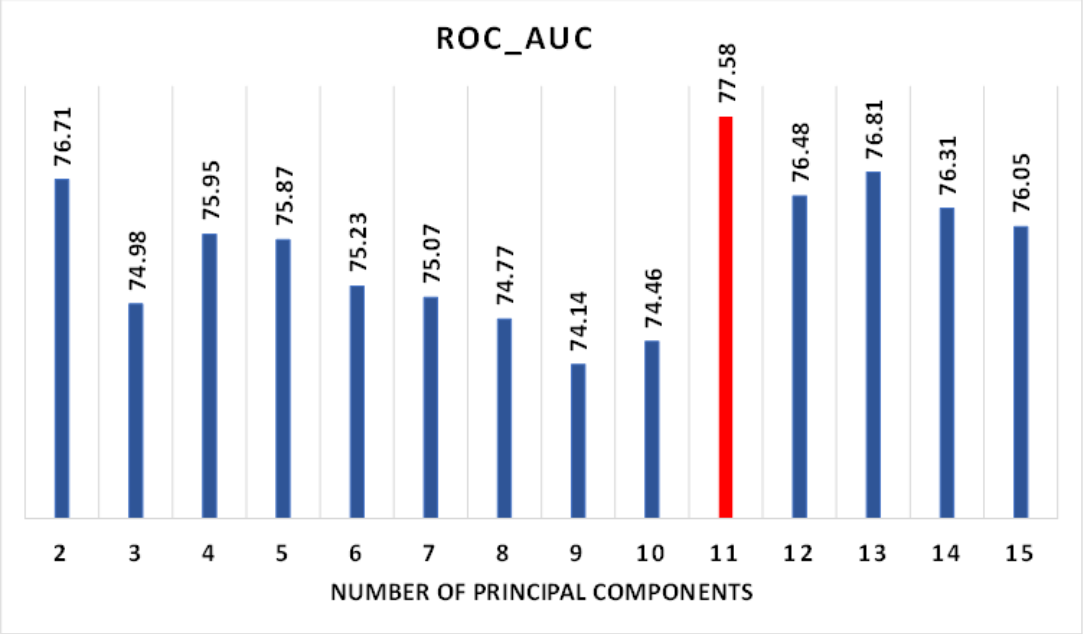


Figure 3.22: CICIDS2017 ROC-AUC Results for iForest-Based Workflow

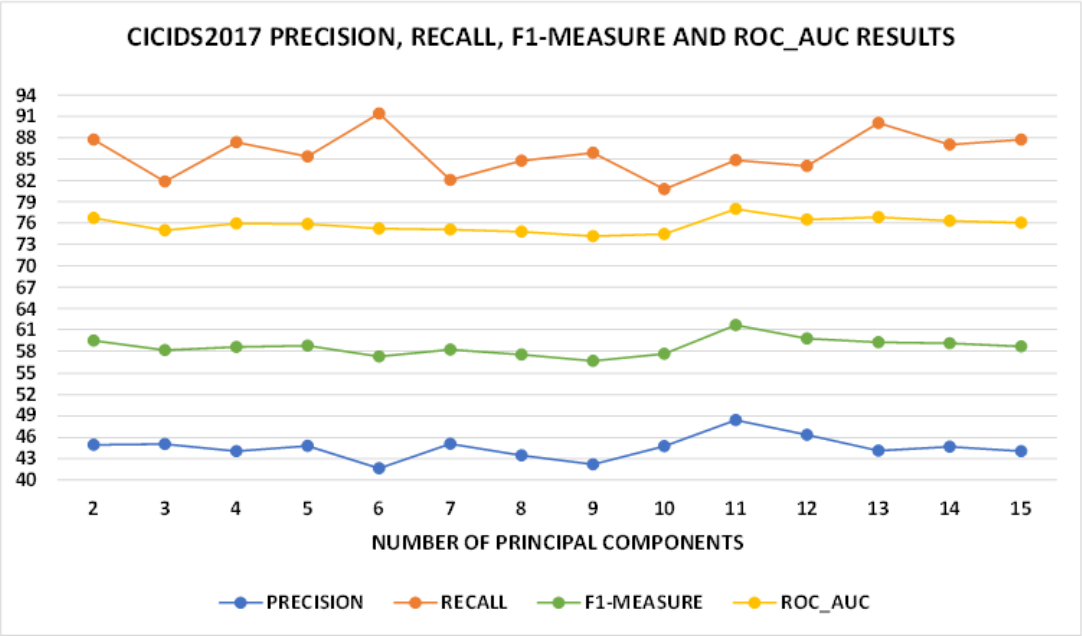


Figure 3.23: CICIDS2017 Precision, Recall, F1-score AND ROC-AUC Results For iForest

NSL-KDD Results

Table 3.10 shows iForest overall experimental results for the NSL-KDD dataset.

Table 3.10: NSL-KDD iForest Overall Experimental Results (in %)

PCA	n_estimators	Contamination	max_samples	Accuracy	Precision	Recall	F1-score	ROC-AUC score
2	150	0.15	0.50	87.68	84.63	91.92	88.12	87.70
3	350	0.05	1	92.05	92.91	90.97	91.93	92.05
4	450	0.11	1	90.15	87.88	93.03	90.38	90.17
5	600	0.08	1	91.80	90.62	93.15	91.87	91.80
6	300	0.06	1	92.19	92.37	91.88	92.13	92.18
7	100	0.07	1	91.79	91.52	92.03	91.77	91.79
8	600	0.10	1	90.75	89.15	92.70	90.89	90.76
9	100	0.09	1	90.92	89.90	92.09	90.99	90.93
10	600	0.10	1	90.68	89.14	92.55	90.81	90.69
11	100	0.10	1	91.17	89.25	**93.52**	91.33	91.18
12	100	0.08	1	91.92	91.02	92.94	91.97	91.93
13	600	0.07	1	91.76	91.55	91.92	91.73	91.76
14	200	0.06	1	92.02	92.08	91.87	91.97	92.02
15	100	0.03	1	92.04	**94.76**	88.92	91.75	92.03
16	100	0.05	1	92.93	93.13	92.63	**92.88**	**92.93**
17	50	0.04	1	92.69	93.85	91.29	92.55	92.69

Table 3.10 shows that the highest precision was obtained using 15 PCs (with a contamination value of 0.03, 100 estimators and 100% max samples). Additionally, using 17 and 16 PCs also resulted in high precision results (with contamination values of 0.04 and 0.05 and

100% max samples for both and 150 and 600 estimators, respectively). Concerning the recall results, using 11 PCs produced the highest results (with a contamination value of 0.10, 100 estimators and 100% max samples). The second highest recall was achieved using 5 PCs (with contamination value of 0.08, 600 estimators and 100% max samples) followed by 4 PCs (with contamination parameter of 0.11, 450 estimators and 100% max samples). Finally, for F1-score and ROC-AUC score results, the highest results were achieved using 16 PCs (with contamination value of 0.05, 100 estimators and 100% max samples), followed by 17 and 6 PCs, respectively. In contrast, using 2 PCs had the lowest precision, F1-score and ROC-AUC score, while using 15 PCs had the lowest recall results. Hence, based on the F1-score results, the optimal number of PCs for the iForest algorithms for the NSL-KDD dataset is 16.

Figures 3.24 – 3.27 show the best results for each number of PCs used; the red bar represents the highest results obtained for each measure. Figure 3.28 depicts the results of precision, recall, F1-score and ROC-AUC measures.

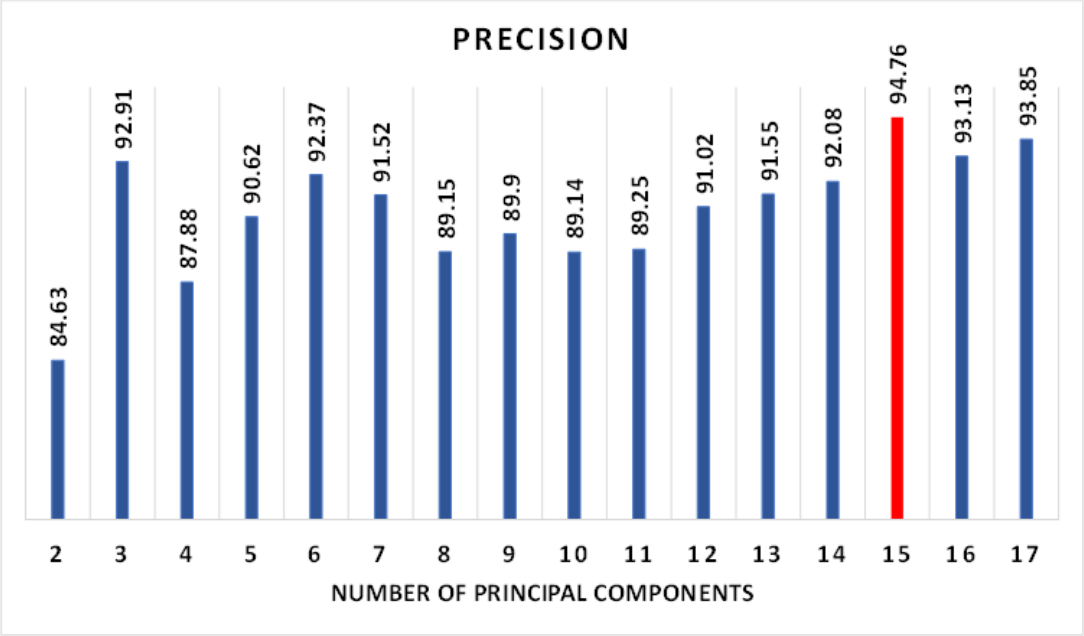


Figure 3.24: NSL-KDD Precision Results for iForest-Based Workflow

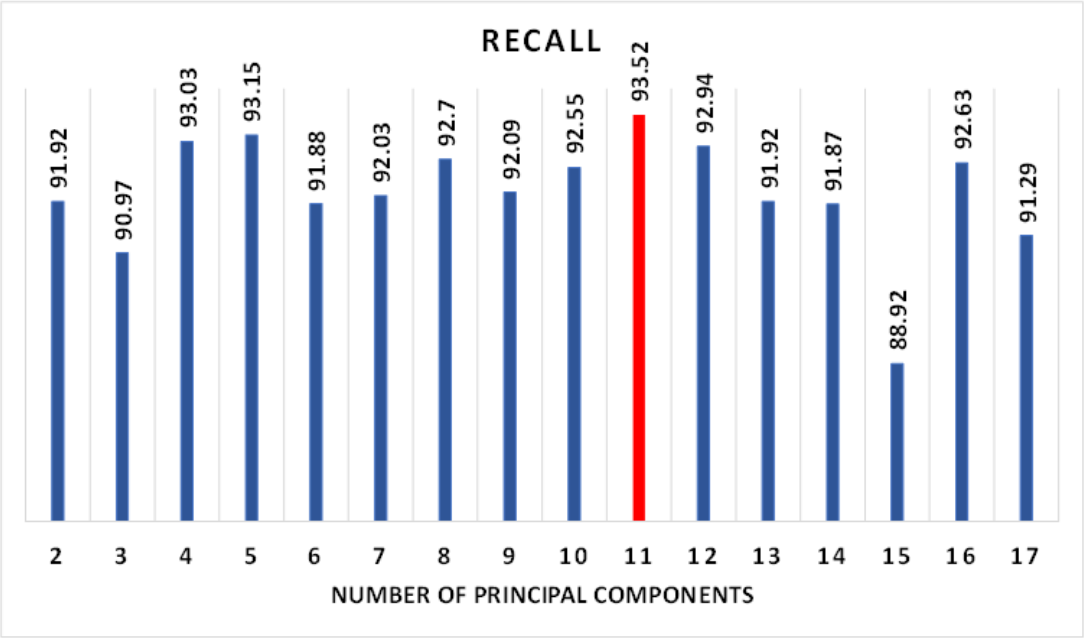


Figure 3.25: NSL-KDD Recall Results for iForest-Based Workflow

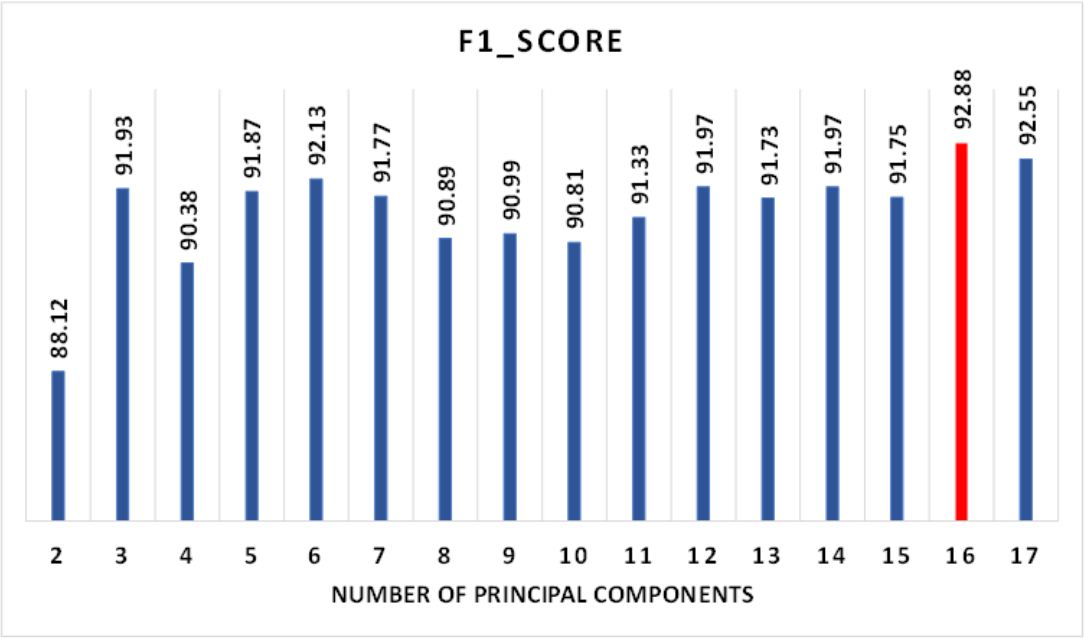


Figure 3.26: NSL-KDD F1-score Results for iForest-Based Workflow

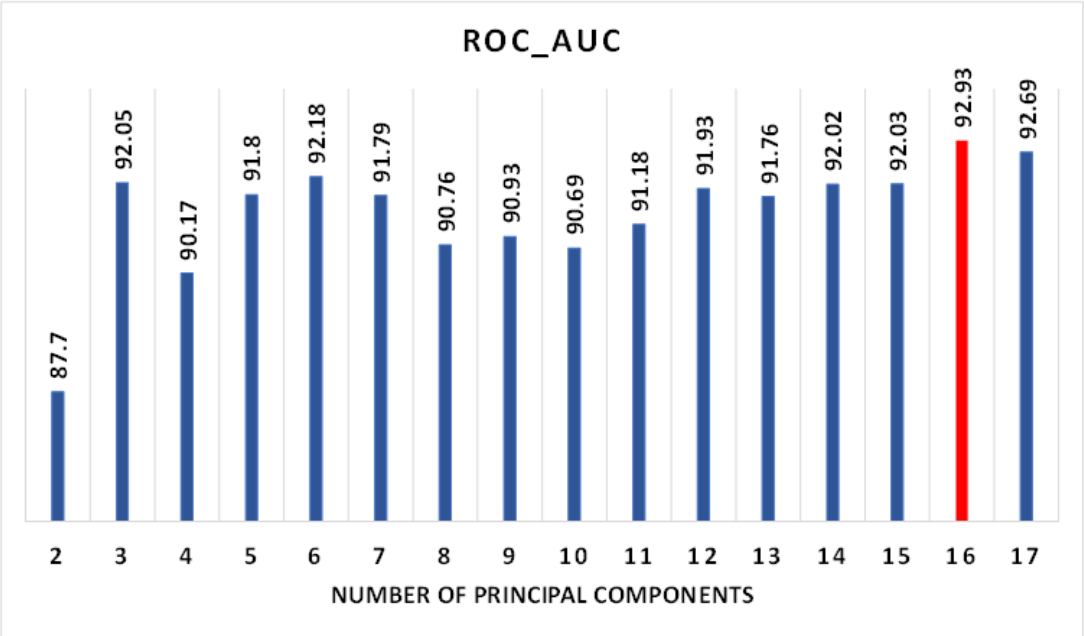


Figure 3.27: NSL-KDD ROC-AUC Results for iForest-Based Workflow

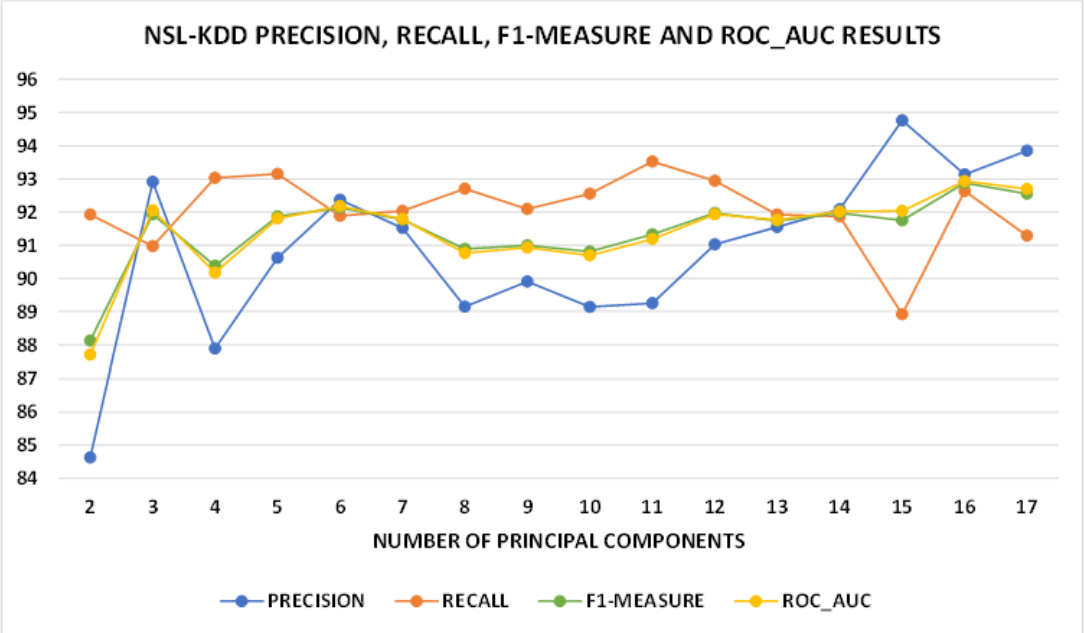


Figure 3.28: NSL-KDD Precision, Recall, F1-score AND ROC-AUC Results For iForest

3.5.3 Elliptic Envelope

The EE module from scikit-learn [147] was used. The EE hyper-parameter is *contamination*, which is the same in LOF and iForest. No knowledge about the proportion of outliers

in the training data was assumed. The *contamination* parameter value was selected from 0.01 to 0.5 in steps of 0.01. Once the *contamination* parameter was optimised and its best value determined, it was applied to the test set for every number of PCs ranging between 2–15 for the CICIDS2017 dataset and 2–17 for the NSL-KDD dataset.

CICIDS2017 Results

Table 3.11 shows EE's overall experimental results for the CICIDS2017 dataset.

Table 3.11: CICIDS2017 EE Overall Experimental Results (in %)

PCA	Contamination	Accuracy	Precision	Recall	F1-score	ROC-AUC score
2	0.33	71.55	**44.88**	76.57	56.59	73.26
3	0.38	68	41.81	81.97	55.37	72.75
4	0.44	69.47	42.79	77.43	55.12	72.18
5	0.42	64.43	40.11	95.06	56.41	74.85
6	0.29	66.68	41.90	**97.17**	**58.55**	**77.06**
7	0.38	64.41	40.10	95.08	56.41	74.85
8	0.47	64.41	40.09	94.97	56.38	74.81
9	0.49	66.64	40.77	83.32	54.75	72.32
10	0.39	62.88	37.94	83.79	52.23	69.99
11	0.45	61.86	37.52	86.44	52.33	70.23
12	0.45	65.40	40.77	94.75	57.01	75.38
13	0.37	68.05	42.20	86.33	56.69	74.27
14	0.49	68.08	42.22	86.39	56.72	74.31
15	0.49	67.71	42.09	88.62	57.07	74.83

Table 3.11 shows that the highest F1-score and ROC-AUC score was seen for 6 PCs (with a contamination value of 0.44), followed by 15 and 12 for the F1-score and 12, 5 and 7 for the ROC-AUC score. Furthermore, 10, 11 and 9 PCs produced the lowest F1-score and ROC-AUC score results. Similarly, the highest recall was seen for 6 PCs, followed by 7 and 5 PCs, while 3, 4 and 2 had the lowest recall. Moreover, the highest precision was observed for 2 PCs (with a contamination value of 0.33), followed by 4 and 14, respectively. In contrast, the lowest precision was observed for 11, 10 and 8, respectively. Therefore, based on the F1-score results, the optimal number of PCs for the EE algorithms for the CICIDS2017 dataset is 11.

Figures 3.29 – 3.32 show the best results for each number of PCs used; the red bar represents the highest results obtained for each measure. Figure 3.33 depicts the results of

precision, recall, F1-score and ROC-AUC measures.

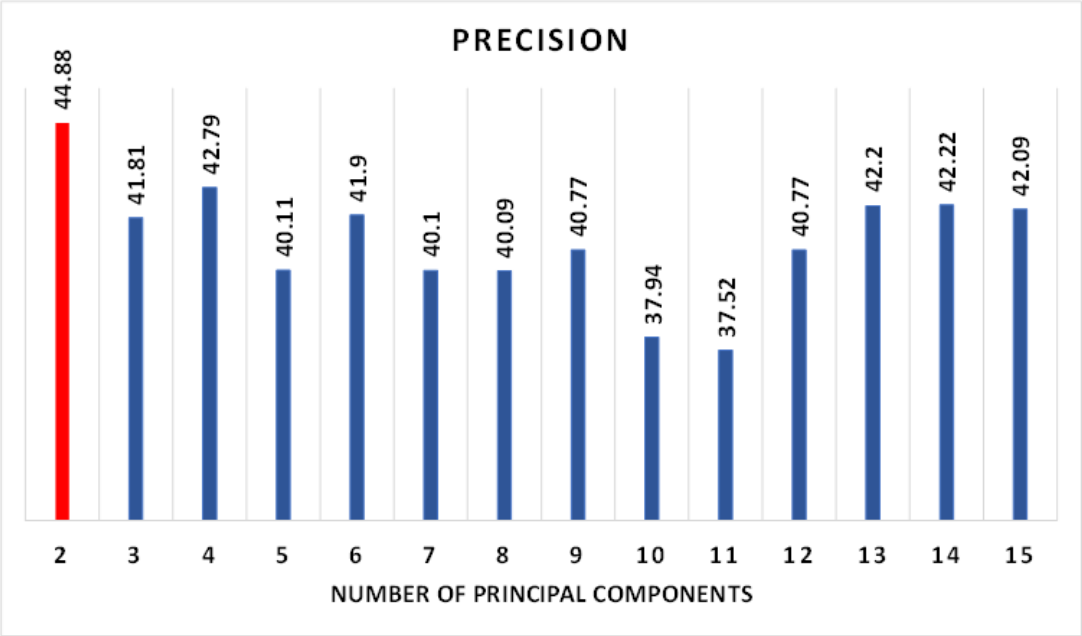


Figure 3.29: CICIDS2017 Precision Results for EE-Based Workflow

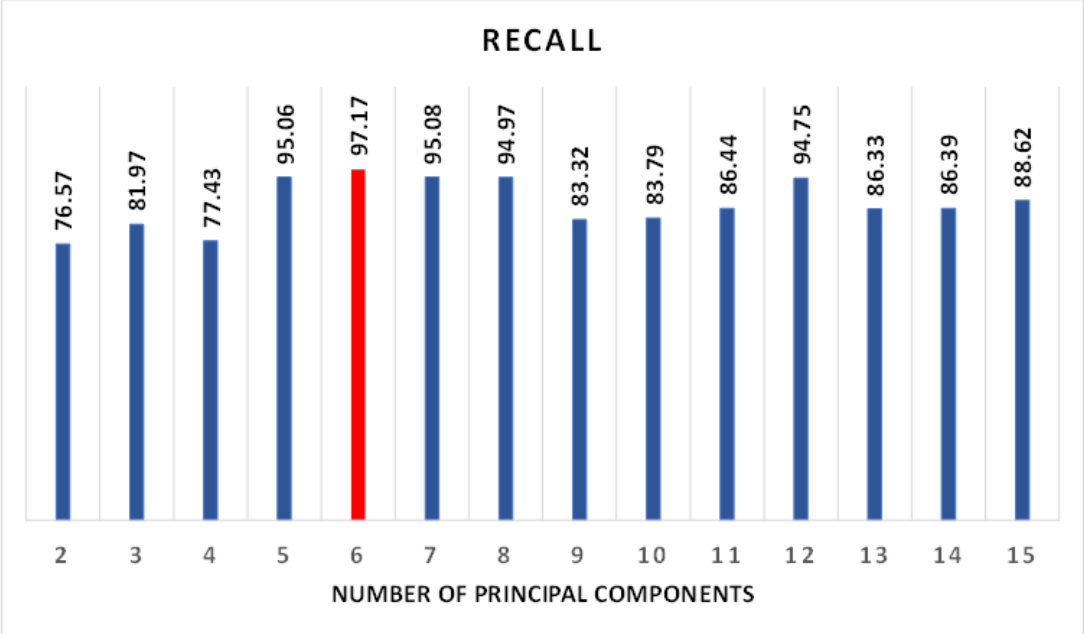


Figure 3.30: CICIDS2017 Recall Results for EE-Based Workflow

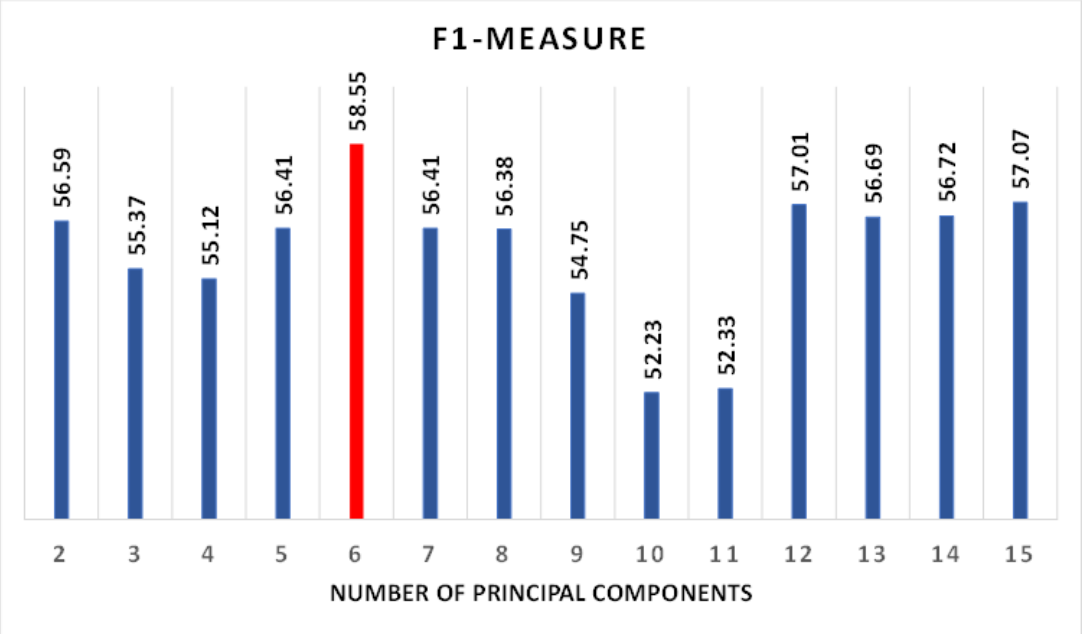


Figure 3.31: CICIDS2017 F1-score Results for EE-Based Workflow

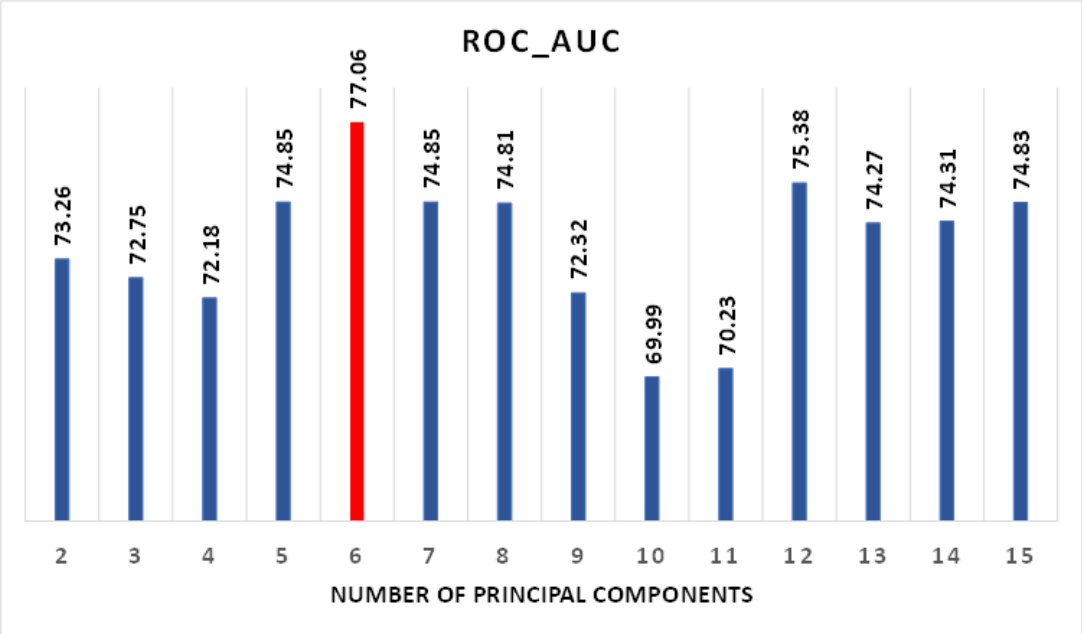


Figure 3.32: CICIDS2017 ROC-AUC Results for EE-Based Workflow

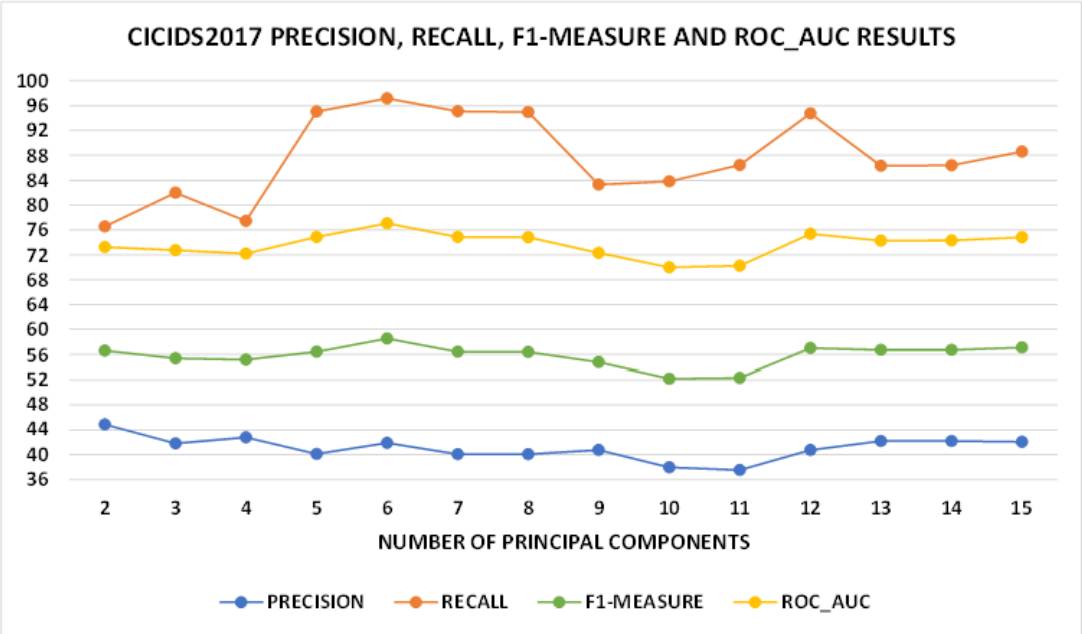


Figure 3.33: CICIDS2017 Precision, Recall, F1-score AND ROC-AUC Results For EE

NSL-KDD results

NSL-KDD Elliptic Envelope's overall experimental results are shown in Table 3.12.

Table 3.12: NSL-KDD EE Overall Experimental Results (in %)

PCA	Contamination	Accuracy	Precision	Recall	F1-score	ROC-AUC score
2	0.34	79.91	75.03	89.57	81.66	79.63
3	0.46	77.44	70.27	95.78	81.07	77.29
4	0.46	76.84	70	94.61	80.46	76.69
5	0.40	77.75	71.68	92.38	80.72	77.63
6	0.44	76.82	70.41	93.18	80.21	76.68
7	0.48	77.86	69.79	98.26	81.73	77.68
8	0.48	77.85	69.96	98.25	81.73	77.67
9	0.48	77.84	69.96	98.23	81.72	77.67
10	0.46	78.89	70.75	**99.12**	82.57	78.72
11	0.26	81.93	80.13	85.32	**82.64**	**81.90**
12	0.29	79.03	77.52	82.26	79.82	79
13	0.25	81.28	80.26	83.38	81.79	81.26
14	0.30	79	77.21	82.78	79.90	78.97
15	0.23	80.33	**80.75**	80.07	80.41	80.33
16	0.31	78.82	75.83	85.14	80.21	78.77
17	0.27	79.04	78.53	80.42	79.94	79.03

Table 3.12 shows that the highest precision was achieved using 15 PCs (with a contamina-

tion value of 0.23). The second and third highest were 13 and 11 PCs (with contamination parameters of 0.25 and 0.26, respectively). Regarding the recall results, using 10 PCs had the highest results (with a contamination value of 0.46). Furthermore, results were also high using 8 PCs and 7 PCs (with a contamination value of 0.08 for each). Finally, concerning the F1-score and ROC-AUC score results, the highest results were achieved when using 11 PCs (with a contamination value of 0.26), followed by 10 and 13 PCs for the F1-score and 13 and 15 PCs for the ROC-AUC score.

On the other hand, using 8 PCs and 9 PCs had the lowest precision, while using 15 PCs had the lowest recall results. Finally, using 17 PCs resulted in the lowest F1-score and 2 PCs the lowest ROC-AUC score. Therefore, based on the F1-score results, the optimal number of PCs for the EE algorithms for the NSL-KDD dataset is 11.

Figures 3.34 – 3.37 show the best results for each number of PCs used; the red bar represents the highest results obtained for each measure. Figure 3.38 depicts the results of precision, recall, F1-score and ROC-AUC measures.

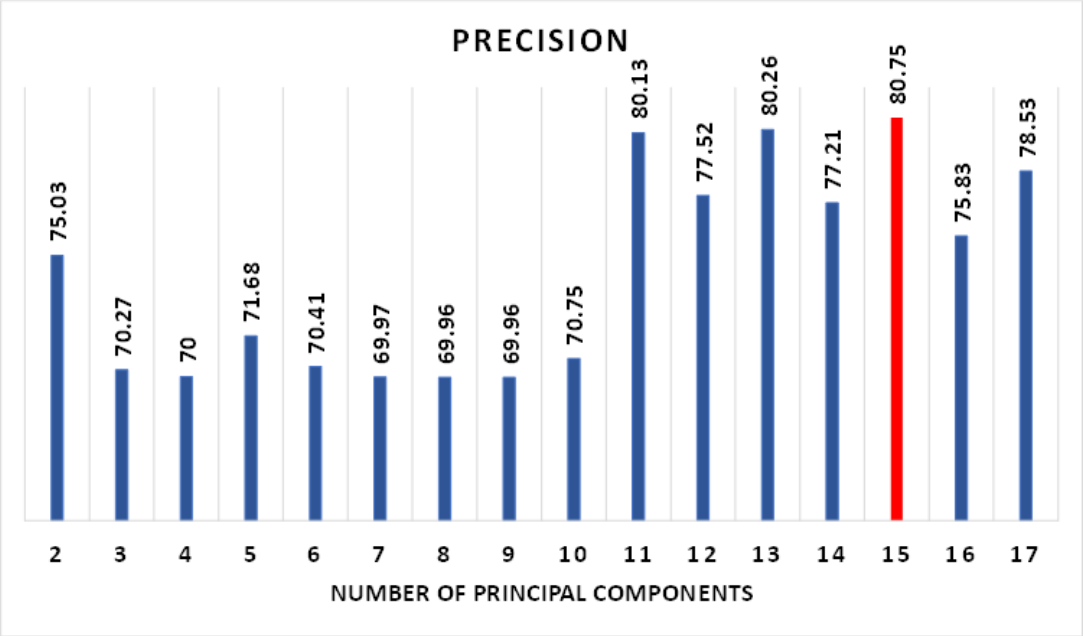


Figure 3.34: NSL-KDD Precision Results for EE-Based Workflow

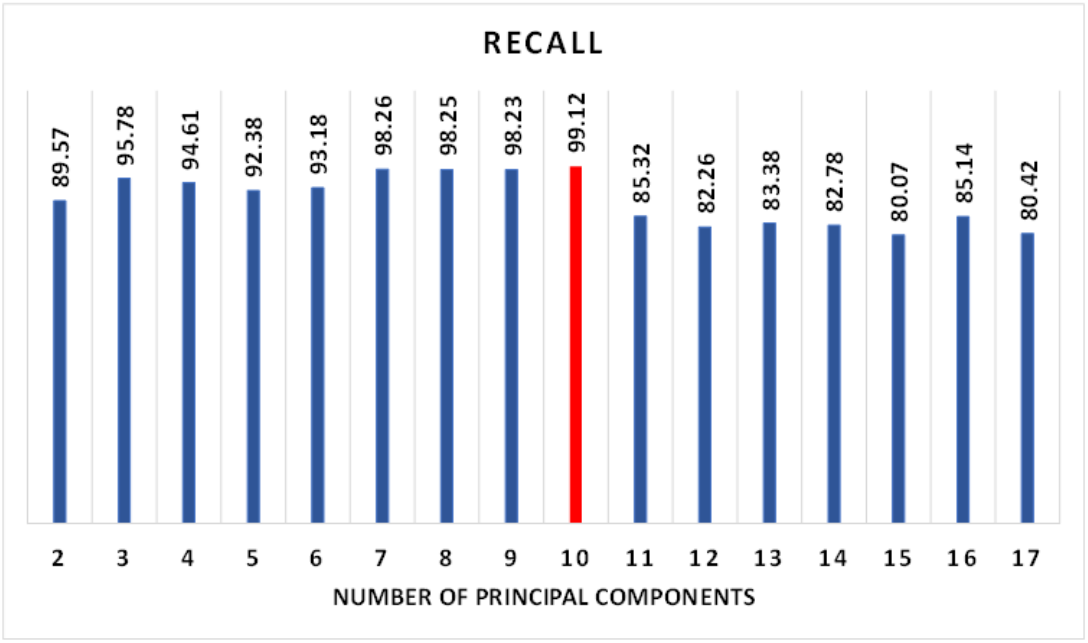


Figure 3.35: NSL-KDD Recall Results for EE-Based Workflow

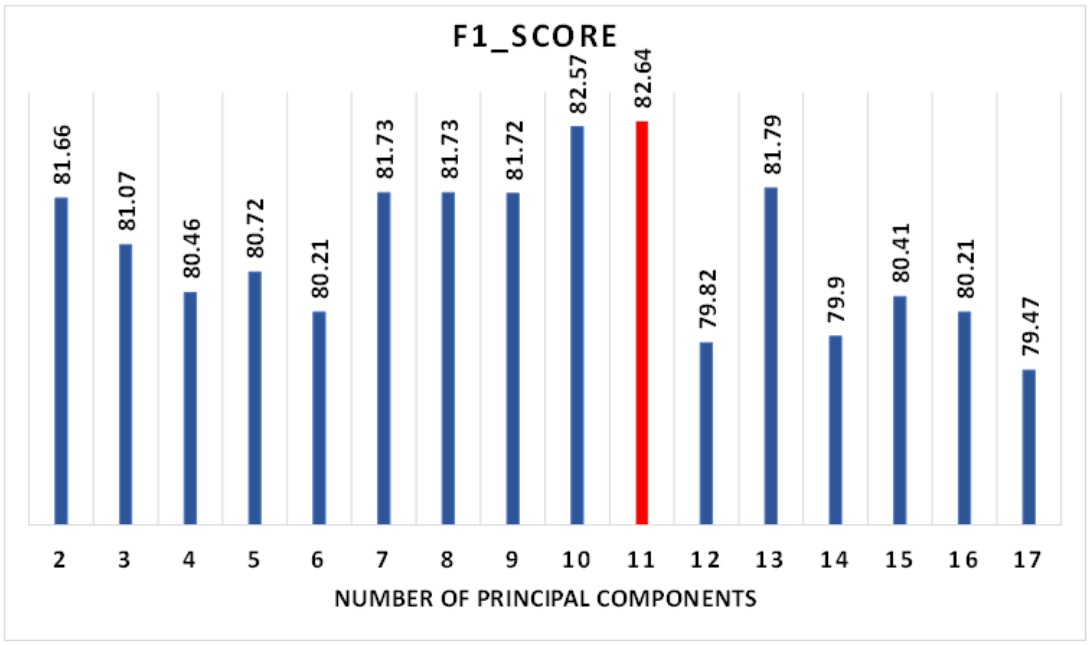


Figure 3.36: NSL-KDD F1-score Results for EE-Based Workflow

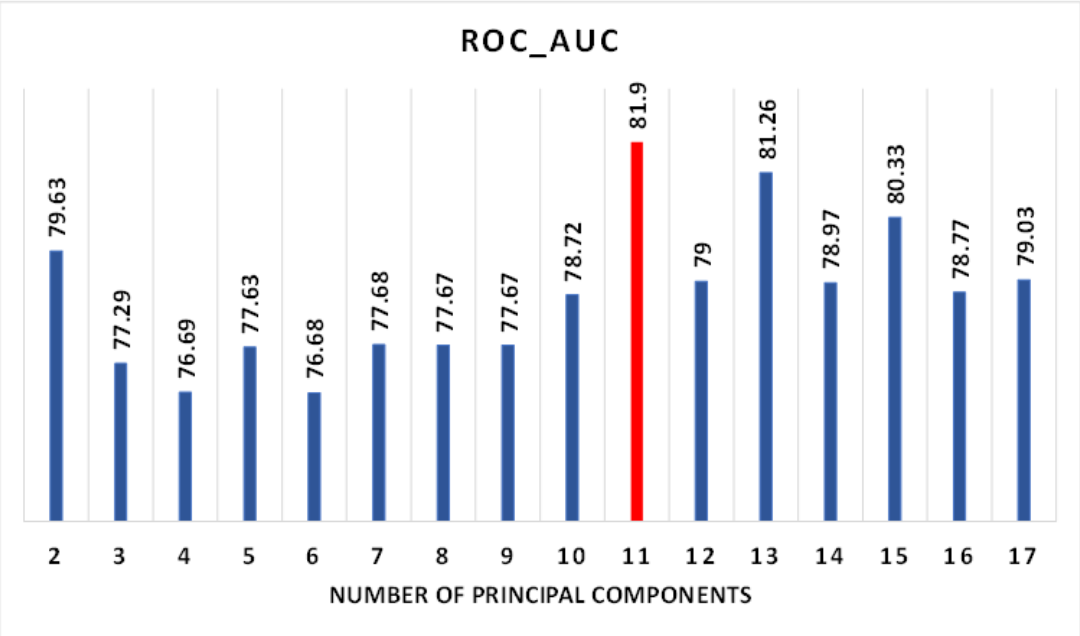


Figure 3.37: NSL-KDD ROC-AUC Results for EE-Based Workflow

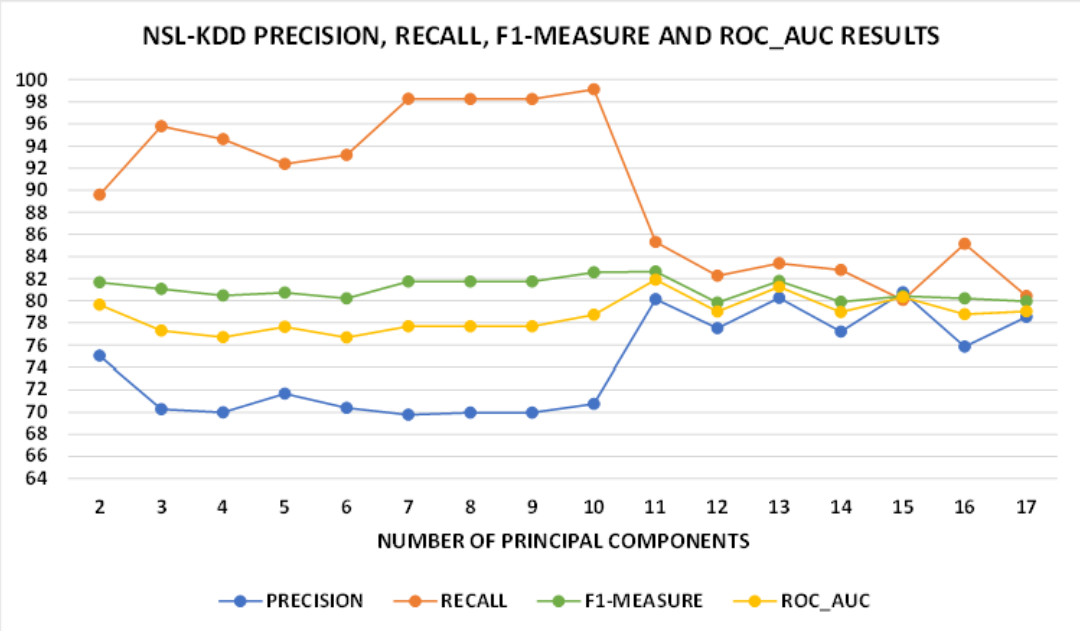


Figure 3.38: NSL-KDD Precision, Recall, F1-score AND ROC-AUC Results For EE

3.6 Initial Experiments: Evaluation and Discussion

The results presented in Table 3.13 and Table 3.14 depict the highest results achieved for the evaluated anomaly detection algorithms for the CICIDS2017 and the NSL-KDD datasets. LOF and iForest were chosen as base anomaly detectors for UNAD.

Table 3.13: Classifiers' Highest Results for the CICIDS2017 (in %)

Classifier	PCA	Accuracy	Precision	Recall	F1-score	ROC-AUC score
LOF	7	85.65	66.19	83.27	73.76	84.84
iForest	11	73.82	47.73	84.86	61.09	77.58
EE	6	66.65	41.90	97.17	58.55	77.06

Table 3.14: Classifiers highest Results for the NSL-KDD (in %)

Classifier	PCA	Accuracy	Precision	Recall	F1-score	ROC-AUC score
LOF	7	85.53	81.05	92.55	86.42	85.57
iForest	16	92.93	93.13	92.63	92.88	92.93
EE	11	81.93	80.13	85.32	82.64	81.90

LOF was chosen because, in the CICIDS2017, it achieved a relatively good F1-score at 7 PCs on its own, at 73.76% and a relatively high recall with 83.27%. Furthermore, the precision of LOF is moderate, at 66.19%. Likewise, LOF achieved comparatively high results in all measures using 7 PCs in the NSL-KDD dataset. The F1-score obtained 86.42%, recall obtained 92.55% and precision was 81.05%.

Concerning the evaluation of the iForest algorithm, for the CICIDS2017 dataset, the F1-score is moderate at 61.09% using 11 PCs. The recall is high at 84.86%, yet precision is relatively low at 47.73%, meaning that about half the anomaly alarms are false alarms. On the other hand, iForest performed very well and achieved high results on the NSL-KDD dataset, with all measures reaching over 90% using 16 PCs. The F1-score was 92.88%, and the results achieved for the recall and precision were 92.63% and 93.13%, respectively.

The EE achieved the lowest F1-score of all anomaly detectors for both datasets. It achieved 58.55% at 6 PCs and 82.64% at 11 PCs for CICIDS2017 and NSL-KDD datasets, respectively. The precision results for CICIDS2017 were 41.90% and 80.13% for the NSL-KDD

dataset. The recall measure had the highest recall among all anomaly detectors. The CICIDS2017 dataset achieved 97.17%, and the NSL-KDD dataset achieved 85.32%.

Overall, EE achieved a very low precision and the lowest F1-score among all anomaly detectors in the CICIDS2017 dataset. Similarly, EE achieved the lowest results on all measures in the NSL-KDD dataset. Furthermore, since EE is more effective on Gaussian distributed datasets, it will not perform well on data streams because the data stream distribution can change over time due to concept drift. Hence, the EE classifier is highly likely to be counterproductive in the UNAD ensemble and, therefore, was excluded.

Although the anomaly detector candidates were optimised with F1-score as a target, ROC-AUC was included in the evaluation metrics since it is frequently used in ML literature. However, the ROC-AUC measure is used in the case of having a balanced dataset. Interestingly, in all cases, using ROC-AUC rather than F1-score would have led to the same outcomes.

3.7 Initial Experiments: Summary

This section summarises the preliminary experiments that evaluated the outlier detection algorithms. Table 3.15 shows the classifiers' hyperparameters range values and the number of steps used in the experiments.

Table 3.15: Classifiers Hyperparameter Range Values

Classifier	Hyperparameter	Range	Step Size
LOF	Contamination	[0-0.5]	0.01
	n_neighbors	[5-50]	5
iForest	Contamination	[0-0.5]	0.01
	n_estimators	50-600	50
	max_samples	auto, 25%-100%	25%
EE	Contamination	[0-0.5]	0.01

Table 3.16 depicts the classifiers' best hyperparameters settings and the chosen principal components for each classifier for the CICIDS2017 dataset.

Table 3.16: CICIDS2017 Best Hyperparameter values and Principal Components

Classifier	PCA	Hyperparameter	Value
LOF	7	Contamination	0.07
		n_neighbors	30
iForest	11	Contamination	0.24
		n_estimators	400
		max_samples	25%
EE	6	Contamination	0.44

Table 3.17 illustrates the classifiers' best hyperparameters settings and the chosen principal components for each classifier for the NSL-KDD dataset.

Table 3.17: NSL-KDD Best Hyperparameter values and Principal Components

Classifier	PCA	Hyperparameter	Value
LOF	7	Contamination	0.14
		n_neighbors	5
iForest	16	Contamination	0.05
		n_estimators	100
		max_samples	100%
EE	11	Contamination	0.26

3.8 Chapter Summary

This chapter introduced the research methodology with its three components to address the research hypotheses. Next, it provided in-depth details regarding each component of the system. The first component was the unsupervised bagging ensemble UNAD, which aims to detect unknown network attacks, and the second is the supervised component which aims to improve the overall detection rate. The third component aims to explain the predictions made by the model to the domain expert locally and globally.

In addition, this chapter introduced the evaluated anomaly detection algorithms. Furthermore, the datasets used to evaluate the system's framework were introduced by providing an in-depth description and analysis of the type and the number of attacks and features for

each dataset. This chapter also explained the preprocessing steps applied for each dataset before training the models. It then presented the results of the evaluated anomaly detection algorithms and the adopted ones. Finally, the initial experiments were summarised, showing the classifiers' best hyperparameter values and principal components used in each dataset.

Chapter 4

Unsupervised Ensemble Learner Architecture for Unknown Attack Detection

This chapter aims to answer *RQ1* and *RQ2* of this thesis. First, it discusses the proposed unsupervised ensemble learner UNAD's architecture and workflow. It then compares its results with stand-alone algorithms LOF and iForest, which performed the best in the F1-score results in the initial experiments. It also provides an empirical evaluation of the UNAD ensemble using two results combiner methods, Majority Voting and Weighted Majority Voting. Furthermore, this chapter compares and summarises the results of these two methods. This chapter concludes by recommending the best methods to use for UNAD as a results combiner, based on the results.

4.1 The UNAD Approach

4.1.1 UNAD Workflow

For the UNAD workflow, both CICIDS2017 and NSL-KDD datasets were preprocessed as described in **Section 3.4**. Next, each dataset was projected on the best number of PCs

achieved in the experiments outlined in Chapter 3. For the CICIDS2017 dataset, that means 7 PCs for the LOF and 11 for the iForest, and for the NSL-KDD dataset, 7 PCs for the LOF and 16 for the iForest. Furthermore, diversity among each type of base learner was created through the bagging ensemble method.

Bagging was chosen over other methods, as discussed in **Section 2.5**, since it reduces variance and thus avoids overfitting [81]. A boosting method may decrease the model's generalisation performance, hence, overfitting the model [167]. In addition, boosting methods perform poorly in noisy datasets [168]. Overall, boosting methods aim to build an ensemble from weak learners that performs well on the dataset, but creating many iterations for the boosting ensemble may produce a very complex classifier, leading to a considerably less accurate model than a stand-alone classifier [167].

Concerning the option of a stack generalisation ensemble method, this method is limited in that the best or optimal combination of base learner classifiers and the meta-classifier must be chosen [169]. Furthermore, tuning the classifiers' hyperparameters in the stack generalisation ensemble is a time-consuming process [169]. Although this problem can be resolved using exhaustive search methods such as genetic algorithms, ant colonies or artificial bee colonies, these methods have the limitation of a high computational complexity [170]. Moreover, the stack generalisation ensemble requires a minimum of three algorithms to create the ensemble two base classifiers and a meta-classifier (2+1). The UNAD ensemble, by contrast, comprises two algorithms only, LOF and iForest, which performed the best in the initial experiments.

For each base learner, bagging was applied to benign/normal data instances. Figure 4.1 illustrates the UNAD's Workflow.

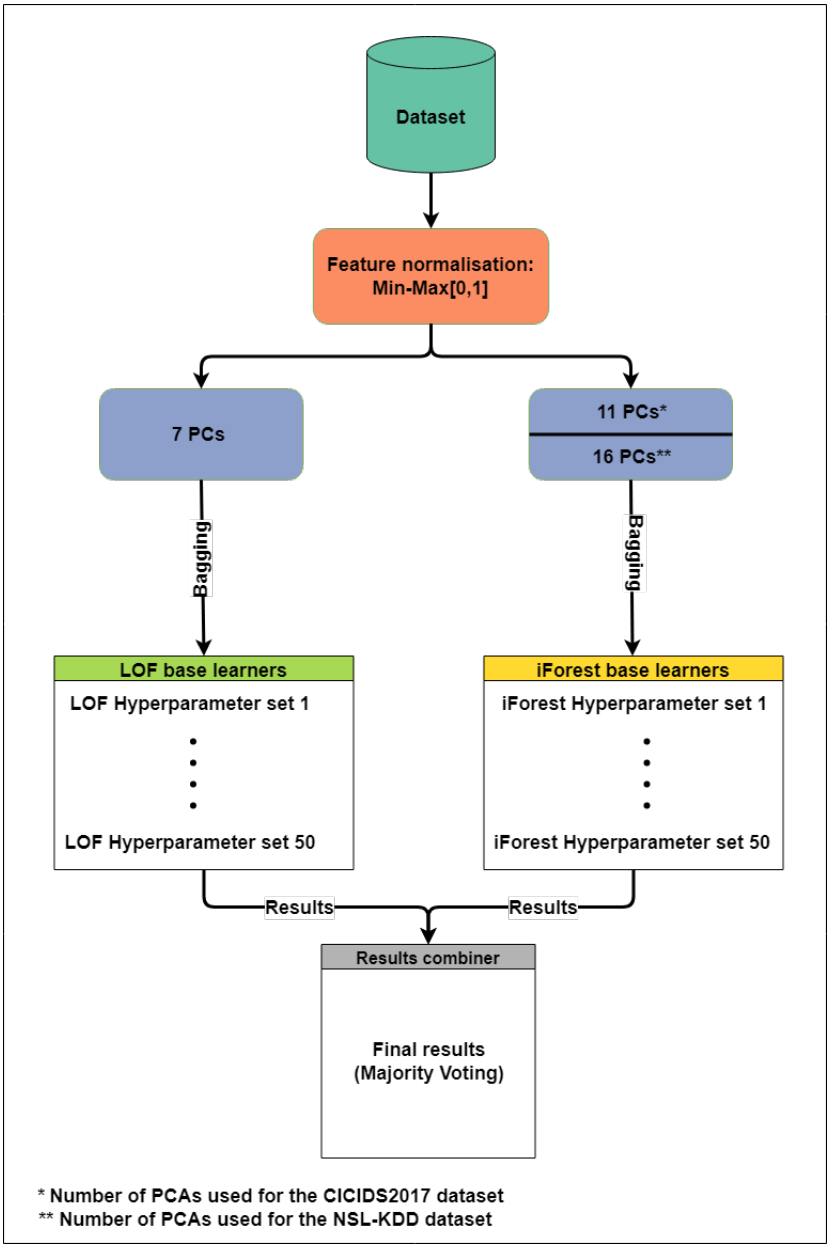


Figure 4.1: Proposed UNAD workflow

UNAD is formed of 100 base learners—50 LOF and 50 iForest; the number of base learners was selected based on the optimum results achieved, which are discussed in the remainder of this chapter. UNAD uses the bootstrap sample with replacement method; hence, each base learner in UNAD is trained on random data samples from the training set. Furthermore, the set of parameters that produced the highest results in the initial experiments stage is used for UNAD's base learners. Although the combination of parameters might be selected again for other base learners, however, as UNAD uses random samples with replacement, these base learners will be trained on different training samples.

UNAD uses a Majority Voting method of all 100 base learners as a results combiner to classify the flow as either benign/normal or attack. Further, there is an equal vote per base learner instance and per classification. The exact number of base learners for LOF and iForest was chosen to mitigate bias towards one type of base learners; hence there is an even number of base learners.

UNAD includes a set of heterogeneous base learners for the LOF algorithm by selecting a different combination of hyperparameter values for each base learner. Concerning the CICIDS2017 dataset, the hyperparameter values that produced the top three results using 7 PCs are considered, which are as follows:

- **Nearest Neighbours**: 25, 30 and 40
- **Contamination**: 0.06, 0.07 and 0.08

Furthermore, for the NSL-KDD dataset, like the CICIDS2017 dataset, the hyperparameter values that produced the top three results using 7 PCs are considered, which are as follows:

- **Nearest Neighbours**: 5
- **Contamination**: 0.14, 0.15 and 0.16

Regarding the iForest algorithm, as with the LOF, UNAD includes a set of heterogeneous base learners for the iForest algorithm by selecting a different combination of hyperparameter values for each base learner.

Concerning the CICIDS2017 dataset, the hyperparameter values that produced the top three results using 11 PCs are considered, which are as follows:

- **Number of estimators**: 150, 350 and 400.
- **Max_samples**: 25%
- **Contamination**: 0.24

Finally, for the NSL-KDD dataset, the hyperparameter values that produced the top three results using 16 PCs are considered, which are as follows:

- **Number of estimators**: 100, 150 and 200.
- **Max_samples**: 100%
- **Contamination**: 0.05 and 0.06.

4.1.2 Experimental Evaluation of UNAD

CICIDS2017 Results

Table 4.1 depicts UNAD experiment results for the CICIDS2017 dataset compared with the best stand-alone LOF and iForest results.

Table 4.1: CICIDS2017 LOF, iForest and UNAD Results Comparison (in %)

Measure(%) Method	**LOF**	**iForest**	**UNAD**
Accuracy	85.65	73.82	87.23
Precision	66.19	47.73	70.99
Recall	83.27	84.86	79.92
F1-score	73.76	61.09	75.19
ROC-AUC	84.84	77.58	84.74

Although the UNAD's recall was slightly lower than that of its stand-alone algorithms (79.92%), LOF and iForest, the precision was considerably improved (70.99%), and there was also improvement in the F1-score (84.74%). The improvement in the F1-score is due to the considerable improvement in the precision results.

Figure 4.2 illustrates the percentage of identified benign cases and detected attacks using the UNAD ensemble.

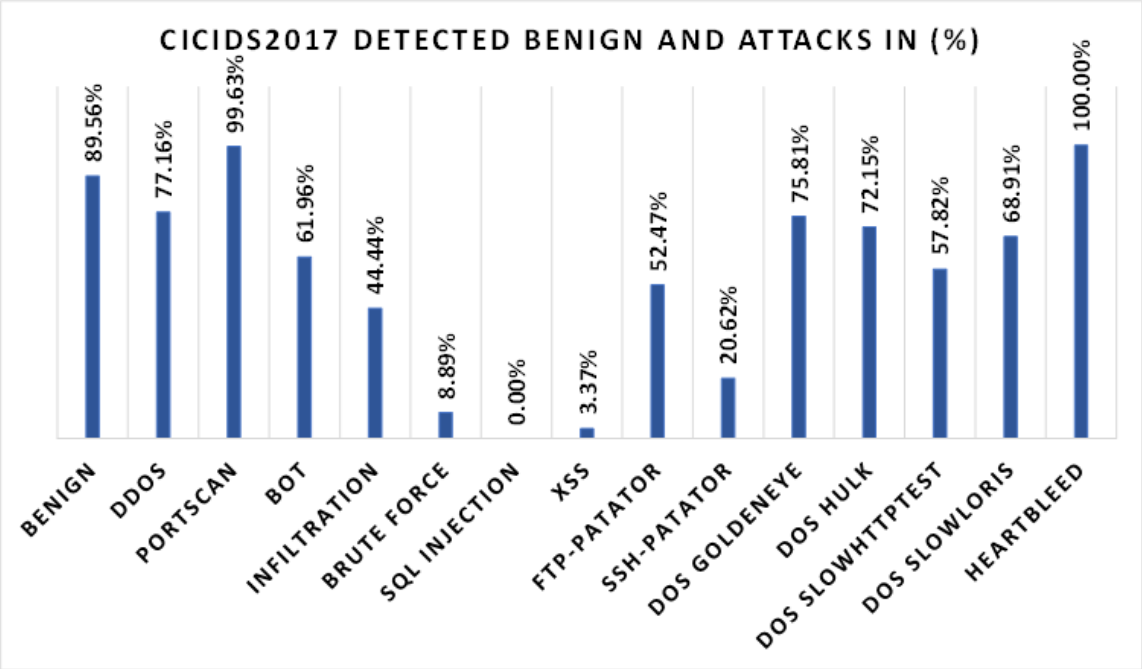


Figure 4.2: UNAD Detected Benign and Attacks on CICIDS2017 Dataset

UNAD detected all the heartbleed attacks and almost all the portscan attacks (99.63%). UNAD was also able to identify 89.56% of the benign flow. DDoS and DoS detection rates were between 77.16% and 72.15% except for DoS Slowloris and DoS Slowhttptest, which were 68.19% and 57.82%, respectively. Attacks under the Web Attack category were the least detected, with 8.89% for Brute Force, 3.37% for XSS, and none of the SQL Injection attacks was detected.

NSL-KDD Results

Table 4.2 depicts the ensemble experiments results for the NSL-KDD dataset compared with the best stand-alone LOF and iForest results.

Table 4.2: NSL-KDD LOF, iForest and UNAD Results Comparison (in %)

Measure(%) Method	**LOF**	**iForest**	**UNAD**
Accuracy	85.53	92.93	93.45
Precision	81.05	93.13	93.90
Recall	92.55	92.63	92.86
F1-score	86.42	92.88	93.38
ROC-AUC	85.57	92.93	93.44

Table 4.2 shows that the iForest algorithm performed better than the LOF algorithms in all measures, with all measures above 92%. Furthermore, UNAD outperformed its base learners on all measures, with all measures over 93% except for the recall, which was 92.86%.

Figure 4.3 shows the percentage of identified normal cases and detected attacks using the UNAD ensemble.

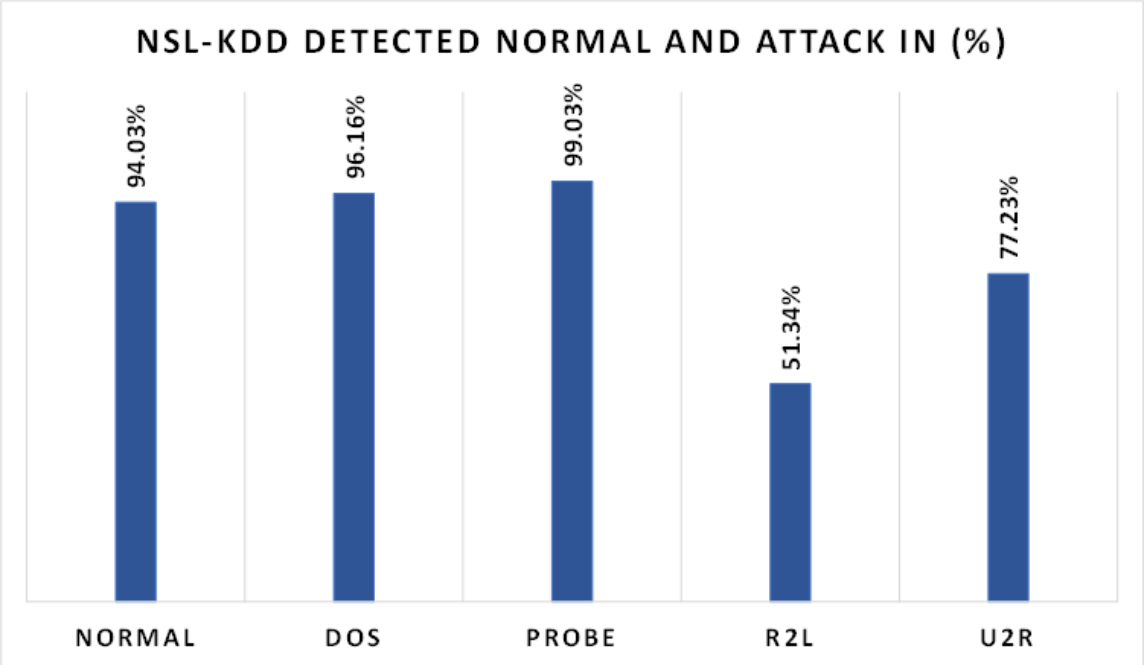


Figure 4.3: UNAD-Detected Benign and Attacks on NSL-KDD Dataset

UNAD detected almost all the probe attacks (99.03%) and identified more than 90% of the normal flow and DoS attacks with 94.03% and 96.16% detection rates. The detection rate for the U2R was high at 77.23%. Finally, UNAD detected over 50% of R2L attacks, which was the lowest detection rate among all other attack types.

Although there were an overall high precision, recall and F1-scores, the proportion of identified attacks varied significantly from one attack type to another. Nevertheless, all attack types were identified by UNAD (except the SQL injection on the CICIDS2017 dataset). However, considering that UNAD had never been introduced to or trained on any of the attacks included in the test set, it performed relatively well, finding all attacks with high precision, recall and F1-score. In addition, almost all attack types are represented by UNAD.

4.1.3 UNAD Current Limitation

Since UNAD uses the Majority Voting as a results combiner and each base learner has equal voting (50 base learners for LOF and 50 for iForest), ties were a significant limitation for UNAD, causing it to abstain from classification when uncertain. The UNAD majority vote combiner is biased towards benign/normal flow, represented by the value of '0' in the label feature in the dataset, meaning that if a tie occurs, UNAD will choose '0' for that instance. Table 4.3 and Table 4.4 depict analysis of the tie instances for both datasets, showing the count and percentage of the abstained attack type.

Table 4.3: CICIDS2017 Traffic Type Instances Abstained from Detection

Type	Count	Percentage (%)
BENIGN	164,062	18.8
DDoS	2,660	4.1
FTP-Patator	1,552	39.1
DoS Slowhttptest	1,157	42.1
SSH-Patator	962	32.6
DoS GoldenEye	882	17.1
DoS Hulk	753	0.7
DoS Slowloris	632	21.8
Web Attack: Brute Force	607	80.5
Web Attack: XSS	303	92.9
PortScan	260	0.3
Bot	19	1.9
Infiltration	8	44.4
Web Attack: SQL Injection	2	20
Total	**173,859**	
Total Abstained (%)	**15.1**	
Ratio Abstained (Benign : Attack)	**17:1**	

Table 4.4: NSL-KDD Traffic Type Instances Abstained from Detection

Type	Count	Percentage (%)
Normal	453	2.6
DoS	137	1.1
Probe	3	0.09
R2L	294	2.7
U2R	1	0.5
Total	**888**	
Total Abstained (%)	**2.5**	
Ratio Abstained (Normal : Attack)	**1:1**	

Table 4.3 shows that UNAD abstained from detecting 173,859 instances, or 15.1% of the entire testing set for the CICIDS2017. Furthermore, the ratio of benign flow to attacks in the abstained data was 17:1. Concerning the NSL-KDD dataset, Table 4.4 indicates that UNAD abstained from detecting 888 instances, or 2.5% of the entire testing set. Additionally, the ratio of normal flow to attacks in the abstained data was equal, with a ratio of 1:1.

4.2 UNAD with Weighted Majority Voting to Overcome Abstaining Limitation

4.2.1 The UNAD WMV Workflow

As previously discussed, UNAD's major limitation is that it abstains from classifying some data instances due to the equal number of base learners. Therefore, to overcome this limitation, a Weighted Majority Voting function was implemented, in which the F1-score of each base learner in the UNAD is used as the weighted vote for each data instance. Hence, the problem of ties and biased voting resulting from the use of Majority Voting in the UNAD ensemble was eliminated. Figure 4.4 shows the updated UNAD workflow in which the Majority Voting method was replaced by the Weighted Majority Voting method as a results combiner.

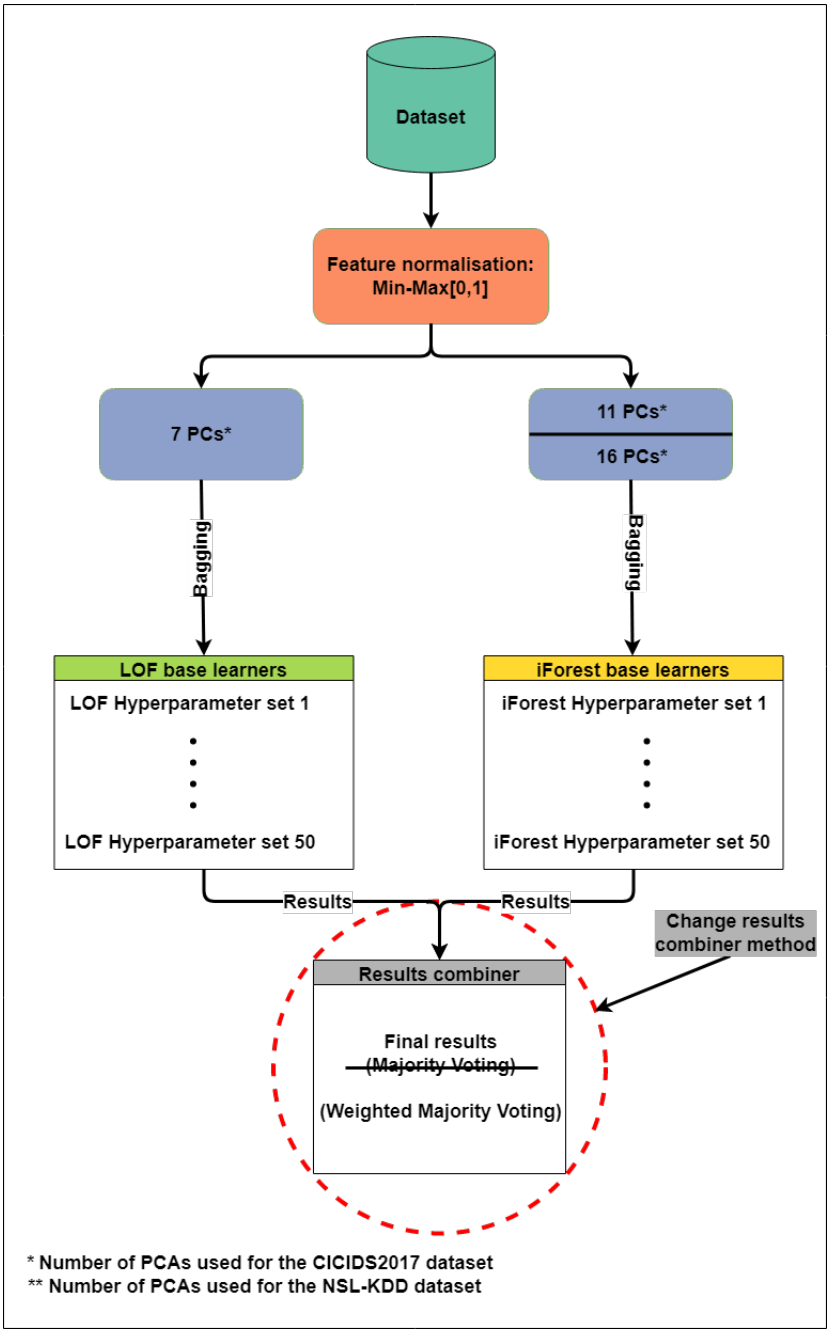


Figure 4.4: updated UNAD Workflow

4.2.2 Comparative Analysis of UNAD with Majority Voting versus UNAD with Weighted Majority Voting

Once the results combiner method was changed, the same experiments were repeated as described in **Section 4.2.1**.

CICIDS2017 Results

Table 4.5 compares the stand-alone algorithms used as base learners in the UNAD, the UNAD results using the Majority Voting method (UNAD MV) as a results combiner and the UNAD results using the Weighted Majority Voting method (UNAD WMV) as a results combiner, all evaluated on the CICIDS2017 dataset.

Table 4.5: Comparison of Stand-alone Algorithms, UNAD MV and UNAD WMV on CICIDS2017

Measure	LOF	iForest	UNAD MV	UNAD WMV
Accuracy	85.65	73.82	87.23	86.84
Precision	66.19	47.73	70.99	69.57
Recall	83.27	84.86	79.92	81.14
F1-score	73.76	61.09	75.19	74.91
ROC-AUC	84.84	77.58	84.74	84.90

The results indicate that UNAD MV method results were marginally higher than the UNAD WMV method results on all measures except for the recall measure. However, UNAD WMV performed better than the stand-alone algorithms, except for the recall measure.

Figure 4.5 illustrates the detection percentage using UNAD MV and UNAD WMV for the CICIDS2017 dataset.

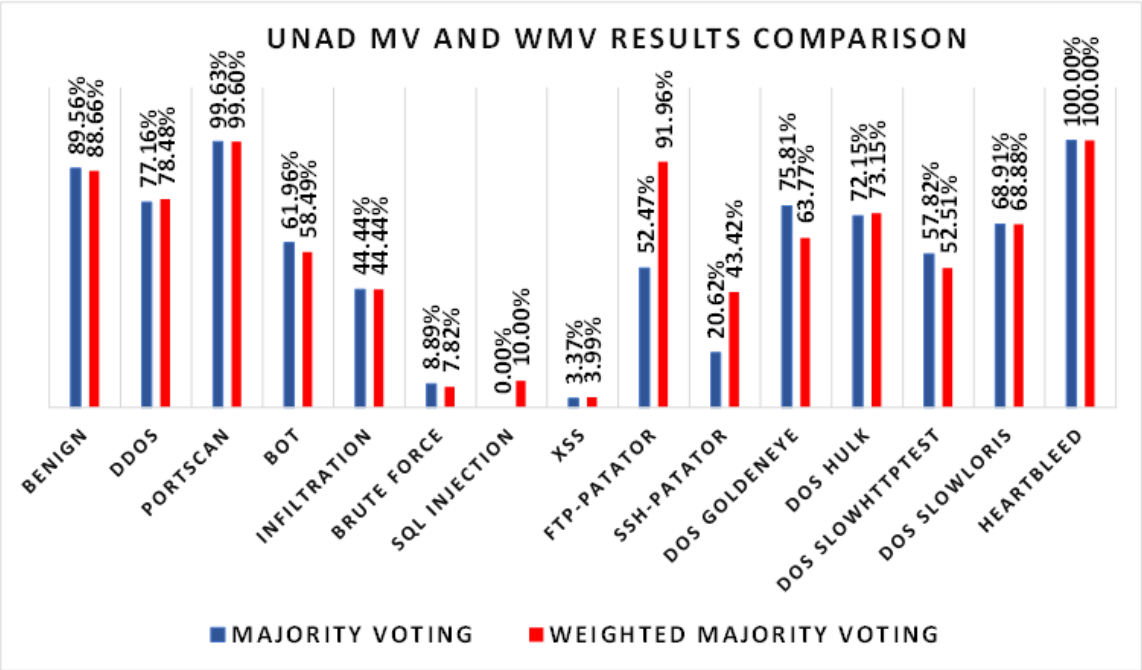


Figure 4.5: Comparison of UNAD MV and WMV Results for CICIDS2017 Dataset

The results for both methods were nearly identical, with an average difference of around 1% for most attacks. MV performed better in detecting Bot and DoS Goldeneye attacks at 61.96% and 75.81%, respectively. MV and WMV showed the same detection rate for Infiltration (44.44%) and Heartbleed attacks (100%). However, WMV outperformed MV in detecting FTP-Patator and SSH-Patator attacks; the detection rate for WMV for the SSH-Patator attack was more than double that of the MV. Finally, WMV detected 10% of the SQL injection attacks whereas, as previously pointed out, UNAD MV did not detect any of the SQL injection attacks.

NSL-KDD Results

Table 4.6 compares the stand-alone algorithms used as base learners in the UNAD, UNAD MV as a results combiner and UNAD WMV as a results combiner, evaluated on the NSL-KDD dataset.

Table 4.6: Comparison of Stand-alone Algorithms, UNAD MV and UNAD WMV on NSL-KDD Dataset

Measure	LOF	iForest	UNAD MV	UNAD WMV
Accuracy	85.53	92.93	93.45	93.22
Precision	81.05	93.13	93.90	93.52
Recall	92.55	92.63	92.86	92.80
F1-score	86.42	92.88	93.38	93.16
ROC-AUC	85.57	92.93	93.44	93.22

The table shows that UNAD MV results were slightly higher than UNAD WMV on all measures. However, UNAD WMV performed better than the stand-alone algorithms.

Figure 4.6 shows the detection percentage using UNAD MV and UNAD WMV for the NSL-KDD dataset.

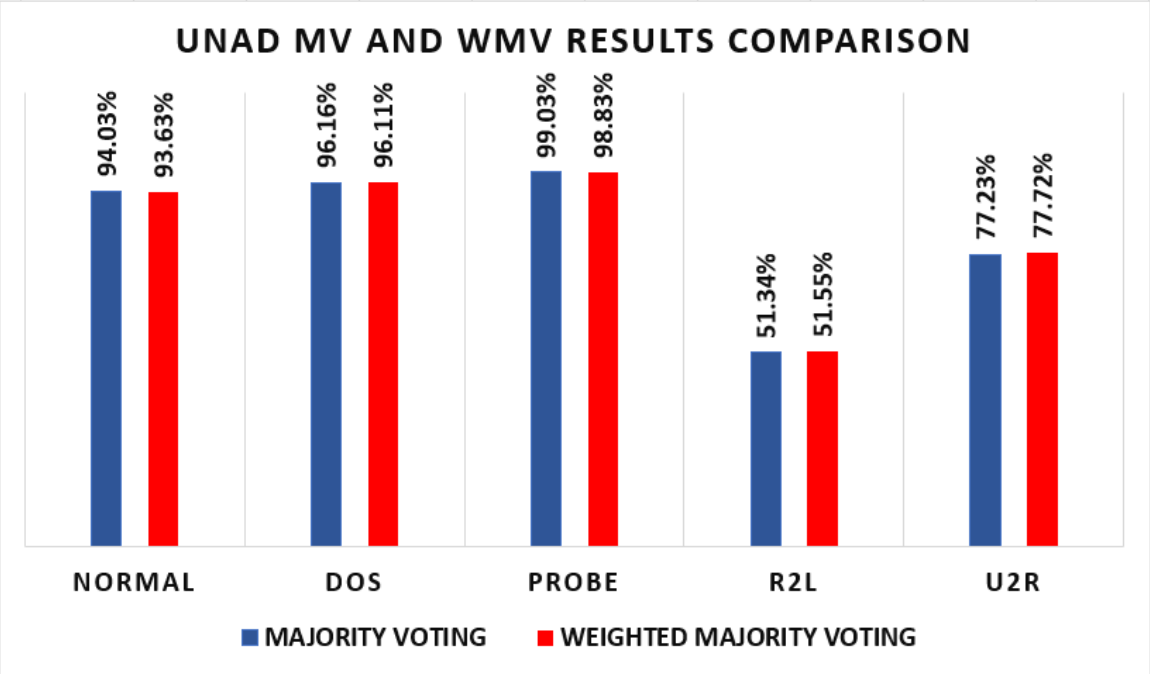


Figure 4.6: UNAD MV and WMV Results Comparison for NSL-KDD Dataset

Probe attacks had the highest detection rate for both methods, with 99.03% for MV and 98.83% for WMV, followed by DoS attacks, with just over 96% detection rate for each. Moreover, the detection rate for normal flow was very high (at 94.09% and 93.63% for MV and WMV, respectively). Furthermore, the U2R attack detection rate was high, just over 77% for each. In contrast, R2L attacks had the lowest detection rate for both methods, with

just over 51%. Overall, both MV and WMV performed nearly the same in detecting all attack types in the NSL-KDD dataset.

In conclusion, although UNAD WMV performed slightly less well than UNAD MV, it can be seen that WMV overcame two major limitations of WV. First, the MV would abstain from voting when encountered ties; this limitation led to the second major limitation, which is that WV was biased towards benign/normal flow when abstaining from voting. This bias explains the slightly higher results, as the number of benign/normal flow is high in the dataset. By contrast, WMV produced more accurate, unbiased results by using the weight as a mechanism for voting. Finally, UNAD WMV was able to detect one SQL injection (1 out 10), which might help the system's second component (the supervised classifier) to detect any future attack of the same type.

4.3 Chapter Summary

This chapter addressed *RQ1* and *RQ2* of this thesis by introducing UNAD, the unsupervised ensemble learner for detecting unknown attacks, which acts as the first component of the system. Furthermore, it explained the workflow and internal process of UNAD, showing the number of principal components and the set of hyperparameters chosen for each dataset. It also compared UNAD with its stand-alone algorithms' results.

MV was used as the results combiner method for UNAD before the problems of abstention from classification when ties were encountered and biased voting surfaced. Thus, WMV was implemented to mitigate these limitations. Furthermore, a comparative analysis has been conducted to compare the results of the two approaches; the results showed that the WMV overcame the MV issue and provided more solid and accurate results. Another significant improvement for UNAD when using WMV as a results combiner was that it detected SQL injection attacks, which UNAD MV could not detect. In conclusion, this chapter shows that the developed UNAD ensemble can detect new attack types that have not been encountered with a high detection rate. The next chapter will introduce and discuss the system's second and third components.

Chapter 5

Improving UNAD Detections and the System Transparency

This chapter aims to answer *RQ3* and *RQ4* of this thesis. First, it presents the supervised model, the second component, where the primary goal is to boost the overall results and improve the detection rate. It also presents the explainable component, the third component, which aims to explain the decision made by the model in a human-understandable way.

First, this chapter explains the second component's workflow and discusses its steps in detail. Next, it presents the evaluated classifiers and compares and summarises their results. In addition, it provides further analysis of the second component's selected classifier by showing the percentage of the detected attacks for both CICIDS2017 and NSL-KDD datasets. Regarding the third component, the chapter discusses and highlights the importance of explainability in ML and the benefits of including it as part of the system. Finally, this chapter describes the two types of explainability included in this thesis—Local and Global—and provides an example of both types.

5.1 Detailed Workflow of the Second Component

Once the UNAD step is completed, the detected data instances in this step are passed to the second component of the system. However, before proceeding with the second component, a domain expert with knowledge of networks and ML techniques checks a subsample of UNAD's results, ensuring that the supervised model is given error-free data instances. Once verified, this data is combined with UNAD's training set, which contains only benign/normal flow. Afterwards, the features are normalised between [0-1] using MinMaxScaler from scikit-learn [147] so that they have equal importance, while keeping the format of the original data distribution.

Next, feature selection was performed to reduce the number of features by eliminating any redundant or irrelevant features, thus enhancing the model performance. In addition, feature selection lowers the computational cost and the required storage [118, 119, 120], which makes it suitable for dealing with a large and complex dataset. For this, the filter feature selection method is considered. The filter method is fast and can help overcome the overfitting issue when training the model [118]. Furthermore, in the filter method, the selection of the features is independent of the ML model, since it can select the relevant features in general regardless of the chosen ML model [118]. Information Gain (IG) feature selection was used; IG measures the dependency between features and labels by calculating the information gained between the features and the class labels [120]. Consequently, a feature is considered relevant to the class if it has a high IG. IG is known for its computational efficiency and simplicity of interpretation [120].

Table 5.1 and Table 5.2 show the IG for the top 30 features for the CICIDS2017 and NSL-KDD datasets, ordered from the highest to the lowest. All CICIDS2017 and NSL-KDD datasets features and their corresponding IG are available in **Appendix C.1** and **Appendix D.1**.

Table 5.1: CICIDS2017 IG for Top 30 Features

No.	Feature	IG	No.	Feature	IG
1	Average Packet Size	0.5794	**16**	Avg Fwd Segment Size	0.4246
2	Packet Length Variance	0.5758	**17**	Fwd Packet Length Mean	0.4245
3	Packet Length Std	0.5753	**18**	Flow Bytes/s	0.3908
4	Packet Length Mean	0.5545	**19**	Flow IAT Max	0.3894
5	Total Length of Bwd Packets	0.5158	**20**	Flow Duration	0.3613
6	Subflow Bwd Bytes	0.5156	**21**	Fwd IAT Max	0.3478
7	Bwd Packet Length Mean	0.5021	**22**	Flow Packets/s	0.3368
8	Avg Bwd Segment Size	0.5020	**23**	Bwd Header Length	0.3368
9	Total Length of Fwd Packets	0.4973	**24**	Fwd Packets/s	0.3346
10	Init_Win_bytes_backward	0.4825	**25**	Fwd IAT Total	0.3343
11	Bwd Packet Length Max	0.4797	**26**	Bwd Packets/s	0.3334
12	Max Packet Length	0.4790	**27**	Fwd Header Length	0.3236
13	Init_Win_bytes_forward	0.4569	**28**	Fwd IAT Mean	0.3154
14	Fwd Packet Length Max	0.4549	**29**	Flow IAT Mean	0.3148
15	Subflow Fwd Bytes	0.4326	**30**	Flow IAT Min	0.2968

Table 5.2: NSL-KDD IG for Top 30 Features

No.	Feature	IG	No.	Feature	IG
1	src_bytes	0.5414	**16**	dst_host_same_src_port_rate	0.2093
2	dst_bytes	0.4302	**17**	flag_s0	0.1929
3	dst_host_same_srv_rate	0.3372	**18**	dst_host_srv_diff_host_rate	0.1874
4	same_srv_rate	0.3274	**19**	service_private	0.1728
5	dst_host_diff_srv_rate	0.3267	**20**	dst_host_count	0.1447
6	diff_srv_rate	0.3266	**21**	dst_host_rerror_rate	0.1209
7	flag_sf	0.3224	**22**	srv_diff_host_rate	0.1150
8	logged_in	0.2890	**23**	dst_host_srv_rerror_rate	0.1150
9	dst_host_srv_count	0.2829	**24**	rerror_rate	0.0950
10	dst_host_serror_rate	0.2574	**25**	srv_rerror_rate	0.0881
11	count	0.2436	**26**	flag_rej	0.0574
12	serror_rate	0.2303	**27**	srv_count	0.0508
13	dst_host_srv_serror_rate	0.2244	**28**	service_domain_u	0.0494
14	srv_serror_rate	0.2108	**29**	duration	0.0374
15	service_http	0.2093	**30**	service_smtp	0.0321

Subsequently, the ratio of attacks to benign/normal flow is evaluated, because the data in the CICIDS2017 dataset, which comprises 529,445 benign flows, is combined with the detected data instances from UNAD's (*TP*) and (*TN*), and the training set in the NSAL-KDD dataset is combined with the detected data instances from UNAD's (*TP*) and (*TN*). This step is crucial, as it determines if the combined dataset needs to be balanced before training the supervised model, as training the model on an imbalanced dataset can bias

model and lead to incorrect results [122]. Table 5.3 depicts the proportion of UNAD's training set, detected attacks and benign/normal data instances in the CICIDS2017 and NSL-KDD datasets and their ratio.

Table 5.3: CICIDS2017 and NSL-KDD Data Distribution Ratio

Dataset	**CICIDS2017**		**NSL-KDD**	
Class/Label	Attacks	Benign	Attacks	Normal
UNAD's training set	0	529,445	0	40,405
UNAD's TP and TN	225,794	772,077	16,224	16,533
Total	225,794	1,301,522	16,224	56,938
Ratio	1	6	1	3.5

Table 5.3 shows that both CICIDS2017 and NSL-KDD datasets are imbalanced. For the CICIDS2017 dataset, the proportion of attacks to benign flow is 1:6, while for the NSL-KDD, the ratio is 1:3.5. Hence, both datasets require balancing before training the model. SMOTE (Synthetic Minority Oversampling TEchnique) was applied to the training set, after the feature selection process. SMOTE is an oversampling method where the minority class is oversampled by adding more synthetic data instances [171]. The advantage of SMOTE is that it synthetically oversamples the minority class rather than creating duplicates; it creates broader decision regions, therefore ensuring more representation of the minority class and improvement for the minority class accuracy [171].

To perform the oversampling of the minority class, the SMOTE class from the imblearn package [172] is used. The hyperparameters are *k_neighbors*, which assigns the number of nearest neighbours to construct synthetic samples, and the *sampling_strategy*, which deals with sampling information to resample the data [172]. The former parameter was left to its default value (k = 5). In contrast, the latter parameter was set to *minority*, hence oversampling the minority class. Furthermore, the *random_state* parameter was set to a fixed number (42) for results reproducibility. Once applied, attack and benign/normal flow had an equal ratio (1:1).

Concerning the second component's classifier selection, four supervised algorithms were considered, which are expected to provide high performance: RF, AdaBoost, NB and KNN. As mentioned in **Section 3.1**, the second component aims to enhance the system's overall detection. The selection of the supervised model is based on the highest F1-score achieved.

For the feature selection, a "for loop" iterates over the ordered features from the feature selection method (highest IG to lowest IG). One feature from the list is added to the loop in each iteration. The first iteration starts with the highest five IG features and repeats by adding one feature every iteration, until the best 30 features are assessed. Furthermore, to avoid overfitting and to accurately assess the evaluated models, 10-fold cross-validation is applied on every iteration [173]. In addition to cross-validation, the Grid Search method is used to optimise the model hyperparameters and search for the best combination of hyperparameters. Once all sets of features and model hyperparameters are evaluated, the model with the highest F1-score will be used as the second component's classifier. Figure 5.1 illustrates the second component's detailed workflow.

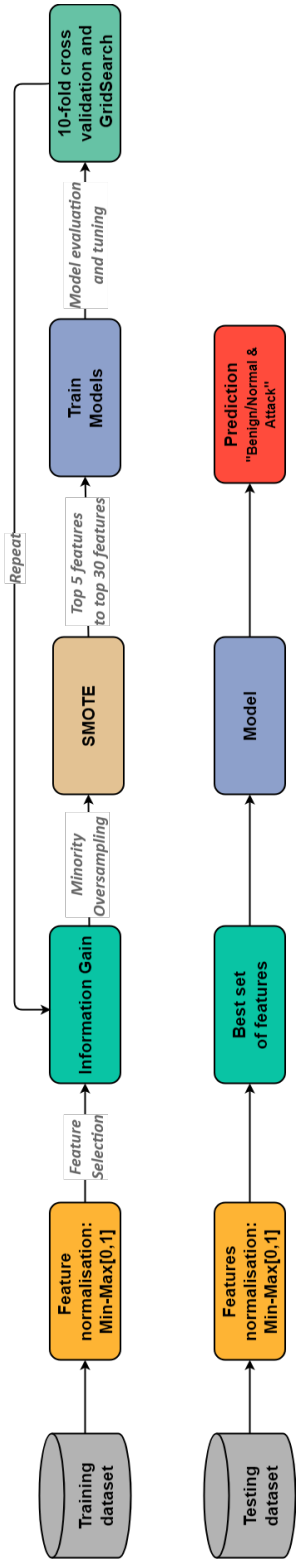


Figure 5.1: Detailed Workflow of Second Component

5.1.1 Research Models' Hyperparameters

The model's hyperparameter tuning is crucial to the model's training phase, as it significantly influences the performance of the ML model. All models were implemented using the scikit-learn library [147]. Hyperparameters for the four models are described in the scikit-learn documentation [147] as follows:

1. ***RandomForest hyperparameters***:
    - **n_estimators**: The number of trees to be built (used) in the forest.
    - **max_depth**: The maximum depth of a tree in the RF.
    - **max_samples**: The number of data samples chosen to train the base estimators.
    - **min_samples_split**: Determines the minimum number of data samples needed to split an internal tree node.
    - **max_features**: Determines the number of random subsets of features to consider when splitting the tree nodes.
    - **min_samples_leaf**: The minimum number of data samples needed to be at a tree leaf node.
2. ***AdaBoost hyperparameters***:
    - **base_estimator**: The algorithm used in the AdaBoost as a base learner.
    - **n_estimators**: The number of base learners used in the AdaBoost ensemble
    - **learning_rate**: Determines the weight applied to each base learner in the boosting process, aiming to reduce the contribution of each base learner in the ensemble.
3. ***Naive Bayes hyperparameters***: No hyperparameters to tune
4. ***KNN hyperparameters***:
    - **n_neighbors**: Number of nearest neighbours needed to classify each data sample

- **weights**: Weight function used in prediction, which has two weighing options:
  - Uniform: All data instances in each neighbourhood have equal weight,
  - Distance: Weight data instances by the opposite of their distance; closer neighbours will have a more significant impact than farther neighbours.

Table 5.4 summarises the classifiers' hyperparameter range values and the number of steps used in the experiments.

Table 5.4: Second Component Classifiers Hyperparameters

Classifier	Hyperparameter	Range	Step Size
RandomForest	n_estimators	100-500	50
	max_depth	Default (None), 5-15	5
	max_samples	Default (None)	-
	min_samples_split	2-8	2
	max_features	Default (sqrt)	-
	min_samples_leaf	Default(1), 2-6	2
AdaBoost	base_estimator	Default (DecisionTreeClassifier)	-
	n_estimators	50-500	50
	learning_rate	0.1 -1	0.1
Naive Bayes	No hyperparameters to tune	-	-
KNN	n_neighbors	5-30	5
	weights	'uniform','distance'	'uniform','distance'

5.1.2 Evaluation and Results of Second Component

This section presents the set of hyperparameters selected for each classifier that provided the highest results, plus the overall results for the CICIDS2017 and NSL-KDD datasets. Table 5.5 shows the classifiers' best set of hyperparameters for the CICIDS2017 and NSL-KDD datasets.

Table 5.5: Classifiers' Best Set of Hyperparameters

Classifier	Hyperparameter	CICIDS2017's value	NSL-KDD's value
RandomForest	n_estimators	100	300
	max_depth	10	15
	max_samples	Default (None)	Default (None)
	min_samples_split	8	4
	max_features	Default (Sqrt)	Default (Sqrt)
	min_samples_leaf	2	1
AdaBoost	base_estimator	Default (DecisionTree)	Default (DecisionTree)
	n_estimators	400	500
	learning_rate	0.9	1
Naive Base	No hyperparameters to tune	-	-
KNN	n_neighbors	15	15
	weights	Uniform	Distance

Table 5.6 and Table 5.7 show the highest results obtained for each classifier on the testing set for CICIDS2017 and NSL-KDD datasets.

Table 5.6: Overall Results: CICIDS2017 Second Component Classifiers (in %)

Classifier	No of features	Accuracy	Precision	Recall	F1-score	ROC-AUC score
RandomForest	18	93.86	96.69	85.22	90.59	91.83
AdaBoost	16	75.85	59.55	94.79	73.15	80.28
Naive Bayes	21	49.12	40.32	97.16	60.37	60.37
KNN	17	84.67	72.52	89.88	80.27	85.89

Table 5.6 shows that the RF classifier achieved the highest results on all measures, using 18 features, compared with the other classifiers with an F1-score of 91.83%. The second best was the KNN classifier, using 17 features with an F1-score of 85.89%. Next came the Adaboost classifier, which achieved an F1-score of 80.28%, using 16 features. Finally, NB had the lowest F1-score of 60.37%, using 21 features.

Table 5.7: Overall Results: NSL-KDD Second Component Classifiers (in %)

Classifier	No of features	Accuracy	Precision	Recall	F1-score	ROC-AUC score
RandomForest	11	74.02	69.76	89.67	78.47	73.10
AdaBoost	23	61.69	63.06	66.22	64.60	61.42
Naive Bayes	10	45.15	48.51	63.51	55.01	44.07
KNN	12	65.04	65.73	70.59	68.07	64.72

Table 5.7 shows that the RF classifier achieved the highest results on all measures, using 11 features, compared to the other three classifiers with an F1-score of 78.47%. Next came the KNN classifier, using 12 features, with an F1-score of 68.07%. However, the F1-score in the RF was significantly higher than in the KNN classifier, with a difference of around 10%. Finally, the AdaBoost classifier achieved an F1-score of 64.60%, using 23 features. NB had the lowest results on all measures, with an F1-score of 55.01

Overall, the RF classifier achieved the highest results on all measures for both datasets, except for the recall measure on the CICIDS2017 dataset, where the NB classifier had the highest recall. However, the RF classifier still provided a high recall (85.22%) and outperformed the NB classifier on all other measures. Therefore, based on the experimental results, the RF classifier is selected as the second component's classifier to boost the overall results and improve the detection rate for the system.

CICIDS2017 Results Analysis

Figure 5.2 illustrates the percentage of identified benign and attacks flow on the CICIDS2017 dataset using the second component's classifier.

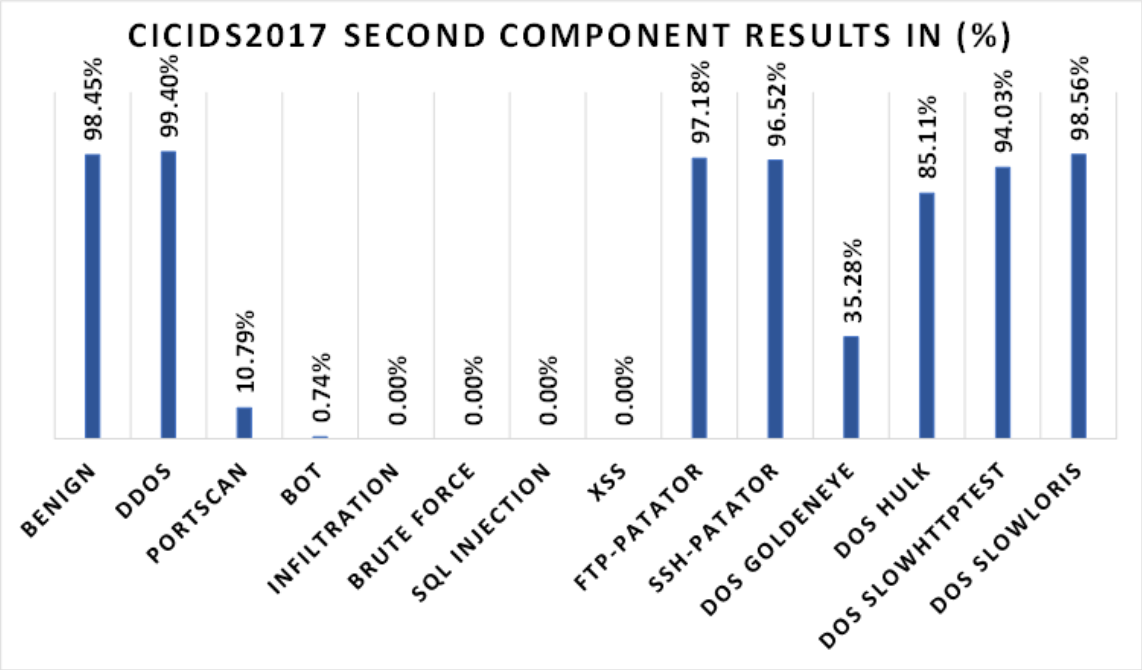


Figure 5.2: Second Component Result Analysis for the CICIDS2017 Dataset

The RF classifier detected most attack types (10 out of 14). It detected almost all DDoS attacks (99.40%). Detection rates for DoS slowloris and benign flows were also high, at 98.56% and 98.45%, respectively, followed by FTP-Patator at 97.18% and SSH-Patator at 96.52%. The detection rate for DoS Slowhttptest was just over 94%, and the detection rate for DoS Hulk was high at 85.11%. DoS GoldenEye and Portscan attacks achieved low results, 35.28% and 10.79%. The Bot attacks' detection rate was under 1%. Finally, the RF classifier could not detect any web attacks (brute force, SQL injection, XSS) or infiltration attacks.

NSL-KDD Results Analysis

Figure 5.3 illustrates the percentage of identified normal and attacks flow on the NSL-KDD dataset using the second component's classifier.

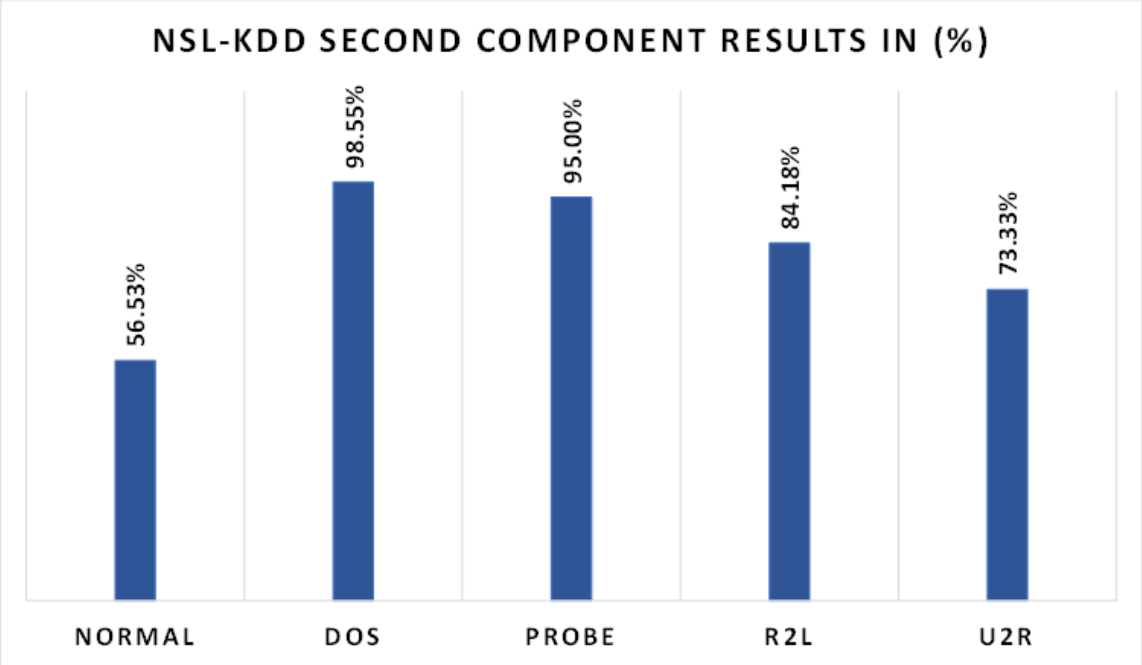


Figure 5.3: Second Component Result Analysis for the NSL-KDD Dataset

The RF classifier detected all attacks on the NSL-KDD attacks. It detected almost all DoS attacks at 98.55% and 95% of the Probe attacks. Furthermore, the detection rate for R2L and U2R attacks was high at 84.18% and 73.33%, respectively. Finally, the RF classifier detected more than half of the normal flow, at 56.53%.

Overall, the RF classifier performed very well with high results for benign/normal and most attack types. Concerning the CICIDS2017 dataset, the results were high for the benign and varied from one attack type to another. For the NSL-KDD, the normal flow result was moderate and high for all attack types. Further discussion of the results for the RF classifier appears in Chapter 6.

5.2 Explainability Component

This section describes the explainable component, the third component of the system. The explainable component provides domain experts with two types of explanations related to the decision made by the classifier. The first type is local explainability, which aims to explain any single prediction made by the model. The second type is global explainability, which aims to explain the entire model. Both are post-hoc explainability methods, which are applied after the second component stage.

The benefit of the local and global explainable components is that they will provide the domain expert with explanations about the decision made by the model. These explanations are essential to confirm model fairness, detect bias in the training data, and verify that the model works as expected [174]. Furthermore, the local and global explainable components allow the model to be tested, audited and debugged, which helps detect faulty model behaviour and identify erroneous predictions, thus improving the model's safety [95]. Furthermore, local and global components will increase domain experts' trust in the system and avoid unclear results by serving as an additional evaluation measure, contributing to the success of the black-box model predictions. The remainder of this section demonstrates how the explainability components help domain experts.

5.2.1 Local Explainability

The local explainable method provides explanations for a single prediction; LIME [3] is used for this purpose. LIME is a reliable algorithm proposed by Ribeiro *et al.* that can explain the predictions of any classifier by matching it locally with an explainable model [3].

As explained by Molnar [88], LIME works by first selecting the desired data sample that needs an explanation. Next, new data samples are generated by perturbing the dataset. Then, it obtains the model predictions for these new data samples and weights the new data samples according to their proximity to the selected data sample. After this, LIME trains the weighted data samples using an explainable ML model. Finally, it explains the prediction using the explainable ML model [88]. The advantages of using LIME are that it is easy to implement and use; it can also be used on various types of data, such as tabular data, text and images [88]. Furthermore, LIME has a reusability functionality, meaning if the black-box ML model is changed, no alteration or modification is required to the LIME settings [88]. Therefore, LIME will illuminate the logic behind the decision made by the model regarding any individual or particular data instance within the testing set. An alternative to LIME is SHAP [175]; however, SHAP is known for its high computational complexity [176].

Explainability Examples on the CICIDS2017 Dataset

Figures 5.4 – 5.7 illustrate examples of model explainability on the CICIDS2017 dataset.

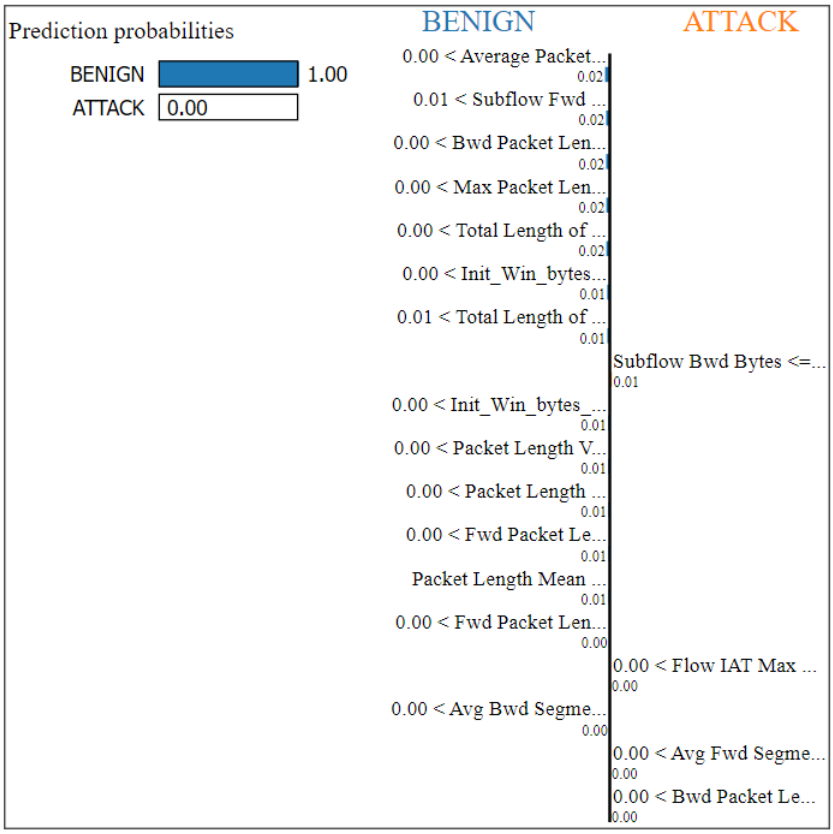


Figure 5.4: Explanation of Correctly Detected Benign Flow on CICIDS2017 Dataset

Figure 5.4 shows an example of correctly detected benign flow. The prediction probability for this instance was 100% towards benign flow. Furthermore, almost all the features selected this instance as benign except *Subflow Bwd Bytes, Flow IAT Max, Avg Bwd Segment Size* and *Bwd Packet Length Max* features.

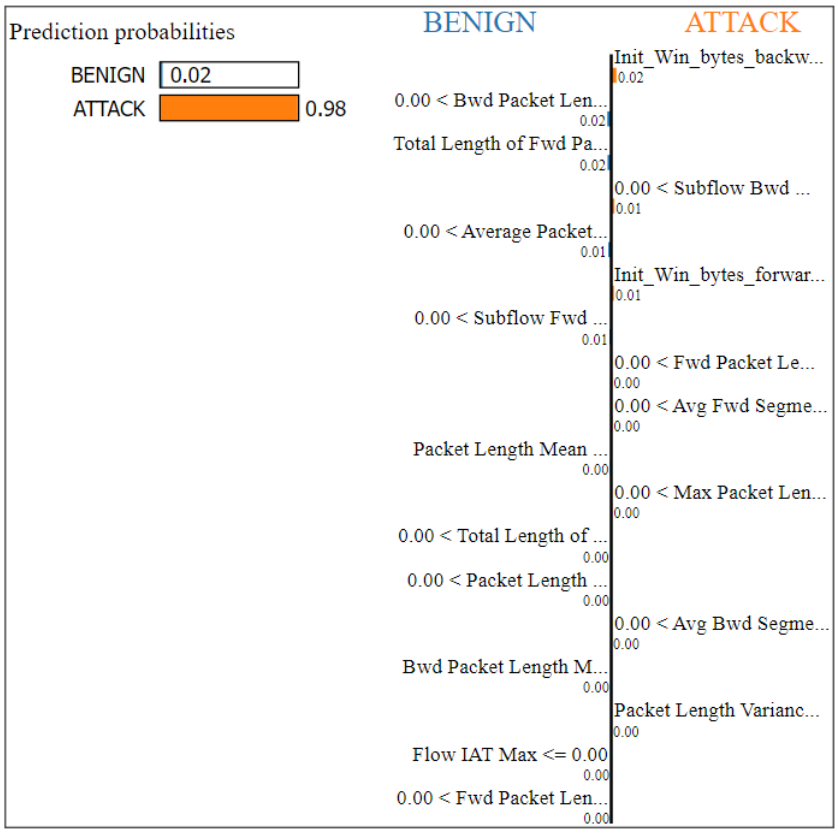


Figure 5.5: Explanation of Correctly Detected Attack Flow on CICIDS2017 Dataset

Figure 5.5 hows an example of a correctly detected FTP-Patator attack instance. The prediction probability for this instance was 98% towards attack flow. Although most of the features selected this instance as benign, the other remaining had a more substantial influence in classifying this instance as attack flow.

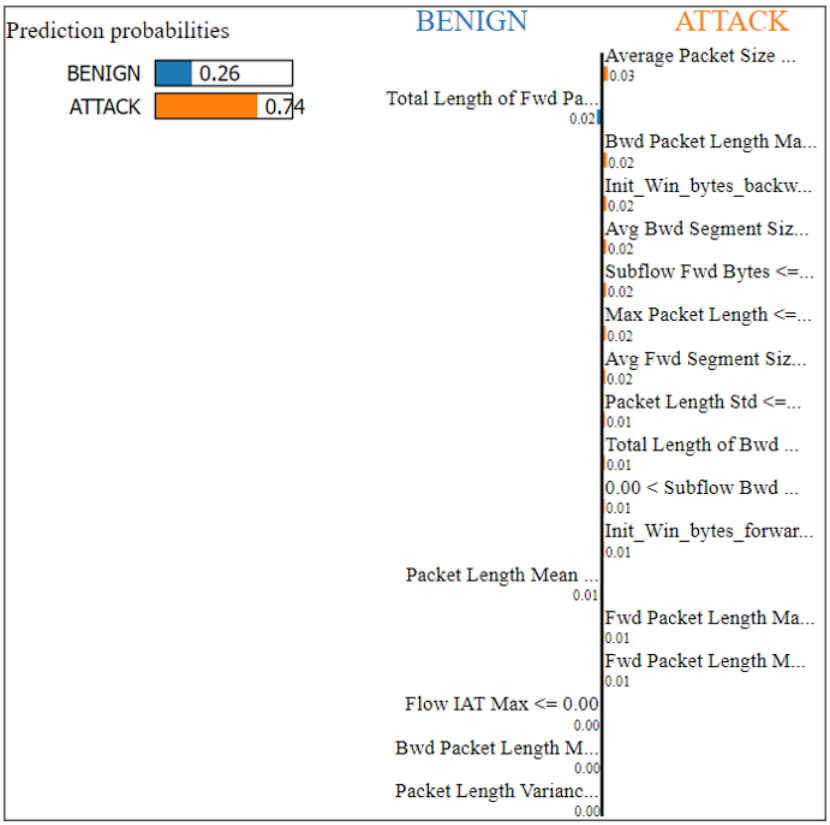


Figure 5.6: Explanation of Incorrectly Detected Benign Flow on CICIDS2017 Dataset

Figure 5.6 shows an example of an incorrectly detected benign flow instance. The prediction probability for this instance was 74% towards attack flow. Furthermore, almost all the features selected this instance as attack except *Total Length of Fwd Packets, Packet Length Mean, Flow IAT Max, Bwd Packet Length Mean* and *Packet Length Variance* features.

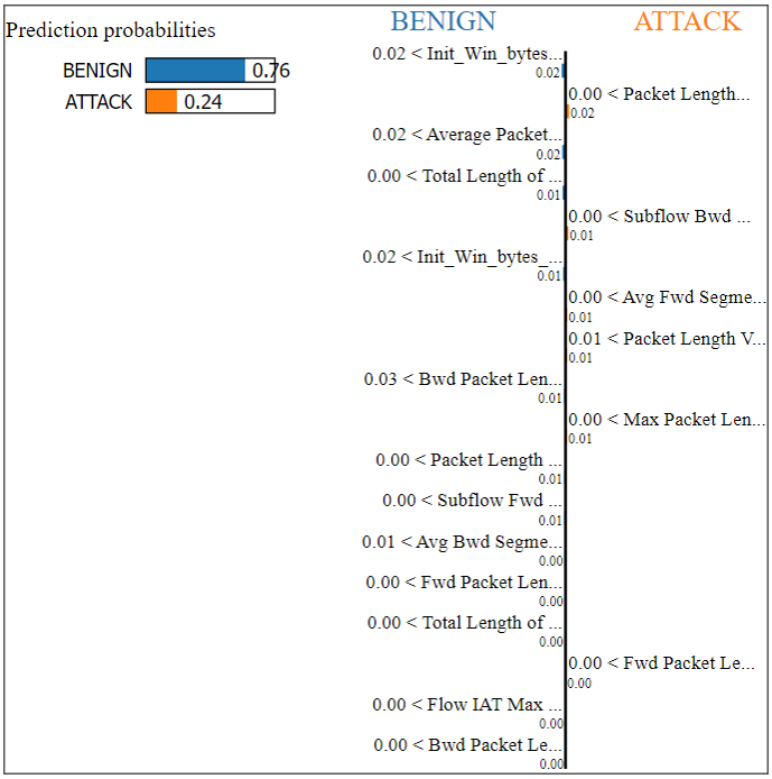


Figure 5.7: Explanation of Incorrectly Detected Attack Flow on CICIDS2017 Dataset

Figure 5.7 illustrates an example of incorrectly detected SSH-Patator attack instance. The prediction probability for this instance was 76% towards benign flow. The incorrect classification occurred because most of the features selected this instance as benign.

Explainability Examples on the NSL-KDD Dataset

Figures 5.8 – 5.11 illustrate examples of model explainability on the NSL-KDD dataset.

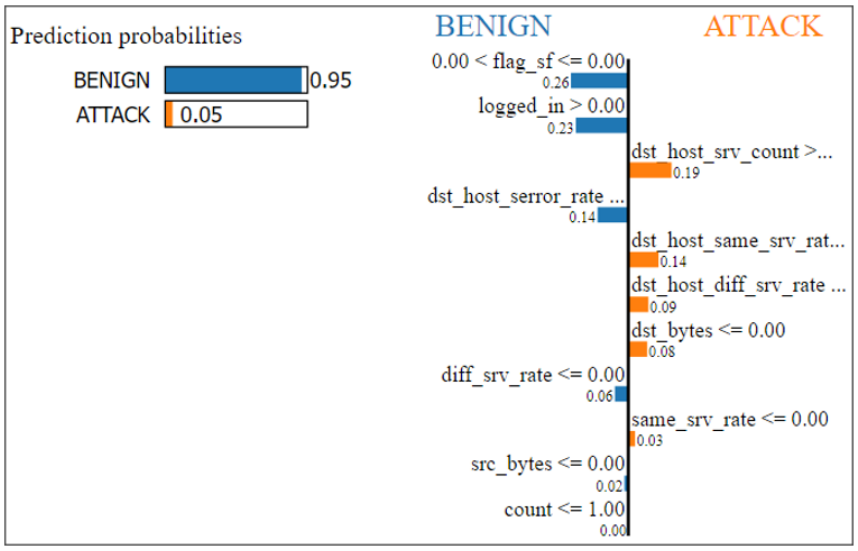


Figure 5.8: Explanation of Correctly Detected Normal Flow on NSL-KDD Dataset

Figure 5.8 shows an example of correctly detected normal flow. The predication probability for this instance was 95% towards normal flow. Further, *flaf_sf, logged_in* and *dst_host_serror_rate* features had the greatest influence on classifying this instance as normal flow.

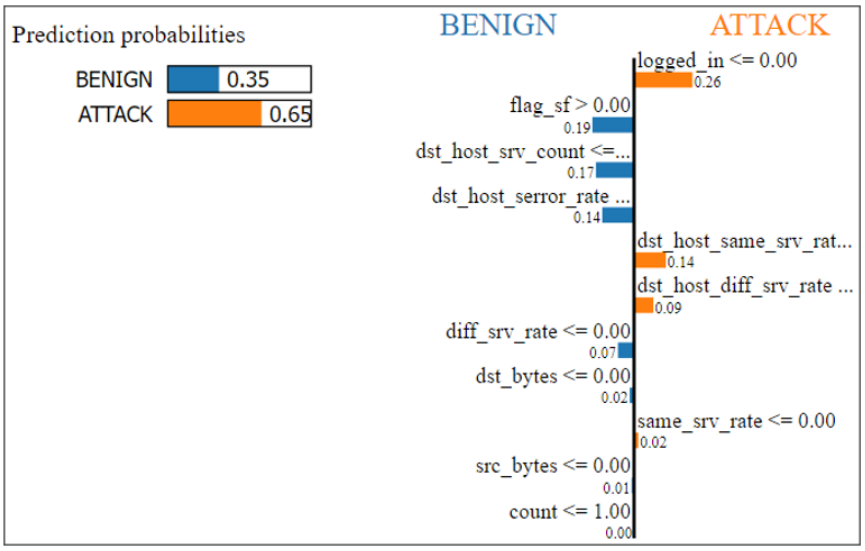


Figure 5.9: Explanation of Correctly Detected Attack Flow on NSL-KDD Dataset

Figure 5.9 shows an example of a correctly detected DoS attack instance. The prediction probability for this instance was 65% towards DoS flow. Although most of the features selected this instance as normal, the *logged_in, dst_host_same_srv_rate, dst_host_diff_srv_rate*

and *same_srv_rate* features were more influential in classifying this instance as a DoS attack.

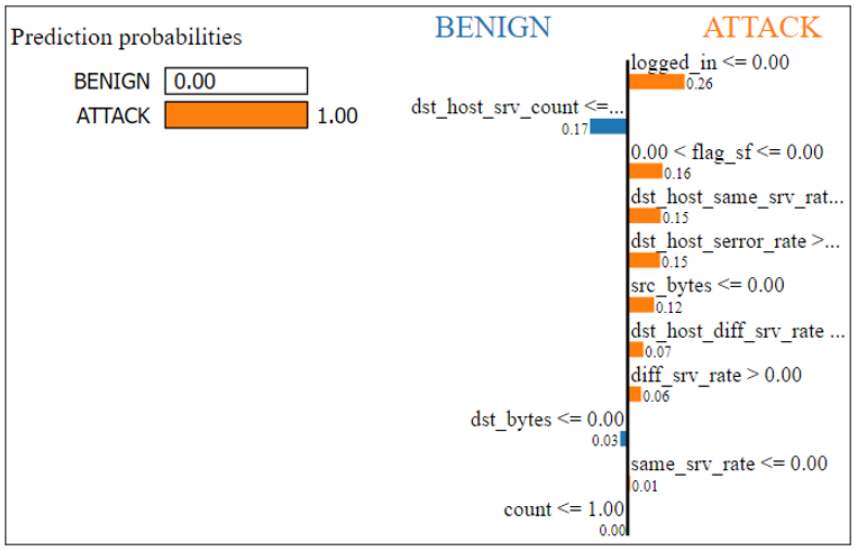


Figure 5.10: Explanation of Incorrectly Detected Normal Flow on NSL-KDD Dataset

Figure 5.10 shows an incorrectly detected normal flow instance. The prediction probability for this instance was 100% towards attack flow. Furthermore, almost all the features selected this instance as attack except *dst_host_srv_count, dst_bytes* and *count*.

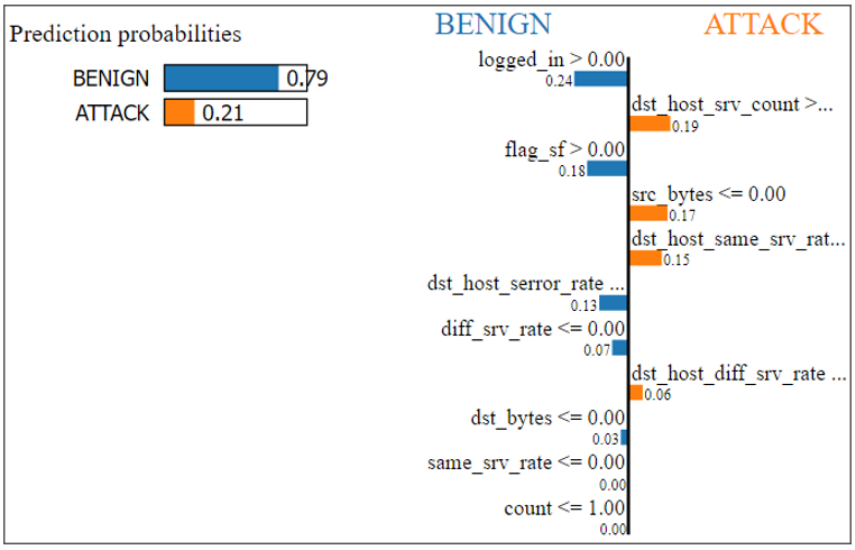


Figure 5.11: Explanation of Incorrectly Detected Attack Flow on NSL-KDD Dataset

Figure 5.11 illustrates an example of incorrectly detected DoS attack instance. The predication probability for this instance was 79% towards normal flow. The reason for the

incorrect classification was that most of the features selected this instance as normal, except *dst_host_srv_count, src_bytes, dst_host_same_srv_rate* and *dst_host_diff_srv_rate* features.

5.2.2 Global Explainability

Global explainability aims to provide a comprehensive explanation of the model. For this, the global surrogate method is adopted; this method explains a model using a self-explanatory model (intrinsic model) [99, 100, 177]. Examples include linear, decision tree and rule-based models [178].

The global surrogate method task begins by training an explainable model on the same dataset used to train the black-box model [179]. Next, predictions of the black-box model as the target dataset (the testing set) for the explainable model are assigned. The final step is an evaluation, known as fidelity, of how well the explainable model can approximate the black-box model predictions [179].

DT has been used as the explainable model for the global surrogate method. DT is a transparent algorithm that can extract rules in a human-readable way, explaining the knowledge obtained from the black-box model [91]. For this, a rule extractor function is implemented, which extracts the rules of the DT in a human-readable format and generates a CSV report for a domain expert to explain the model predictions and assess the results. The DT was implemented using its default hyperparameters settings, which are the same as the RF hyperparameters except for the *n_estimators* parameter, which is excluded. Descriptions of the RF hyperparameters were provided in **Section 5.1.1**. For model fidelity, the accuracy measure was used as an evaluation metric. The higher the accuracy score, the better the explainable model can approximate the black-box model. The accuracy score achieved for CICIDS2017 and NSL-KDSS datasets was over 99%, meaning that the DT model can very accurately approximate the RF model predictions.

Figures 5.12 and 5.13 illustrate the DT and network flow analysis report for CICIDS2017, and Figures 5.14 and 5.15 illustrate the DT and network flow analysis report for NSL-KDD.

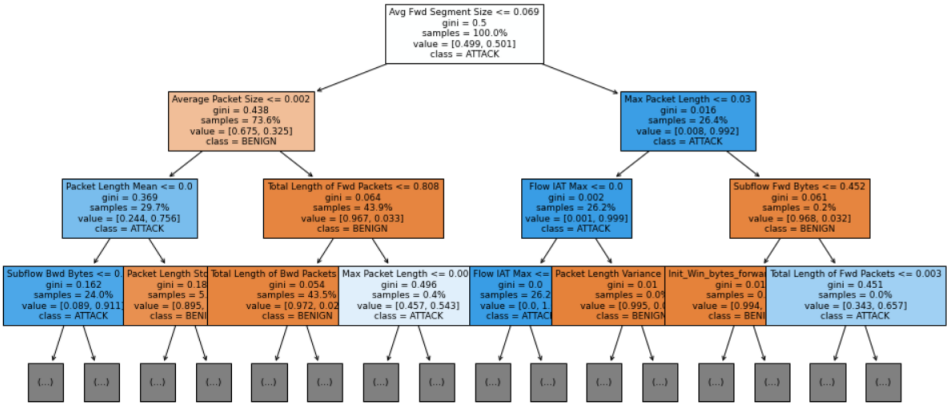


Figure 5.12: CICIDS2017 Decision Tree

	A
1	**Network Flow Analysis Report**
2	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets > 0.002 AND the Subflow Fwd Bytes <= 0.334 AND the Packet Length Mean <= 0.003 AND the To
3	IF the Avg Fwd Segment Size > 0.069 AND the Max Packet Length <= 0.03 AND the Flow IAT Max <= 0.0 AND the Flow IAT Max > 0.0 AND the Packet Length Mean <= 0.004 AND the Packet Length Mean <= 0.004 THEN the flow is an ATTACK
4	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean <= 0.0 AND the Subflow Bwd Bytes > 0.004 AND the Packet Length Std > 0.0 AND the Subflow Bwd Bytes <= 0.452 AND the Subflow Bwd Bytes > 0.00
5	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets > 0.002 AND the Subflow Fwd Bytes <= 0.334 AND the Packet Length Mean > 0.003 AND the Pac
6	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets <= 0.002 AND the Subflow Fwd Bytes <= 0.001 AND the Packet Length Std <= 0.019 AND the Ma
7	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean > 0.0 AND the Packet Length Std <= 0.0 AND the Packet Length Mean > 0.004 AND the Init_Win_bytes_forward <= 0.0 AND the Total Length of Fwd Pa
8	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean <= 0.0 AND the Subflow Bwd Bytes > 0.004 AND the Packet Length Std > 0.0 AND the Subflow Bwd Bytes <= 0.452 AND the Subflow Bwd Bytes > 0.00
9	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean <= 0.0 AND the Subflow Bwd Bytes > 0.004 AND the Packet Length Std <= 0.0 AND the Subflow Bwd Bytes <= 0.005 AND the Total Length of Fwd Pack
10	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean > 0.0 AND the Packet Length Std <= 0.0 AND the Packet Length Mean <= 0.004 AND the Packet Length Mean <= 0.004 AND the Total Length of Fwd Pa
11	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean <= 0.0 AND the Subflow Bwd Bytes > 0.004 AND the Packet Length Std <= 0.0 AND the Subflow Bwd Bytes > 0.005 AND the Total Length of Fwd Packe
12	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size <= 0.002 AND the Packet Length Mean <= 0.0 AND the Subflow Bwd Bytes <= 0.004 AND the Total Length of Bwd Packets <= 0.001 AND the Total Length of Fwd Packets <= 0.799 AND the Fw
13	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets <= 0.002 AND the Subflow Fwd Bytes > 0.001 AND the Max Packet Length <= 0.001 AND the Sub
14	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets > 0.002 AND the Subflow Fwd Bytes <= 0.334 AND the Packet Length Mean > 0.003 AND the Pac
15	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets > 0.002 AND the Subflow Fwd Bytes <= 0.334 AND the Packet Length Mean <= 0.003 AND the To
16	IF the Avg Fwd Segment Size <= 0.069 AND the Average Packet Size > 0.002 AND the Total Length of Fwd Packets <= 0.808 AND the Total Length of Bwd Packets <= 0.002 AND the Subflow Fwd Bytes > 0.001 AND the Max Packet Length > 0.001 AND the Avg

Figure 5.13: CICIDS2017 Network Flow Analysis Report

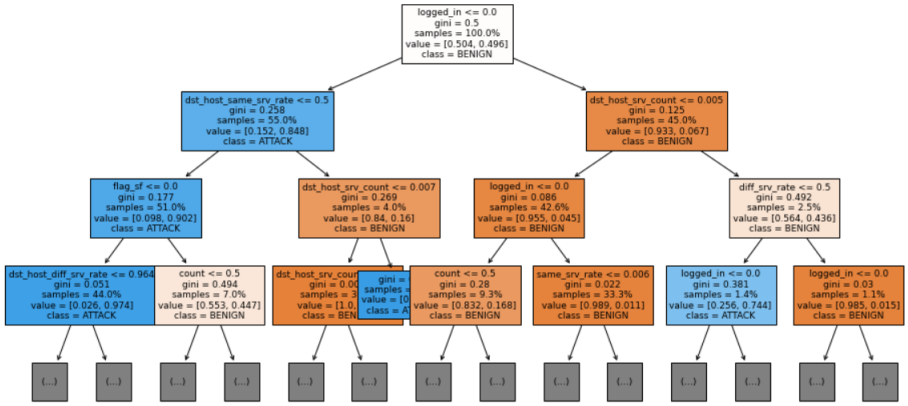
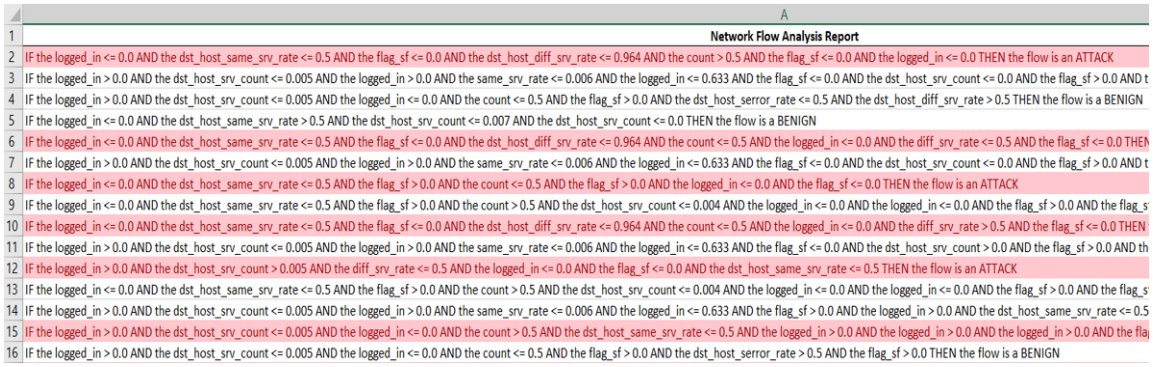


Figure 5.14: NSL-KDD Decision Tree



	A
1	**Network Flow Analysis Report**
2	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate <= 0.5 AND the flag_sf <= 0.0 AND the dst_host_diff_srv_rate <= 0.964 AND the count > 0.5 AND the flag_sf <= 0.0 AND the logged_in <= 0.0 THEN the flow is an ATTACK
3	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in > 0.0 AND the same_srv_rate <= 0.006 AND the logged_in <= 0.633 AND the flag_sf <= 0.0 AND the dst_host_srv_count <= 0.0 AND the flag_sf > 0.0 AND t
4	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in <= 0.0 AND the count <= 0.5 AND the flag_sf > 0.0 AND the dst_host_serror_rate <= 0.5 AND the dst_host_diff_srv_rate > 0.5 THEN the flow is a BENIGN
5	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate > 0.5 AND the dst_host_srv_count <= 0.007 AND the dst_host_srv_count <= 0.0 THEN the flow is a BENIGN
6	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate <= 0.5 AND the flag_sf <= 0.0 AND the dst_host_diff_srv_rate <= 0.964 AND the count <= 0.5 AND the logged_in <= 0.0 AND the diff_srv_rate <= 0.5 AND the flag_sf <= 0.0 THEN
7	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in > 0.0 AND the same_srv_rate <= 0.006 AND the logged_in <= 0.633 AND the flag_sf <= 0.0 AND the dst_host_srv_count <= 0.0 AND the flag_sf > 0.0 AND t
8	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate <= 0.5 AND the flag_sf > 0.0 AND the count <= 0.5 AND the flag_sf > 0.0 AND the logged_in <= 0.0 AND the flag_sf <= 0.0 THEN the flow is an ATTACK
9	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate <= 0.5 AND the flag_sf > 0.0 AND the count > 0.5 AND the dst_host_srv_count <= 0.004 AND the logged_in <= 0.0 AND the logged_in <= 0.0 AND the flag_sf > 0.0 AND the flag_s
10	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate <= 0.5 AND the flag_sf <= 0.0 AND the dst_host_diff_srv_rate <= 0.964 AND the count <= 0.5 AND the logged_in <= 0.0 AND the diff_srv_rate > 0.5 AND the flag_sf <= 0.0 THEN
11	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in > 0.0 AND the same_srv_rate <= 0.006 AND the logged_in <= 0.633 AND the flag_sf <= 0.0 AND the dst_host_srv_count > 0.0 AND the flag_sf > 0.0 AND th
12	IF the logged_in > 0.0 AND the dst_host_srv_count > 0.005 AND the diff_srv_rate <= 0.5 AND the logged_in <= 0.0 AND the flag_sf <= 0.0 AND the dst_host_same_srv_rate <= 0.5 THEN the flow is an ATTACK
13	IF the logged_in <= 0.0 AND the dst_host_same_srv_rate <= 0.5 AND the flag_sf > 0.0 AND the count > 0.5 AND the dst_host_srv_count <= 0.004 AND the logged_in <= 0.0 AND the logged_in <= 0.0 AND the flag_sf > 0.0 AND the flag_s
14	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in > 0.0 AND the same_srv_rate <= 0.006 AND the logged_in <= 0.633 AND the flag_sf > 0.0 AND the logged_in > 0.0 AND the dst_host_same_srv_rate <= 0.5
15	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in <= 0.0 AND the count > 0.5 AND the dst_host_same_srv_rate <= 0.5 AND the logged_in > 0.0 AND the logged_in > 0.0 AND the fla
16	IF the logged_in > 0.0 AND the dst_host_srv_count <= 0.005 AND the logged_in <= 0.0 AND the count <= 0.5 AND the flag_sf > 0.0 AND the dst_host_serror_rate > 0.5 AND the flag_sf > 0.0 THEN the flow is a BENIGN

Figure 5.15: NSL-KDD Network Flow Analysis Report

Once the DT graph is generated (Figures 5.12 and 5.14) the rule extractor function extracts the rules and converts them into a CSV report (Figures 5.13 and 5.15). In this report, each row represents a rule in conditional format. Furthermore, each rule comprises the features and their range of values. If a rule results in an attack, then the rule row is highlighted in red in the CSV report.

5.3 Chapter Summary

This chapter answered *RQ3* and *RQ4* of this thesis by presenting the second and third components of the system. Concerning the second component, four different supervised classifiers were evaluated. The classifiers were trained and optimised on UNAD's correctly detected attacks, plus the training set used to train UNAD, which comprised only benign/normal flow. Next, the classifiers were evaluated on UNAD's not-detected data instances. The classifiers were assessed in terms of the F1-score measure. The RF classifier produced the highest result of the three; thus, it was selected for the system's second component model. Data analysis showed that using a supervised classifier improves the detection of most attack types.

For the third component, two types of explainability have been introduced—local and global—to provide the domain expert with explanations about the decision made by the model. For local explainability, LIME was implemented, which provides explanations for a single prediction, while global explainability provides a comprehensive explanation of the model. For this, the global surrogate method was adopted, and DT was used as the

explainable model, in which, after generating the tree, a rule extractor function extracts the rules and generates a human-readable report for the domain expert. This component can help domain experts assess a threat level and understand how the model made its decisions. The following chapter will present and discuss the thesis's overall boosted results.

As previously mentioned, with enough experience in networking and ML techniques, a human expert can use the local and global parts interchangeably. For example, in Figure 5.9, the flow was misclassified as an attack flow (DoS) instead of a normal flow. For this, the human expert can compare the set of features that influenced the misclassification from the local explanation part with the list of rules report from the global part, highlight any inconsistencies between the two parts, and find what contributed to the misclassification. This way, the human expert will obtain the knowledge/insight to audit the system, hence improving the system's overall performance, re-train and re-tune the system to avoid any future prediction errors, and helping the system correctly classify attack types.

Chapter 6

Overall Results and Discussion

This chapter aims to show the overall boosted results after implementing the second component of the system and combining its results with UNAD. Chapter 6 will present UNAD, the second component classifier and the overall boosted results for both CICIDS2017 and NSL-KDD datasets. This chapter revisits the data analysis from Chapters 4 and 5 to compare these results with the overall obtained results. Furthermore, this chapter provides a detailed analysis of the detection rate for benign/normal and all attack types after combining the second component classifier results with UNAD results. Finally, this chapter discusses possible reasons why some of the attacks in the CICIDS2017 dataset were not detected by the second component classifier.

6.1 CICIDS2017 Results and Analysis

Table 6.1 illustrates the results for UNAD, the second component classifier and the overall boosted results after combining the second component classifier's results with UNAD's results on the CICIDS2017 dataset.

Table 6.1: CICIDS2017 Overall Results (in %)

Measure	UNAD WMV	Second component classifier	Overall results
Accuracy	86.84	93.86	99.19
Precision	69.57	96.69	99.44
Recall	81.14	85.22	99.21
F1-score	74.91	90.59	98.31
ROC-AUC	84.90	91.83	98.52

The overall obtained results were significantly higher for all measures. F1-score results improved from 74.91% to 98.31%, as the second component managed to detect 90.59% of UNAD's not-detected attacks. The precision measure results were also considerably improved, from 69.57% on UNAD to 99.44%. Furthermore, the overall results for the recall measure improved from 81.14% to 99.21%. Finally, the system achieved an overall accuracy of 99.19% and ROC-AUC 98.52%.

Figures 6.1 and 6.2 revisit the results analysis for UNAD (*C1*) and the second component classifier (*C2*) for the CICIDS2017 dataset. Figure 6.3 represents the overall boosted results using the second component classifier for the CICIDS2017, while Figure 6.4 shows the results of UNAD, the second component and the overall results.

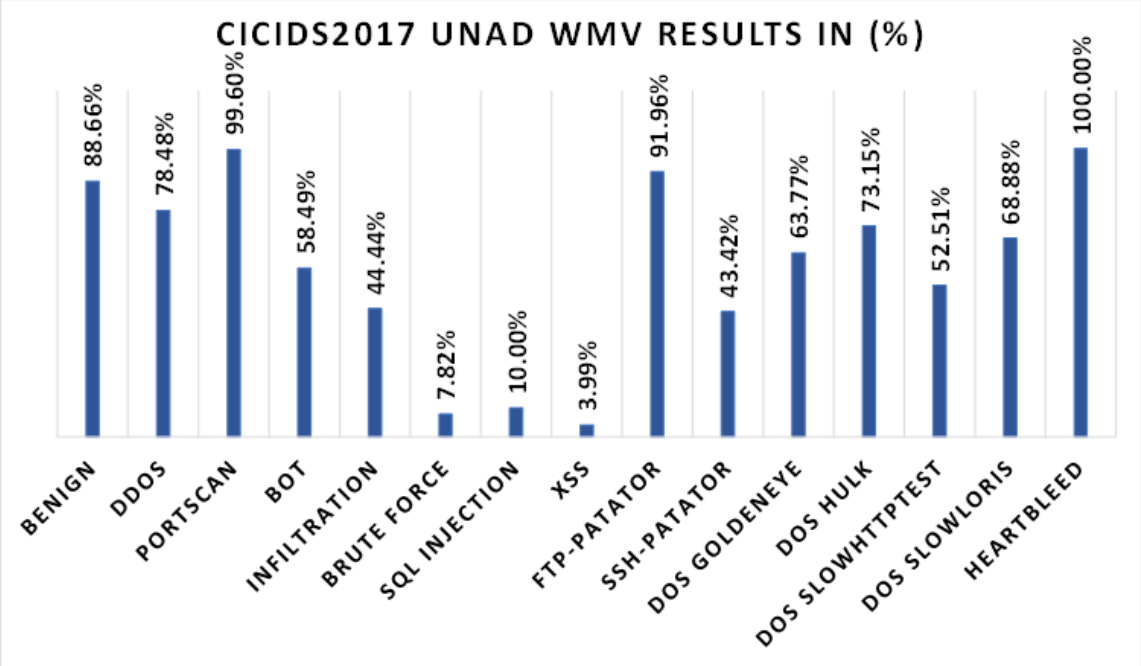


Figure 6.1: UNAD WMV Results for CICIDS2017 Dataset (in %)

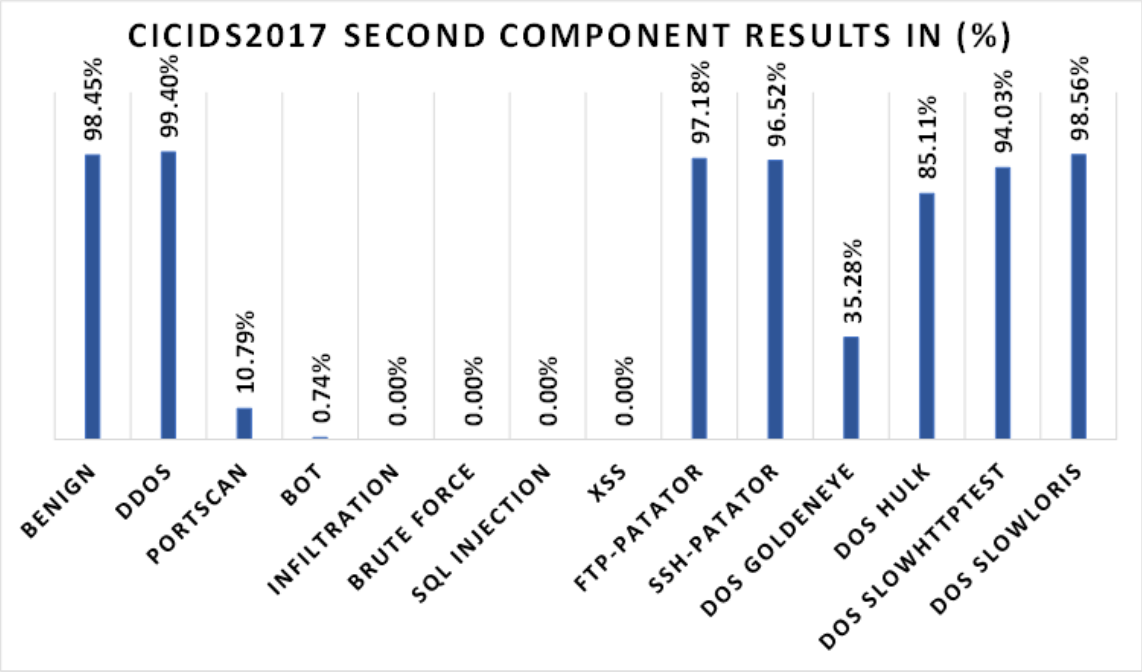


Figure 6.2: CICIDS2017 Second Component Results (in %)

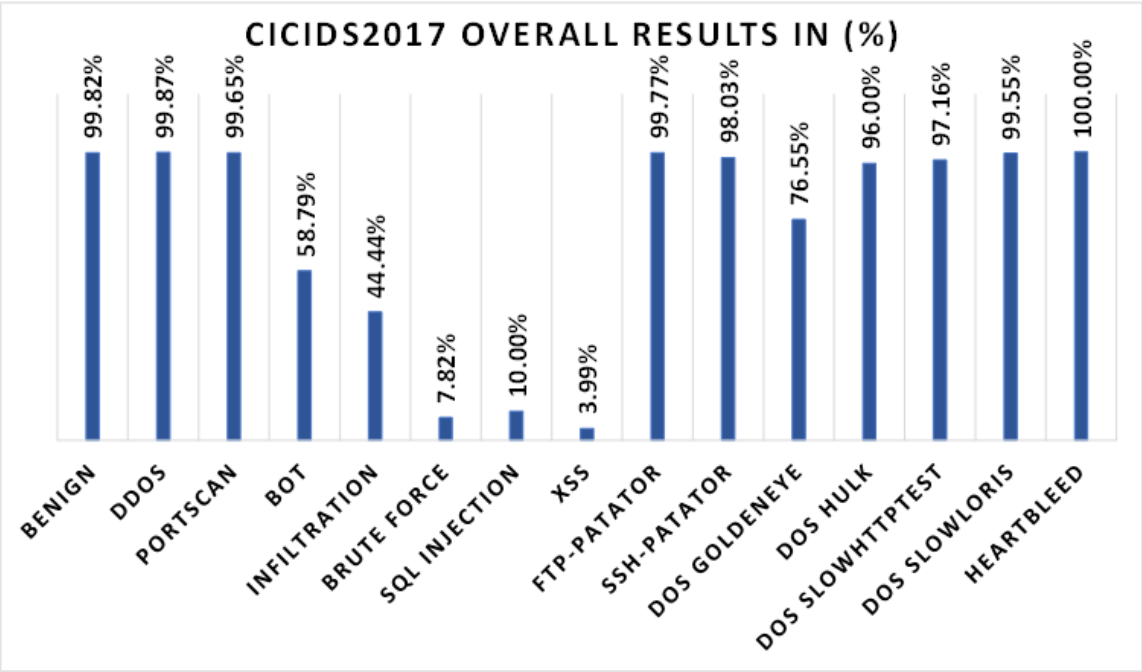


Figure 6.3: CICIDS2017 Overall Results (in %)

Figure 6.4 shows that the detection rate for benign and most attack types improved after implementing the second component classifier. As pointed out in Chapter 4, the UNAD ensemble can detect new attack types that have not been encountered before with a high

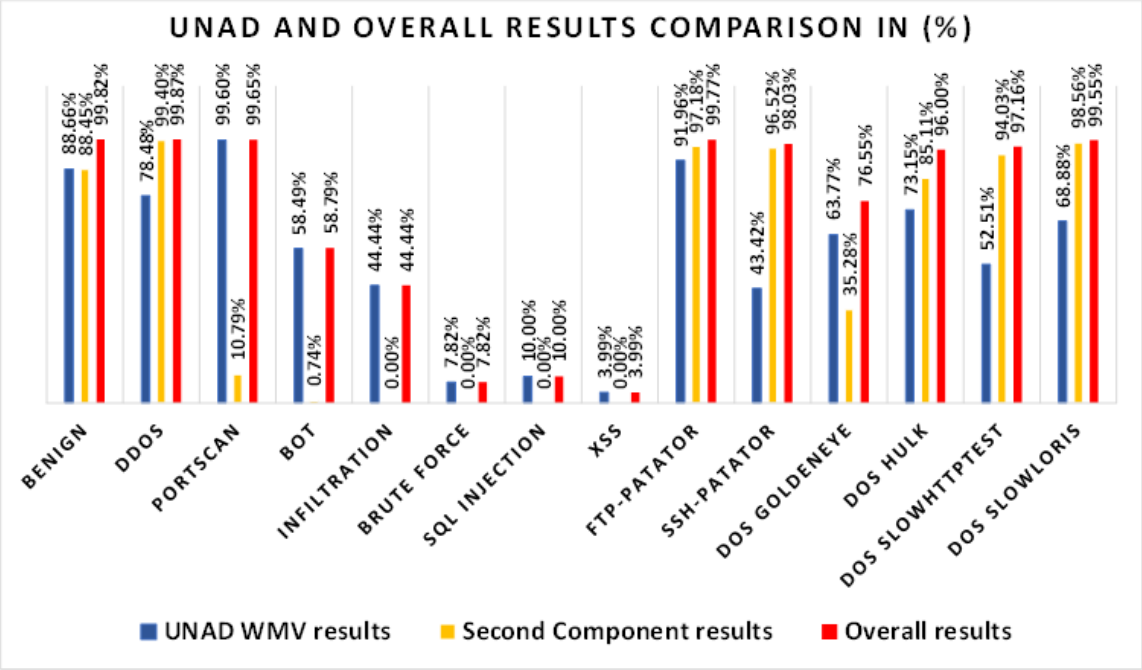


Figure 6.4: CICIDS2017 UNAD, second component and Overall Results (in %)

detection rate for benign and most of the attack types (blue bar). Furthermore, as seen in Figure 6.4, using the second component (orange bar) improved the detection rate for benign and most attack types (red bar). The detection rate for the benign flow improved by just over 11%, from 88.66% to 99.82%. Similarly, the DoS GoldenEye detection rate increased by 12%, from 63.77% to 76.55%. The detection rate for DDoS and DoS Hulk attacks was enhanced by more than 20%, with 99.87% and 96% detection rates, respectively. Moreover, DoS Slowhttptest and DoS SlowLoris detection rates were boosted by 44% and 30%, respectively. SSH-Patator showed a significant improvement in its detection rate, from 43.42% to 98.03%, with more than 54% improvement in the detection rate. FTP-Patator detection rate improved by around 8%. In contrast, Portscan and Bot attacks improved slightly, with under 0.5% of enhancement in the detection rate. Finally, none of the web attacks (Brute Force, SQL Injection, XSS) or infiltration attacks saw any improvements in the detection rate as the second component couldn't detect any of these attacks.

Further Analysis of Attack Types with No Improvement in Overall Detection Rate

As mentioned in Chapter 5 and seen in Figure 6.2, the second component classifier could not detect any of the web attacks (brute force, SQL injection, XSS) or infiltration attacks.

Hence, it did not improve the overall detection rate for these attacks. Further analysis of these attack types showed that the second component classifier did not detect them because a low proportion of data instances of these attacks was detected by UNAD, which was used to train the second component classifier. The low detection by UNAD can be due to losing some information that helps in recognising such attacks after applying the dimensionality reduction technique (PCA), which was aimed to reduce the number of features, hence, avoid the curse of dimensionality problem. For example, UNAD detected only 59 brute force attacks, which were used to train the second component classifier to detect 695 data instances of the same attack type. Similarly, the RF classifier was trained on 13 XSS attacks to detect 313 data instances.

Furthermore, the second component classifier was trained on only 8 infiltration data instances to detect 10 data instances. Likewise, the second component classifier was trained on only one SQL injection data instance in order to detect nine instances. In contrast, the second component classifier was trained, for example, on 1,280 SSH-Patator attacks, 1,444 DoS Slowhttptest attacks, 50,236 DDoS attacks and 772,077 benign flow; therefore, these attacks saw a significant improvement in their detection rate.

6.2 NSL-KDD Results and Analysis

Table 6.2 shows the results for UNAD, the second component classifier and the overall boosted results after combining the second component classifier's results with UNAD's results on the NSL-KDD dataset.

Table 6.2: NSL-KDD Overall Results (in %)

Measure	UNAD WMV	Second component classifier	Overall results
Accuracy	93.22	74.02	98.24
Precision	93.52	69.76	97.26
Recall	92.80	89.67	99.26
F1-score	93.16	78.47	98.25
ROC-AUC	93.22	73.10	98.24

As with the CICIDS2017 results, the overall results for the NSL-KDD dataset were also

very high for all measures after using the second component classifier. For example, the F1-score was enhanced by just over 5% from 93.16% in UNAD to 98.25%. Furthermore, the precision measure results increased from 93.52% on UNAD to 97.26%. The overall results for the recall measure improved from 92.80% on UNAD to 99.26%. Finally, the system achieved an overall accuracy and ROC-AUC of 99.19% and 98.24%, respectively.

Figures 6.5 and 6.6 revisit the results obtained from the data analysis of UNAD (*C1*) and the second component classifier (*C2*) for the NSL-KDD dataset. Figure 6.7 represents the overall boosted results using the second component classifier for the NSL-KDD dataset, and Figure 6.8 shows the results of UNAD, the second component and the overall results.

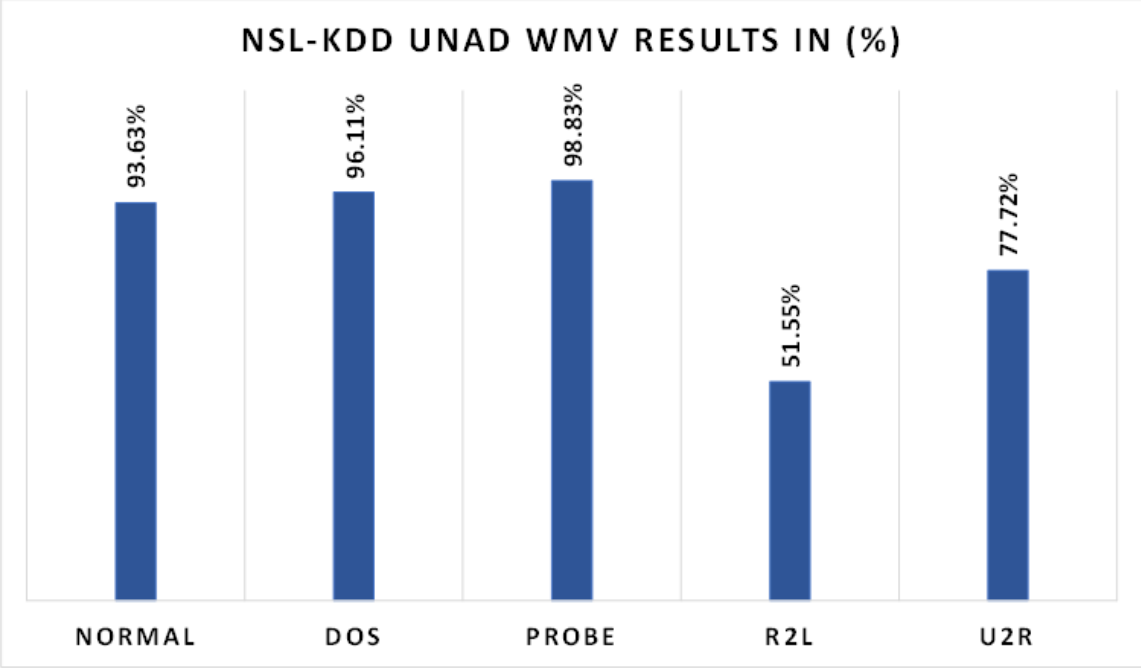


Figure 6.5: UNAD WMV Results for NSL-KDD Dataset (in %)

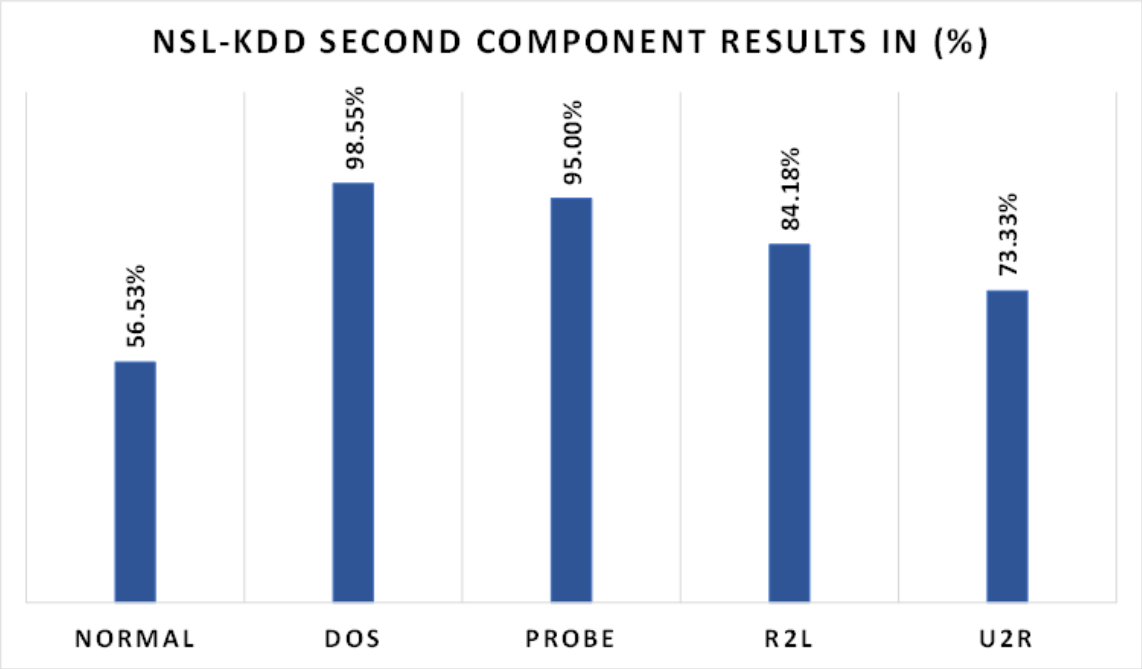


Figure 6.6: UNAD NSL-KDD Second Component Results (in %)

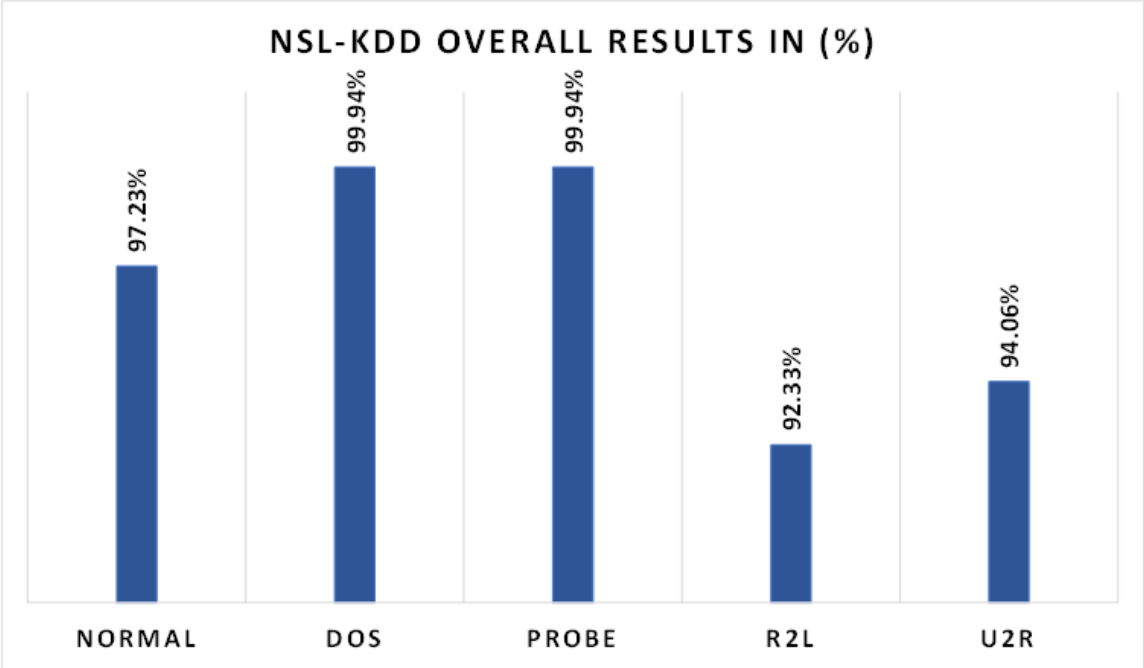


Figure 6.7: NSL-KDD Overall Results (in %)

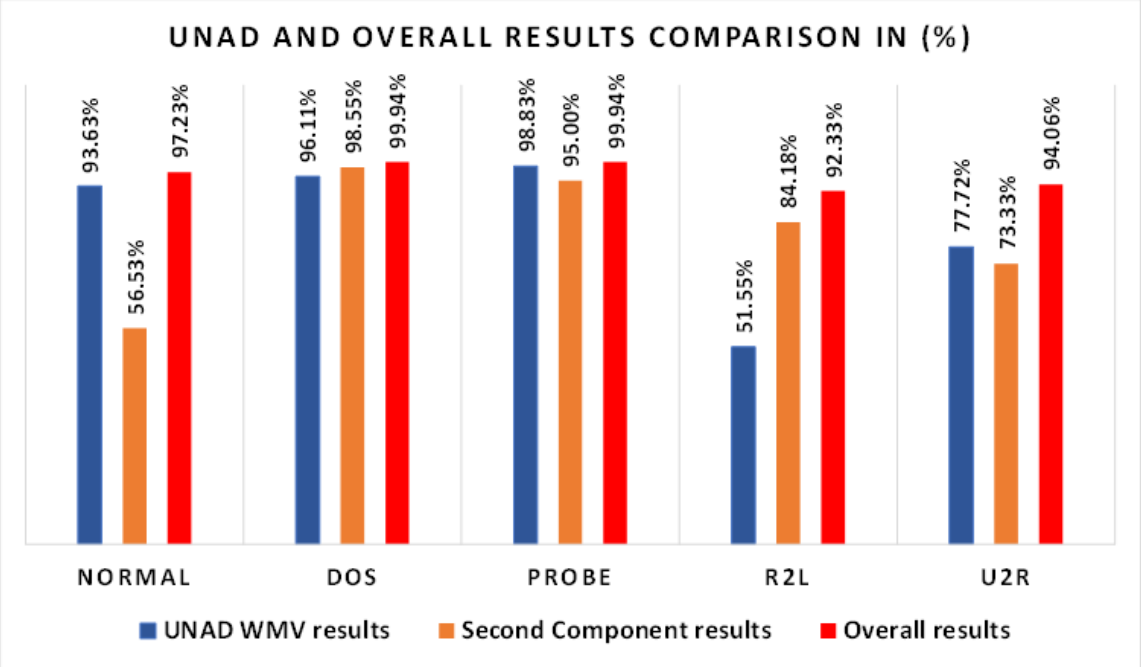


Figure 6.8: NSL-KDD UNAD, second component and Overall Results (in %)

Figure 6.8 shows that the detection rate for normal and all attack types have improved after implementing the second component classifier. The blue bar represents UNAD detection results, the orange bar represents the second component detection results and the red bar depicts the overall results after using the second component classifier. As seen in the figure, using the second component improved the detection rate for normal and all attack types. R2L attacks showed the most significant improvement in the detection rate, which was enhanced by more than 40%, from 51.55% to 92.33%, followed by U2R, which was boosted by more than 16%, from 77.72% to 94.06%. Normal and DoS attack detection rates improved by more than 3% , to 97.23% and 99.94%, respectively. Probe attacks detection rate were enhanced by 1%, from 98.83% to 99.94%.

Overall, it is evident that the second component classifier effectively enhanced the detection rate and improved the overall results for both datasets, which can be seen in the data analysis presented in this section. The improvement in the detection rate varied among the attacks. However, around 68% of the total attack types were detected at rates ranging from 92% to just below 100% after being enhanced via the second component classifier. Furthermore, as previously pointed out, some attack types did not show any improvement,

due to the very low number of data instances detected by UNAD that were used to train the second component classifier.

6.3 Chapter Summary

This chapter presented the overall boosted results after implementing the second component of the system and combining its results with UNAD. In addition, this chapter combined the data analysis acquired from Chapters 4 and 5 with the overall results. Then, this chapter illustrated the detection rate for benign/normal and all attack types after merging the second component classifier results with UNAD results. Finally, this chapter further examined attack types in the CICIDS2017 dataset that showed no improvement in overall detection rates, such as infiltration, brute force, SQL injection and XSS. The analysis showed that these attacks were not detected by the second component classifier because a low proportion of data instances of these attacks was detected by UNAD, which had been used to train the second component classifier.

Chapter 7

Conclusion and Future Work

This chapter concludes this thesis; it addresses the research questions and hypotheses and their outcomes. Furthermore, this chapter discusses the limitations of the research presented in this thesis. Finally, this chapter discusses further work that could build on this research.

7.1 Thesis Summary

The threat of network attacks increases every day. These attacks originate from hackers with various motives, such as stealing victims' bank account details, committing financial fraud, or for political reasons. As a result, governments and businesses have begun allocating money to strengthen network infrastructure against these attacks. One of the methods which can be used to fight network attacks are Intrusion Detection Systems. At present, IDSs using ML are receiving much research attention. Many researchers have proposed systems that use supervised ML. However, these systems are limited to detecting attacks that have been previously encountered and are well known; hence, they cannot detect new or unknown network attacks. Unsupervised ML is the key to overcoming this limitation, but unsupervised ML typically suffers from many false positives [2], low precision and recall results.

The research presented in this thesis aimed to develop a network IDS using ML methods to detect unknown and new attack types, while maintaining a low false positive rate for the system. To achieve this goal, four research questions were proposed and three hypotheses were investigated.

To investigate and answer the research questions and achieve the research objectives, the following Chapters were presented. First, a literature review was conducted in **Chapter 2**. This chapter provided a comprehensive overview of the use of ML in the intrusion detection domain. **Chapter 2** introduced types of intrusion detections systems and the measures used to evaluate their performance. In addition, **Chapter 2** reviewed anomaly detection techniques and described some of the supervised and unsupervised learning algorithms used in the anomaly detection domain.

Chapter 2 also explained the ensemble techniques and the methods used to combine its results. Also, **Chapter 2** discussed model explainability in ML and the preprocessing methods used in this thesis. Finally, **Chapter 2** reviewed and evaluated literature related to unsupervised intrusion detection to understand the domain challenges and limitations.

In **Chapter 3**, a novel Network Intrusion Detection System framework was proposed (Contribution 1). The framework was explained at high and low levels. This framework consisted of three components: *C1, C2* and *C3*, each with a different purpose. For example, *C1* was used to detect new and unknown network attacks. *C2* was used to boost the overall results and improve the detection rate, and *C3* was used to explain the model's decision in a human-understandable way. In addition, **Chapter 3** explained the preliminary experimental setup and the datasets used in this thesis, including the type of attacks, number of attack instances and the feature description. Furthermore, the preprocessing steps and the workflow applied to the datasets used in this thesis were explained. In addition, **Chapter 3** introduced the anomaly detection algorithms used in the first component as base learners and examined, evaluated and compared their results (Contribution 2).

Chapter 4 addressed ***RQ1*** and ***RQ2***:

> ***RQ1:*** *Is it possible to develop an unsupervised Network Intrusion Detection System that can exhibit a high detection performance in terms of precision, recall and F1-score while maintaining good performance over time with the current complexity in network attacks?*
>
> ***RQ2:*** *To what extent can the developed unsupervised Network Intrusion Detection System accurately detect attacks that have not been encountered before?*

Chapter 4 answered *RQ1* and *RQ2* by introducing a novel heterogeneous unsupervised bagging ensemble UNAD (Contribution 3), the unsupervised bagging ensemble learner for detecting unknown attacks, which, as previously pointed out, acts as the first component of the system. UNAD consists of a set of heterogeneous LOF and iForest base learners and uses Weighted Majority Voting as a results combiner. The data analysis showed that the UNAD ensemble can accurately detect new attack types that have not been previously encountered and maintain a high detection rate for most of the evaluated attack types. Hence, the results of **Chapter 4** confirm **Hypothesis 1**:

> ***Hypothesis 1:*** *"Anomaly detection methods can be adapted to detect new and previously unknown network attacks as new attacks are expected to be an anomaly to the normal network flow pattern. Moreover, the detection performance can be improved by constructing an ensemble-based model consisting of anomaly detection techniques."*

Chapter 5 addressed ***RQ3*** and ***RQ4***:

> ***RQ3:*** *Is it possible to improve the system's detection accuracy after the initial discovery of a new type of attack using supervised methods?*
>
> ***RQ4:*** *Can a mechanism within the IDS that explains attack detections help a domain expert to assess the threat level and understand how the model's decisions are made?*

Chapter 5 answered *RQ3* and *RQ4* by introducing the second and third components. The second component, which addresses *RQ3*, is formed of the supervised algorithm. The main goal of this component is to boost the overall results and improve the detection rate by being trained on UNAD's True Positive and True Negative, in addition to the training set used to train UNAD, which comprises only benign/normal flow. The RF classifier was chosen for the second component as the supervised classifier because it achieved the highest results compared with the other evaluated classifiers. Furthermore, the data analysis showed that implementing the RF classifier improved the detection rate for most of the evaluated attack types (Contribution 4). Therefore, the results of **Chapter 5** partially confirm **Hypothesis 2**:

> ***Hypothesis 2:*** *"Having a supervised model will assist in detecting attacks that have been encountered before, since these attacks become known to the system, thus improving the overall detection results."*

RQ4 was addressed by implementing the third component, which aims to explain the decision made by the model in a human-understandable way using ML explainability methods (Contribution 5). Two types of explanation were used in the third component, local and global. The local explainability used LIME [3], which provides an explanation for the domain expert for any single prediction made by the system. For global explainability, the global surrogate method was applied. For this, a Decision Tree was used to explain the entire model. Once the DT graph is created, a rule extractor function extracts the rules and generates a human-readable report for the domain expert in CSV format. Thus, the third component can help domain experts assess threat levels and understand how the model made its decisions. Accordingly, the outcome of **Chapter 5** confirms **Hypothesis 3**.

> ***Hypothesis 3:*** *"It is possible to obtain some explanation from the developed model to support domain experts in evaluating the level of threats and understanding the decisions made by the model."*

Lastly, **Chapter 6** presented the thesis's overall boosted results after implementing the second component to the system and combining its results with UNAD. Furthermore, **Chapter 6** revisited the results analysis acquired from Chapters 4 and 5 and presented these results

with the overall results. In addition, **Chapter 6** further examined the detection rate for benign/normal and all attack types after combining the second component classifier results with UNAD results. Finally, **Chapter 6** discussed the possible reason that led to some of the attacks in the CICIDS2017 dataset, such as infiltration, brute force, SQL injection and XSS, going undetected by the second component classifier and thus not improving the overall detection results.

7.2 Limitations

This thesis successfully answered the research questions, addressed the hypotheses and achieved its aim and objectives. However, two limitations were identified:

1. A real dataset cannot be used to evaluate ML models due to privacy and security issues. Hence, publicly available synthetic benchmarking datasets were used in this thesis.

2. The second component, which uses the RF classifier, cannot effectively detect an attack if trained on a low proportion of that attack and therefore cannot boost its overall results.

7.3 Future Work

For future work, the contribution of this thesis can be extended in the following directions:

1. As new attack types keep emerging, test and evaluate the system's effectiveness on these new attack types. Furthermore, deploy, test and evaluate the current system in the production environment (real-network environment). For an actual network, the system would be deployed behind the firewall to allow monitoring of the inbound and outbound traffic. This way, the system will be able to monitor, analyse and check any malicious attempts on the organisation's network and directly interact with the firewall in case of detecting an attack so the firewall can block it. Further, a

packet sniffer such as Wireshark will be implemented between the firewall and the system, capturing the traffic in real-time and forwarding it to the system. Further, this system should be managed by a human expert with knowledge of networks and ML techniques. Therefore, the human expert will be able to analyse the network traffic, interpret the outcome of the system's third component report, investigate and confirm any attack incidents and train/tune the system accordingly.

2. Although UNAD performed very well in detecting new and unknown attacks, further improvement of the overall system performance can be made by first evaluating and analysing the performance of other anomaly detection algorithms to be included in the UNAD ensemble as additional heterogeneous base learners. Secondly, empirically evaluate and analyse different supervised ML algorithms that can achieve the RF results or higher; but can overcome the issue of not detecting some attacks if UNAD was trained on a low proportion of data instances.

3. Consider using a computer cluster, if possible, to split the model training and algorithms hyperparameter tuning on compute nodes when dealing with a large dataset, which will reduce computational time for such processes.

4. Enhance the system to act in real-time (real-time IDS) in case an attack incident occurs. This could be done by adopting data stream mining and concept drift detection methods. In addition, the real-time IDS system could include features such as real-time alert notification and an automatic quick response mechanism that shuts down the system to mitigate damage from attacks.

5. Test and evaluate the model in other domains, such as the healthcare (identifying diseases and outbreaks) and financial sectors (fraud detection), and evaluate how well it can generalise and perform in those domains.

References

[1] Ashima Chawla, Brian Lee, Sheila Fallon, and Paul Jacob. Host based intrusion detection system with combined CNN/RNN model. In Carlos Alzate, Anna Monreale, Haytham Assem, Albert Bifet, Teodora Sandra Buda, Bora Caglayan, Brett Drury, Eva García-Martín, Ricard Gavaldà, Stefan Kramer, Niklas Lavesson, Michael Madden, Ian M. Molloy, Maria-Irina Nicolae, and Mathieu Sinn, editors, *ECML PKDD 2018 Workshops - Nemesis 2018, UrbReas 2018, SoGood 2018, IWAISe 2018, and Green Data Mining 2018, Dublin, Ireland, September 10-14, 2018, Proceedings*, volume 11329 of *Lecture Notes in Computer Science*, pages 149–158. Springer, 2018. doi: 10.1007/978-3-030-13453-2_12. URL https://doi.org/10.1007/978-3-030-13453-2_12.

[2] Weiwei Chen, Fangang Kong, Feng Mei, Guiqin Yuan, and Bo Li. A novel unsupervised anomaly detection approach for intrusion detection system. In *2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids)*, pages 69–73, 2017. doi: 10.1109/BigDataSecurity.2017.56.

[3] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages

1135–1144. ACM, 2016. doi: 10.1145/2939672.2939778. URL https://doi.org/10.1145/2939672.2939778.

[4] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 108–116. SciTePress, 2018. doi: 10.5220/0006639801080116. URL https://doi.org/10.5220/0006639801080116.

[5] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009, Ottawa, Canada, July 8-10, 2009*, pages 1–6. IEEE, 2009. doi: 10.1109/CISDA.2009.5356528. URL https://doi.org/10.1109/CISDA.2009.5356528.

[6] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of tor traffic using time based features. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Proceedings of the 3rd International Conference on Information Systems Security and Privacy, ICISSP 2017, Porto, Portugal, February 19-21, 2017*, pages 253–262. SciTePress, 2017. doi: 10.5220/0006105602530262. URL https://doi.org/10.5220/0006105602530262.

[7] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of encrypted and VPN traffic using time-related features. In Olivier Camp, Steven Furnell, and Paolo Mori, editors, *Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016, Rome, Italy, February 19-21, 2016*, pages 407–414. SciTePress, 2016. doi: 10.5220/0005740704070414. URL https://doi.org/10.5220/0005740704070414.

[8] NSL KDD. NSL KDD feature description. http://kdd.ics.uci.edu/databases/kddcup99/task.html. Accessed: 05.02.2022.

[9] Adetunmbi A Olusola, Adeola S Oladele, and Daramola O Abosede. Analysis of kdd'99 intrusion detection dataset for selection of relevance features. In *Proceedings of the world congress on engineering and computer science*, volume 1, pages 20–22. WCECS, 2010.

[10] Joseph Johnson. Global digital population. https://www.https://www.statista.com/statistics/617136/digital-population-worldwide/. Accessed: 08.01.2021.

[11] Cedric Nabe. Impact of COVID-19 on Cybersecurity. https://www2.deloitte.com/ch/en/pages/risk/articles/impact-covid-cybersecurity.html. Accessed: 04.02.2022.

[12] UK HM Treasury. Autumn Budget and Spending Review 2021. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/1043689/Budget_AB2021_Web_Accessible.pdf. Accessed: 04.02.2022.

[13] Steve Morgan. Cybercrime To Cost The World $10.5 Trillion Annually By 2025. https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/. Accessed: 02.12.2020.

[14] Zeeshan Ahmad, Adnan Shahid Khan, Cheah Wai Shiang, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Trans. Emerg. Telecommun. Technol.*, 32(1), 2021. doi: 10.1002/ett.4150. URL https://doi.org/10.1002/ett.4150.

[15] BBC. Ticketmaster. https://www.bbc.co.uk/news/technology-54931873, . Accessed: 02.12.2020.

[16] BBC. Flightradar24 Cyber-attack. https://www.bbc.co.uk/news/technology-54337980, . Accessed: 02.12.2020.

[17] BBC. Air India cyber-attack. https://www.bbc.co.uk/news/world-asia-india-57210118, . Accessed: 05.02.2022.

[18] BBC. Sunderland University cyber-attack. https://www.bbc.co.uk/news/uk-england-tyne-58925807, . Accessed: 05.02.2022.

[19] BBC. VoIP. https://www.bbc.co.uk/news/technology-59053876, . Accessed: 02.12.2020.

[20] The Guardian. Colonial Pipeline cyber-attack. https://www.theguardian.com/us-news/2021/may/12/us-fuel-shortages-pipeline-hack-drivers. Accessed: 05.02.2022.

[21] Gaku Kotani and Yuji Sekiya. Unsupervised scanning behavior detection based on distribution of network traffic features using robust autoencoders. In Hanghang Tong, Zhenhui Jessie Li, Feida Zhu, and Jeffrey Yu, editors, *2018 IEEE International Conference on Data Mining Workshops, ICDM Workshops, Singapore, Singapore, November 17-20, 2018*, pages 35–38. IEEE, 2018. doi: 10.1109/ICDMW.2018.00013. URL https://doi.org/10.1109/ICDMW.2018.00013.

[22] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognit.*, 65:211–222, 2017. doi: 10.1016/j.patcog.2016.11.008. URL https://doi.org/10.1016/j.patcog.2016.11.008.

[23] David Gunning and David W. Aha. Darpa's explainable artificial intelligence (XAI) program. *AI Mag.*, 40(2):44–58, 2019. doi: 10.1609/aimag.v40i2.2850. URL https://doi.org/10.1609/aimag.v40i2.2850.

[24] Saif Alzubi, Frederic T. Stahl, and Mohamed Medhat Gaber. Towards intrusion detection of previously unknown network attacks. In Khalid Al-Begain, Mauro Iacono, Lelio Campanile, and Andrzej Bargiela, editors, *Proceedings of the 35th International ECMS International Conference on Modelling and Simulation, ECMS 2021, Virtual Event, UK, May 31 - June 2, 2021*, pages 35–41. European Council for Modeling and Simulation, 2021. doi: 10.7148/2021-0035. URL https://doi.org/10.7148/2021-0035.

[25] Michael West. Chapter 2 - preventing system intrusions. In John R. Vacca, editor, *Network and System Security (Second Edition)*, pages 29–56. Syngress, Boston,

second edition edition, 2014. ISBN 978-0-12-416689-9. doi: https://doi.org/10.1016/B978-0-12-416689-9.00002-2. URL https://www.sciencedirect.com/science/article/pii/B9780124166899000022.

[26] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. Taxonomy and survey of collaborative intrusion detection. *ACM Comput. Surv.*, 47(4):55:1–55:33, 2015. doi: 10.1145/2716260. URL https://doi.org/10.1145/2716260.

[27] Monowar H. Bhuyan, D. K. Bhattacharyya, and Jugal K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Commun. Surv. Tutorials*, 16(1):303–336, 2014. doi: 10.1109/SURV.2013.052213.00046. URL https://doi.org/10.1109/SURV.2013.052213.00046.

[28] James P Anderson. Computer security threat monitoring and surveillance, james p. *Anderson Co., Fort Washington, PA*, 1980.

[29] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM Comput. Surv.*, 48(1):12:1–12:41, 2015. doi: 10.1145/2808691. URL https://doi.org/10.1145/2808691.

[30] Prachi Deshpande, Subhash Chander Sharma, Sateesh Kumar Peddoju, and S. Junaid. HIDS: A host based intrusion detection system for cloud computing environment. *Int. J. Syst. Assur. Eng. Manag.*, 9(3):567–576, 2018. doi: 10.1007/s13198-014-0277-7. URL https://doi.org/10.1007/s13198-014-0277-7.

[31] Nour Moustafa, Jiankun Hu, and Jill Slay. A holistic review of network anomaly detection systems: A comprehensive survey. *J. Netw. Comput. Appl.*, 128:33–55, 2019. doi: 10.1016/j.jnca.2018.12.006. URL https://doi.org/10.1016/j.jnca.2018.12.006.

[32] Firkhan Ali Bin Hamid Ali and Yee Yong Len. Development of host based intrusion detection system for log files. In *2011 IEEE Symposium on Business, Engineering*

and Industrial Applications (ISBEIA), pages 281–285, 2011. doi: 10.1109/ISBEIA.2011.6088821.

[33] Chirag Modi, Dhiren R. Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *J. Netw. Comput. Appl.*, 36(1):42–57, 2013. doi: 10.1016/j.jnca.2012.05.003. URL https://doi.org/10.1016/j.jnca.2012.05.003.

[34] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecur.*, 2(1):20, 2019. doi: 10.1186/s42400-019-0038-7. URL https://doi.org/10.1186/s42400-019-0038-7.

[35] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong-Jian Chen, Jui-Lin Lai, and Citra Dwi Perkasa. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert Syst. Appl.*, 38(1):306–313, 2011. doi: 10.1016/j.eswa.2010.06.066. URL https://doi.org/10.1016/j.eswa.2010.06.066.

[36] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *J. Netw. Comput. Appl.*, 36(1):16–24, 2013. doi: 10.1016/j.jnca.2012.09.004. URL https://doi.org/10.1016/j.jnca.2012.09.004.

[37] Berkah I. Santoso, M. Rien Suryatama Idrus, and Irwan Prasetya Gunawan. Designing network intrusion and detection system using signature-based method for protecting openstack private cloud. In *2016 6th International Annual Engineering Seminar (InAES)*, pages 61–66, 2016. doi: 10.1109/INAES.2016.7821908.

[38] Masoud Ghorbanian, Bharanidharan Shanmugam, Ganthan Narayansamy, and Norbik Bashah Idris. Signature-based hybrid intrusion detection system (hids) for android devices. In *2013 IEEE Business Engineering and Industrial Applications Colloquium (BEIAC)*, pages 827–831, 2013. doi: 10.1109/BEIAC.2013.6560251.

[39] Francisco Muñoz Cortés and Natalia Gaviria Gómez. A hybrid alarm management strategy in signature-based intrusion detection systems. In *2019 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–6, 2019. doi: 10.1109/ColComCon.2019.8809121.

[40] Snort. Snort Intrusion Detection System. https://www.snort.org/. Accessed: 04.12.2020.

[41] Shelly Xiaonan Wu and Wolfgang Banzhaf. The use of computational intelligence in intrusion detection systems: A review. *Appl. Soft Comput.*, 10(1):1–35, 2010. doi: 10.1016/j.asoc.2009.06.019. URL https://doi.org/10.1016/j.asoc.2009.06.019.

[42] R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019. doi: 10.1109/ACCESS.2019.2895334. URL https://doi.org/10.1109/ACCESS.2019.2895334.

[43] Pedro Garcia-Teodoro, Jesús Esteban Díaz Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Comput. Secur.*, 28(1-2):18–28, 2009. doi: 10.1016/j.cose.2008.08.003. URL https://doi.org/10.1016/j.cose.2008.08.003.

[44] Pedro Garcia-Teodoro, Jesús Esteban Díaz Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Comput. Secur.*, 28(1-2):18–28, 2009. doi: 10.1016/j.cose.2008.08.003. URL https://doi.org/10.1016/j.cose.2008.08.003.

[45] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009. doi: 10.1145/1541880.1541882. URL https://doi.org/10.1145/1541880.1541882.

[46] Sultan Zavrak and Murat Iskefiyeli. Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access*, 8:108346–108358,

2020. doi: 10.1109/ACCESS.2020.3001350. URL https://doi.org/10.1109/ACCESS.2020.3001350.

[47] Alka Chaudhary, V.N. Tiwari, and Anil Kumar. Design an anomaly based fuzzy intrusion detection system for packet dropping attack in mobile ad hoc networks. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 256–261, 2014. doi: 10.1109/IAdCC.2014.6779330.

[48] Frank J Anscombe. Rejection of outliers. *Technometrics*, 2(2):123–146, 1960.

[49] Arthur Zimek and Peter Filzmoser. There and back again: Outlier detection between statistical reasoning and data mining algorithms. *WIREs Data Mining Knowl. Discov.*, 8(6), 2018. doi: 10.1002/widm.1280. URL https://doi.org/10.1002/widm.1280.

[50] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *J. Netw. Comput. Appl.*, 60:19–31, 2016. doi: 10.1016/j.jnca.2015.11.016. URL https://doi.org/10.1016/j.jnca.2015.11.016.

[51] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: identifying density-based local outliers. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 93–104. ACM, 2000. doi: 10.1145/342009.335388. URL https://doi.org/10.1145/342009.335388.

[52] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. Incremental local outlier detection for data streams. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2007, part of the IEEE Symposium Series on Computational Intelligence 2007, Honolulu, Hawaii, USA, 1-5 April 2007*, pages 504–515. IEEE, 2007. doi: 10.1109/CIDM.2007.368917. URL https://doi.org/10.1109/CIDM.2007.368917.

[53] Mahsa Salehi, Christopher Leckie, James C. Bezdek, Tharshan Vaithianathan, and Xuyun Zhang. Fast memory efficient local outlier detection in data streams. *IEEE*

Trans. Knowl. Data Eng., 28(12):3246–3260, 2016. doi: 10.1109/TKDE.2016.2597833. URL https://doi.org/10.1109/TKDE.2016.2597833.

[54] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1):3:1–3:39, 2012. doi: 10.1145/2133360.2133363. URL https://doi.org/10.1145/2133360.2133363.

[55] Peter J. Rousseeuw and Katrien van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999. doi: 10.1080/00401706.1999.10485670. URL https://doi.org/10.1080/00401706.1999.10485670.

[56] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alexander J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, 2001. doi: 10.1162/089976601750264965. URL https://doi.org/10.1162/089976601750264965.

[57] Imran Razzak, Khurram Zafar, Muhammad Imran, and Guandong Xu. Randomized nonlinear one-class support vector machines with bounded loss function to detect of outliers for large scale iot data. *Future Gener. Comput. Syst.*, 112:715–723, 2020. doi: 10.1016/j.future.2020.05.045. URL https://doi.org/10.1016/j.future.2020.05.045.

[58] Aya Ayadi, Oussama Ghorbel, Mohammed S. BenSaleh, Abdelfateh Obeid, and Mohamed Abid. Performance of outlier detection techniques based classification in wireless sensor networks. In *13th International Wireless Communications and Mobile Computing Conference, IWCMC 2017, Valencia, Spain, June 26-30, 2017*, pages 687–692. IEEE, 2017. doi: 10.1109/IWCMC.2017.7986368. URL https://doi.org/10.1109/IWCMC.2017.7986368.

[59] Lev Faivishevsky. Information theoretic multivariate change detection for multisensory information processing in internet of things. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 6250–6254. IEEE, 2016. doi: 10.1109/ICASSP.2016.7472879. URL https://doi.org/10.1109/ICASSP.2016.7472879.

[60] Lotfi Mhamdi, Desmond C. McLernon, Fadi El-Moussa, Syed Ali Raza Zaidi, Mounir Ghogho, and Tuan A. Tang. A deep learning approach combining autoencoder with one-class SVM for ddos attack detection in sdns. In *Eighth IEEE International Conference on Communications and Networking, ComNet 2020, Virtual Event, Tunisia, October 28-30, 2020*, pages 1–6. IEEE, 2020. doi: 10.1109/ComNet47917.2020.9306073. URL https://doi.org/10.1109/ComNet47917.2020.9306073.

[61] Riccardo Guidotti, Anna Monreale, Franco Turini, Dino Pedreschi, and Fosca Giannotti. A survey of methods for explaining black box models. *CoRR*, abs/1802.01933, 2018. URL http://arxiv.org/abs/1802.01933.

[62] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011. ISBN 978-0123814791. URL http://hanj.cs.illinois.edu/bk3/.

[63] Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers - a survey. *IEEE Trans. Syst. Man Cybern. Part C*, 35(4):476–487, 2005. doi: 10.1109/TSMCC.2004.843247. URL https://doi.org/10.1109/TSMCC.2004.843247.

[64] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[65] Robert E. Schapire. Explaining adaboost. In Bernhard Schölkopf, Zhiyuan Luo, and Vladimir Vovk, editors, *Empirical Inference - Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. Springer, 2013. doi: 10.1007/978-3-642-41136-6_5. URL https://doi.org/10.1007/978-3-642-41136-6_5.

[66] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[67] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[68] Andrzej Zolnierek and Bartlomiej Rubacha. The empirical study of the naive bayes classifier in the case of markov chain recognition task. In Marek Kurzynski, Edward Puchala, Michal Wozniak, and Andrzej Zolnierek, editors, *Computer Recognition Systems, Proceedings of the 4th International Conference on Computer Recognition Systems, CORES'05, May 22-25, 2005, Rydzyna Castle, Poland*, volume 30 of *Advances in Soft Computing*, pages 329–336. Springer, 2005. doi: 10.1007/3-540-32390-2_38. URL https://doi.org/10.1007/3-540-32390-2_38.

[69] Saurabh Mukherjee and Neelam Sharma. Intrusion detection using naive bayes classifier with feature reduction. *Procedia Technology*, 4:119–128, 2012. ISSN 2212-0173. doi: https://doi.org/10.1016/j.protcy.2012.05.017. URL https://www.sciencedirect.com/science/article/pii/S2212017312002964. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

[70] Daniele Soria, Jonathan M. Garibaldi, Federico Ambrogi, Elia Biganzoli, and Ian O. Ellis. A 'non-parametric' version of the naive bayes classifier. *Knowl. Based Syst.*, 24(6):775–784, 2011. doi: 10.1016/j.knosys.2011.02.014. URL https://doi.org/10.1016/j.knosys.2011.02.014.

[71] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus F. M. Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael S. Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, 2008. doi: 10.1007/s10115-007-0114-2. URL https://doi.org/10.1007/s10115-007-0114-2.

[72] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognit. Lett.*, 27(8):861–874, 2006. doi: 10.1016/j.patrec.2005.10.010. URL https://doi.org/10.1016/j.patrec.2005.10.010.

[73] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.*, 21(9):1263–1284, 2009. doi: 10.1109/TKDE.2008.239. URL https://doi.org/10.1109/TKDE.2008.239.

[74] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *CoRR*, abs/2010.16061, 2020. URL https://arxiv.org/abs/2010.16061.

[75] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. GP ensemble for distributed intrusion detection systems. In Peng Wang, Maneesha Singh, Chidanand Apté, and Petra Perner, editors, *Pattern Recognition and Data Mining, Third International Conference on Advances in Pattern Recognition, ICAPR 2005, Bath, UK, August 22-25, 2005, Proceedings, Part I*, volume 3686 of *Lecture Notes in Computer Science*, pages 54–62. Springer, 2005. doi: 10.1007/11551188_6. URL https://doi.org/10.1007/11551188_6.

[76] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 157–166. ACM, 2005. doi: 10.1145/1081870.1081891. URL https://doi.org/10.1145/1081870.1081891.

[77] Matthew Norton and Stan Uryasev. Maximization of AUC and buffered AUC in binary classification. *Math. Program.*, 174(1-2):575–612, 2019. doi: 10.1007/s10107-018-1312-2. URL https://doi.org/10.1007/s10107-018-1312-2.

[78] Jesse Davis and Mark Goadrich. The relationship between precision-recall and ROC curves. In William W. Cohen and Andrew W. Moore, editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 233–240. ACM, 2006. doi: 10.1145/1143844.1143874. URL https://doi.org/10.1145/1143844.1143874.

[79] José Hernández-Orallo, Peter A. Flach, and César Ferri. ROC curves in cost space. *Mach. Learn.*, 93(1):71–91, 2013. doi: 10.1007/s10994-013-5328-9. URL https://doi.org/10.1007/s10994-013-5328-9.

[80] José Ramón Pasillas-Díaz and Sylvie Ratté. Bagged subspaces for unsupervised outlier detection. *Comput. Intell.*, 33(3):507–523, 2017. doi: 10.1111/coin.12097. URL https://doi.org/10.1111/coin.12097.

[81] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman amp; Hall/CRC, 1st edition, 2012. ISBN 1439830037.

[82] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, 1996. doi: 10.1007/BF00058655. URL https://doi.org/10.1007/BF00058655.

[83] Robert E. Schapire. *The Boosting Approach to Machine Learning: An Overview*, pages 149–171. Springer New York, New York, NY, 2003. ISBN 978-0-387-21579-2. doi: 10.1007/978-0-387-21579-2_9. URL https://doi.org/10.1007/978-0-387-21579-2_9.

[84] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In Lorenza Saitta, editor, *Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), Bari, Italy, July 3-6, 1996*, pages 148–156. Morgan Kaufmann, 1996.

[85] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. doi: 10.1016/S0893-6080(05)80023-1. URL https://doi.org/10.1016/S0893-6080(05)80023-1.

[86] Xuefeng Zhang, Penghui Wang, Lan Du, and Hongwei Liu. New method for radar hrrp recognition and rejection based on weighted majority voting combination of multiple classifiers. In *2011 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4, 2011. doi: 10.1109/ICSPCC.2011.6061765.

[87] Hongwei Li and Bin Yu. Error rate bounds and iterative weighted majority voting for crowdsourcing. *CoRR*, abs/1411.4086, 2014. URL http://arxiv.org/abs/1411.4086.

[88] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[89] Erico Tjoa and Cuntai Guan. A survey on explainable artificial intelligence (XAI): toward medical XAI. *IEEE Trans. Neural Networks Learn. Syst.*, 32(11):4793–4813, 2021. doi: 10.1109/TNNLS.2020.3027314. URL https://doi.org/10.1109/TNNLS.2020.3027314.

[90] Wilson Silva, Kelwin Fernandes, Maria João Cardoso, and Jaime S. Cardoso. Towards complementary explanations using deep neural networks. In Danail Stoyanov, Zeike Taylor, Seyed Mostafa Kia, Ipek Oguz, Mauricio Reyes, Anne L. Martel, Lena Maier-Hein, Andre F. Marquand, Edouard Duchesnay, Tommy Löfstedt, Bennett A. Landman, M. Jorge Cardoso, Carlos A. Silva, Sérgio Pereira, and Raphael Meier, editors, *Understanding and Interpreting Machine Learning in Medical Image Computing Applications - First International Workshops MLCN 2018, DLF 2018, and iMIMIC 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16-20, 2018, Proceedings*, volume 11038 of *Lecture Notes in Computer Science*, pages 133–140. Springer, 2018. doi: 10.1007/978-3-030-02628-8_15. URL https://doi.org/10.1007/978-3-030-02628-8_15.

[91] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, 58:82–115, 2020. doi: 10.1016/j.inffus.2019.12.012. URL https://doi.org/10.1016/j.inffus.2019.12.012.

[92] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38, 2019. doi: 10.1016/j.artint.2018.07.007. URL https://doi.org/10.1016/j.artint.2018.07.007.

[93] Johanna D Moore and William R Swartout. Explanation in expert systemss: A survey. Technical report, UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST, 1988.

[94] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine

learning. *Commun. ACM*, 63(1):68–77, 2020. doi: 10.1145/3359786. URL https://doi.org/10.1145/3359786.

[95] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8), 2019. ISSN 2079-9292. doi: 10.3390/electronics8080832. URL https://www.mdpi.com/2079-9292/8/8/832.

[96] Riccardo Guidotti, Anna Monreale, Franco Turini, Dino Pedreschi, and Fosca Giannotti. A survey of methods for explaining black box models. *CoRR*, abs/1802.01933, 2018. URL http://arxiv.org/abs/1802.01933.

[97] David Martens, Jan Vanthienen, Wouter Verbeke, and Bart Baesens. Performance of classification models from a user perspective. *Decis. Support Syst.*, 51(4):782–793, 2011. doi: 10.1016/j.dss.2011.01.013. URL https://doi.org/10.1016/j.dss.2011.01.013.

[98] Amand F Schmidt and Chris Finan. Linear regression and the normality assumption. *Journal of clinical epidemiology*, 98:146–151, 2018.

[99] Radwa El Shawi, Mouaz H. Al-Mallah, and Sherif Sakr. On the interpretability of machine learning-based model for predicting hypertension. *BMC Medical Informatics Decis. Mak.*, 19(1):146:1–146:32, 2019. doi: 10.1186/s12911-019-0874-0. URL https://doi.org/10.1186/s12911-019-0874-0.

[100] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2021. doi: 10.3390/e23010018. URL https://doi.org/10.3390/e23010018.

[101] Jianglin Huang, Yan-Fu Li, and Min Xie. An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Inf. Softw. Technol.*, 67:108–127, 2015. doi: 10.1016/j.infsof.2015.07.004. URL https://doi.org/10.1016/j.infsof.2015.07.004.

[102] Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*, volume 72 of *Intelligent Systems Reference Library*. Springer, 2015. ISBN 978-3-319-10246-7. doi: 10.1007/978-3-319-10247-4. URL https://doi.org/10.1007/978-3-319-10247-4.

[103] Shimon Whiteson, Brian Tanner, Matthew E. Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning, ADPRL 2011, Paris, France, April 12-14, 2011*, pages 120–127. IEEE, 2011. doi: 10.1109/ADPRL.2011.5967363. URL https://doi.org/10.1109/ADPRL.2011.5967363.

[104] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *J. Mach. Learn. Res.*, 20:53:1–53:32, 2019. URL http://jmlr.org/papers/v20/18-444.html.

[105] Daniel Berrar. Cross-validation. In Shoba Ranganathan, Michael Gribskov, Kenta Nakai, and Christian Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 542 – 545. Academic Press, Oxford, 2019. ISBN 978-0-12-811432-2. doi: https://doi.org/10.1016/B978-0-12-809633-8.20349-X. URL http://www.sciencedirect.com/science/article/pii/B978012809633820349X.

[106] Tzu-Tsung Wong and Po-Yang Yeh. Reliable accuracy estimates from k-fold cross validation. *IEEE Trans. Knowl. Data Eng.*, 32(8):1586–1594, 2020. doi: 10.1109/TKDE.2019.2912815. URL https://doi.org/10.1109/TKDE.2019.2912815.

[107] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4(0):40–79, 2010. ISSN 1935-7516. doi: 10.1214/09-ss054. URL http://dx.doi.org/10.1214/09-SS054.

[108] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009. doi: 10.1007/978-0-387-39940-9_565. URL https://doi.org/10.1007/978-0-387-39940-9_565.

[109] Sukirty Jain, Sanyam Shukla, and Rajesh Wadhvani. Dynamic selection of normalization techniques using data complexity measures. *Expert Syst. Appl.*, 106:252–262, 2018. doi: 10.1016/j.eswa.2018.04.008. URL https://doi.org/10.1016/j.eswa.2018.04.008.

[110] Nikolay Chumerin and Marc M. Van Hulle. Comparison of two feature extraction methods based on maximization of mutual information. In *2006 16th IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing*, pages 343–348, 2006. doi: 10.1109/MLSP.2006.275572.

[111] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012. ISBN 0262018020.

[112] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016. doi: 10.1098/rsta.2015.0202. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2015.0202.

[113] Wei Wang and Roberto Battiti. Identifying intrusions in computer networks with principal component analysis. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, The International Dependability Conference - Bridging Theory and Practice, April 20-22 2006, Vienna University of Technology, Austria*, pages 270–279. IEEE Computer Society, 2006. doi: 10.1109/ARES.2006.73. URL https://doi.org/10.1109/ARES.2006.73.

[114] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Comput. Networks*, 51(12):3448–3470, 2007. doi: 10.1016/j.comnet.2007.02.001. URL https://doi.org/10.1016/j.comnet.2007.02.001.

[115] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000. doi: 10.1016/S0893-6080(00)00026-5. URL https://doi.org/10.1016/S0893-6080(00)00026-5.

[116] Hongtao Du and Hairong Qi. An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images. In *2004 IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2004, Anchorage, Alaska, USA, 20-24 September 2004*, pages 3257–3260. IEEE, 2004. doi: 10.1109/IGARSS.2004.1370396. URL https://doi.org/10.1109/IGARSS.2004.1370396.

[117] Jing Wang and Chein-I Chang. Independent component analysis-based dimensionality reduction with applications in hyperspectral image analysis. *IEEE Trans. Geosci. Remote. Sens.*, 44(6):1586–1600, 2006. doi: 10.1109/TGRS.2005.863297. URL https://doi.org/10.1109/TGRS.2005.863297.

[118] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, 2003. URL http://jmlr.org/papers/v3/guyon03a.html.

[119] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Comput. Electr. Eng.*, 40(1):16–28, 2014. doi: 10.1016/j.compeleceng.2013.11.024. URL https://doi.org/10.1016/j.compeleceng.2013.11.024.

[120] Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. In Charu C. Aggarwal, editor, *Data Classification: Algorithms and Applications*, pages 37–64. CRC Press, 2014. doi: 10.1201/b17320-3. URL http://www.crcnetbase.com/doi/abs/10.1201/b17320-3.

[121] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Comput. Surv.*, 50(6):94:1–94:45, 2018. doi: 10.1145/3136625. URL https://doi.org/10.1145/3136625.

[122] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explor.*, 6(1):1–6, 2004. doi: 10.1145/1007730.1007733. URL https://doi.org/10.1145/1007730.1007733.

[123] David A. Cieslak, Nitesh V. Chawla, and Aaron Striegel. Combating imbalance in network intrusion datasets. In *2006 IEEE International Conference on Granular Computing, GrC 2006, Atlanta, Georgia, USA, May 10-12, 2006*, pages 732–737. IEEE, 2006. doi: 10.1109/GRC.2006.1635905. URL https://doi.org/10.1109/GRC.2006.1635905.

[124] T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. Learning from streaming data with concept drift and imbalance: an overview. *Prog. Artif. Intell.*, 1(1): 89–101, 2012. doi: 10.1007/s13748-011-0008-0. URL https://doi.org/10.1007/s13748-011-0008-0.

[125] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Comput. Intell.*, 20(1):18–36, 2004. doi: 10.1111/j.0824-7935.2004.t01-1-00228.x. URL https://doi.org/10.1111/j.0824-7935.2004.t01-1-00228.x.

[126] Hao Zhang, Shumin Dai, Yongdan Li, and Wenjun Zhang. Real-time distributed-random-forest-based network intrusion detection system using apache spark. In *37th IEEE International Performance Computing and Communications Conference, IPCCC 2018, Orlando, FL, USA, November 17-19, 2018*, pages 1–7. IEEE, 2018. doi: 10.1109/PCCC.2018.8711068. URL https://doi.org/10.1109/PCCC.2018.8711068.

[127] Christopher B. Freas, Robert W. Harrison, and Yuan Long. High performance attack estimation in large-scale network flows. In Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He, and Jeffrey S. Saltz, editors, *IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018*, pages 5014–5020. IEEE, 2018. doi: 10.1109/BigData.2018.8622125. URL https://doi.org/10.1109/BigData.2018.8622125.

[128] Yuyang Zhou, Guang Cheng, Shanqing Jiang, and Mian Dai. Building an efficient intrusion detection system based on feature selection and ensemble classifier.

Comput. Networks, 174:107247, 2020. doi: 10.1016/j.comnet.2020.107247. URL https://doi.org/10.1016/j.comnet.2020.107247.

[129] Maxime Labonne, Alexis Olivereau, Baptiste Polvé, and Djamal Zeghlache. Unsupervised protocol-based intrusion detection for real-world networks. In *International Conference on Computing, Networking and Communications, ICNC 2020, Big Island, HI, USA, February 17-20, 2020*, pages 299–303. IEEE, 2020. doi: 10.1109/ICNC47757.2020.9049796. URL https://doi.org/10.1109/ICNC47757.2020.9049796.

[130] Joohwa Lee, Ju-Geon Pak, and Myungsuk Lee. Network intrusion detection system using feature extraction based on deep sparse autoencoder. In *International Conference on Information and Communication Technology Convergence, ICTC 2020, Jeju Island, Korea (South), October 21-23, 2020*, pages 1282–1287. IEEE, 2020. doi: 10.1109/ICTC49870.2020.9289253. URL https://doi.org/10.1109/ICTC49870.2020.9289253.

[131] Lotfi Mhamdi, Desmond C. McLernon, Fadi El-Moussa, Syed Ali Raza Zaidi, Mounir Ghogho, and Tuan A. Tang. A deep learning approach combining autoencoder with one-class SVM for ddos attack detection in sdns. In *Eighth IEEE International Conference on Communications and Networking, ComNet 2020, Virtual Event, Tunisia, October 28-30, 2020*, pages 1–6. IEEE, 2020. doi: 10.1109/ComNet47917.2020.9306073. URL https://doi.org/10.1109/ComNet47917.2020.9306073.

[132] Quoc Thong Nguyen, Kim Phuc Tran, Philippe Castagliola, Truong Thu Huong, Minh Kha Nguyen, and Salim Lardjane. Nested one-class support vector machines for network intrusion detection. In *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, pages 7–12, 2018. doi: 10.1109/CCE.2018.8465718.

[133] Hui-Hao Chou and Sheng-De Wang. An adaptive network intrusion detection approach for the cloud environment. In *International Carnahan Conference on Secu-*

rity Technology, ICCST 2015, Taipei, Taiwan, September 21-24, 2015, pages 1–6. IEEE, 2015. doi: 10.1109/CCST.2015.7389649. URL https://doi.org/10.1109/CCST.2015.7389649.

[134] Jiong Zhang and Mohammad Zulkernine. Anomaly based network intrusion detection with unsupervised outlier detection. In *Proceedings of IEEE International Conference on Communications, ICC 2006, Istanbul, Turkey, 11-15 June 2006*, pages 2388–2393. IEEE, 2006. doi: 10.1109/ICC.2006.255127. URL https://doi.org/10.1109/ICC.2006.255127.

[135] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In Vladimir Estivill-Castro, editor, *Computer Science 2005, Twenty-Eighth Australasian Computer Science Conference (ACSC2005), Newcastle, NSW, Australia, January/February 2005*, volume 38 of *CRPIT*, pages 333–342. Australian Computer Society, 2005. URL http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV38Leung.html.

[136] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 94–105. ACM Press, 1998. doi: 10.1145/276304.276314. URL https://doi.org/10.1145/276304.276314.

[137] Guo Pu, Lijuan Wang, Jun Shen, and Fang Dong. A hybrid unsupervised clustering-based anomaly detection method. *Tsinghua Science and Technology*, 26(2):146–153, 2021. doi: 10.26599/TST.2019.9010051.

[138] Ming Zhang, Boyi Xu, and Jie Gong. An anomaly detection model based on one-class SVM to detect network intrusions. In *11th International Conference on Mobile Ad-hoc and Sensor Networks, MSN 2015, Shenzhen, China, December 16-18, 2015*, pages 102–107. IEEE Computer Society, 2015. doi: 10.1109/MSN.2015.40. URL https://doi.org/10.1109/MSN.2015.40.

[139] Vitor Hugo Bezerra, Victor Guilherme Turrisi da Costa, Sylvio Barbon Junior, Rodrigo Sanches Miani, and Bruno Bogaz Zarpelão. Iotds: A one-class classification approach to detect botnets in internet of things devices. *Sensors*, 19(14):3188, 2019. doi: 10.3390/s19143188. URL https://doi.org/10.3390/s19143188.

[140] Ebu Yusuf Güven, Sueda Gülgün, Ceyda Manav, Behice Bakır, and Zeynep Gürkaş Aydın. Multiple classification of cyber attacks using machine learning. *Electrica*, 22(2):313–320, 2022.

[141] Li Sun, Steven Versteeg, Serdar Boztas, and Asha Rao. Detecting anomalous user behavior using an extended isolation forest algorithm: An enterprise case study. *CoRR*, abs/1609.06676, 2016. URL http://arxiv.org/abs/1609.06676.

[142] Zekun Xu, Deovrat Kakde, and Arin Chaudhuri. Automatic hyperparameter tuning method for local outlier factor, with applications to anomaly detection. In Chaitanya K. Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay Ak, Yuanyuan Tian, Roger S. Barga, Carlo Zaniolo, Kisung Lee, and Yanfang (Fanny) Ye, editors, *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 4201–4207. IEEE, 2019. doi: 10.1109/BigData47090.2019.9006151. URL https://doi.org/10.1109/BigData47090.2019.9006151.

[143] Zhangyu Cheng, Chengming Zou, and Jianwei Dong. Outlier detection using isolation forest and local outlier factor. In Chih-Cheng Hung, Qianbin Chen, Xianzhong Xie, Christian Esposito, Jun Huang, Juw Won Park, and Qinghua Zhang, editors, *Proceedings of the Conference on Research in Adaptive and Convergent Systems, RACS 2019, Chongqing, China, September 24-27, 2019*, pages 161–168. ACM, 2019. doi: 10.1145/3338840.3355641. URL https://doi.org/10.1145/3338840.3355641.

[144] Santosh Kumar Sahu, Durga Prasad Mohapatra, and Niranjan K. Ray. An ensemble-based outlier detection approach on intrusion detection. In *19th OITS International Conference on Information Technology, OCIT 2019, Bhubaneswar, India, December*

16-18, 2021, pages 404–409. IEEE, 2021. doi: 10.1109/OCIT53463.2021.00085. URL https://doi.org/10.1109/OCIT53463.2021.00085.

[145] Xu-Ying Liu, Qian-Qian Li, and Zhi-Hua Zhou. Learning imbalanced multi-class data with optimal dichotomy weights. In Hui Xiong, George Karypis, Bhavani Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, pages 478–487. IEEE Computer Society, 2013. doi: 10.1109/ICDM.2013.51. URL https://doi.org/10.1109/ICDM.2013.51.

[146] Google. What is Google Colaboratory? https://research.google.com/colaboratory/faq.html. Accessed: 22.06.2022.

[147] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. doi: 10.5555/1953048.2078195. URL https://dl.acm.org/doi/10.5555/1953048.2078195.

[148] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 582–588. The MIT Press, 1999. URL http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.

[149] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 413–422. IEEE Computer Society, 2008. doi: 10.1109/ICDM.2008.17. URL https://doi.org/10.1109/ICDM.2008.17.

[150] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *WIREs Data Mining*

Knowl. Discov., 8(4), 2018. doi: 10.1002/widm.1249. URL https://doi.org/10.1002/widm.1249.

[151] Kaspersky. Brute Force Attack Definition. https://www.kaspersky.com/resource-center/definitions/brute-force-attack. Accessed: 02.12.2020.

[152] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. The matter of heartbleed. In Carey Williamson, Aditya Akella, and Nina Taft, editors, *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, pages 475–488. ACM, 2014. doi: 10.1145/2663716.2663755. URL https://doi.org/10.1145/2663716.2663755.

[153] Nazrul Hoque, Dhruba K. Bhattacharyya, and Jugal K. Kalita. Botnet in ddos attacks: Trends and challenges. *IEEE Commun. Surv. Tutorials*, 17(4):2242–2270, 2015. doi: 10.1109/COMST.2015.2457491. URL https://doi.org/10.1109/COMST.2015.2457491.

[154] Glenn Carl, George Kesidis, Richard R. Brooks, and Suresh Rai. Denial-of-service attack-detection techniques. *IEEE Internet Comput.*, 10(1):82–89, 2006. doi: 10.1109/MIC.2006.5. URL https://doi.org/10.1109/MIC.2006.5.

[155] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE Commun. Surv. Tutorials*, 15(4):2046–2069, 2013. doi: 10.1109/SURV.2013.031413.00127. URL https://doi.org/10.1109/SURV.2013.031413.00127.

[156] William G Halfond, Jeremy Viegas, Alessandro Orso, et al. A classification of sql-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering*, volume 1, pages 13–15. IEEE, 2006.

[157] KirstenS. Cross Site Scripting (XSS). https://owasp.org/www-community/attacks/xss/. Accessed: 02.05.2020.

[158] KDD99. The UCI KDD Archive. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: 18.06.2020.

[159] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by lincoln laboratory. *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000. doi: 10.1145/382912.382923. URL https://doi.org/10.1145/382912.382923.

[160] Osama Faker and Erdogan Dogdu. Intrusion detection using big data and deep learning techniques. In Dan Lo, Donghyun Kim, and Eric Gamess, editors, *Proceedings of the 2019 ACM Southeast Conference, ACM SE '19, Kennesaw, GA, USA, April 18-20, 2019*, pages 86–93. ACM, 2019. doi: 10.1145/3299815.3314439. URL https://doi.org/10.1145/3299815.3314439.

[161] scikit learn. scikit-learn dataset feature format. https://scikit-learn.org/stable/faq.html. Accessed: 10.04.2022.

[162] Fadi Salo, Ali Bou Nassif, and Aleksander Essex. Dimensionality reduction with IG-PCA and ensemble classifier for network intrusion detection. *Comput. Networks*, 148:164–175, 2019. doi: 10.1016/j.comnet.2018.11.010. URL https://doi.org/10.1016/j.comnet.2018.11.010.

[163] Liqun Liu, Bing Xu, Xiaoping Zhang, and Xianjun Wu. An intrusion detection method for internet of things based on suppressed fuzzy clustering. *EURASIP J. Wirel. Commun. Netw.*, 2018:113, 2018. doi: 10.1186/s13638-018-1128-z. URL https://doi.org/10.1186/s13638-018-1128-z.

[164] K. Keerthi Vasan and B. Surendiran. Dimensionality reduction using principal component analysis for network intrusion detection. *Perspectives in Science*, 8:510–512, 2016. ISSN 2213-0209. doi: https://doi.org/10.1016/j.pisc.2016.05.010. URL https://www.sciencedirect.com/science/article/pii/S2213020916301446. Recent Trends in Engineering and Material Sciences.

[165] M. Shyu, S. Chen, K. Sarinnapakorn, and L. Chang. A novel anomaly detection scheme based on principal component classifier. In *Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with the Third IEEE International Conference on Data Mining (ICDM03)*, page 172–179, 2003.

[166] S. Almotairi, A. Clark, G. Mohay, and J. Zimmermann. A technique for detecting new attacks in low-interaction honeypot traffic. In *2009 Fourth International Conference on Internet Monitoring and Protection*, pages 7–13, 2009. doi: 10.1109/ICIMP.2009.9.

[167] Lior Rokach. Ensemble methods for classifiers. In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 957–980. Springer, 2005.

[168] Gonzalo Martínez-Muñoz, Daniel Hernández-Lobato, and Alberto Suárez. An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(2):245–259, 2009. doi: 10.1109/TPAMI.2008.78. URL https://doi.org/10.1109/TPAMI.2008.78.

[169] P. Shunmugapriya and S. Kanmani. Optimization of stacking ensemble configurations through artificial bee colony algorithm. *Swarm Evol. Comput.*, 12:24–32, 2013. doi: 10.1016/j.swevo.2013.04.004. URL https://doi.org/10.1016/j.swevo.2013.04.004.

[170] M. Paz Sesmero Lorente, Agapito Ledezma, and Araceli Sanchis. Generating ensembles of heterogeneous classifiers using stacked generalization. *WIREs Data Mining Knowl. Discov.*, 5(1):21–34, 2015. doi: 10.1002/widm.1143. URL https://doi.org/10.1002/widm.1143.

[171] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16:321–357, 2002. doi: 10.1613/jair.953. URL https://doi.org/10.1613/jair.953.

[172] Guillaume Lemaitre, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *J. Mach. Learn. Res.*, 18:17:1–17:5, 2017. URL http://jmlr.org/papers/v18/16-365.html.

[173] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[174] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael A. Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In Francesco Bonchi, Foster J. Provost, Tina Eliassi-Rad, Wei Wang, Ciro Cattuto, and Rayid Ghani, editors, *5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018*, pages 80–89. IEEE, 2018. doi: 10.1109/DSAA.2018.00018. URL https://doi.org/10.1109/DSAA.2018.00018.

[175] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4765–4774, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[176] Kjersti Aas, Martin Jullum, and Anders Løland. Explaining individual predictions when features are dependent: More accurate approximations to shapley values. *Artif. Intell.*, 298:103502, 2021. doi: 10.1016/j.artint.2021.103502. URL https://doi.org/10.1016/j.artint.2021.103502.

[177] Erik Strumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowl. Inf. Syst.*, 41(3):647–665, 2014. doi: 10.1007/s10115-013-0679-x. URL https://doi.org/10.1007/s10115-013-0679-x.

[178] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2021. doi: 10.3390/e23010018. URL https://doi.org/10.3390/e23010018.

[179] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. Interpretable machine learning - A brief history, state-of-the-art and challenges. In Irena Koprinska, Michael Kamp, Annalisa Appice, Corrado Loglisci, Luiza Antonie, Albrecht Zimmermann, Riccardo Guidotti, Özlem Özgöbek, Rita P. Ribeiro, Ricard Gavaldà, João Gama, Linara Adilova, Yamuna Krishnamurthy, Pedro M. Ferreira, Donato Malerba, Ibéria Medeiros, Michelangelo Ceci, Giuseppe Manco, Elio Masciari, Zbigniew W. Ras, Peter Christen, Eirini Ntoutsi, Erich Schubert, Arthur Zimek, Anna Monreale, Przemyslaw Biecek, Salvatore Rinzivillo, Benjamin Kille, Andreas Lommatzsch, and Jon Atle Gulla, editors, *ECML PKDD 2020 Workshops - Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2020): SoGood 2020, PDFL 2020, MLCS 2020, NFMCP 2020, DINA 2020, EDML 2020, XKDD 2020 and INRA 2020, Ghent, Belgium, September 14-18, 2020, Proceedings*, volume 1323 of *Communications in Computer and Information Science*, pages 417–431. Springer, 2020. doi: 10.1007/978-3-030-65965-3_28. URL https://doi.org/10.1007/978-3-030-65965-3_28.

Appendices

Appendix A

CICIDS2017 feature description

Table A.1: CICIDS2017 feature description [6, 7]

No.	Feature Name	Feature Description
1	FlowID	Flow ID number
2	Source IP	Source IP address
3	Source Port	Source port number
4	Destination IP	Destination IP address
5	Destination Port	Destination port number
6	Protocol	Protocol
7	Timestamp	Timestamp of the flow
8	Total Fwd Packets	Total packets in the forward direction
9	Total Backward Packets	Total packets in the backward direction
10	Total Length of Fwd Packets	Total size of packet in forward direction
11	Total Length of Bwd Packets	Total size of packet in backward direction
12	Fwd Packet Length Max	Maximum size of packet in forward direction
13	Fwd Packet Length Min	Minimum size of packet in forward direction
14	Fwd Packet Length Mean	Average size of packet in forward direction
15	Fwd Packet Length Std	Standard deviation size of packet in forward direction
16	Bwd Packet Length Max	Maximum size of packet in backward direction
17	Bwd Packet Length Min	Minimum size of packet in backward direction
18	Bwd Packet Length Mean	Mean size of packet in backward direction
19	Bwd Packet Length Std	Standard deviation size of packet in backward direction
20	Flow Bytes/s	Number of packets transferred per second
21	Flow Packets/s	Number of packets transferred per second
22	Fwd Packets/s	Number of forward packets per second
23	Bwd Packets/s	Number of backward packets per second
24	Min Packet Length	Minimum length of a packet
25	Max Packet Length	Maximum length of a packet
26	Packet Length Mean	Mean length of a packet
27	Packet Length Std	Standard deviation length of a packet
28	Packet Length Variance	Variance length of a packet
29	Down/Up Ratio	Download and upload ratio
30	Avg Fwd Segment Size	Average size observed in the forward direction
31	Avg Bwd Segment Size	Average size observed in the backward direction
32	Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction
33	Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction
34	Fwd Avg Bulk Rate	Average number of bulk rates in the forward direction
35	Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction
36	Bwd Avg Packets/Bulk	Average number of packets bulk rate in the backward direction
37	Bwd Avg Bulk Rate	Average number of bulk rates in the backward direction
38	Init Win bytes forward	Number of bytes sent in initial window in the forward direction
39	Init Win bytes backward	Number of bytes sent in initial window in the backward direction
40	Act data pkt fwd	Number of packets with at least 1 byte of TCP data payload in the forward direction
41	Min seg size forward	Minimum segment size observed in the forward direction
42	Flow Duration	Flow duration
43	Flow IAT Mean	Average time between two packets sent in the flow
44	Flow IAT Std	Standard deviation time between two packets sent in the flow
45	Flow IAT Max	Maximum time between two packets sent in the flow
46	Flow IAT Min	Minimum time between two packets sent in the flow
47	Fwd IAT Total	Total time between two packets sent in the forward direction
48	Fwd IAT Mean	Mean time between two packets sent in the forward direction
49	Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
50	Fwd IAT Max	Maximum time between two packets sent in the forward direction
51	Fwd IAT Min	Minimum time between two packets sent in the forward direction
52	Bwd IAT Total	Total time between two packets sent in the backward direction
53	Bwd IAT Mean	Mean time between two packets sent in the backward direction
54	Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
55	Bwd IAT Max	Maximum time between two packets sent in the backward direction
56	Bwd IAT Min	Minimum time between two packets sent in the backward direction

57	Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
58	Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
59	Fwd URG Flags	Number of times the URG ag was set in packets travelling in the forward direction (0 for UDP)
60	Bwd URG Flags	Number of times the URG ag was set in packets travelling in the backward direction (0 for UDP)
61	FIN Flag Count	Number of packets with FIN
62	SYN Flag Count	Number of packets with SYN
63	RST Flag Count	Number of packets with RST
64	PSH Flag Count	Number of packets with PUSH
65	ACK Flag Count	Number of packets with ACK
66	URG Flag Count	Number of packets with URG
67	CWE Flag Count	Number of packets with CWE
68	ECE Flag Count	Number of packets with ECE
69	Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
70	Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
71	Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
72	Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
73	Fwd Header Length	Total bytes used for headers in the forward direction
74	Bwd Header Length	Total bytes used for headers in the backward direction
75	Average Packet Size	Packet average size
76	Active Mean	Mean time a flow was active before becoming idle
77	Active Std	Standard deviation time a flow was active before becoming idle
78	Active Max	Maximum time a flow was active before becoming idle
79	Active Min	Minimum time a flow was active before becoming idle
80	Idle Mean	Mean time a flow was idle before becoming active
81	Idle Std	Standard deviation time a flow was idle before becoming active
82	Idle Max	Maximum time a flow was idle before becoming active
83	Idle Min	Minimum time a flow was idle before becoming active
84	Label	Indicates whether the flow is an attack or not

Appendix B

NSL-KDD feature description

Table B.1: NSL-KDD feature description [8, 9]

No.	Feature Name	Feature Description
1	Duration	Length of the connection in seconds.
2	protocol_type	Type of the protocol.
3	Service	Network service on the destination.
4	src_bytes	The number of data bytes transferred from source to destination.
5	dst_bytes	The number of data bytes transferred from destination to source.
6	Flag	Connection status (normal or error).
7	Land	If the connection is from/to the same host/port, then 1 or 0 otherwise.
8	wrong_fragment	The number of wrong fragments.
9	Urgent	The number of urgent packets.
10	Hot	The number of hot indicators.
11	num_failed_logins	The number of failed login attempts.
12	logged_in	If successfully logged in, then 1 or 0 otherwise.
13	num_compromised	The number of compromised conditions.
14	root_shell	If root shell is obtained, then 1 or 0 otherwise.
15	su_attempted	If su root command is attempted, then 1 or 0 otherwise.
16	num_root	The number of root accesses.
17	num_file_creations	The number of file creation operations.
18	num_shells	The number of shell prompts.
19	num_access_files	The number of operations on access control files.
20	num_outbound_cmds	The number of outbound commands in an ftp session.
21	is_hot_login	If the login belongs to the hot list, then 1 or 0 otherwise.
22	is_guest_login	If the login is a guest login, then 1 or 0 otherwise.
23	Count	The number of connections to the same host as the current connection in the past two seconds.
24	serror_rate	The percentage of connections that have SYN errors.
25	rerror_rate	The percentage of connections that have REJ errors.
26	same_srv_rate	The percentage of connections to the same service.
27	diff_srv_rate	The percentage of connections to different services
28	srv_count	The number of connections to the same service as the current connection in the past two seconds.
29	srv_serror_rate	The percentage of connections that have SYN errors.
30	srv_rerror_rate	The percentage of connections that have REJ errors.
31	srv_diff_host_rate	The percentage of connections to different hosts.
32	dst_host_count	The total number of connections having the same destination and host IP address.
33	dst_host_srv_count	The total number of connections having the same port number.
34	dst_host_same_srv_rate	The percentage of connections that were to the same service among the connections collected in dst_host_count.
35	dst_host_diff_srv_rate	The percentage of connections that were to different services among the connections collected in dst_host_count.
36	dst_host_same_src_port_rate	The percentage of connections that were to the same source port among the connections collected in dst_host_srv_count.
37	dst_host_srv_diff_host_rate	The percentage of connections that were to different destination machines among the connections collected in dst_host_srv_count.
38	dst_host_serror_rate	The percentage of connections that have activated the flag (s0, s1, s2 or s3) among the connections collected in dst_host_count.
39	dst_host_srv_serror_rate	The percentage of connections that have activated the flag (s0, s1, s2 or s3) among the connections collected in dst_host_srv_count.
40	dst_host_rerror_rate	The percentage of connections that have activated the flag (REJ) among the connections collected in dst_host_count.
41	dst_host_srv_rerror_rate	The percentage of connections that have activated the flag (REJ) among the connections collected in dst_host_srv_count.
42	Class	Indicates if the flow is an attack or not.
43	difficulty_level	Class/flow difficulty level.

Appendix C

CICIDS2017 Information Gain

Table C.1: CICIDS2017 Information Gain

No.	Feature	IG	No.	Feature	IG
1	Average Packet Size	0.5794	**39**	Subflow Bwd Packets	0.2569
2	Packet Length Variance	0.5758	**40**	Bwd IAT Total	0.2507
3	Packet Length Std	0.5753	**41**	Bwd IAT Mean	0.2354
4	Packet Length Mean	0.5545	**42**	Fwd Packet Length Std	0.2307
5	Total Length of Bwd Packets	0.5158	**43**	Bwd Packet Length Min	0.2261
6	Subflow Bwd Bytes	0.5156	**44**	Subflow Fwd Packets	0.2162
7	Bwd Packet Length Mean	0.5021	**45**	Total Fwd Packets	0.2161
8	Avg Bwd Segment Size	0.5020	**46**	Fwd IAT Std	0.2152
9	Total Length of Fwd Packets	0.4973	**47**	Active Min	0.1788
10	Init_Win_bytes_backward	0.4825	**48**	Idle Max	0.1787
11	Bwd Packet Length Max	0.4797	**49**	Active Mean	0.1768
12	Max Packet Length	0.4790	**50**	Active Max	0.1720
13	Init_Win_bytes_forward	0.4569	**51**	Idle Mean	0.1689
14	Fwd Packet Length Max	0.4549	**52**	Idle Min	0.1665
15	Subflow Fwd Bytes	0.4326	**53**	Bwd IAT Std	0.1533
16	Avg Fwd Segment Size	0.4246	**54**	min_seg_size_forward	0.1453
17	Fwd Packet Length Mean	0.4245	**55**	Down/Up Ratio	0.1333
18	Flow Bytes/s	0.3908	**56**	PSH Flag Count	0.1015
19	Flow IAT Max	0.3894	**57**	act_data_pkt_fwd	0.0976
20	Flow Duration	0.3613	**58**	Idle Std	0.0418
21	Fwd IAT Max	0.3478	**59**	ACK Flag Count	0.0356
22	Flow Packets/s	0.3368	**60**	URG Flag Count	0.0348
23	Bwd Header Length	0.3368	**61**	Active Std	0.0328
24	Fwd Packets/s	0.3346	**62**	FIN Flag Count	0.0247
25	Fwd IAT Total	0.3343	**63**	SYN Flag Count	0.0077
26	Bwd Packets/s	0.3334	**64**	Fwd PSH Flags	0.0076
27	Fwd Header Length	0.3236	**65**	Fwd Avg Packets/Bulk	0.0005
28	Fwd IAT Mean	0.3154	**66**	Bwd PSH Flags	0.0003
29	Flow IAT Mean	0.3148	**67**	Bwd Avg Packets/Bulk	0.0003
30	Flow IAT Min	0.2968	**68**	Fwd Avg Bulk Rate	0.0002
31	Bwd Packet Length Std	0.2913	**69**	CWE Flag Count	0.0001
32	Fwd IAT Min	0.2805	**70**	ECE Flag Count	0.0001
33	Bwd IAT Min	0.2702	**71**	Bwd Avg Bulk Rate	0.0001
34	Fwd Packet Length Min	0.2690	**72**	Bwd URG Flags	0
35	Min Packet Length	0.2665	**73**	Fwd URG Flags	0
36	Bwd IAT Max	0.2665	**74**	RST Flag Count	0
37	Flow IAT Std	0.2577	**75**	Bwd Avg Bytes/Bulk	0
38	Total Backward Packets	0.2570	**76**	Fwd Avg Bytes/Bulk	0

Appendix D

NSL-KDD Information Gain

Table D.1: NSL-KDD Information Gain

No.	Feature	IG	No.	Feature	IG
1	src_bytes	0.5414	**62**	flag_sh	0.0046
2	dst_bytes	0.4302	**63**	service_urp_i	0.0045
3	dst_host_same_srv_rate	0.3372	**64**	service_login	0.0044
4	same_srv_rate	0.3274	**65**	service_name	0.0044
5	dst_host_diff_srv_rate	0.3267	**66**	service_daytime	0.0044
6	diff_srv_rate	0.3266	**67**	service_netbios_dgm	0.0043
7	flaf_sf	0.3224	**68**	service_sql_net	0.0042
8	logged_in	0.2890	**69**	service_kshell	0.0041
9	dst_host_srv_count	0.2829	**70**	service_hostnames	0.0041
10	dst_host_serror_rate	0.2574	**71**	service_echo	0.0041
11	count	0.2436	**72**	service_bgp	0.0039
12	serror_rate	0.2303	**73**	num_root	0.0038
13	dst_host_srv_serror_rate	0.2244	**74**	service_ldap	0.0038
14	srv_serror_rate	0.2108	**75**	service_netbios_ns	0.0037
15	service_http	0.2093	**76**	service_netbios_domain	0.0037
16	dst_host_same_src_port_rate	0.2086	**77**	service_mtp	0.0035
17	flag_s0	0.1929	**78**	flag_s3	0.0034
18	dst_host_srv_diff_host_rate	0.1874	**79**	service_ssh	0.0033
19	service_private	0.1728	**80**	service_nnsp	0.0033
20	dst_host_count	0.1447	**81**	wrong_fragment	0.0033
21	dst_host_rerror_rate	0.1209	**82**	service_gopher	0.0032
22	srv_diff_host_rate	0.1150	**83**	service_urh_i	0.0030
23	dst_host_srv_rerror_rate	0.1128	**84**	num_access_files	0.0027
24	rerror_rate	0.0950	**85**	num_compromised	0.0026
25	srv_rerror_rate	0.0881	**86**	flag_oth	0.0026
26	flag_reg	0.0574	**87**	urgent	0.0026
27	srv_count	0.0508	**88**	service_klogin	0.0025
28	service_domain_u	0.0494	**89**	service_tim_i	0.0024
29	duration	0.0374	**90**	service_ctf	0.0024
30	service_smtp	0.0321	**91**	service_exec	0.0023
31	protocol_type_udp	0.0313	**92**	service_pm_dump	0.0022
32	protocol_type_icmp	0.0194	**93**	service_netbios_ssn	0.0020
33	service_ftp_data	0.0174	**94**	service_harvest	0.0019
34	flag_rsto	0.0169	**95**	service_remote_job	0.0018
35	service_telnet	0.0151	**96**	num_shells	0.0017
36	service_eco_i	0.0143	**97**	service_time	0.0016
37	protocol_type_tcp	0.0142	**98**	service_nntp	0.0014
38	flag_rstr	0.0134	**99**	service_shell	0.0012
39	service_ecr_i	0.0133	**100**	service_discard	0.0012
40	service_imap4	0.0096	**101**	service_systat	0.0012
41	hot	0.0087	**102**	service_printer	0.0012
42	service_csnet_ns	0.0084	**103**	service_http_2784	0.0010
43	service_iso_tsap	0.0082	**104**	service_X11	0.0007
44	service_finger	0.0082	**105**	service_rje	0.0006
45	service_sunrpc	0.0079	**106**	service_http_8001	0.0004
46	service_Z39_50	0.0073	**107**	is_host_login	0.0004
47	service_courier	0.0069	**108**	service_ntp_u	0.0003
48	service_supdup	0.0068	**109**	root_shell	0.0001
49	service_vmnet	0.0059	**110**	num_file_creations	0
50	service_ftp	0.0059	**111**	su_attempted	0
51	service_efs	0.0059	**112**	service_tftp_u	0
52	service_auth	0.0057	**113**	flag_s1	0
53	service_pop_3	0.0057	**114**	flag_s2	0
54	service_link	0.0055	**115**	service_red_i	0
55	service_uucp_path	0.0054	**116**	land	0
56	service_http_443	0.0053	**117**	service_aol	0
57	service_vmnet	0.0051	**118**	service_pop_2	0
58	service_uucp	0.0050	**119**	is_guest_login	0
59	service_other	0.0048	**120**	service_IRC	0
60	num_failed_logins	0.0047	**121**	flag_rstos0	0
61	service_netstat	0.0047			