A Systematic Methodology to Evaluating Optimised Machine Learning Based Network Intrusion Detection Systems

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

of Rhodes University

Hatitye Chindove

Grahamstown, South Africa
June 20, 2022

Abstract

A network intrusion detection system (NIDS) is essential for mitigating computer network attacks in various scenarios. However, the increasing complexity of computer networks and attacks makes classifying unseen or novel network traffic challenging. Supervised machine learning techniques (ML) used in a NIDS can be affected by different scenarios. Thus, dataset recency, size, and applicability are essential factors when selecting and tuning a machine learning classifier.

This thesis explores developing and optimising several supervised ML algorithms with relatively new datasets constructed to depict real-world scenarios. The methodology includes empirical analyses of systematic ML-based NIDS for a near real-world network system to improve intrusion detection. The thesis is experimental heavy for model assessment. Data preparation methods are explored, followed by feature engineering techniques. The model evaluation process involves three experiments testing against a validation, untrained, and retrained set. They compare several traditional machine learning and deep learning classifiers to identify the best NIDS model.

Results show that the focus on feature scaling, feature selection methods and ML algorithm hyper-parameter tuning per model is an essential optimisation component. Distance based ML algorithm performed much better with quantile transformation whilst the tree based algorithms performed better without scaling. Permutation importance performs as a feature selection method compared to feature extraction using Principal Component Analysis (PCA) when applied against all ML algorithms explored. Random forests, Support Vector Machines and recurrent neural networks consistently achieved the best results with high macro f1-score results of 90% 81% and 73% for the CICIDS 2017 dataset; and 72% 68% and 73% against the CICIDS 2018 dataset.

Acknowledgements

I want to acknowledge the following in their contributions to this thesis:

First, I would like to extend my gratitude to my supervisor, Dr Dane Brown, who provided extended support throughout this thesis and provided feedback whenever I needed guidance.

I want to thank Professor Barry Irwin and the rest of the Rhodes University Computer Science department for an excellent opportunity to complete this Qualification. The department support made it possible for me to complete this research successfully.

Contents

List of Figures

List of Tables

List of source codes

List of Acronyms

DNN	Deep Neural Networks
LR	Logistic Regression
SVM	Support Vector Machines
MLP	Multi-Layer Perceptrons
NB	Gaussian Naive Bayes
DT	Decision Trees
RT	Random Tree
DL	Deep Learning
GA	Genetic Algorithm
IDS	Intrusion Detection System
KNN	K Nearest Neighbour
RNN	Recurrent Neural Network
ML	Machine Learning
NIDS	Network Intrusion Detection System
PCAP	Network Packet Capture
PCA	Principal Component Analysis
LDA	Linear Discriminant Analysis
RF	Random Forests
ROC	Receiver Operating Characteristic
FI	Feature Importance
PI	Permutation Importance
IG	Information Gain
LSTM	Long Short Term Memory
PI	Permutation Importance
OvA	One versus All

1 Introduction

As the world has become more dependent on computer systems, the scale of security attacks on computer infrastructure has increased (Kizza, 2013, Schneier, 2015). A 2021 IBM report[1] shows that successful attacks which result in a data breach have an average of 287 days from detection to breach containment. Early detection in these cases can assist in improving security incident response times for systems seeking to maintain a high-security availability, integrity and confidentiality posture (Denning, 1987, Zhang *et al.*, 2018). It follows that intrusion detection is an essential part of monitoring computer events. Intrusion detection systems typically execute real-time traffic classification to determine security incidents or attacks that may threaten the computer system (Valenti *et al.*, 2013). A Network-based Intrusion Detection System (NIDS) is a security tool that identifies an inside attack, outside attack and unauthorised access into a computer network.

NIDS generally use two methods to detect attacks, namely, signature-based and anomaly-based detection. Signature-based detection uses sets of collected attacker patterns implemented as pre-installed rules to detect attacks. On the other hand, anomaly-based attack detection uses traffic shape observations to measure the deviation from normal traffic (Hayes and Capretz, 2014). The use of anomaly-based detection adaptions through

[1]https://www.ibm.com/za-en/security/data-breach

machine learning (ML) techniques has shown promising results in recent studies (Ren *et al.*, 2019, Rosay *et al.*, 2020).

A key characteristic of ML algorithms is their ability to learn and improve their performance over time (Tsai *et al.*, 2009). Common strategies for ML emphasise building a framework that enhances its execution based on previous results – changing execution strategy based on recently acquired data – to classify an intrusion. ML techniques have the advantage of adaptability and capture inter-dependencies when effectively implemented. However, an immediate challenge with ML is the general need to compromise between model complexity and training duration. Selecting appropriate algorithms for NIDS is vital in addressing this problem.

Generally, using ML for intrusion detection has potential. However, it has challenges and limitations that require comprehensive review before developing ML-based NIDS.

1.1 Problem Statement

This research evaluates multiple variants of ML algorithms and their applicability to various network attack profiles. The problem statements are detailed below:

1. In real-world systems, the vast majority of traffic NIDS encounter is ‘Benign’; as such, identifying malicious traffic may be challenging.
2. NIDS constantly encounter novel attacks, and thus, it follows that they should adapt to (detect) new scenarios.

1.2 Research Question

Based on the above statements, the following research question can be formulated: ‘How can effective ML-based NIDS be developed for use in real-world systems’. This research question can be broken down into the following sub-questions:

1. What supervised ML algorithms are better fitted for improved NIDS detection rates?
2. How can ML models be optimised to achieve consistent best detection results and how can this be effectively measured?

1.3 Research Objectives

The proposed research aims to achieve the following objectives:

1. Conduct systematic experiments to investigate, design and construct a system for ML-based NIDS that addresses the NIDS problems based on the literature.
2. Investigate methods that cater for the significant data imbalance and distribution when developing ML models for NIDS.
3. Conduct an experiment to validate the ML models on a subset of the first dataset by performing parameter tuning, biased towards data generalisation.
4. Determine the most effective NIDS performance metrics and ML algorithms on modelled and unmodelled data from a different dataset.

The findings of the above objectives will reflect on each model's classification capability.

1.4 Research Contributions and Limits

Based on the first objective, this research aims to develop systematic ML techniques for NIDS that have consistency in yielding results applicable real-world data and aid future studies. Although further analysis may benefit inefficient methods, this research will not cover this in-depth analysis. Consequently, the methods used to fulfil the research follow a scientific approach of testing and observing results.

1.5 Document Structure

The remaining chapters are as follows:

Chapter 2 *Concepts and Literature Review*: Introduces the various concepts about NIDS development and other related implementations.

Chapter 3 *Experimental Design*: In addition to the testbed, the chapter presents the methodology used to construct the proposed system.

Chapters 4 *Results*: Presents the results obtained by the experiments.

Chapter 5 *Discussion*: Discusses the results and compares them with related studies.

Chapter 6 *Conclusion*: This chapter concludes the thesis, highlights the research contributions and provides directions for future work.

2 Concepts and Literature Review

This chapter provides a broad overview of intrusion detection systems and their related literature. Section 2.1 introduces the concept of intrusion detection systems. Section 2.2 provides a brief outline of the methods adopted for Network Intrusion Detection System (NIDS) development. Section 2.3 explains concepts of the standard machine learning (ML) methodology. Section 2.4 discusses related NIDS that are considered for adoption in Chapter 3. Focus is placed on methods that contribute towards ML-based NIDS.

2.1 Intrusion Detection Systems

The history of the intrusion detection system (IDS) dates back to the 1980s, where they were termed “Intrusion Detection Expert System” (Lunt and Jagannathan, 1988). An IDS monitors the information flowing through the network and alerts users of a probable computer or network misuse event (Denning, 1987, Thakkar and Lohiya, 2020). IDS performs network and user activity analysis through information made available by monitored systems. Such activities may originate from an internal or external computer network.

For an IDS, a ‘Benign’ class is a regular normal event, while a ‘Malicious’ class may indicate misuse or abuse on a computer network (Javaid *et al.*, 2016). IDS capabilities are achieved through activity classification techniques broadly categorised as signature-based or anomaly-based methods (Liao *et al.*, 2013).

The signature-based method involves the process of comparing pre-captured patterns of monitored events – known as 'Benign' or 'Malicious' patterns – against captured events to recognise possible intrusions (Jones and Sielken, 2000). An anomaly-based method involves monitoring deviations from a known behaviour derived from regular network activities over time (Javitz *et al.*, 1991, Axelsson, 2000). As a result, anomaly-based detection draws on the potential to detect new intrusion events (García-Teodoro *et al.*, 2009), which has led to more research into this area. In summary, signature-based detection compares activity to established rules, while anomaly-based detection compares activity to profiles.

Network administrators generally deploy these detection techniques to monitor a computer application, client or host, network communications equipment or a hybrid method. A NIDS captures information passing through network communications equipment by analysing a stream of data packets (Liao *et al.*, 2013). The focus of this thesis will be on aspects related only to NIDS development.

2.2 NIDS Model Development Methods

This section briefly describes the influential signature-based and anomaly-based detection studies.

2.2.1 Signature-based Detection

Signature-based detection analyses activity characteristics of captured network data that resemble a predefined pattern or string of an intrusion event (Kayacik *et al.*, 2005). The method fundamentally assumes that malicious activities have a pattern that differs from the norm.

Signature-based NIDS, such as Zeek (formerly Bro)[1], Snort[2] and Suricata[3], are examples of tools that have received widespread adoption (García-Teodoro *et al.*, 2009). Studies by Cardenas *et al.* (2006) and Hofstede *et al.* (2017) have shown that signature-based methods obtain high accuracy on malicious pattern detection of known attacks and provide ease of access for detail used for contextual analysis.

However, this approach has had challenges related to occasionally incorrectly classifying benign traffic, leading to high false-positive rates (Chowdhury *et al.*, 2017). This is most apparent in detection tasks for unknown attacks or partially modelled data (Pérez *et al.*, 2017, Xian *et al.*, 2018, Zhang *et al.*, 2020). Liao *et al.* (2013) mention other challenges such as the lack of understanding of network states and protocols, keeping signatures up to date, and knowledge maintenance as time-consuming. Based on this viewpoint, alternatives that may cater to these challenges appear attractive.

2.2.2 Anomaly-based Detection

Anomaly-based detection analyses the deviation of known behaviours and profiling of the expected behaviours derived from monitoring regular activities (Javitz *et al.*, 1991, Doshi *et al.*, 2018). The method fundamentally assumes that malicious behaviour profiles differ from benign behaviour, thus has theoretical effectiveness at detecting new scenarios (Wang and Stolfo, 2004). However, based on observed security events in literature (Liao *et al.*, 2013), the method occasionally yields weak model performance.

Anomaly-based techniques broadly include state transition analysis, expert systems, and signature analysis. These techniques possess high accuracy and low false-positive rates. The techniques pose a knowledge maintenance challenge for each attack, due to the dependency on manual, careful and detailed analysis (Debar *et al.*, 2000). Anomaly-based systems also use ML techniques that can learn and improve performance on specific tasks or task groups over time (Tsai *et al.*, 2009).

[1]https://zeek.org/
[2]https://www.snort.org/
[3]https://suricataids.org/

ML has the advantage of flexibility, adaptability, and capturing of inter-dependencies when effectively implemented. However, selecting appropriate algorithms is essential in ML implementations that yield good performance.

2.3 Machine Learning-Based NIDS

This section explains basic concepts of ML and general supporting literature on ML-based NIDS.

2.3.1 Overview of Machine Learning

ML is an approach to developing computer system models that optimise a performance criterion, using example data or experience, without explicit programming (Valiant, 1984). The model definition has a set of parameters, and learning is the execution of a computer program to optimise the model's parameters using training data or experience. Edgar and Manz (2017) defines it as "a field of study that uses computational algorithms to turn empirical data into usable models." Generally, ML models are applied to make predictions, gain knowledge, or both based on the training data provided.

ML often uses statistical models that make inferences from a sample (Russell and Norvig, 2002, Edgar and Manz, 2017, Alpaydin, 2020). The primary aim is to allow the computers to learn without human assistance and adjust actions accordingly. ML can be broadly classified into three main types, supervised, unsupervised and reinforcement learning (Russell and Norvig, 2002).

Supervised learning models development involves using a training dataset to create a function in which each training data contains a pair of the input and output vectors – features and class labels, respectively (Tsai *et al.*, 2009). The learning task is to compute the approximate relation between the input-output examples to create a classifier (model). A trained model can classify unknown examples into learned class labels. The ability to learn and adapt without explicit programming is not easy when using signature-based methods (Laskov *et al.*, 2005).

There is a lack of prominent open-source ML-based NIDS in the literature reviewed. However, a growing number of implementations exist on a proprietary commercial scale, such as DarkTrace[4], Awake Security[5] and AWS Guard Duty[6]. These proprietary solutions are generally viewed as a 'black box' due to the lack of open literature about their ML application techniques. These solutions are not explored any further in this thesis. Note, Section 2.4 introduces several ML algorithms suited for ML-based NIDS.

Furthermore, Sommer and Paxson (2010) highlights challenges in evaluating NIDS performance. As such, a comprehensive review is required to ascertain the strengths and weaknesses of more modern ML-based NIDS – including the more recent Deep Learning (DL) algorithms.

2.3.2 Data Preparations

Datasets

Based on the problem statement in Section 1.1, the datasets used against NIDS must depict real-world systems. It follows that, research on datasets used in NIDS experiment environments have focused on the need to generate synthetic traffic to represent real-world scenarios (García-Teodoro *et al.*, 2009). Abdulraheem and Ibraheem (2019) focused on improving dataset quality and established crucial criteria[7] for building a reliable NIDS research dataset.

These dataset quality criteria concur with Corona *et al.* (2013)'s study on attribute stipulations for real-world intrusion datasets used in research that aims to resolve security problems. However, capturing these datasets may prove a challenge because of the rapid

[4]www.darktrace.com/

[5]www.awakesecurity.com/

[6]www.aws.amazon.com/guardduty/

[7]Criteria include network configuration, traffic representation, labelling, network interaction, packet capture, protocol availability, attack diversity, anonymity, heterogeneity, feature set, and metadata.

evolution of intrusions. NIDS development falls in this class of security problems. Therefore, selecting a dataset that fits these scenarios is crucial for practical implementations.

The KDD 99 dataset has been the most popular for NIDS research and has 41 features (Aksu *et al.*, 2018b). The significant challenge of this dataset is in the presence of redundancies, which has made ML model training biased towards the most represented classes resulting in poor model performance against underrepresented classes. Viegas *et al.* (2017) and Ring *et al.* (2019) mention several other limitations which make the dataset results unrealistic for real-world use. Most studies reviewed use the KDD 99 or its variants – including UNSW-NB[8], Kyoto 2006+[9] and AWID[10] – that supplement the original dataset's limitations (Stein *et al.*, 2005, Su, 2011, Li *et al.*, 2012, Almseidin *et al.*, 2017, Chuan-long *et al.*, 2017, Zhang *et al.*, 2018).

More recent studies (Ahmim *et al.*, 2019, Abdulhammed *et al.*, 2019, Rosay *et al.*, 2020, Kurniabudi *et al.*, 2020) make use of the CICIDS 2017[11] and CICIDS 2018[12] datasets. Both datasets are composed of more than 80 features, possess more attack categories, and account for more of Abdulraheem and Ibraheem (2019)'s quality criteria than KDD 99 and its variants (Thakkar and Lohiya, 2020).

Balancing and Sampling Methods

NIDS datasets that have real-world depiction are likely to be imbalanced due to the expected majority of benign traffic, as is the case for CICIDS 2017 (Panigrahi and Borah, 2018). The imbalance leads to a severe skew in class distribution overly emphasising majority classes. Imbalances may negatively affect ML modelling; however, this better depicts the real-world traffic (Singh *et al.*, 2015). Random Undersampling, Random Oversampling and Synthetic Minority Oversampling Technique (SMOTE) are some resampling strategies that can be used to mitigate class imbalance (Branco *et al.*, 2015).

[8]https://www.unsw.adfa.edu.au/unswcanberracyber/cybersecurity/ADFANB15Datasets
[9]http://www.takakura.com/Kyoto_data/
[10]http://icsdweb.aegean.gr/awid/
[11]https://www.unb.ca/cic/datasets/ids2017.html
[12]https://www.unb.ca/cic/datasets/ids2018.html

The **Random Undersampling** strategy deletes majority class instances at random until a more balanced class distribution is reached (Wang and Yao, 2009). It has the advantage of removing intentional sampling bias due to its random nature.

The **Random Oversampling** strategy randomly selects minority class instances with replacements and adds them to training datasets . The method increases the risk of overfitting since the instances generated are copies of randomly selected minority classes (Songwattanasiri and Sinapiromsaran, 2010).

The **SMOTE** strategy selects samples in close feature space proximity to create a synthetic sample (Bowyer *et al.*, 2011). Similar synthetic instance generation methods include Borderline SMOTE, ADASYN, SMOTE NC and SVM SMOTE. Note; synthetic oversampling methods are vulnerable to introducing ambiguous synthetic instances that may substantially overlap with other original classes.

2.3.3 Feature Scaling and Transformation

Scaling and transformation are crucial parts for data representation and standardisation for numeric feature creation, rescaling, and adjusting skewed features for ML modelling (Tax and Duin, 2000), as depicted from a research question in Section 1.2. Standardisation assures better input data quality and probability distribution to fit ML algorithms (Rosay *et al.*, 2020). Several scaling and transformation algorithms including Standard, Quantile, Power and Robust methods can be used[13].

2.3.4 Feature space Reduction

Feature Space reduction is used to deal with mapping original feature space from a high dimensional space of features to a reduced dimensional space (Mladenić, 2006). The process generally involves selecting a subset of original features and/or developing new dimensions. The use of feature selection and feature extraction methods leads to dimensionality reduction which can be leveraged to address the research question in Section 1.2.

[13]https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

Feature Selection

Feature selection is a data preprocessing activity involving the search for features present in a dataset that best reflect the difference between classes by adopting important attributes only and removing redundant data, thus giving ML algorithms unbiased results (Panda *et al.*, 2012, Siva *et al.*, 2012). Due to evolving attack scenarios, establishing feature selection criteria for NIDS is challenging (Javaid *et al.*, 2016, Viegas *et al.*, 2017).

Feature selection methods usually rely on feature importance (Reis *et al.*, 2019). Common feature importance methods include **Gini Importance (GI)** and **Permutation Importance (PI)** (Louppe *et al.*, 2013, Kurniabudi *et al.*, 2020).

GI calculates importance based on the total decrease in node impurity weighted – more details on GI are presented under Decision Trees in Section 2.3.6 – by the probability of reaching a node averaged over all decision trees of an ensemble (Menze *et al.*, 2009). PI is an iterative function that focuses on the importance based on the result produced after training a model (Altmann *et al.*, 2010). If the order of a given feature value results in less accurate predictions drawn from a model, then that feature has less importance as it reduces performance.

Feature Extraction

Several feature extraction algorithms exist, but literature has commonly adopted **Principal Component Analysis (PCA)**. The application of PCA reduces dataset complexity and maximises the total variance of all data samples (Lakhina *et al.*, 2010, Heba *et al.*, 2010). On its own, the technique is not a classifier but works as an auxiliary to other classifiers in an unsupervised manner.

PCA reduces dataset complexity by transforming correlated features into a reduced number. Subsequently, the ML classifier adopts the transformed data, which can now be trained on as uncorrelated linear combinations of the original features, reducing model complexity.

2.3.5 Model Evaluation

Performance Metrics

NIDS are prone to detection errors, and as such, a good metric choice can assist in evaluating classification algorithms' ability to fulfil the desired task. A classification model has the following desired prediction outcome of (Ferri *et al.*, 2009):

- True Positive (TP) - is an outcome where the model class prediction matches the actual class.
- True Negative (TN) - is an outcome where the prediction correctly matches the negative class.

A perfect classification model is one that only possesses TP and TN. In the case of an error, the following predictions occur on the classification model:

- False Positive (FP) - the model incorrectly predicts the negative class as the actual class.
- False Negative (FN) - the model incorrectly predicts the positive class as a negative class.

Literature commonly adopts metrics such as accuracy, precision, recall, f1-scores and ROC curves for additional model analysis (Davis and Goadrich, 2006, Ferri *et al.*, 2009, Xin *et al.*, 2018).

Accuracy is the ratio of the number of correct predictions to the total number of input samples as shown in Equation 2.1.

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

Precision is the ratio of correctly classified attack flows, in front of all the classified flows as shown in Equation 2.2.

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

Recall is the ratio of correctly classified attack flows in front of all generated flows as shown in Equation 2.3.

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

F1-score (f_1) is the harmonic mean of the precision and recall for a classification model. This metric presents a balanced metric for classifiers as shown in Equation 2.4.

$$f_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.4}$$

The **ROC curve** is a probability curve that plots the true positive rate (TPR) against the false positive rate (FPR) of a classifier. The Area Under the Curve (AUC) is a measure that represents the degree or measure of separability of classes.

2.3.6 Classification Algorithms

The rest of this section describes several supervised ML classification algorithms used in the proposed systems, including K-Nearest Neighbour, Support Vector Machines, Decision Trees, Random Forests, Multi-layer Perceptrons, Logistic Regression and Recurrent Neural Networks.

K-Nearest Neighbour

K-Nearest Neighbour (KNN) is a learning algorithm that assumes similarities to exist at a close distance. The model picks k entries in a dataset closest to the new data point using distance measures such as Euclidean or Manhattan distance. A majority vote of the most common classes among those k entries will be the class of the new data point. Figure 2.1 shows an example 2D plane representation of KNN. The Euclidean distance

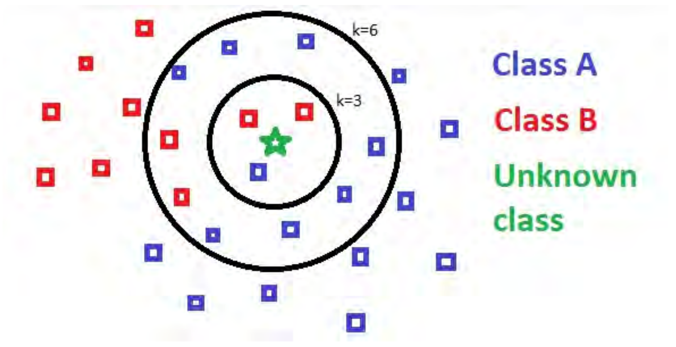

Figure 2.1: KNN Example[14].

is calculated as shown in Equation 2.5.

$$d(x, y) = \sqrt{\sum_{i=1}^{k} (a_i x - a_i y)^2} \tag{2.5}$$

where $d(x, y)$ is the Euclidean distance between two data instances x and y. Manhattan distance between two points (x_1, y_1) and (x_2, y_2) can be calculated as Equation 2.6.

$$\mid x_1 - x_2 \mid + \mid y_1 - y_2 \mid \tag{2.6}$$

For Equation 2.5, KNN makes use of Lazy learning[15] to store training data until prediction time. To assign the most common class of x, KNN sets neighbours as shown in Equation 2.7.

$$c(x) = \arg\max_{c \in C} \sum_{i=1}^{k} \delta\left(c, c\left(y_i\right)\right) \tag{2.7}$$

where $y_1, y_2, ..., y_k$ are the k nearest neighbours of x, k is the number of neighbours and $\delta\left(c, c\left(y_i\right)\right) = 1$ if c$=$ $c\left(y_i\right)$ and $\delta\left(c, c\left(y_i\right)\right) = 1$ otherwise (Jiang *et al.*, 2007).

Support Vector Machines

Support Vector Machines (SVM) is a learning method that uses a statistical learning

[14]https://towardsdatascience.com/knn-using-scikit-learn-c6bed765be7

[15]'Memorizes' data instead of learning a discriminative function.

model for classification and regression problems (Chapelle *et al.*, 1999). SVM first maps the input vector into a higher dimensional feature space and obtains an optimal hyperplane. This hyperplane aims to separate two classes of data points by finding the biggest margin between two points, as shown in Figure 2.2.

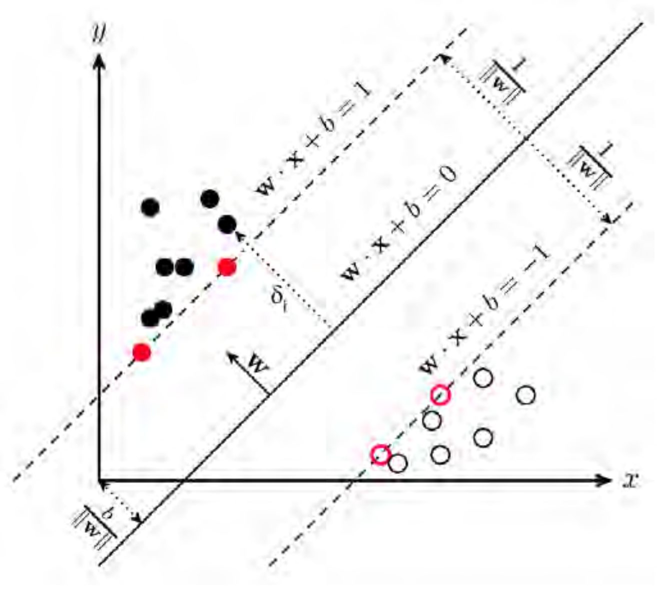

Figure 2.2: SVM Hyperplane Example[16].

Given classes $S^+ = \{x_i \mid y_i = 1\}$ and $S^- = \{x_i \mid y_i = -1\}$ are linearly separable, at least one boundary can be formed between them (Chapelle *et al.*, 1999). Support vectors are the data points for sets S^+ and S^- that are located on the boundaries of the margin and are coloured in solid and non-solid red, respectively. Rescaling of $\boldsymbol{w}$ for all x_i that are support vectors holds:

$$\boldsymbol{w} \cdot x_i + b = 1 \tag{2.8}$$

$$\boldsymbol{w} \cdot x_i + b = -1 \tag{2.9}$$

The distance d decision boundary for the margin can be represented as:

$$d = \frac{2}{\|\boldsymbol{w}\|} \tag{2.10}$$

An optimal hyperplane is a decision boundary that achieves the maximum margin between

[15]https://svm.michalhaltuf.cz

sets S^+ and S^-. SVM has the advantage of high generalisation performance without prior knowledge, even when the dimension of input space is high.

Decision Trees

Decision Trees (DT) are a learning technique based on a divide and conquer strategy, which utilises decision nodes and leaf nodes where a decision node represents a test over one of the attributes and a leaf node represents the class value (Stein *et al.*, 2005, Tsai *et al.*, 2009). Figure 2.3 shows an example of a DT used to classify weather.

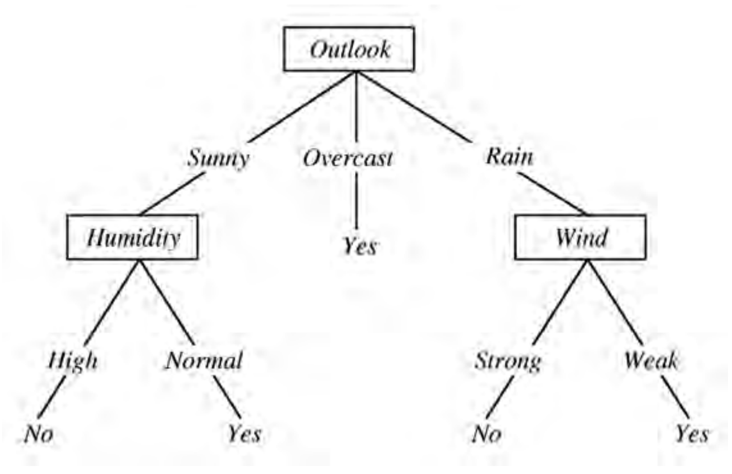

Figure 2.3: Decision Tree Example[17].

DT node splitting into multiple sub-nodes is implemented through continuous target variables or categorical target variables. Examples of categorical target variables include Gini Impurity, Information Gain and Chi-Squared.

Equation 2.11 shows the calculation for Gini Impurity.

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{2.11}$$

where p_i represents the relative frequency of a class in a dataset under observation and c is the count of classes.

[17]https://towardsdatascience.com/decision-tree-in-machine-learning-e380942a4c96

Equation 2.12 shows the calculation for Information Gain.

$$\text{Information Gain} = info(T) - \sum_{i=1}^{s} \frac{|T_i|}{|T|} \times info\,(T_i) \tag{2.12}$$

where T marks the set of decision cases and T_i(i = 1 to s) are a subset of T with value attribute A. Equation 2.13 shows the entropy function as info (T).

$$info(T) = -\sum_{j=1}^{Nclass} \frac{freq\,(C_j, T)}{|T|} \times \log_2 \left(\frac{freq\,(C_j, T)}{|T|} \right) \tag{2.13}$$

Equation 2.14 shows the calculation for Chi-Squared.

$$\text{Chi-Squared} = \sqrt{\frac{(Actual - Expected)^2}{Expected}} \tag{2.14}$$

where *Actual* is the actual child node class and *Expected* is the expected child node class based on the distribution of classes in the parent node.

Random Forests

Random Forests (RF) is a learning algorithm that consists of a large volume of individual DT that operate as an ensemble at training time (Resende and Drummond, 2018, Reis *et al.*, 2019). The training algorithm applies Bagging or Bootstrapping methods to DT learners to improve model performance by decreasing the variance of the model without increasing the bias. The training methods involve selecting and replacing a random sample from the original training dataset. Each tree in the model is then trained independently to generate results. The RF model outputs a class prediction for each tree, and through the aggregation step, the class with the most votes becomes the models' prediction, as shown in Figure 2.4.

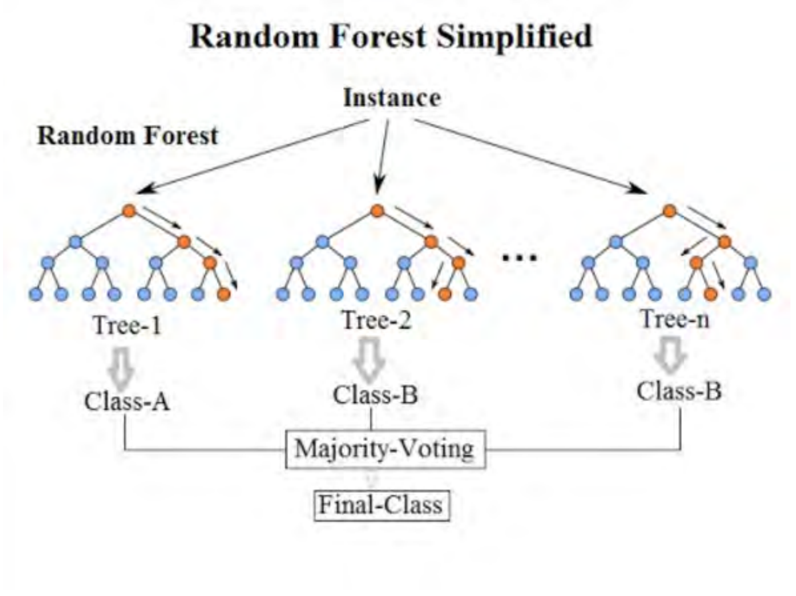

Figure 2.4: Random Forest Example[18].

RF often uses Gini Impurity for node splitting on its trees due to its less mathematically intensive nature shown in Equation 2.11.

Logistic Regression

Logistic Regression (LR) is a statistical learning model that measures the relationship between the dependent variable and one or more independent variables by estimating conditional probabilities between one and zero (Arunraj *et al.*, 2017). The predicted variable is denoted by Y when calculating the probability, and X denotes the predictor. Zero and one represent the two classes of benign and malicious traffic. Figure 2.5 shows an example of LR used to represent expertise.

[18]https://medium.com/williamkoehrsen/random-forest-simple-explanation-377895a60d2d

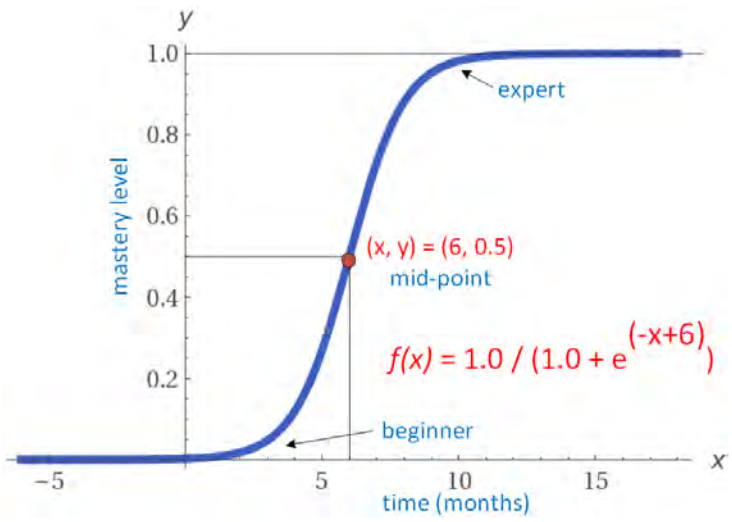

Figure 2.5: Logistic Regression Function Example[19].

Artificial Neural Networks

Artificial Neural Networks (ANN) are a learning algorithm that aims to simulate the operation of the human brain (García-Teodoro *et al.*, 2009, Van *et al.*, 2017, Muhuri *et al.*, 2020). It incorporates a collection of connected units called artificial neurons, which each have a connection between neurons with the ability to transmit a signal to another neuron (Tu, 1996). Figure 2.6 shows a hypothetical representation of an ANN.

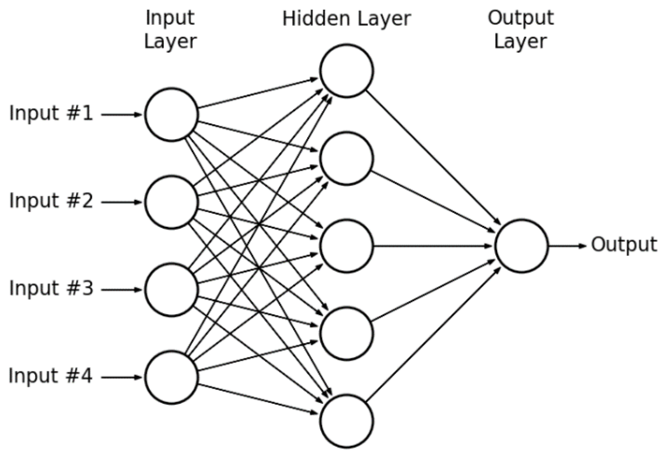

Figure 2.6: Hypothetical ANN Example.

The input layer represents features from the dataset, and the output layer represents the output labels for the respective features. The connections of the layers show weighted

[19]https://jupyter.ai/logistic-regression-ml/

paths in the network that each feature can take to reach the output label. The input layer leads to the hidden layer, which computes the sum of each possible case from the input layer, in a process called forward-propagation, when moving from left to right in Figure 2.6 (García-Teodoro *et al.*, 2009, Vinayakumar *et al.*, 2017).

To effectively utilise the ANN, the model applies an activation function to the hidden layer by providing non-linear change to the output values in the layer, capturing non-linearities within the data (Tu, 1996). Depending on the nature of the problem, several options for activation functions can be used, including Sigmoid, Tanh and ReLu functions.

Gradient descent steps are computed to find the minimum value of the function to minimise the prediction errors for the ML model. Back-propagation gathers gradients of each descent step and updates the weights used in the previous steps. Back-propagation uses the error from the output layer and propagates it back through the hidden layer, and at each of the steps, the weight is updated (García-Teodoro *et al.*, 2009, Tu, 1996).

ANN's have received recent wide adoption in the field of intrusion detection due to their adaptability to change (Chuan-long *et al.*, 2017). An area of ANN that has received recent prominence is the application of Deep Learning (DL) algorithms (Dong and Wang, 2016, Chuan-long *et al.*, 2017, Vinayakumar *et al.*, 2017, Meidan *et al.*, 2018, Shone *et al.*, 2018).

DL uses hierarchical feature learning from observational data, where there is a definition of higher-level features or factors from lower-level ones (Van *et al.*, 2017). DL techniques aim to maximise learning good feature representation of a large amount of data.

Recurrent Neural Network is a DL architecture that has significantly gained momentum in recent NIDS studies (Chuan-long *et al.*, 2017, Vinayakumar *et al.*, 2017, Meidan *et al.*, 2018). Multi-Layer Perceptrons have also seen prominence in the NIDS literature (García-Teodoro *et al.*, 2009, Tu, 1996).

Multi-layer Perceptrons

Multi-layer Perceptrons (MLP) are a feed-forward ANN designed with a series of algorithms that recognise underlying relationships in data that is not necessarily linearly separable (García-Teodoro *et al.*, 2009, Tu, 1996). Back-propagation is commonly used to train MLP models in a supervised manner (Barapatre *et al.*, 2008). An MLP consists of at least the input, hidden and output layers, as shown in Figure 2.7.

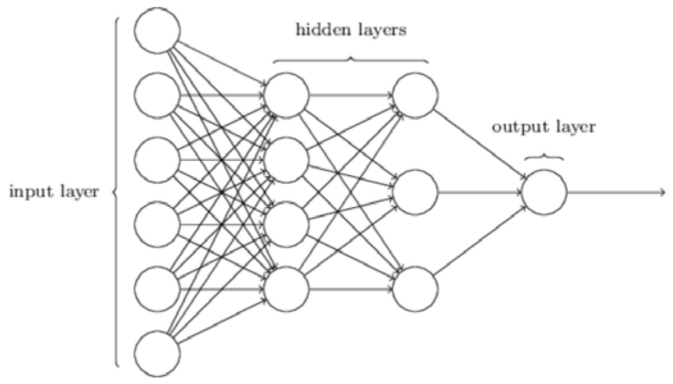

Figure 2.7: MLP Example.

Researchers have used this method to predict commands based on a sequence of previous commands or to identify the malicious behaviour of traffic patterns in NIDS (García-Teodoro *et al.*, 2009, Shenfield *et al.*, 2018).

Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of ANN that contains loops, allowing storage of temporal information within the network (Muhuri *et al.*, 2020). RNN also uses back-propagation learning via storing recurring time sequences. Figure 2.8 shows the basic structure of a RNN.

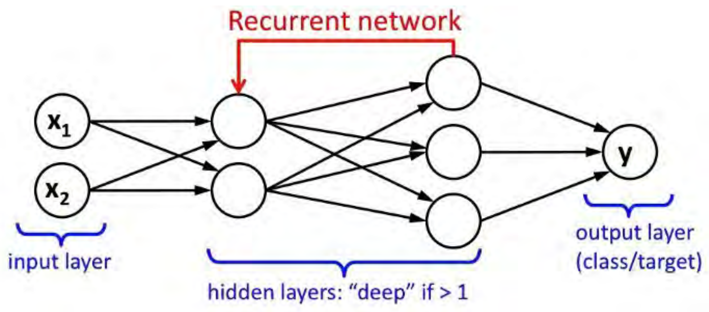

Figure 2.8: RNN Structure Example.

Therefore, for an RNN, if the order of input were to be changed, the model becomes significantly different (Chuan-long *et al.*, 2017).

Training issues can result from exploding gradients[20]. An address of exploding gradients can involve adding Gated Recurrent Units (GRU) – with No Use or No Update values for weights – and/or Long Short Term Memory networks (LSTM) – with Forget Gates and The Output Gate.

Literature has shown that this technique received increased adoption in NIDS development, and RNN variants may provide interesting comparisons against other ML methods (Chuan-long *et al.*, 2017, Vinayakumar *et al.*, 2017, Meidan *et al.*, 2018).

Figure 2.9 shows a visual summary of the grouping the classifiers presented in this section. For the purpose of this research, the classifiers presented can broadly be grouped as Tree-Based and Distance-Based methods.

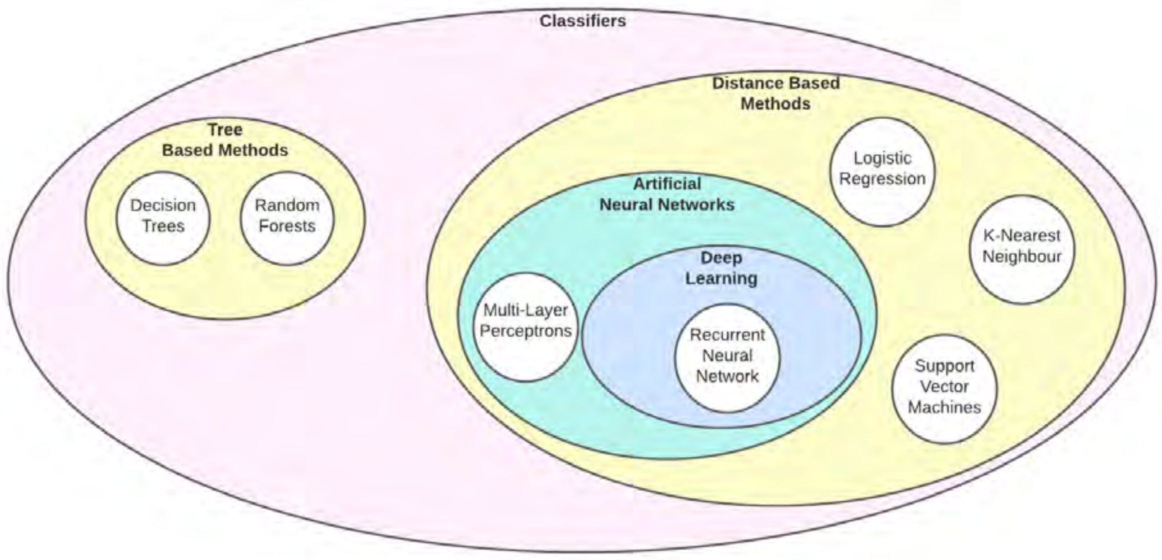

Figure 2.9: Classifiers Discussed.

2.4 Related Studies

The following section relates to ML-based NIDS implementations found in the literature.

[20]Error gradients can accumulate too quickly during weight updates, resulting in unscalable values or buffer overflow.

2.4.1 Related NIDS

Many studies have been conducted to develop ML methods that are used in NIDS implementations. The following studies present notable model development techniques that were able to attain good results.

Kurniabudi *et al.* (2020) studied Information Gain as a feature selection method. They use a filter-based method for attribute ranking and noise reduction, allowing features with the most information base to improve attack detection accuracy. The CICIDS 2017 dataset is used, with only 20% of the dataset for the experiment. They use a train/test split ratio of 0.7/0.3 and 10 fold cross-validation method. After feature selection, several classifiers, including RF, Bayes Network, Random Tree (RT), Naive Bayes (NB) and DT (J48), are evaluated and compared against each other using recall, false negative rate, precision, recall and accuracy metrics.

They found features *Packet Length Std*, *Total Length of Bwd Packets*, *Subflow Bwd Bytes* and *Destination Port* as the most important (Kurniabudi *et al.*, 2020). RF and RT classifiers yielded an accuracy of 0.96. RF, Bayes Network, RT and DT classifiers are able to effectively detect classes Benign, DoS/DDoS, Port Scan, Brute Force, Web XSS, Web Brute Force and Web SQL Injection using feature subsets of 7, 35 or 52. The classifier achieves a high recall of 0.80 using 22 or 35 features. The model obtained perfect accuracy when presented with a feature subset of 52, 57 or 77.

However, data sampling methods used, feature scaling or transformation methods adopted, the effect of the train and test data split, and model hyperparameter tuning used are not referenced in Kurniabudi *et al.* (2020)'s study.

Reis *et al.* (2019) studied Gini, Permutation and Drop-column Importance feature selection methods for optimal feature rankings for an effective ML-based NIDS. Gini is used as the primary feature selection method and used with RF and DT classifiers. The CICIDS 2017 dataset is used for model evaluation. Similar to Kurniabudi *et al.* (2020), feature selection is used to identify irrelevant features.

Duplicate row entries are initially removed from the dataset, and eight[21] of the features available are reinitialised with a '0' value, leaving 69 features. Stratified sampling takes 0.3 of the data in the dataset and creates a new dataset with the same proportion of benign and malicious classes. The proportional cut also reduces attacks of minority classes such as *Heartbleed*, *Infiltration* and *Web SQL Injections* to a mere single digit of samples. Classification for *DDoS*, *Heartbleed*, *Infiltration* and *SSH-Patator* were the best-detected attacks, whilst the *Web Brute Force* attacks were harder to detect. RF had the best performance with the insignificant classification performance between 10 and 70 feature selection. A difference of f1-score 0.002, an FPR of ≈ 0 and a false negative rate of 0.007 were observed over the feature selection range.

Reis *et al.* (2019) study has no discussions on how feature scaling, hyperparameter tuning and classifier evaluation criteria are adopted. Aspects of a class imbalance concerning the dataset are also not discussed, thus making the experiment challenging to reproduce.

Rosay *et al.* (2020) studied MLP classifiers to quantify the benefits of a DL-based NIDS. The method details the steps for data preprocessing through formatting, cleaning, sampling, training data split, cross-validation, test data splits and feature scaling. Feature selection and extraction methods are evaluated against the CICIDS 2017 dataset. Random sampling is used to create a training set by selecting 0.5 of each class instance. The same strategy yields a 0.25 cross-validation set and 0.25 test set. The adopted sampling method aims to reduce the benign to malicious class imbalance in the training data while maintaining the original imbalance on the test and validation sets. A Standard Scaler is used to provide better performance results compared to other techniques. The MLP classifier uses two hidden layers and 256 nodes with a Dropout used as a regularisation technique to prevent over adjustment on training data in the MLP. Rosay *et al.* (2020) divides the training set into a mini-batch of 32 instances. Weight tuning between ANN nodes is applied to reduce the cross-entropy loss function. Similar to Reis *et al.* (2019), eight insignificant features are discarded from the training set.

A subsets of 70 and 73 features achieved the best f1-score of ≈ 0.99 with features *IP*

[21]*Bwd PSH Flags, Bwd URG Flags, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk* and *Bwd Avg Bulk Rate*

Address and *Source Port* as the most important features. The authors state that the prominence of the two features must be a result of the MLP model learning of an attacking *IP address* and not necessarily the other features. *IP Address* and *Source Port* are concluded as the most important features in helping to detect intrusion detection but they may not be suitable for real-world implementation as they vary by environment.

The sampling methods used are not able to systematically address the class imbalance. Feature scaling methods and the metrics for evaluation are also limited.

Abdulhammed *et al.* (2019) studied Auto-Encoder and PCA as dimensionality reduction methods for NIDS against the CICIDS 2017 dataset. PCA reduced the original 81 features to 10 features compared to 59 features from Auto-Encoders.

RF classifier achieved the best multi-class result of ≈ 0.99 accuracy and f1-score using 30 principal components from PCA. The results confirmed Abdulhammed *et al.* (2019)'s framework for feature dimensionality reduction as the best, with precision value of 0.99 and an FPR of 0.01. PCA was a superior dimensionality reduction method compared to Auto-Encoder due to its faster training times and interpretability of results.

Ahmim *et al.* (2019) studied a binary classifier, a multi-class classifier using all dataset features and a hierarchical model to correctly classify each attack and provide a low FPR and high recall against the CICIDS 2017 dataset. As part of the dataset preprocessing procedure, the author removes duplicate rows in addition to the same insignificant features as those identified in other studies (Reis *et al.*, 2019, Rosay *et al.*, 2020).

RF and MLP achieve an accuracy of 0.95 and 0.84, and an FPR of 0.02 and 0.07, respectively. Ahmim *et al.* (2019)'s study lacks evidence of classifier development methodology for comparative models for optimal results. Such evidence involves description hyperparameter tuning, classifier choice, scaling method, feature selection or extraction methods.

Aksu *et al.* (2018a) studied the Fisher score algorithm to determine the effect of each feature on classification. The algorithm uses weighted vectors produced by Linear Discriminant Analysis (LDA) for classification evaluated using the CICIDS 2017 dataset and feature importance ranking to determine optimal features for classification algorithms.

Aksu *et al.* (2018a) uses accuracy, precision, recall, and f1-score to measure performance. They found a selection of 30 features producing the best f1-scores for KNN, SVM and DT of 0.99, 0.65 and 0.99, respectively.

The crucial features found in Aksu *et al.* (2018a)'s study resemble the same features raised by Reis *et al.* (2019). Aksu *et al.* (2018a) highlight how feature selection trains optimal classification models in ML. While 30 features produced the best accuracy, they are not listed explicitly. The feature scaling or transformation methods used and hyperparameter tuning activities and motivation for adopting LDA are also not stated.

Bisht and Ahmad (2017) studied DT, bootstrap aggregation (bagging), Adaboost, RF, MultiBoostAB, Rotation Forest, and Random SubSpace modules ensemble methods for NIDS classifiers. The authors use a maximum of ten classifier ensembles and the default parameters of Weka – ML software.

Bisht and Ahmad (2017) use the KDD 99 dataset and two additional KDD 99 variant testing datasets. The one testing dataset is created by removing all redundant records in KDD 99 and using 21 classifiers to divide the testing dataset into five groups based on prediction difficulty. DT achieved the best result of 0.83 f1-score and RF with a 0.80 f1-score, which appears to be lower than the work by Zhang *et al.* (2018). However, there is no discussion on how this method may perform with different feature scaling or extraction methods in the experiment.

Chuan-long *et al.* (2017) studied a DL-based RNN architecture as a binary and multi-class classification NIDS and compared the findings with RF, MLP and DT performance. Chuan-long *et al.* (2017)'s literature review shows that previous studies mainly used DL methods for preprocessing and the classification performed by a supervised classifier. Chuan-long *et al.* (2017) evaluate algorithms against the NSL-KDD dataset – a KDD 99 variant – by preprocessing the dataset through numericalisation and normalisation.

Numericalisation involves working with the three non-numeric features in a dataset. Due to the need for RNN numeric matrix input value, non-numeric features, such as 'protocol type', 'service' and 'flag', need to be converted into numeric form. Features present minimum and maximum values where a significant difference in the logarithmic scaling method is applied to obtain the ranges for the duration, source bytes and destination bytes, which is linearly mapped to a range between 0 and 1.

RNN produced the best accuracy and training time using 80 hidden nodes and a 0.1 learning rate. The best binary class performance of RNN had an accuracy of 0.83 compared to RF, which had the second-best accuracy of 0.81. For multi-class findings, RNN had an accuracy of 0.81 compared to MLP, with 0.78 as the second-best. Chuan-long *et al.* (2017) identifies RNN as a candidate with solid modelling ability for NIDS. It is crucial to note that comprehensive analysis is challenging as other metrics such as the f1-score or precision and recall are not used as measures. Moreover, Chuan-long *et al.* (2017) makes no mention of any model tuning of the other models to aid in a fair comparison.

Stein *et al.* (2005) studied genetic algorithm-based (GA) feature selection, which implements a wrapper model of search and evaluation components. As a measure, Stein *et al.* (2005) uses error rates drawn from a hybrid approach using the KDD 99 dataset. The GA and DT hybrid outperformed the DT without feature selection. The result attributes the hybrid approach focusing on relevant features and eliminating unnecessary or distracting features where the initial filtering can improve the classification abilities of DT.

Su (2011) studied GA in combination with KNN to detect large-scale attacks, such as DoS attacks, in real-time. An accuracy rate of 0.97 for known attacks is achieved, with only the top 19 features considered and for unknown attacks. A 0.78 accuracy is achieved using the top 28 features. These result shows the potential of focusing on feature selection for ML to yield optimised classification models.

Atefi *et al.* (2019) studied KNN and Deep Neural Network (DNN) NIDS using the CICIDS 2017 dataset. Atefi *et al.* (2019) use a fraction of 0.8 for training each iteration, with the remaining 0.2 for testing in a process. KNN and DNN achieved an accuracy of 0.90 and 0.96, respectively. KNN and DNN achieved a recall of 0.91 and 0.96, respectively, and precision of 0.90 and 0.96, respectively.

Doshi *et al.* (2018) studied the use of IoT-specific network behaviours to inform feature selection for high accuracy DDoS detection in IoT network traffic. They use various ML algorithms, where linear SVM performed the worst in detecting DDoS. DT and KNN classifiers attained a 0.99 accuracy. The finding attributed to the possibility of the models' data segmentation into a higher feature space.

Almseidin *et al.* (2017) studied MLP classifier for NIDS against the KDD 99 dataset. They found MLP not to suitable for classifying remote to local and user to root attacks but more acceptable when handling DoS and Probe attacks. The results also showed that no single algorithm could effectively handle all types of attacks. RF and Naive Bayes achieved an accuracy of 0.93 and 0.91, respectively.

2.4.2 Discussion of Related NIDS

Generally, studies showed the process of feature selection as a crucial phase when handling a dataset (Reis *et al.*, 2019, Rosay *et al.*, 2020, Kurniabudi *et al.*, 2020). A number of the studies were consistent in the redundant and non-relevant features dropped in their dataset, especially for the CICIDS 2017 dataset. Studies observed consistency in the application of a standardisation method for training data; however, most studies made no mention of pre-training methods applied for feature processing (Aksu *et al.*, 2018a, Zhang *et al.*, 2018).

A discussion of feature selection methods in a number of the studies and their benefits of them were shown (Stein *et al.*, 2005, Su, 2011, Abdulhammed *et al.*, 2019, Reis *et al.*, 2019, Aksu *et al.*, 2018a, Kurniabudi *et al.*, 2020). However, most of the studies did not entirely refer to the reproducible processes of investigating feature importance, regardless of the importance of this phase. The lack of detail in the feature selection processes has a potential shortfall to the findings, which warrants further research.

As most ML methods have various means of hyperparameter tuning for optimal model performance, most studies made no emphasise handling these activities in training. Most of the studies explored use the KDD 99 or other variations in use, such as NSL-KDD. Datasets such as CICIDS 2017 were not variations of KDD 99 as they also possessed different classes.

Most of the studies explored were not consistent with the metrics used when evaluating the performance of a NIDS. Moreover, whether the findings were a weighted, micro or average macro metric was not provided, especially for the recall and precision scores. Therefore, the related studies lacked the development of a systematic ML-based NIDS.

2.5 Literature Review Summary

This chapter first introduced the background and fundamental concepts of network intrusion detection systems. The signature-based and anomaly-based detection methods were highlighted as the prominent detection methods used for NIDS. The detection methods were discussed with a focus on their strengths and shortfalls.

ML concepts related to NIDS were presented as a derivative of the anomaly-based detection methods. The chapter described datasets that have been adopted for ML in NIDS related studies. Aspects of dataset balancing and sampling methods were discussed as a means to assist in ML-based NIDS development. Feature selection and feature extraction methods are discussed as feature space reduction methods that could be leveraged for better ML model training. Common performance metrics used in ML-based NIDS assessments were then presented. The related ML concepts were closed with an overview of ML classification algorithms identified in literature.

The chapter finally discusses related studies of interest found in the reviewed literature. The overview provided a highlight of related ML-based NIDS found in literature. Discussions of the shortfalls and opportunities from these related studies were presented. The chapter closed with discussions highlighting the key areas that needed to be addressed to develop a systematic ML-based NIDS.

3 Experimental Design

This chapter presents and discusses the approach used to create an adaptive ML-based NIDS that addresses the problem statement described in Chapter 1. The system design choices are explained in the following sections with data – data preparation, feature engineering and ML algorithm implementation processes are outlined.

3.1 Methodology

Training an NIDS model to achieve the set objective from Section 1.3 requires selecting an appropriate processing methodology and establishing relevant datasets. The success of the proposed system implies that the algorithms and techniques chosen must perform well for network traffic used by NIDS.

3.1.1 Overview

The proposed systems' methodology involves much experimentation into several ML classifiers and optimisation methods stemming from literature reviewed in Chapter 2.

Figure 3.1 shows the general approach for creating the proposed NIDS. The proposed system pipeline takes several stages, namely: 1) Data Preparation, 2) Feature Engineering, and 3) Model Tuning and Evaluation. A modelling pipeline evaluation method is adopted to assist with data leakage prevention in the test harness by ensuring that data preparation is constrained to each fold of the cross-validation procedure.

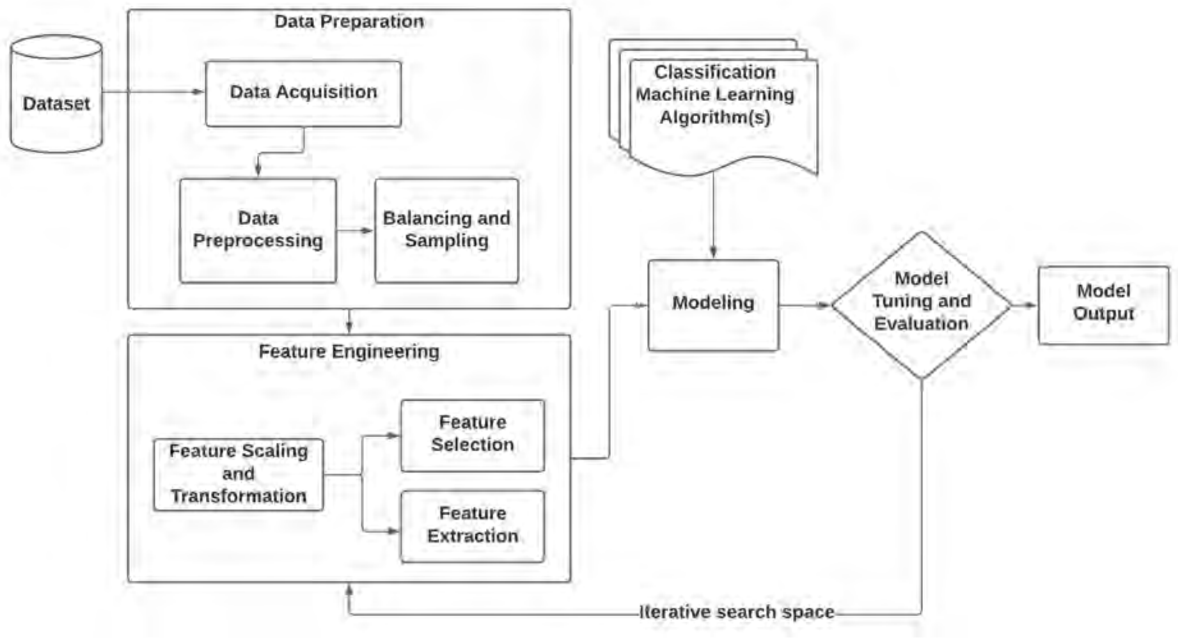

Figure 3.1: Experiment Design Overview.

In the **data preparation stage**, datasets CICIDS 2017 and CICIDS 2018 are presented initially in network packet captured files (PCAPs) format. The dataset authors initially conducted feature extraction using CICFlowmeter[1] to produce comma separated values (CSV) format files. Section 3.1.2 provides an overview of datasets and additional early processing of the dataset.

Data preprocessing includes data merging, formatting, cleansing and missing value treatment. Possible data contamination for the proposed systems is avoided during this phase by splitting training and test data before any additional data preparation activities are completed. To address the imbalanced nature of the dataset, the proposed system rebalances the data through oversampling and undersampling of the training sets.

[1]https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter

The **feature engineering** implementation stage is the first component of an iterative model search that ends in the model tuning and evaluation stage. This stage involves *feature scaling or transformation*, *feature selection* and *feature extraction* methods for *feature space reduction*. The output of the feature engineering stage is combined with a ML classification algorithm to form the model for the modeling phase.

The final stage of the pipeline focuses on **model tuning and evaluation** – extends from the modeling phase. The proposed system completes a first series of experiments to determine the optimal parameters for the base model by testing on a validation set. The second experiment set involves running evaluation tests for the base model against the unseen CICIDS 2018 dataset. The last set of experiments involves retraining the base model with the CICIDS 2018 dataset without additional hyperparameter tuning to gauge the original model tuning shortfalls.

3.1.2 Datasets

To address the research question posed in Section 1.2, the datasets selected for use are required to reflect real-world systems. The CICIDS 2017 and CICIDS 2018 datasets were selected as the ideal datasets to help address the research question.

The CICIDS 2017 dataset was captured by simulating an attack network and victim network (Sharafaldin *et al.*, 2018). The dataset was prelabeled by the authors of the dataset and adopted in this research. The processed PCAPs are statistical features calculated separately for forward and reverse network communication – a bidirectional network. The final dataset comprises a classification label and 83 statistical features, each calculated separately for forward and reverse directions. The dataset is composed of 15 classes, namely: Benign, Bot, DDoS, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS slowloris, FTP-Patator, Heartbleed, Infiltration, PortScan, SSH-Patator, Web Brute Force, Web SQL Injection and Web XSS.

The CICIDS 2018 dataset follows a similar data gathering methodology as CICIDS 2017 but in a different setting. This research adopts context data of the attacks provided by the dataset authors for the defined scenarios in the best effort to label the CICIDS 2018 dataset using the time windows.

The dataset includes different attack scenarios from the CICIDS 2017 dataset plus larger attack and victim networks. Since it contains almost the same feature set, testing this dataset will utilise the 69 common features on the models developed from the CICIDS 2017 dataset. The dataset is composed of 15 classes, namely: Benign, FTP-BruteForce, SSH-Bruteforce, DoS-GoldenEye, DoS-Slowloris, DoS-SlowHTTPTest, DoS-Hulk, DDoS-LOIC-HTTP, DDoS-LOIC-UDP, DDOS-HOIC, Brute Force-Web, Brute Force-XSS, SQL Injection, Infiltration and Bot. For more details on the datasets, interested readers can refer to Appendix A.1.1 and A.1.2.

The preliminary data preprocessing of CICIDS 2018 also involves the use of semantic information from the extracted data using rules from regular expressions to determine different variants of classes, both known and unknown. The following preprocessing steps are completed during relabeling.

- Select classes that are suspected to be similar to those already modelled on CICIDS 2017 in the previous experiments, as shown in Listing 1, undergo regular expression matching in Pandas.
- Class labels are matched to the models' labels and included for subsequent classification of those specific labels.

The caveat of the adopted labeling approach lies in possible inaccurate relabeling for CICIDS 2018. However, the experiment presents the possible inaccuracy as a likely representation of novel attack scenarios through an approach that targets a challenge in real-world scenarios for NIDS.

The regular expression matching patterns used for relabeling of CICIDS 2018 in Listing 1 are based on the dataset authors' tool and attack descriptions with a bias towards realignment with CICIDS 2017.

```
data['Label'] = data['Label'].replace("Brute Force -Web", "Web Brute Force",
                    regex=True)
data['Label'] = data['Label'].replace("Brute Force -XSS", "Web XSS", regex=True)
data['Label'] = data['Label'].replace("DDOS attack-HOIC", "DDoS", regex=True)
data['Label'] = data['Label'].replace("DDOS attack-LOIC-UDP", "DDoS", regex=True)
data['Label'] = data['Label'].replace("DoS attacks-GoldenEye", "DoS GoldenEye",
                    regex=True)
data['Label'] = data['Label'].replace("DoS attacks-Hulk", "DoS Hulk", regex=True)
data['Label'] = data['Label'].replace("DoS attacks-SlowHTTPTest", "DoS Slowhttptest",
                    regex=True)
data['Label'] = data['Label'].replace("DoS attacks-Slowloris", "DoS slowloris",
                    regex=True)
data['Label'] = data['Label'].replace("FTP-BruteForce", "FTP-Patator", regex=True)
data['Label'] = data['Label'].replace("Infilteration", "Infiltration", regex=True)
data['Label'] = data['Label'].replace("SQL Injection", "Web Sql Injection",
                regex=True)
data['Label'] = data['Label'].replace("SSH-Bruteforce", "SSH-Patator", regex=True)
```

Listing 1: Snippet of regular expression matching for CICIDS 2018.

Note, the use of the term ‘Malicious’ may reflect non-benign traffic classes in the datasets regardless of some activities possibly not necessarily being attacker patterns in the real world. Examples may be dataset class instances of Port Scan and Bots.

3.2 Implementation

The following subsections specify the details of the implementation of the proposed system introduced in Section 3.1.

3.2.1 Test Bed

Table 3.1 shows all the hardware and software tools and components required to complete the experiments, develop and test the proposed systems’.

Component	**Description**
Server:	AMD Ryzen 9 3950X 3.5 GHz
Memory:	128 GB 3200MHz DDR4 RAM
Operating System:	Ubuntu 20.04 LTS
Programming Language:	Python 3[2]
Core Libraries:	Scikit, Keras, Numpy, Panda

Table 3.1: Test Bed Architecture.

The complete version of the programming code of this research is available on GitHub repo[3].

3.2.2 Data Preparation

A detailed implementation of the proposed system begins with data preparation which includes data acquisition, data preprocessing, the training set sampling and balancing processes.

Data Acquisition

Initially datasets undergo data analysis through preliminary dataset assessments and exploration to assist in describing dataset characterizations to better understand the nature of the data. As a result, numeric features from the dataset were loaded as floats and integers, leading to an observation that downcasting all numeric data to integers results in no measurable change in classification performance when using the same global random seed of 42.

Listing 2 shows the fast loading method used in the proposed system. The method achieves approximately a 5-fold improvement in speed across the different datasets by loading all data as integers except the 'Label' column. The procedure was necessary to address time constraints and memory limitations during experimentation.

[3]https://github.com/hatityechindove/ml-based-nids-research

```
def fast_load_data(data_path, non_numeric=[`Label'], dtype=`object'):
    columns_to_skip = non_numeric
    df = pd.read_csv(data_path, engine='c', dtype=dtype,
        usecols=lambda x: x not in columns_to_skip)
        df2 = pd.read_csv(data_path, engine='c', dtype='object',
        usecols=lambda x: x in columns_to_skip)

    con = pd.concat([df, df2], axis=1)
    con = con[-con['Label'].isin(ex_class)]
    if ex_class != None
    else
        print()
    return con
```

Listing 2: Dataset File Merging (2017 dataset).

Data Preprocessing

The first step involves merging data into a single dataset by leveraging fast-loading for all the CSV files on data acquisition – highlighted in previous Section 3.2.2. Each file is loaded into data frames[4] and merged into a single data frame. The first field is typically a header containing the feature names per column. After combining the files to create a single dataset, duplicate header fields are removed, resulting in a dataset containing 2,660,377 samples and 15 classes from CICIDS 2017.

Trailing and leading whitespaces are removed as part of data formatting to avoid ambiguities by Python during string manipulation or interpretation. The proposed system prunes duplicate records similar to that of Reis *et al.* (2019) and Rosay *et al.* (2020) – who argued that the data loss is necessary for accurate real-world application, discussed in Section 2.4. As a result, there were 69 functional features used during the ML training process. Listing 3 shows the resulting 69 features of the original 83 statistical features presented by the dataset authors.

Some instances contained (typically one) features with a *NaN* value, while the rest of the features were unaffected. Instead of discarding those instances, the feature value

[4]A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

```
"Dst Port", "Flow Duration", "Tot Fwd Pkts", "Tot Bwd Pkts",
"TotLen Fwd Pkts", "TotLen Bwd Pkts", "Fwd Pkt Len Max",
"Fwd Pkt Len Min", "Fwd Pkt Len Mean", "Fwd Pkt Len Std",
"Bwd Pkt Len Max", "Bwd Pkt Len Min", "Bwd Pkt Len Mean",
"Bwd Pkt Len Std", "Flow Byts/s", "Flow Pkts/s", "Flow IAT Mean",
"Flow IAT Std", "Flow IAT Max", "Flow IAT Min", "Fwd IAT Tot",
"Fwd IAT Mean", "Fwd IAT Std", "Fwd IAT Max", "Fwd IAT Min",
"Bwd IAT Tot", "Bwd IAT Mean", "Bwd IAT Std", "Bwd IAT Max",
"Bwd IAT Min", "Fwd PSH Flags", "Fwd URG Flags",
"Fwd Header Len", "Bwd Header Len", "Fwd Pkts/s",
"Bwd Pkts/s", "Pkt Len Min", "Pkt Len Max", "Pkt Len Mean",
"Pkt Len Std", "Pkt Len Var", "FIN Flag Cnt", "SYN Flag Cnt",
"RST Flag Cnt", "PSH Flag Cnt", "ACK Flag Cnt", "URG Flag Cnt",
"CWE Flag Count", "ECE Flag Cnt", "Down/Up Ratio", "Pkt Size Avg",
"Fwd Seg Size Avg", "Bwd Seg Size Avg", "Subflow Fwd Pkts",
"Subflow Fwd Byts", "Subflow Bwd Pkts", "Subflow Bwd Byts",
"Init Fwd Win Byts", "Init Bwd Win Byts", "Fwd Act Data Pkts",
"Fwd Seg Size Min", "Active Mean", "Active Std", "Active Max",
"Active Min", "Idle Mean", "Idle Std", "Idle Max", "Idle Min"
```

Listing 3: 69 Features adopted for Modelling.

was encoded (replaced) with a large integer since it may be malicious. Formatting also involved class encoding to integer outputs from string outputs. The exercise reduced the number of instances by 1,538, resulting in 2,658,839 instances of data applicable for training.

Sampling and Balancing

Stratified sampling was applied to the dataset to separate training and test data. Post-strata creation, the sample is proportionally selected randomly with a seed/state of 42 for reproduction purposes. The proposed system evaluates the effects of training data size on the model's performance by splitting training/test data splits of 10/90, 30/70 and 50/50 percentage. The validation set was created by splitting the training set to 70/30, where the validation set is 30% of the training data. Sampling and testing leveraged cross-validation during model development.

The CICIDS 2017 dataset has significant class imbalance – later visually shown in Figure 3.2. Benign data is the most represented class, as expected since this dataset emulates a real-world representation.

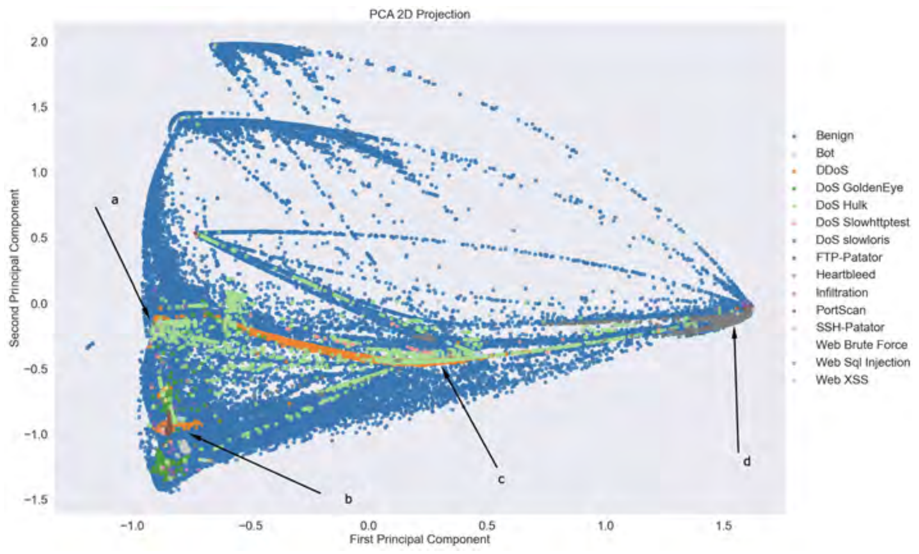

Figure 3.2: Original dataset distribution.

The experiment defines *Majority* classes as those with more than 5000 samples and classes with fewer instances as *Minority* classes. Visually, minority classes such as Infiltration and Web SQL Injection are barely noticeable with their overlap with the Benign class.

SMOTE is applied systematically by performing a grid search in steps of 10 synthetic samples to minority classes for validation. The best result of the Grid Search was 200 synthetic samples during preliminary tests. Adding synthetic samples to minority classes yielded insignificant changes to classification performance on the validation set and was thus avoided. Random Undersampling follows after the SMOTE process, which removes excess samples from majority classes based on the earlier specified global random seed if they contain more than 50000 samples. As a point of emphasis, SMOTE was applied after the split to avoid data contamination for a fair comparison. Figure 3.3 shows the visual effects of rebalancing the original dataset.

An extreme example of oversampling with SMOTE is the Heartbleed class, which contains only one training sample. A preliminary test showed that contaminating the test data and adding synthetic instances to the original dataset made it possible to achieve results up to 0.94 macro f1-score. However, this is excluded in the proposed implementation. Data contamination is a common mistake in ML training that should be avoided (Bowyer *et al.*, 2011).

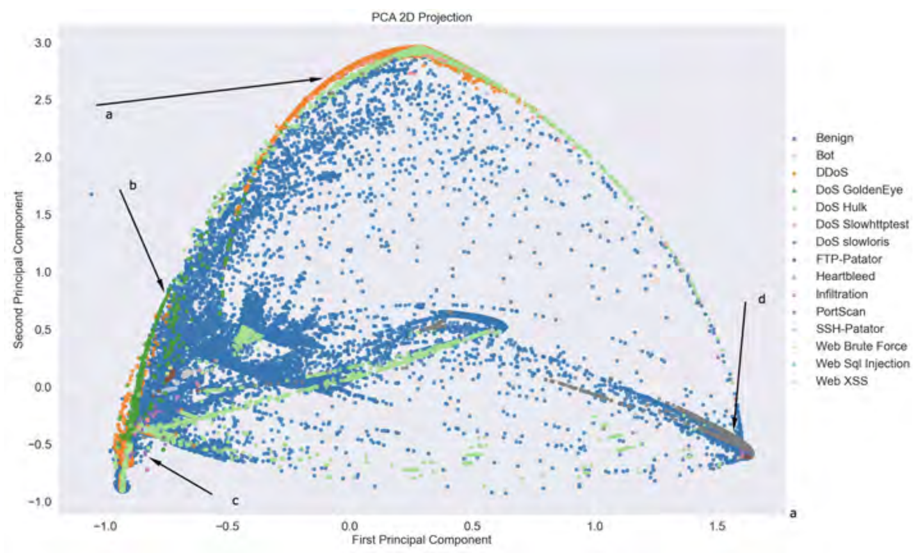

Figure 3.3: After Applying SMOTE and Undersampling.

The experiment used a minimum distributable train/test split ratio – 10/90 – to test model performance against limited data. The assumption was if the model performs well with limited data it is likely to perform similarly on unseen data. Additionally, the experiment uses a macro f1-score biased towards minority classes to influence the model towards achieving better individual categorisation of classes versus binary classification. Furthermore, if the experiment attained a good f1-score with a minimum distributable train/test split ratio, it would reflect that the features selected are the most effective and thus easily more applicable to real-world scenarios.

However, favouring a model that produces high macro f1-score on limited training data, may not necessarily scale well with more training data. The experiment factors in that the use of macro f1-score can lead to potential minority classes bias resulting in the loss of accuracy for the model (Opitz and Burst, 2019). Similar considerations are made with SMOTE as it may also reduce the overall f1-score while improving the minority classes detection (Ramezankhani *et al.*, 2014).

3.2.3 Feature Engineering

The following subsection specifies feature scaling and transformation for feature engineering applied to the collected dataset.

Feature Scaling and Transformation

The Standard, Power, Quantile and Robust[5] scaling techniques – described in Section 2.3.3 – are evaluated for optimal distribution of functional statistical features in this step of the experiment. The primary scaling method used Quantile scaling as it yielded the best results across most classification models.

Note, feature scaling and transformation was applied to distance-based classifiers and not focused on for tree-based classifiers due to these methods not being affected by the data distribution. The effect of each scaling method, when applied to the resampled dataset, is represented in Appendix A.2.1.

3.2.4 Feature Space Reduction

Feature space reduction – feature selection and feature extraction – activities were executed (separately) in parallel. There was a complete evaluation of the difference in outcome between the two systems on the validation set. Of note, isolation forests[6] were fit on the training data but resulted in an average of 8.83% (+/- 3.07%) macro f1 score reduction.

Feature Selection Methods

The evaluation of feature importance is a key part of the methodology. As part of the experiment, two feature importance methods undergo evaluation, namely, Gini Importance (GI) and Permutation Importance (PI). These methods were applied to the base

[5]Robust was not considered further because it was consistently performing poorly.
[6]Isolation forest is an anomaly detection algorithm

models[7] for all scalers to determine which approach to take. Using the *Python* time module, calculating the PI function took 302.19 seconds, whilst the GI function took 2.12 seconds. The research adopts PI as detailed in Section 2.3.4. Feature selection ranks the best features in each model by permuting over 34 (features / 2) iterations and explicitly selecting features that produce the best f1-score for a particular base model.

Figure 3.4 shows the preliminary distribution of the RF model's 26 most important features before feature rescaling.

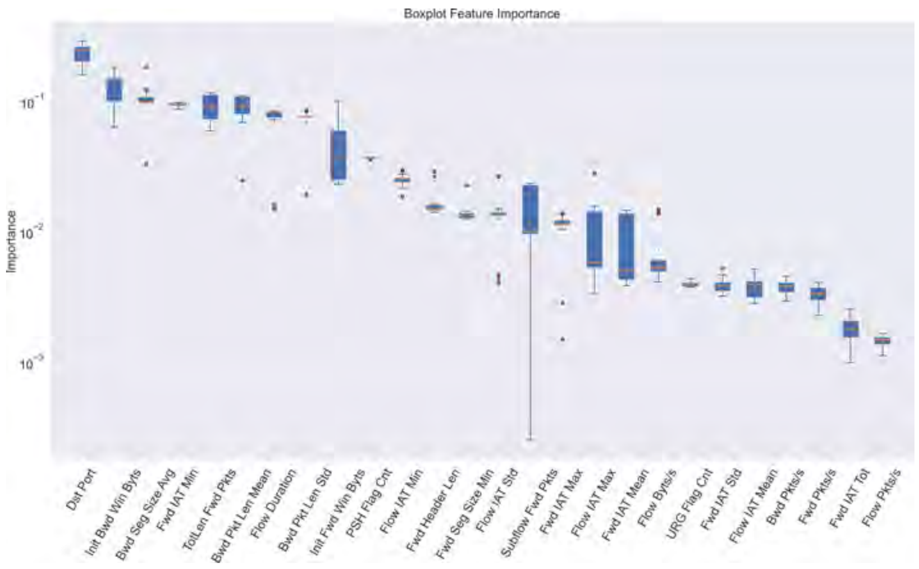

Figure 3.4: RF: Boxplot of top 26 permutation importances per feature.

The distribution shows that each feature ranking has differing quantile ranges and extreme lower values in some cases. The shorter the range of the box, the higher the compatibility with other combinations of features in terms of achieving optimal classification performance (Altmann *et al.*, 2010). Selected features undergo rescaling using a Standard scaler to achieve zero mean and a unit of variance.

Feature Extraction Methods

As an alternative to feature selection methods, the dataset undergoes a dimensionality reduction using the PCA feature extraction method. Other reduction methods[8] were tested

[7]Classifiers use default parameters of *Scikit*, as established by best practices in the literature.
[8]LDA, LLE, TSNE and UMAP

on the validation set but not presented in the findings due to practical limitations, including the tendency to overfit, deficient classification performance, and high computational requirements. To identify the number of required components for PCA, Figure 3.5 shows that the first 26 principal components explain 0.99 of the variance.

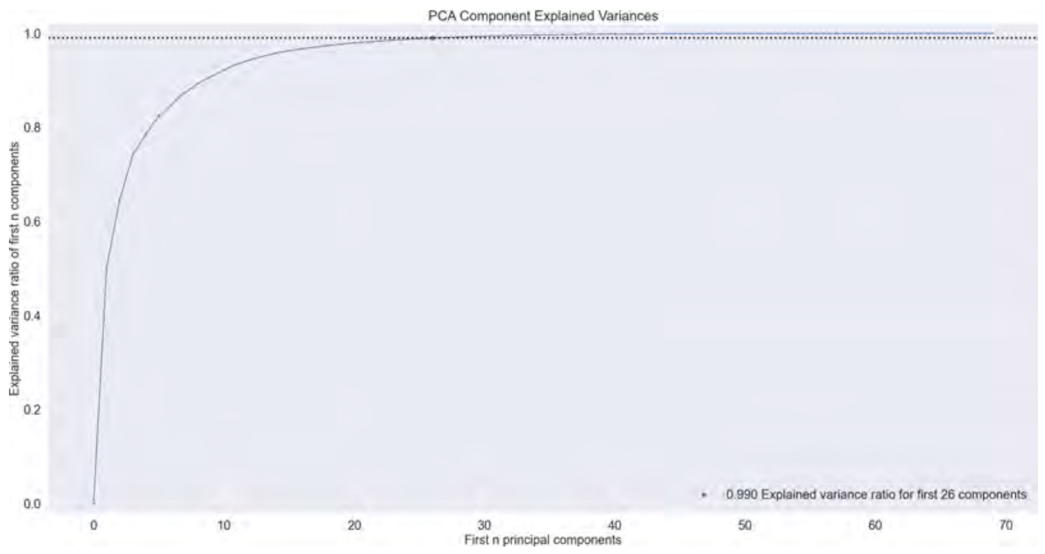

Figure 3.5: Experiment Variance.

Note, the evaluation of PI and PCA methods uses hyperparameter tuning on each scaler-classifier combination.

3.2.5 Model Evaluation

In addition to each classifier's diverse hyperparameter tuning parameters, they are all exhaustively searched using Grid Search[9]. The phase involves conducting a series of training experiments for each tuned ML classification algorithm against select feature engineering and feature space reduction methods. The study uses the findings for comparison purposes of each output ML model.

For the tables presented in Chapter 4, *Processing* is used to refer to feature space reduction activities presented in Section 3.2.3. For example, **26PCA** represents a feature extraction

[9]First Random Search is used to reduce search space, followed by Grid Search, which iterates exhaustively through the best subset of the hyperparameters of a target algorithm.

method where it is read as 26 components used with PCA, whilst **26PI** represents the first 26 most important features using PI. Moreover, **Q-26PI-MLP** represents a complete model application, where the first character represents the Scaler or Transform method – Quantile as Q – followed by the feature processing method – Permutation Importance (PI) – and finally, the ML classification algorithm – Multilayer Perceptrons (MLP).

Performance Metrics

Multiple metrics are used to gauge the performance of a given model. The proposed system used macro f1-score, micro precision, micro f1-score and ROC curves.

A macro f1-score is used in multi-class experiment to reflect on average model performance. A macro-f1 is the harmonic mean between precision and recall, where the average is calculated per label and then averaged across all labels. If pj and rj are the precision and recall for all $\mathrm{\lambda_j} \in \mathrm{h\,(xi)}$ from $\lambda j \in y_i$, the macro f1-score is shown in Equation 3.1:

$$\text{macro f1-score} = \frac{1}{Q}\sum_{j=1}^{Q}\frac{2 \times pj \times rj}{pj + rj} \tag{3.1}$$

The macro f1-score provides equal importance to all classes, regardless of their sample size. A macro-average will compute the metric independently for each class and then take the average (treating all classes equally), regardless of underrepresentation.

In contrast, a micro-average will aggregate the contributions of all classes to compute the average metric. The research thus places emphasis and preference on the macro score versus the weighted score.

Equations 3.2, 3.3 and 3.4 represent the calculation for the micro-average metrics, where ‘c’ is the class label.

$$micro\ precision = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c} \tag{3.2}$$

$$micro\ recall = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FN_c} \tag{3.3}$$

$$\text{micro f1-score} = 2 \times \frac{\text{micro precision} \times \text{micro recall}}{\text{micro precision } + \text{micro recall}} \tag{3.4}$$

ROC curves do not depend on the class distribution, thus making it helpful in evaluating classifiers predicting rare events such as underrepresented classes. The area under the curve is a measure that represents the degree or measure of separability of classes (Benign/Malicious). ROC curves evaluate model performance on unseen data and AUC values to gauge unbiased performance.

Due to the general high performance of the models, the ROC curves are rescaled to ease the interpretation of the findings. Hosmer Jr *et al.* (2013)'s general binary rules for AUC are adapted for a multi-class problem area below:

- Values equal to 0.5 suggests that the model is no better than predicting an outcome than random chance – represented as **no discrimination**.
- Values 0.5 < x < 0.7 means slightly better than random chance – represented as **poor**.
- Values 0.7 <= x < 0.8 indicate an acceptable discrimination – represented as **acceptable**.
- Values 0.8 <= x < 0.9 indicate an excellent model – represented as **exceptional**.
- Values >= 0.9 indicates an outstanding model – represented as **outstanding**.

3.2.6 Experiments

To address the objectives set in Section 1.3, the following three experiments are completed:

- Experiment 1 - Compares the performance of models trained against CICIDS 2017.
- Experiment 2 - Compares the performance of the CICIDS 2017 models tested against the unseen CICIDS 2018 data.
- Experiment 3 - Compares the performance of the CICIDS 2017 models retrained against CICIDS 2018.

The results of these experiments are presented in Sections 4.3.1, 4.3.2 and 4.3.3, respectively.

Experiment 1: Training models with CICIDS 2017 dataset

The One versus All (OvA) strategy is used as a training strategy for all classifiers. The strategy involves training a single classifier per class, with the class samples as positive samples and all other samples as negatives.

For the untrained data, the CICIDS 2017 has three train/test data splits of increasing size, mentioned in Section 3.2.2. These are evaluated separately following the implementation in Section 3.2.5 to determine the effect of having limited training data, as opposed to up to half of the total samples in the dataset.

Experiment 2: CICIDS 2017 Model testing with CICIDS 2018 dataset

Using the semantic information from features extracted during data acquisition in Section 3.2.2, the same models developed for CICIDS 2017 are used on CICIDS 2018 to evaluate the model effectiveness against untrained data. Reusing the CICIDS 2017 models on a new dataset, the CICIDS 2018 dataset is processed in the same way as the above, but with the following difference to allow for automated reuse of a model on different data during classification.

Additionally, this approach seeks to prove the concept of detecting novel scenarios by adopting class variants into a parent class, broadly classifying DDoS from variants DDoS

LOIC and HOIC. In cases where detection is successful, it is different from signature-based detection, which will require new patterns developed. An interested reader may refer to the dataset authors[10] to describe the missing labels.

Experiment 3: Training and Testing with CICIDS 2018

The final experiment evaluates the classification performance of CICIDS 2018 by following the same hyperparameter tuning and training methodology as the first experiment. For remodelling, the study uses the grouping of classes during the regular expression matching detailed in Section 3.1.2.

3.3 Experiment Design Summary

This chapter first provides a high-level view of the methodology used for the proposed NIDS. This included the datasets, implemented algorithms and evaluation methods used.

The second section involved a detailed description of the implementation of the proposed system including the testbed, data preparation, feature engineering, feature space reduction methods and finally model evaluations. The focus was placed on how data preparations were completed for the two datasets used in this experiment.

The chapter moves on to introduce and discuss metrics used for model evaluation. The chapter closes with a description of the experiments that were conducted for model evaluation. The experiments were set to directly address the objectives set for this research.

[10]https://www.unb.ca/cic/datasets/ids-2018.html

4

Results

This chapter analyses and discusses findings from the experiments on NIDS development conducted in Chapter 3. Results of feature engineering, model tuning and evaluation are presented against the CICIDS 2017 and further evaluated for presentation against the CICIDS 2018 datasets, respectively. The core model evaluation in this chapter uses the metrics presented in Sections 2.3.5 and 3.2.5.

4.1 Feature Engineering Results

Table 4.1 shows an extract of the top five most important features per classifier using PI. *Dst Port* – destination port – was the most consistent contributing feature for the evaluated models. Given that the feature represents a common entry point onto a network, it is sensible that the feature holds significance. These findings of feature ranking also concur with that of the information gain method by Kurniabudi *et al.* (2020), where the destination port is also one of the top-ranked features.

Several features including *Init Bwd Win Byts*, *Fwd IAT Min*, *Bwd Pkt Len Std* and *Fwd Pkt Len Max* were preferred by RF, DT, KNN and MLP classifiers. These were

Classifier	**Top Five Features**
RF	Dst Port, Init Bwd Byts Bwd Seg Size Avg, Fwd IAT Min Tot Len Pkts
SVM	Bwd Pkt Len Std, Dst Port PSH Flag Cnt, Fwd IAT Std Fwd Pkt Len Max
MLP	Dst Port, Fwd Act Data Pkts Fwd Pkt Len Max, PSH Flag Cnt Init Fwd Win Byts
KNN	Dst Port Flow IAT Min, Fwd IAT Min Bwd Header Len, Init Bwd Win Byts
DT	Fwd Pkt Len Max, Bwd Pkt Len Std TotLen Bwd Pkts, Init Bwd Win Byts Dst Port
LR	PSH Flag Cnt, Dst Port Bwd Pkt Len Min, Fwd Pkt Len Max Fwd Pkt Len Std
RNN	*N/A*

Table 4.1: Top Five Most Influential Features.

also observed by Kurniabudi *et al.* (2020), Reis *et al.* (2019) and Aksu *et al.* (2018a) as preferred features during feature selection.

Dimensionality reduction using PCA produced better results compared to other[1] extraction methods, especially for the LR classifier. The performance of PCA against scaling methods like Quantile reflect that the approach may be transforming the data into a more uniform distribution[2].

In general, feature selection had varying effects per ML model with better performance than PCA. Readers may refer to Appendix A.3.1 for further analysis of feature selection findings.

[1]LDA, LLE, TSNE and UMAP

[2]PCAs dominant components resulted in a uniform distribution.

4.2 Model Evaluation

This section details the findings from the experiments of the complete model evaluation from Section 3.2.5.

4.2.1 Model Tuning

This subsection presents hyperparameters that were effective for each ML classifier. Hyperparameters bias/variance trade-off balance by selecting the value where training and cross-validation scores peak for part of this section's basis. Provision of best results is on a case by case basis due to the many combinations drawn from hyperparameter tuning.

Table 4.2 shows classifiers with their respective most influential parameter. Note, the parameter tuning experiments were repeated the standard ten times and the best parameters were taken based on the result. The following paragraphs provide details for each classifier.

RF accuracy is not improved significantly with changes to parameters except when maximum depth and/or feature count are changed. In as much as the effect observed was small, both Kurniabudi *et al.* (2020) and Ahmim *et al.* (2019) make no suggestion of hyperparameter tuning value when evaluating RF models.

SVM results show that a low gamma of seven improves and stabilises the f1-score across validation folds. Generally, a low-value gamma reflects that the model is too constrained and cannot capture the complexity of the data from the linearity of the training data (Cherkassky and Ma, 2004).

Cross-validation macro f1-score is not improved by a gamma value above six. Of note, the gamma value is multiplied by (1 - training split)/2) to allow the SVM algorithm to train in a reasonable amount of time when given increasing amounts of data. The lower gamma value reduces the complexity of the Gaussian kernel function (Ring and Eskofier, 2016).

Classifier	**Parameter(s)**	**Accuracy**

RF	n-estimators = 300
	max-features = sqrt
	max-depth = 25
		≈ 1

SVM	kernel = rbf
	gamma = 7
	C = 100
		≈ 1

MLP	solver = adam
	learning-rate = constant
	hidden-layer-sizes = (200, 200, 200)
	alpha = 0.0001
	activation = relu
		0.887

KNN	weights = distance
	n-neighbors = 1
	metric = manhattan
		≈ 1

DT	min-samples-split = 3
	min-samples-leaf = 3
	max-features = sqrt
	max-depth = 25
	criterion = gini
		0.93

LR	C = 800
	min-samples-leaf = 3
	solver = lbfgs
		0.887

RNN	epochs = 500
	batch-size = 64
	dropout-rate = 0.2
	learn-rate = 0.001
	neurons = 128
		≈ 1

Table 4.2: Most Influential Hyperparameters using 26PI.

The best-tuned parameters included using a radial basis function (RBF) kernel, a class weight for the imbalanced data, and a ‘C’ parameter to trade-off correct classification of training examples against maximisation of the decision function’s margin. The ‘C’ value of 100 was optimal, and higher values led to overfitting.

As was the case with RF, related SVM based NIDS did not describe their parameter tuning process. It is thus challenging to compare aspects of the studies that led to good model development.

MLP results show that the optimal tuned parameters include: using the ADAM solver, a constant learning rate and adopting the 'ReLu' activation function. This result is similar to SVM. The selection of the activation function is identical to that by Muhuri *et al.* (2020) and indicates that this may be appropriate for this kind of data.

KNN results show that the model performs best with three neighbours on the validation set. The Manhattan distance worked best for the multiclass. However, preliminary tests showed that Euclidean distance works well on binary classification problems.

DT shows the optimal performing parameters for the model included: 'Gini' impurity function, square root for the best split to maximise on feature search, minimum sample split and leafs of three. There is no evidence of a better yield from using 'Entropy' as an impurity function from preliminary tests.

LR yields the best result when using a 'C' value – Inverse of regularisation strength – of 800. Values lower or higher than this result in decline in the f1-score. Additional parameters used to achieve the best results included: using l2 for penalty normalising, LBFGS optimisation algorithm and no class weighting.

RNN tuning involved adjusting the batch size for training and the updates on the number of epochs for how many times the weights for the neural network were updated during training. Higher epochs generally yielded better results when there was a minimum 500 epochs. The model uses all 69 features with tuned parameters, including dropout rate varied between 0.1 and 0.5, the learning rate between 0.0001 and 0.001, and batch sizes between 32 and 512. Best findings had epoch between 100 and 1500, where 1000 was optimal before overfitting. Given all parameters are influential in this model, a validation curve for hyperparameters was not used.

In general, a focus on hyperparameter tuning was able to show that this step is essential for better assurance to optimised ML NIDS. Interestingly, no emphasise is placed on hyperparameter tuning step from Section 2.4 in the literature review. Appendix A.4.1 shows other results related to hyperparameter tuning.

4.3 Experiment Results

The following subsections detail the results of the three experiments presented in Section 3.2.6.

4.3.1 Experiment 1: Training models with CICIDS 2017 dataset

Findings for each model in this subsection are developed and validated against the CICIDS 2017 dataset. The rest of the section details findings of limited data for train/test splits and the effects of varying train/test data splits on model outputs.

Minimum Distributable/Limited Data

In this section, the best performing models that use the minimum distributable – also referred to as limited data – train/test split ratio – 0.1/0.9 – are analysed to prove effectiveness against training data dependence. Note that the training subset was further split and used as a validation set. For multi-class classification, Hosmer Jr *et al.* (2013)'s class discrimination rules, introduced in Section 3.2.5, are used to summarise performance. Multiclass classification models are trained for all the classes[3] in the CICIDS 2017 dataset, as presented in Section 2.3.2.

Table 4.3 shows the limited data best results for the NIDS models where all models achieve a macro f1 score above 0.70. Model 26PI-RF achieves the best macro f1-score of 0.87 with a precision as high as 0.90.

The rest of the section details findings per model. Models are analysed using a table of results and/or ROC curves. For model contrasts, the FPR and TPR per class due to the high accuracy of the models, are zoomed for ROC curve analysis. There is a provision of accuracy and other metrics for comparison with other studies.

[3]Benign, Bot, DDoS, DoS GoldenEye, DoS Hulk, DoS Slowhttptest, DoS Slowloris, FTP-Patator, Heartbleed, Infiltration, PortScan, SSH-Patator, Web Brute Force, Web SQL Injection and Web XSS

Model	**Accuracy**	**Precision**	**Recall**	**f1-score** (macro)
26PI-RF	1	0.90	0.86	0.87
Q-26PI-KNN	1	0.80	0.86	0.82
Q-36PI-MLP	1	0.83	0.85	0.81
Q-26PI-SVM	1	0.82	0.84	0.81
16PI-DT	1	0.78	0.83	0.76
Q-26PCA-LR	0.99	0.69	0.78	0.73
Q-69PI-RNN	1	0.74	0.82	0.73

Table 4.3: Best results minimum distributable train/test split ratio models.

Due to evident class imbalance highlighted in Section 3.2.2, the f1-score is macro-averaged except when compared to findings in literature. The wiki section of the code repository[4] shows references to additional visual results for the models. For the reader interested in more details, refer to Appendix A.5 through A.7.

RF: 26PI-RF yields the best results with limited data with a perfect (rounded) accuracy, outperforming the 0.97 accuracies achieved by Kurniabudi *et al.* (2020), which was the best observed in the literature. Generally, for feature space reduction, RF prefers feature selection over feature extraction methods. It is not clear why this is the case, but this may serve as a potential future study outside the scope of this study. The model performs exceptionally well for the majority classes and to a fair extent in minority class cases. It follows that RF is a strong detection model.

Figure 4.1 illustrates the ROC curve for 26PI-RF which reflects findings at a micro averaged accuracy. The near perfect AUCs for the model indicates outstanding discrimination for all classes except for *Web Sql Injection*.

In the latter instance, the curve is not completely visible due to the FPR tending to 1; hence the AUC value is explicitly given in parentheses. RF tends to perform better with significant class representation in training sets; however, it is interesting to note that minority classes *Heartbleed* and *Web Brute Force* also had good performance. It is not clear why minority classes performed well. RF is the strongest detection model option

[4]https://github.com/hatityechindove/ml-based-nids-research/wiki

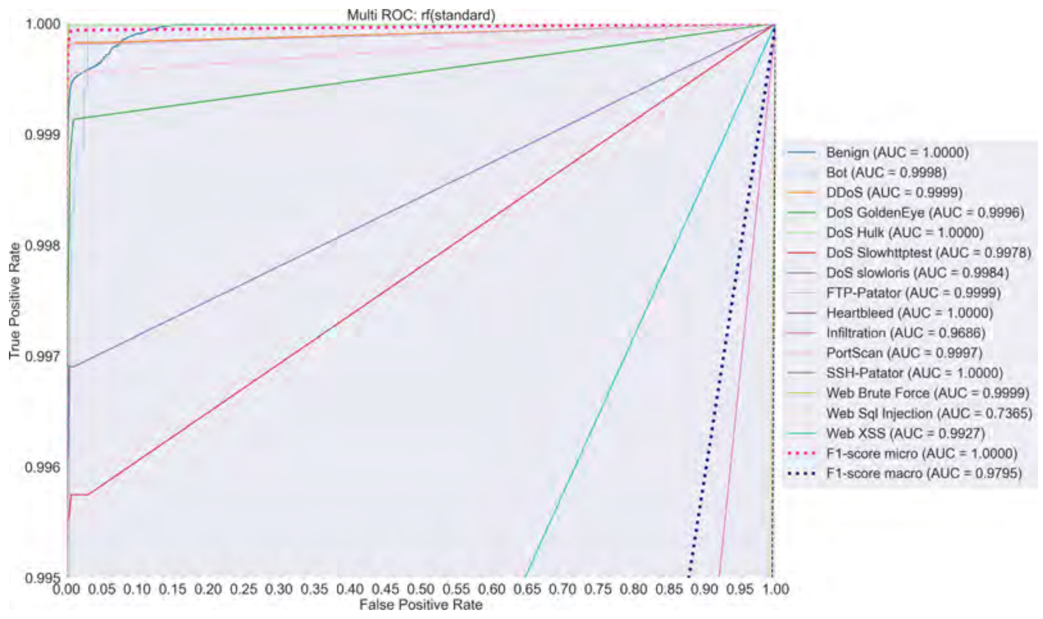

Figure 4.1: 26PI-RF: ROC curve of the best performing RF model on limited data.

for ML methods not using Deep Learning. Appendix A.7.1 shows additional information related to the RF.

KNN: Table 4.4 shows that Q-26PI-KNN achieves the highest macro f1-score using limited data. Both models achieve a perfect (rounded-off) accuracy like the best RF model.

Processing	**Accuracy**	**Precision**	**Recall**	**f1-score**
Q-26PI-KNN	1	0.80	0.86	0.82
S-26PI-KNN	0.99	0.74	0.87	0.78
P-26PI-KNN	1	0.75	0.85	0.79
P-26PCA-KNN	1	0.71	0.82	0.74
Q-26PCA-KNN	1	0.75	0.81	0.77
S-26PCA-KNN	0.99	0.7	0.79	0.74

Table 4.4: Best CICIDS 2017 limited data results for KNN model.

During validation testing, the Quantile transformation set with a uniform output consistently yielded better results across all feature selection and extraction methods, whilst the Standard scaler produced the lowest. Given that the macro averages of the model perform well, it goes to show that the model is appropriate for use in binary classification and weaker for categorisation of classes. In general, KNN models perform lower across all classes when compared to the RF model analysed earlier. There were no other findings of interest. Appendix A.7.2 presents additional analysis related to the KNN.

MLP: Table 4.5 shows that Q-36PI-MLP achieves the highest macro f1-score of 0.81. Similar to KNN, Quantile is able to yield better results for both feature space reduction methods, whilst the Standard Scaler achieves the lowest f1-score results. Chuan-long *et al.* (2017)'s study achieved a recall of 0.78, which is lower compared to those achieved by all evaluated MLP models.

Processing	**Accuracy**	**Precision**	**Recall**	**f1-score**
P-36PI-MLP	1.00	0.80	0.84	0.79
Q-36PI-MLP	1.00	0.83	0.85	0.81
S-36PI-MLP	0.98	0.70	0.81	0.73
P-26PCA-MLP	1.00	0.76	0.79	0.75
Q-26PCA-MLP	1.00	0.77	0.82	0.78
S-26PCA-MLP	0.99	0.66	0.78	0.68

Table 4.5: Best CICIDS 2017 limited data results for MLP model.

The model prefers more features than the other applied ML models evaluated for optimal results. An in-depth review show that MLP performs worst against *Infiltration* and all Web Attacks. This finding may reflect that ANN's may generally perform better with a wider feature space, as observed for RNN findings. Appendix A.7.3 highlights additional information related to the performance analysis.

SVM: Table 4.6 shows that Q-26PI-SVM was the best SVM model with a 0.81 macro f1-score. Interestingly, SVM findings exhibit trends different to RF's, in that SVM's recall is significantly better in many cases, and conversely, precision is worse. Furthermore, differences in f1-scores are much lower across other feature selection combinations than RF. This difference indicates that the SVM is relatively sensitive to features included in the model.

Of note, all minority classes had low accuracy except for *Heartbleed*. The model may not serve as a good option for categorising but is better suited for binary classification. For the interested reader, refer to Appendix A.7.4 for additional analysis.

DT: Table 4.7 shows that 16PI-DT is the best performing model with accuracy, precision,

Processing	**Accuracy**	**Precision**	**Recall**	**f1-score**
Q-26PI-SVM	1.00	0.82	0.84	0.81
S-26PI-SVM	0.98	0.74	0.77	0.71
P-26PI-SVM	1.00	0.84	0.80	0.79
P-26PCA-SVM	1.00	0.84	0.78	0.80
Q-26PCA-SVM	1.00	0.84	0.80	0.80
S-26PCA-SVM	0.98	0.69	0.75	0.69

Table 4.6: Best CICIDS 2017 limited data results for SVM model.

recall and f1-score of 1, 0.78, 0.83 and 0.76, respectively. P-16PI-DT attains a similar f1-score but has a lower precision and recall. In this case, results show that Quantile performs best when used with feature space reduction.

Processing	**Accuracy**	**Precision**	**Recall**	**f1-score**
Q-16PI-DT	1	0.72	0.82	0.75
16PI-DT	1	0.78	0.83	0.76
P-16PI-DT	1	0.74	0.82	0.76
P-26PCA-DT	0.99	0.66	0.73	0.68
Q-26PCA-DT	0.99	0.72	0.77	0.74
S-26PCA-DT	0.99	0.70	0.73	0.69

Table 4.7: DT limited data best results.

All feature selection models achieved perfect (rounded) accuracy for DT regardless of the data scaling method. DT findings exhibit trends where its recall is significantly better in most cases than RF (an ensemble method). Moreover, results show that ensemble methods prefer using no scaler or the Standard Scaler over other scaling methods. Of note, DT preferred fewer features in both validation and these experiments compared to the other models evaluated. This finding is advantageous in cases where a different extraction method may be used compared to the CICFlowMeter used in this study. There is a smaller feature space selection.

Further testing showed that adding more training samples (70% in total) drastically improved the model to be on par with other classifiers like RF. However, it was impractical to compare, as there are very few test data samples remaining when performing testing this way. More analysis information can be review in Appendix A.7.5.

LR: Table 4.8 shows that Q-26PCA-LR is the best performing LR model compared to the rest of the classifier's implementations with limited data. Different from the other models, the PCA feature space reduction methods performed best. The micro f1-score performed marginally lower than other best models that achieved a perfect score. This model has the advantage of having the fastest training times compared to other models evaluated.

Processing	**Accuracy**	**Precision**	**Recall**	**f1-score**
S-36PI-LR	0.96	0.56	0.64	0.59
P-36PI-LR	0.99	0.67	0.79	0.71
Q-36PI-LR	0.99	0.68	0.78	0.70
P-26PCA-LR	0.99	0.64	0.77	0.68
Q-26PCA-LR	0.99	0.69	0.78	0.73
S-26PCA-LR	0.95	0.56	0.64	0.58

Table 4.8: LR limited data best results.

There were no other results of significance observed for the model. For additional information, interested readers can refer to Appendix A.7.6.

RNN: Different from the rest of the evaluated models, RNN was presented with all the features and did not undergo the same process of feature selection and extraction. Due to several factors such as the training methods, loop structure of the neural network, layers, dropout layers presented in Section 2.3.6, more focus was placed on model tuning.

Q-69PI-RNN was the best RNN model with accuracy, precision, recall and f1-score of 1, 0.74, 0.82 and 0.73, respectively. Figure 4.2 shows a ROC curve for the Q-69PI-RNN used to identify and evaluate how each class performs at a micro-level. The ROC curve shows an AUC greater than 0.90 across all classes except for *Web SQL Injection* that had 0.88. Overall, the model has outstanding discrimination for all classes with exceptional discrimination as the minimum performance. RNN generally requires large training sets, and it was interesting to observe the minority classes with good performance.

Further analysis of the ROC curve shows the exceptional performance for most classes that achieve a TPR greater than 0.99 and FPR less than 0.05. *Infiltration* and *Web SQL*

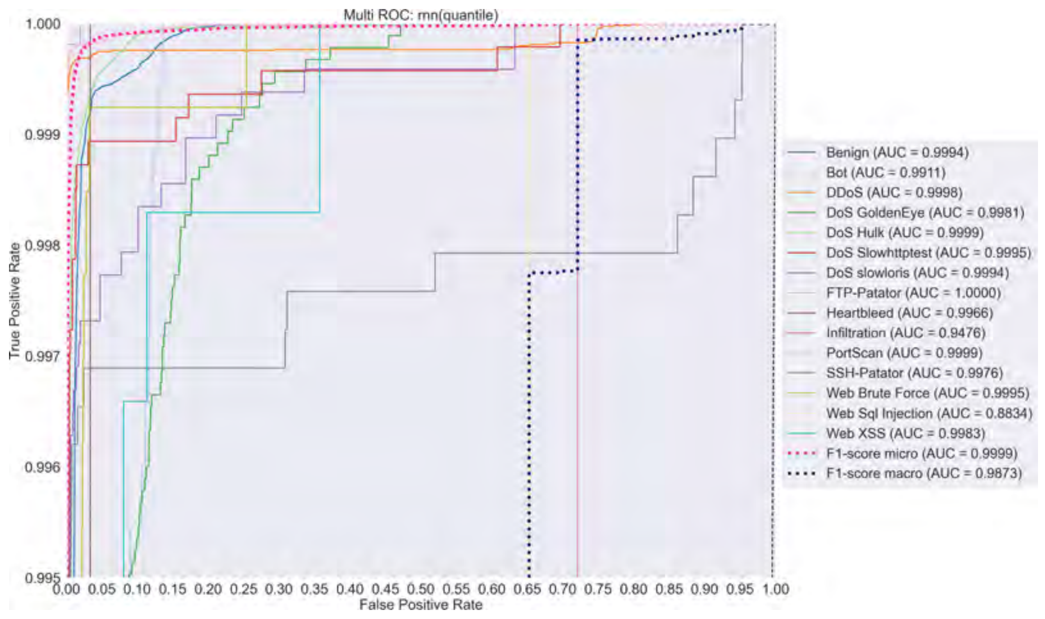

Figure 4.2: Q-69PI-RNN: ROC curve of the best performing RF model on limited data.

Injection appear to be the anomalies due to their higher FPR. The anomalies can result from significantly imbalanced class representation even after oversampling.

The model was effective at macro level detection but had poor, inconsistent detection of minority classes. The results show a downside on the model complexity in establishing the optimal parameters and the training times.

Train/Test Data Split Effects

Table 4.9 shows no significant consistent improvement in model performance from increasing training data for most cases. In general, there was a consistent observation of the benefits of adding more test data except for RF and DT based models. The findings show that six models prefer feature selection instead of the PCA dimensionality reduction method.

There are no significant additional findings on the effects of the train/test data split effects observed. For the interested reader, refer to Appendix A.7.8 on results of more training data.

Model	**Accuracy**	**f1-score (macro)**	**Split**
26PI-RF	1	0.90	0.5
Q-26PI-KNN	1	0.82	0.1
Q-36PI-MLP	1	0.81	0.1
Q-26PI-SVM	1	0.81	0.1
16PI-DT	1	0.76	0.1
Q-69PI-RNN	1	0.73	0.1
Q-26PCA-LR	0.99	0.73	0.1

Table 4.9: Summary of best results based on the extraction processing method.

4.3.2 Experiment 2: CICIDS 2017 Model testing with CICIDS 2018 dataset

Table 4.11 shows test results for the best CICIDS 2017 classifiers tested against the unseen CICIDS 2018 dataset. Note, the models did not undergo additional hyperparameter tuning and used the parameters from Section 4.2.1. Q-36PI-MLP achieves the best macro f1-score of 0.22.

Model	**Precision**	**Recall**	**f1-score (macro)**	**f1-score (micro)**
Q-36PI-MLP	0.22	0.25	0.22	0.81
Q-26PI-SVM	0.29	0.18	0.19	0.9
Q-26PI-KNN	0.23	0.21	0.17	0.66
Q-69PI-RNN	0.14	0.15	0.13	0.78
16PI-DT	0.08	0.13	0.09	0.62
26PI-RF	0.06	0.07	0.06	0.81
Q-26PCA-LR	0.06	0.04	0.05	0.46

Table 4.10: Model tests against CICIDS 2018 dataset.

Q-26PI-SVM and Q-26PI-KNN performed marginally lower than the best model, whilst the rest had a deficient performance. This low performance can be a result of the adopted attribute learning method, which used regular expression matching during data acquisition in Section 3.2.2. As stated in Section 3.2.6, the approach served to cater for the novel scenarios that differentiate the adaptive ML capabilities to the signature-based detection.

For future studies, a review into Pérez *et al.* (2017)'s approach to semantic information may be explored as a different method to try and improve the results.

In general, for unmodelled data, at a micro level, all models are able to predict the *Benign* class with excellent accuracy. MLP, SVM and KNN detected the majority classes. However, all models failed to predict the minority classes with high precision. Regardless of the generally low performance of the ML methods, MLP, SVM and KNN models proved better suited to detect new attack scenarios. Furthermore, these findings show the model's potential to be adopted as a binary classifier for 'Benign' versus 'Non-Benign' traffic classification.

These findings from the proposed model address the problem statement raised in Section 1.1, where expectations of NIDS are to adapt to new scenarios. This achieves of one of the set objectives as the findings show that NIDS models presented with unmodeled data (new scenarios) show potential to detect intrusions.

Table 4.11 shows sample results of Q-26PI-SVM model's detailed results of the multiple classes. The model presented the best micro results against the CICIDS 2018 dataset.

Class	**Precision**	**Recall**	f1-score (macro)
Benign	0.90	0.99	0.94
Bot	0	0	0
DDoS	0.90	0.19	0.32
DoS GoldenEye	0.99	0.21	0.34
DoS Hulk	0.55	0.84	0.67
DoS Slowhttptest	0	0	0
DoS slowloris	0.66	0.23	0.34
FTP-Patator	0	0	0
Infiltration	0	0	0
SSH-Patator	0	0	0
Web Brute Force	0	0	0
Web Sql Injection	0	0	0
Web XSS	0	0	0

Table 4.11: Q-26PI-SVM: Detection performance against the CICIDS 2018 dataset.

The model performs exceptionally well for the 'Benign' class at a micro-level. The *DoS*

GoldenEye class also achieved a high detection together with the other 'DoS' classes which place the model as a model that can move towards generalising its detection capability.

The model had poor detection for *Bot*, *DoS Slowhttptest, FTP-Patator*, *Infiltration*, *SSH-Patator*, *Web Brute Force*, *Web Sql Injection* and *Web XSS*. Similar findings were identified for the RF, RNN and MLP models. These results of the proposed model established that detection on unmodeled data has potential in real-world application.

4.3.3 Experiment 3: Training and Testing with CICIDS 2018

This subsection discusses the model post retraining using the CICIDS 2018 dataset. Table 4.12 shows the best models for each classifier after retraining with the CICIDS 2018 dataset. Note, the micro f1-score is not presented as it is similar to the accuracy.

Model	**Accuracy**	**Macro f1-score**
Q-69PI-RNN	0.98	0.73
26PI-RF	0.78	0.72
Q-26PI-SVM	0.73	0.68
Q-26PCA-MLP	0.78	0.67
Q-26PI-KNN	0.77	0.66
16PI-DT	0.77	0.64
P-26PCA-LR	0.70	0.62

Table 4.12: CICIDS 2018 Best Model Per Classifier.

Q-69PI-RNN provides the best results with an f1-score of 0.73, which persisted with results from the CICIDS 2017 limited data in Section 4.3.1. Marginally lower was the 26PI-RF model with an f1-score of 0.72. The performance of the RF model declined by 0.16 against 0.88 from the CICIDS 2017 RF model. Finally, the new SVM model achieved an f1-score of 0.68, down from the 0.82 from the best CICIDS 2017 SVM model.

An interesting observation is that PCA with a Quantile transform performs well against the CICIDS 2018 dataset. The observation suggests that PCA makes the dataset more uniform in distribution for the transformation to have a better effect.

RNN: After model retraining, at a macro level, RNN classifier findings remains similar to those from the CICIDS 2017 dataset, especially when contrasted with other classification models. Regardless of the model not having the best macro f1-score, findings show stability when presented with a different dataset. In these cases, the model is more consistent when compared to the rest of the models. Thus, RNN is a robust detection model choice for multiple categories of intrusions events.

The RNN model analysis shows outstanding discrimination across all classes at a micro class level. This finding assures the significance of RNN when applied to NIDS problems. For the interested reader, refer to Appendix A.3 on the results of the remodelled classifiers.

RF: Q-26PI-RF was the best performing RF model after remodelling with a different dataset. Unlike the CICIDS 2017 RF model, the Quantile Scaler was the better performer on this dataset. The inability of the model to retain the same engineered features as those used by the best CICIDS 2017 model shows that the model cannot be easily generalised. Regardless of the weakness in generalisation, the classifier continues to show strong candidacy for a NIDS problem but requires significant attention on feature engineering.

At a micro class level, the model achieves outstanding discrimination across all classes except an exceptional rating for the *Infiltration* class. However, this performance was lower than the CICIDS 2018 RNN model achieved. Regardless the model remains a strong option for a NIDS model.

SVM: Q-26PI-SVM performs high when presented with different datasets. The remodelled results show Q-26PI-SVM as being more insensitive to feature engineering than the RF counterpart. As much as RF had a better detection rate with a larger macro f1-score, the SVM model has the advantage of being easily generalised. Due to feature engineering not being applied to the RNN, there is no comparison with the RNN in the study.

Similar to RF, micro class analysis results show the model had outstanding discrimination across all classes except for *Infiltration*, which attained exceptional discrimination. Similarly, the model retains strong candidacy for as NIDS model.

There were no other results of significance observed in the table presented for the other classifiers.

4.4 Model Comparisons

Table 4.13 shows a representation of the macro-level best-evaluated model results against the CICIDS 2017 and CICIDS 2018 datasets. Q-69PI-RNN is the most insensitive and stable model that can be generalised. It can maintain its feature engineering properties and a persistent macro-level f1-score.

Model	**Macro f1-score (2017)**	**Model**	**Macro f1-score (2018)**
26PI-RF	0.90	26PI-RF	0.72
Q-26PI-KNN	0.82	Q-26PI-KNN	0.66
Q-36PI-MLP	0.81	Q-26PCA-MLP	0.67
Q-26PI-SVM	0.81	Q-26PI-SVM	0.68
16PI-DT	0.76	16PI-DT	0.64
Q-69PI-RNN	0.73	Q-69PI-RNN	0.73
Q-26PCA-LR	0.73	P-26PCA-LR	0.62

Table 4.13: Best model implementations across the CICIDS 2017 and 2018 datasets.

Regardless of performance degrading across different datasets, SVM, KNN and DT also show better generalisation as they were not very sensitive feature engineering changes for their best performers. There was an observation of instability across the rest of the classifiers because of the change in feature engineering for different datasets best performers. As such, these models cannot be generalised in the context.

4.5 Summary of Results

This chapter presented the findings of the processes and algorithms implemented during this research for constructing ML-based NIDS models. First, the chapter presents preliminary results from feature engineering. Results of the overall best performers against the

CICIDS 2017 dataset for the evaluated models against limited data are presented. The findings of the models are extended to evaluation against the CICIDS 2018 dataset. The chapter closes by presenting the micro class level modelled and remodelled class analysis.

5 Discussion

This study aimed to evaluate and adopt systematic ML-based methods to NIDS development. The chapter discusses the findings from Chapter 4 and its related literature. Section 5.5 briefly discusses feature selection and model hyperparameter tuning, and model training findings. Also included in the discussion was a section on the comparison with the related NIDS presented in Section 2.4. The chapter concludes with a summary of Chapter 5 discussions.

5.1 Introduction

Some noticeable differences between the findings compared to existing studies was in the lack of emphasis on a systematic approach to feature selection and metric selection (Stein *et al.*, 2005, Bisht and Ahmad, 2017, Chuan-long *et al.*, 2017, Aksu *et al.*, 2018a, Reis *et al.*, 2019, Atefi *et al.*, 2019, Subasi, 2020, Rosay *et al.*, 2020). Kurniabudi *et al.* (2020)'s study reflected on similar emphasis, where comparison and assessment with this study evolved to be more straightforward.

The rest of the sections in this chapter extend and highlight areas where this study observed gaps identified in the literature.

5.2 Data Preparation

Basic research considerations and understanding of the performance of each model against individual classes formed part of essential comprehensive NIDS reviews (Sommer and Paxson, 2010). Similar to Kurniabudi *et al.* (2020), findings presented in Section 4.3.1 and 4.3.3 reflected on the performance of each model against individual classes.

Due to significant class imbalance and minority instances count for classes like *Heartbleed* and *Web Brute Force*, the use of SMOTE for oversampling had a positive effect on classification models. Regardless of the class representation, RF and RNN models appeared not to show significant bias towards either minority or majority classes, which concurred with the findings from Reis *et al.* (2019) and Kurniabudi *et al.* (2020).

When reviewed against the CICIDS 2018 dataset, RNN showed minority class *Web XSS* as being able to achieve a high f1-score. This finding suggested that RNN could work with minimal data and achieve better results compared to the rest of evaluated models. In general, the training methods could get the most out of the algorithms and the dataset provided. Minority classes continued to be a challenge, but the findings showed the effects of applying sampling techniques to improve the training.

5.3 Feature Engineering

The effect of Quantile transformation with uniform output on distance-based classifiers – KNN, MLP, SVM, RNN and NB – showed superiority in the macro f1 of the models. The finding suggested that a uniform distribution transform to the training dataset inherently benefited the models.

The Standard Scaler performed best when applied to RF and DT classifiers. The scaling method generally performed worst against the distance-based classifiers. In such a case, it was better to use the base model, which had default parameters set according to best practices, instead of using the Standard Scaler. The Quantile Scaler consistently remained optimal, with five models adopting the method.

5.4 Feature Space Reduction

Features *DST Port*, *PSH Flag Cnt* and *Init Fwd Wind Byts* show a classifier agnostic trend. These were the most important features across most of the models. The features may prove useful on other datasets outside of this study, as they have repeatable properties by nature, unlike situational features such as *SRC Port* ranked by Rosay *et al.* (2020), which are environment/context-specific. Contrary to Rosay *et al.* (2020)'s findings on *SRC Port* feature relevance, there was no account of the criticality of feature ranking during the training phase as evaluated in this study and stated in other studies (Aksu *et al.*, 2018a, Ahmim *et al.*, 2019, Kurniabudi *et al.*, 2020). *DST Port* feature importance concurred with multiple studies where the feature was highly rated (Kurniabudi *et al.*, 2020, Reis *et al.*, 2019). Consequently, the process of detailing feature selection was important to guide systematic ML-based NIDS models development.

The feature importance results generally showed no universal set of equally essential features across all traditional classification models. Depending on the classification algorithm, feature preference may change, and it was essential to conduct a feature importance test when training classifiers for NIDS.

Overall, the use of feature selection with PI generally outperformed the feature extraction methods. Feature count of 26 appeared optimal with an increase in feature count beyond or lower than 26, yielding marginally different scores.

5.5 Model Evaluation

Each model's dimensionality reduction results highlighted the need for hyperparameter tuning. The process required an extensive set of systematic, iterative experimentation to evaluate the effects of each parameter. This process required caution to avoid overfitting or underfitting as this appeared prevalent. Of the related studies reviewed in Section 2.4, the experiment setup focused on the classifiers evaluated but did not discuss the model optimisation process.

It was challenging to assess whether some of the models in the literature produced a fair comparison, given that it was not clear that they were optimised to produce their best results. The findings from this study reflected on providing an opportunity to review better tuning for each NIDS classifier. In itself, there was better objectivity on each classifier's review.

Against the CICIDS 2017 dataset, KNN, MLP and SVM classifiers performed well with a marginal difference at a macro level. Upon inspecting individual classes, RF performed better for the majority classes but much less for the minority classes. MLP performed much better for the minority classes, similarly for SVM and RNN on class inspection.

On the other hand, RNN performed best with high accuracy when remodelled against the CICIDS 2018 dataset. The RF classifier follows, after which SVM and MLP retain high performance as observed for the CICIDS 2017 dataset. In the case of RNN findings, this study may accept errors in a practical scenario. To further improve the classifier, future studies can explore in-depth other supplemental architectures like utilising LSTM and GRU, similar to the work by Muhuri *et al.* (2020). This study did not venture into a detailed review of the effects of methods. Consequently, more research will need to train RNN for significantly underrepresented classes.

On further macro-level class inspection, generally, majority classes achieved higher performance against the minority classes. The RNN classifier had eight of the twelve predictions perfect. RF also performed well, but observations showed weakness towards minority classes similar to the CICIDS 2017 dataset.

Table 5.1 shows a summary of some of pros and cons of each of the models evaluated.

5.6 Related NIDS Comparison

In comparison to Chuan-long *et al.* (2017), the proposed methodology took an objective position on the development of the models. For the study, as part of the evaluation,

Model	**Pros**	**Cons**
RF	Simple model tuning Strong majority class detection No scaling or transformation required Outstanding micro class discrimination	Requires larger training sets Poor detection of minority classes Fail to effectively adjust to unseen data Harder to generalise for NIDS
SVM	Strong on binary classification Adjust fairly to untrained data Easier to generalise Outstanding micro class discrimination	Long training times Poor minority class detection
KNN	Strong on binary classification	Poor in multi-class detection High resource utilisation
MLP	Adapts better to untrained data	Complex model tuning Requires more training features
DT	Works effectively with few training features Less dependency on additional scaling Fast training times	Requires more training data Fail to adjust to untrained
RNN	Consistent feature engineering requirements Good minority class detection Outstanding micro class discrimination	Most complex tuning requirements Computationally intensive training Long training periods
LR	Better leverages feature extraction methods Strong as a binary classifier	Poor minority class detection Weak multi-class classifier

Table 5.1: Pros and Cons of evaluated models.

the experiment automatically seeked to assign repeatable yet dynamic strategies for each model. When the proposed methodology compared, it accounted for each classifier's best.

This study's methodology incorporated a breadth of evaluation criteria: ROC curves, recall, precision, and f1-score (both macro and weighted), which provided more insights into classification areas where RNN performed better than the other models. The results from the methodology followed achieved a perfect accuracy for both datasets, which was better than 0.8328 from Chuan-long *et al.* (2017). The study took a practical consideration of the real-world data, which was generally imbalanced. This consideration was evident in the datasets used, metrics explored beyond the accuracy metric, and class representation issues encountered using RNN.

Similar to the proposed methodology in this study, Kurniabudi *et al.* (2020) made emphasises the breadth of evaluation. Kurniabudi *et al.* (2020) emphasised feature selection by using information gain versus permutation importance and feature extraction methods used in this study. The study's general conclusion was that RF and DT with more features adopted have a better ability to detect Benign, DoS/DDoS, Brute Force and Bot attacks. The proposed methodology also realises this conclusion. However, there was no leverage of any Scaling or Transformation methods in the study including the effects of the train/test ratio.

Similar to the proposed methodology, Reis *et al.* (2019)'s study adopted feature selection as a means of identifying less interpretable predictions. The feature selection method uses statistical and computation methods to determine the importance of a particular predictor of anomalous traffic. This study's findings concurred with Reis *et al.* (2019) where the average of 26 features of the dataset was ideal for the experiment and was able to yield good results.

Rosay *et al.* (2020)'s study developed an MLP model that utilises a Standard Scaler as a means to provide better results. The findings showed that MLP performed well in all classes except for *Infiltration* and all Web Attacks. This finding on *Infiltration* was similar to the findings drawn from the proposed method except for the Web attack classes that were handled well in this study's experiments.

However, the approach taken for this experiment found the Quantile Scaler was a better scaling strategy for working with MLP in this problem set. Despite this finding, Rosay *et al.* (2020)'s model still had a higher f1-score of 0.99 compared to 0.81 achieved in this experiment. The details of how Rosay *et al.* (2020) were able to achieve the score could not be reproduced during the experiment setup followed as part of the experiments.

Lastly, Bisht and Ahmad (2017)'s study motivated an method of combining classifiers as a means to produce better-performing classifiers than a single classifier. This concurred with the study's findings where the RF ensemble method better served as a NIDS when compared against DT. However, this study could not be generalised to allow for better comparison unless there were further exploration.

In general, findings from the proposed method for the NIDS used in the study performed better than those identified in the literature. A more detailed review of the makeup of the proposed NIDS was discussed compared to that in related NIDS. Most of the related NIDS presented and emphasised a single aspect of the ML process – such as feature selection, classifiers or sampling methods – but not the detailed considerations of each step of developing the NIDS. As a result, the study may reveal limitations in the capabilities of the other related systems explored in the literature.

6

Conclusion and Future Work

In conclusion, the primary objective of systematically investigating and developing a supervised ML model for NIDS was completed and presented. The study reviews the experiment's findings to determine the performance of the models used and how they align with the set goals of the research. Findings showed fundamental challenges faced when applying ML to NIDS.

Based on additional gaps found in the literature, future work identified is detailed Section 6.4.

6.1 Summary of Research

This study focused on experimentation with a range of ways necessary for systematic ML-based NIDS classification models. The problem statements presented in Chapter 1 are detailed below:

1. In real-world systems, the vast majority of traffic is 'Benign'; as such, identifying malicious traffic may be challenging.

2. NIDS constantly encounter novel attacks, it follows that they should adapt to (detect) new scenarios.

The study explored several concepts related to effective ML-based NIDS development to address the above. The research considered the shortfalls of previous studies before establishing a proposed system to conduct the study as part of the literature review.

The experiment setup section of the study was as follows. The system went through several stages in the pipeline, namely; 1) Data Preparation, 2) Feature Engineering, and 3) Model Tuning and Evaluation.

Data acquisition and preprocessing activities were completed in the data preparation stage, resulting in 69 features, resampled and rebalanced training datasets. The research followed the feature engineering stage where feature scaling or transformation and dimensionality reduction were applied using PI and PCA. The ML classifier training commenced with the CICIDS 2017 validation set in the first experiment for the NIDS model development. The evaluation tests were then run against the unseen CICIDS 2018 dataset as part of the second experiment. The final experiment had the models retrained with the CICIDS 2018 dataset.

Sampling and balancing methods explored showed potential but there were no substantial improvements noted to allow a particular approach to be generalised. A focus on classifier tuning against the input data for NIDS use case did show the significant benefits to model performance.

The results section then addressed the main research objectives. RF achieves the best macro f1-score of 0.87 with the top 26 most important features using PI. This finding was higher than the previous studies by Almseidin *et al.* (2017), Atefi *et al.* (2019) and Kurniabudi *et al.* (2020). RNN and RF models consistently achieved high f1-score results with macro f1-scores of 0.73 and 0.87 for the CICIDS 2017 dataset; and 0.73 and 0.72 against the CICIDS 2018 dataset, respectively. With the exception of LR, distanced-based methods were able to detect attacks in untrained data in comparison to tree-based methods, however the detection was still poor.

In general, the results demonstrate that there are effective ways that can be adopted in developing ML-based needs that can be used in real-world systems. However, given that CICIDS 2018 was considered unseen for part of the experimentation with the intention of seeing whether the models are scalable to other datasets and new intrusion attacks that are principally different from the one in the datasets used. The findings where not substantial enough to draw a conclusion that the models can be generalised. As such, more research will be required to achieve better scalable models that adapt to different datasets.

6.2 Research Objective

Sections 6.1 and 6.2, reiterated the original objectives of the study based on the research conducted. In response to the problem statement, the research achieved the following objectives:

1. Systematically investigating and developing ML-based NIDS models, including DL.
2. Investigating methods that account for the significant data imbalance and distribution.
3. Performing parameter tuning to validate the models on a subset of the first dataset and biasing towards new data generalisation.
4. Determining the most effective NIDS performance metrics and ML models for modelled dataset against the unmodelled data from another dataset.

The findings concluded that each model will have varying configuration requirements that are best with no single appropriate method. However, a general trend appears for scaling methods, where the quantile transformation has generally yielded the best results for distance-based methods.

6.3 Research Contribution

This research contributed towards the effective adoption of ML-based NIDS capabilities following a systematic process. The major contribution was the methodology for developing a systematic ML-based system for NIDS. Moreover, a results contribution in the form of a comprehensive model comparison was produced and may prove to be useful for future research.

6.4 Future Work

There is a significant scope for future work in this area based on the above conclusion. The study was able to show the benefits of feature selection in the ML-based NIDS model development. A correlation matrix is one method that is worth exploring to identify correlated features to contrast with Permutation Importance.

Additionally, given the widespread adoption of the KDD-99 dataset in related studies, studies to consolidate features and labelling across datasets can be developed to address limitations inherent to the model development with different datasets.

Lastly, the material discussed in the results chapter highlights areas where there are opportunities to conduct future research.

References

Abdulhammed, R., Musafer, H., Alessa, A., Faezipour, M., and Abuzneid, A. Features dimensionality reduction approaches for machine learning based network intrusion detection. *Electronics*, 8(3):322, 2019.

Abdulraheem, M. H. and Ibraheem, N. B. A detailed analysis of new intrusion detection dataset. In *Semantic Scholar*. 2019.

Ahmim, A., Maglaras, L., Ferrag, M. A., Derdour, M., and Janicke, H. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 228–233. IEEE, 2019.

Aksu, D., Ustebay, S., Aydin, M., and Atmaca, T. Intrusion Detection with Comparative Analysis of Supervised Learning Techniques and Fisher Score Feature Selection Algorithm, pages 141–149. Springer International Publishing, Cham, 09 2018a. doi:10.1007/978-3-030-00840-6_16.

Aksu, D., Üstebay, S., Aydin, M. A., and Atmaca, T. Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In **Czachórski, T., Gelenbe, E., Grochla, K., and Lent, R.**, editors, *Computer and Information Sciences*, pages 141–149. Springer International Publishing, Cham, 2018b. ISBN 978-3-030-00840-6.

Almseidin, M., Alzubi, M., Kovacs, S., and Alkasassbeh, M. Evaluation of machine learning algorithms for intrusion detection system. In *2017 IEEE 15th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 000277–000282. Sep. 2017. ISSN 1949-0488. doi:10.1109/SISY.2017.8080566.

Alpaydin, E. *Introduction to machine learning.* MIT press, 2020, 3–7 pages.

Altmann, A., Toloşi, L., Sander, O., and Lengauer, T. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

Arunraj, N., Hable, R., Fernandes, M., Leidl, K., and Heigl, M. Comparison of supervised, semi-supervised and unsupervised learning methods in network intrusion detection system application. *Anwendungen und Konzepte der Wirtschaftsinformatik*, 6:10 – 19, 11 2017.

Atefi, K., Hashim, H., and Kassim, M. Anomaly analysis for the classification purpose of intrusion detection system with k-nearest neighbors and deep neural network. In *2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, pages 269–274. 2019.

Axelsson, S. Intrusion detection systems: A survey and taxonomy. Technical report, Citeseer, 2000.

Barapatre, P., Tarapore, N., Pukale, S., and Dhore, M. Training MLP neural network to reduce false alerts in ids. In *2008 International Conference on Computing, Communication and Networking*, pages 1–7. IEEE, 2008.

Bisht, N. and Ahmad, A. Analysis of classifier ensembles for network intrusion detection systems. *Communications on Applied Electronics*, 6:47–53, 02 2017. doi: 10.5120/cae2017652516.

Bowyer, K. W., Chawla, N. V., Hall, L. O., and Kegelmeyer, W. P. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011. URL http://arxiv.org/abs/1106.1813

Branco, P., Torgo, L., and Ribeiro, R. P. A survey of predictive modelling under imbalanced distributions. *CoRR*, abs/1505.01658, 2015. URL http://arxiv.org/abs/1505.01658

Cardenas, A. A., Baras, J. S., and Seamon, K. A framework for the evaluation of intrusion detection systems. In *2006 IEEE Symposium on Security and Privacy (S P'06)*, pages 15 pp.–77. May 2006. ISSN 2375-1207. doi:10.1109/SP.2006.2.

Chapelle, O., Haffner, P., and Vapnik, V. Support vector machines for histogram-based image classification. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 10:1055–64, 09 1999. doi:10.1109/72.788646.

Cherkassky, V. and Ma, Y. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113 – 126, 2004. ISSN 0893-6080. doi:https://doi.org/10.1016/S0893-6080(03)00169-2.
URL http://www.sciencedirect.com/science/article/pii/S0893608003001692

Chowdhury, M. M. U., Hammond, F., Konowicz, G., Xin, C., Wu, H., and Li, J. A few-shot deep learning approach for improved intrusion detection. In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, pages 456–462. Oct 2017. doi:10.1109/UEMCON.2017.8249084.

Chuan-long, Y., Yue-fei, Z., Jin-long, F., and Xin-zheng, H. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, PP:1–1, 10 2017. doi:10.1109/ACCESS.2017.2762418.

Corona, I., Giacinto, G., and Roli, F. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences*, 239:201 – 225, 2013. ISSN 0020-0255. doi:https://doi.org/10.1016/j.ins.2013.03.022.
URL http://www.sciencedirect.com/science/article/pii/S0020025513002119

Davis, J. and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. 2006.

Debar, H., Dacier, M., and Wespi, A. A revised taxonomy for intrusion-detection systems. In *Annales des télécommunications*, volume 55, pages 361–378. Springer, 2000.

Denning, D. E. An intrusion-detection model. *IEEE Transactions on software engineering*, 1(SE-13):222–232, 1987.

Dong, B. and Wang, X. Comparison deep learning method to traditional methods using for network intrusion detection. In *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pages 581–585. 2016.

Doshi, R., Apthorpe, N., and Feamster, N. Machine learning DDoS detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. 2018.

Edgar, T. W. and Manz, D. O. Chapter 6 - machine learning. In **Edgar, T. W. and Manz, D. O.**, editors, *Research Methods for Cyber Security*, pages 153 – 173. Syngress, 2017. ISBN 978-0-12-805349-2. doi: https://doi.org/10.1016/B978-0-12-805349-2.00006-6.
URL http://www.sciencedirect.com/science/article/pii/B9780128053492000066

Ferri, C., Hernández-Orallo, J., and Modroiu, R. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, 2009.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 28(1):18 – 28, 2009. ISSN 0167-4048. doi:https://doi.org/10.1016/j.cose.2008.08.003.
URL http://www.sciencedirect.com/science/article/pii/S0167404808000692

Hayes, M. A. and Capretz, M. A. M. Contextual anomaly detection framework for big sensor data. *Journal of Big Data*, 2:1–22, 2014.

Heba, F. E., Darwish, A., Hassanien, A. E., and Abraham, A. Principle components analysis and support vector machine based intrusion detection system. In *2010 10th international conference on intelligent systems design and applications*, pages 363–367. IEEE, 2010.

Hofstede, R., Jonker, M., Sperotto, A., and Pras, A. Flow-based web application brute-force attack and compromise detection. *Journal of Network and Systems Manage-*

ment, 25(4):735–758, October 2017. ISSN 1064-7570. doi:10.1007/s10922-017-9421-4. URL https://doi.org/10.1007/s10922-017-9421-4

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. Applied logistic regression, volume 398. John Wiley & Sons, 2013, 160-164 pages.

Javaid, A., Niyaz, Q., Sun, W., and Alam, M. A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS)*, BICT'15, page 21–26. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 2016. ISBN 9781631901003. doi:10.4108/eai.3-12-2015.2262516.
URL https://doi.org/10.4108/eai.3-12-2015.2262516

Javitz, H. S., Valdes, A. *et al.* The SRI IDES statistical anomaly detector. In *IEEE Symposium on Security and Privacy*, pages 316–326. Oakland, 1991.

Jiang, L., Cai, Z., Wang, D., and Jiang, S. Survey of improving k-nearest-neighbor for classification. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2007)*, volume 1, pages 679–683. 2007.

Jones, A. K. and Sielken, R. S. Computer system intrusion detection: A survey. 2000.

Kayacik, H. G., Zincir-Heywood, A. N., and Heywood, M. I. Selecting features for intrusion detection: A feature relevance analysis on kdd 99. In *PST*. 2005.

Kizza, J. M. *Guide to computer network security*. Springer, 2013.

Kurniabudi, Stiawan, D., Darmawijoyo, Bin Idris, M. Y., Bamhdi, A. M., and Budiarto, R. CICIDS-2017 dataset feature analysis with information gain for anomaly detection. *IEEE Access*, 8:132911–132921, 2020. doi:10.1109/ACCESS.2020.3009843.

Lakhina, S., Joseph, S., and Verma, B. Feature reduction using principal component analysis for effective anomaly–based intrusion detection on NSL-KDD. *International Journal of Engineering Science and Technology*, 2(6):1790–1799, 2010.

Laskov, P., Düssel, P., Schäfer, C., and Rieck, K. Learning intrusion detection: Supervised or unsupervised? In **Roli, F. and Vitulano, S.**, editors, *Image Analysis and Processing – ICIAP 2005*, pages 50–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31866-8.

Li, Y., Xia, J., Zhang, S., Yan, J., Ai, X., and Dai, K. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 39(1):424 – 430, 2012. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2011.07.032.
URL http://www.sciencedirect.com/science/article/pii/S0957417411009948

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., and Tung, K.-Y. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

Louppe, G., Wehenkel, L., Sutera, A., and Geurts, P. Understanding variable importances in forests of randomized trees. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, page 431–439. Curran Associates Inc., Red Hook, NY, USA, 2013.

Lunt, T. F. and Jagannathan, R. A prototype real-time intrusion-detection expert system. In *Proceedings. 1988 IEEE Symposium on Security and Privacy*, pages 59–66. 1988.

Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., and Elovici, Y. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, Jul 2018. ISSN 1558-2590. doi:10.1109/MPRV.2018.03367731.

Menze, B. H., Kelm, B. M., Masuch, R., Himmelreich, U., Bachert, P., Petrich, W., and Hamprecht, F. A. A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. *BMC bioinformatics*, 10(1):213, 2009.

Mladenić, D. Feature selection for dimensionality reduction. In **Saunders, C., Grobelnik, M., Gunn, S., and Shawe-Taylor, J.**, editors, *Subspace, Latent Structure and Feature Selection*, pages 84–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-34138-3.

Muhuri, P., Chatterjee, P., Yuan, X., Roy, K., and Esterline, A. Using a long short-term memory recurrent neural network (LSTM-RNN) to classify network attacks. *Information*, 11:243, 05 2020. doi:10.3390/info11050243.

Opitz, J. and Burst, S. Macro f1 and macro f1. 2019. doi:10.48550/ARXIV.1911.03347. URL https://arxiv.org/abs/1911.03347

Panda, M., Abraham, A., and Patra, M. R. A hybrid intelligent approach for network intrusion detection. *Procedia Engineering*, 30:1–9, 2012.

Panigrahi, R. and Borah, S. A detailed analysis of cicids2017 dataset for designing intrusion detection systems. *International Journal of Engineering & Technology*, 7(3.24):479–482, 2018.

Pérez, J. L. R., Ribeiro, B., Chen, N., and Leite, F. S. A grassmannian approach to zero-shot learning for network intrusion detection. *CoRR*, abs/1709.07984, 2017. URL http://arxiv.org/abs/1709.07984

Ramezankhani, A., Pournik, O., Shahrabi, J., Azizi, F., Hadaegh, F., and Khalili, D. The impact of oversampling with SMOTE on the performance of 3 classifiers in prediction of type 2 diabetes. *Medical decision making : an international journal of the Society for Medical Decision Making*, 36, 12 2014. doi:10.1177/0272989X14560647.

Reis, B., Maia, E., and Praça, I. Selection and performance analysis of CICIDS-2017 features importance. In *International Symposium on Foundations and Practice of Security*, pages 56–71. Springer, 2019.

Ren, J., Guo, J., Qian, W., Yuan, H., Hao, X., and Jingjing, H. Building an effective intrusion detection system by using hybrid data optimization based on machine learning algorithms. *Security and Communication Networks*, 2019:1–11, 06 2019. doi:10.1155/2019/7130868.

Resende, P. A. A. and Drummond, A. C. A survey of random forest based methods for intrusion detection systems. *ACM Computing Surveys (CSUR)*, 51(3):1–36, 2018.

Ring, M. and Eskofier, B. M. An approximation of the Gaussian RBF kernel for efficient classification with SVMs. *Pattern Recognition Letters*, 84:107 – 113, 2016. ISSN 0167-8655. doi:https://doi.org/10.1016/j.patrec.2016.08.013.
URL http://www.sciencedirect.com/science/article/pii/S016786551630215X

Ring, M., Wunderlich, S., Scheuring, D., Landes, D., and Hotho, A. A survey of network-based intrusion detection data sets. *Computers and Security*, 86:147–167, 2019.

Rosay, A., Carlier, F., and Leroux, P. Mlp4nids: An efficient MLP-based network intrusion detection for CICIDS2017 dataset. In **Boumerdassi, S., Renault, É., and Mühlethaler, P.**, editors, *Machine Learning for Networking*, pages 240–254. Springer International Publishing, Cham, 2020. ISBN 978-3-030-45778-5.

Russell, S. and Norvig, P. *Artificial intelligence: a modern approach*. Prentice Hall, 2002, 693–765 pages.

Schneier, B. Secrets and lies: digital security in a networked world. John Wiley & Sons, 2015.

Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116. 2018.

Shenfield, A., Day, D., and Ayesh, A. Intelligent intrusion detection systems using artificial neural networks. *ICT Express*, 4(2):95–99, 2018.

Shone, N., Ngoc, T. N., Phai, V. D., and Shi, Q. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41–50, 2018.

Singh, R., Kumar, H., and Singla, R. An intrusion detection system using network traffic profiling and online sequential extreme learning machine. *Expert Systems with Applications*, 42(22):8609–8624, 2015.

Siva, Geetha, and Kannan. Decision tree based light weight intrusion detection using a wrapper approach. *Expert Systems with Applications*, 39(1):129 – 141, 2012. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2011.06.013.
URL http://www.sciencedirect.com/science/article/pii/S0957417411009080

Sommer, R. and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316. 2010.

Songwattanasiri, P. and Sinapiromsaran, K. Smoute: Synthetics minority over-sampling and under-sampling techniques for class imbalanced problem. In *Proceedings of the Annual International Conference on Computer Science Education: Innovation and Technology, Special Track: Knowledge Discovery*, pages 78–83. 2010.

Stein, G., Chen, B., Wu, A. S., and Hua, K. A. Decision tree classifier for network intrusion detection with GA-based feature selection. In *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 2*, ACM-SE 43, page 136–141. Association for Computing Machinery, New York, NY, USA, 2005. ISBN 1595930590. doi:10.1145/1167253.1167288.
URL https://doi.org/10.1145/1167253.1167288

Su, M.-Y. Real-time anomaly detection systems for denial-of-service attacks by weighted k-nearest-neighbor classifiers. *Expert Systems with Applications*, 38(4):3492 – 3498, 2011. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2010.08.137.
URL http://www.sciencedirect.com/science/article/pii/S0957417410009450

Subasi, A. Chapter 3 - machine learning techniques. In **Subasi, A.**, editor, *Practical Machine Learning for Data Analysis Using Python*, pages 91 – 202. Academic Press, 2020. ISBN 978-0-12-821379-7. doi:https://doi.org/10.1016/B978-0-12-821379-7.00003-5.

URL http://www.sciencedirect.com/science/article/pii/B9780128213797000035

Tax, D. and Duin, R. Feature scaling in support vector data descriptions. *Learning from Imbalanced Datasets*, pages 25–30, 2000.

Thakkar, A. and Lohiya, R. A review of the advancement in intrusion detection datasets. *Procedia Computer Science*, 167:636–645, 2020. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2020.03.330. International Conference on Computational Intelligence and Data Science.
URL https://www.sciencedirect.com/science/article/pii/S1877050920307961

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., and Lin, W.-Y. Intrusion detection by machine learning: A review. *expert systems with applications*, 36(10):11994–12000, 2009.

Tu, J. V. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225 – 1231, 1996. ISSN 0895-4356. doi:https://doi.org/10.1016/S0895-4356(96)00002-9.
URL http://www.sciencedirect.com/science/article/pii/S0895435696000029

Valenti, S., Rossi, D., Dainotti, A., Pescapè, A., Finamore, A., and Mellia, M. Reviewing Traffic Classification, pages 123–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-36784-7. doi:10.1007/978-3-642-36784-7_6.
URL https://doi.org/10.1007/978-3-642-36784-7_6

Valiant, L. G. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782. doi:10.1145/1968.1972.
URL https://doi.org/10.1145/1968.1972

Van, N. T., Thinh, T. N., and Sach, L. T. An anomaly-based network intrusion detection system using deep learning. In *2017 International Conference on System Science and Engineering (ICSSE)*, pages 210–214. 2017.

Viegas, E. K., Santin, A. O., and Oliveira, L. S. Toward a reliable anomaly-based intrusion detection in real-world environments. *Comput. Netw.*, 127(C):200–216, November 2017. ISSN 1389-1286. doi:10.1016/j.comnet.2017.08.013.
URL https://doi.org/10.1016/j.comnet.2017.08.013

Vinayakumar, R., Soman, K. P., and Poornachandran, P. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1222–1228. 2017.

Wang, K. and Stolfo, S. J. Anomalous payload-based network intrusion detection. In **Jonsson, E., Valdes, A., and Almgren, M.**, editors, *Recent Advances in Intrusion Detection*, pages 203–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-30143-1.

Wang, S. and Yao, X. Diversity analysis on imbalanced data sets by using ensemble models. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pages 324–331. IEEE, 2009.

Xian, Y., Lampert, C. H., Schiele, B., and Akata, Z. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2251–2265, 2018.

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., and Wang, C. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, 6:35365–35381, 2018. doi:10.1109/ACCESS.2018.2836950.

Zhang, B., Liu, Z., Jia, Y., Ren, J., and Zhao, X. Network intrusion detection method based on PCA and Bayes algorithm. *Security and Communication Networks*, 2018:1–11, 11 2018. doi:10.1155/2018/1914980.

Zhang, Z., Liu, Q., Qiu, S., Zhou, S., and Zhang, C. Unknown attack detection based on zero-shot learning. *IEEE Access*, 8:193981–193991, 2020. doi:10.1109/ACCESS.2020.3033494.

A

Appendix

A.1 Data Preparations

A.1.1 Dataset: CICIDS2017

The CICIDS 2017 dataset is an evaluation dataset created by the Canadian Institute of Cyber Security to reflect modern intrusion events and can be used to develop NIDS. The dataset is presented as initially network packet captured files (PCAPs) that were processed using the CICFlowmeter to produce an output with statistical features calculated separately for forward and reverse network communication for a bidirectional network.

CICFlowmeter is a network bidirectional flow generator, which utilizes the first network packets from a PCAP file to determine the forward direction (source network to the destination network) and backward direction (destination network to source network) of the network data flow. The PCAP files were generated data was captured over several days with different attack profiles and scenarios being simulated and accounted for as part of the process to allow data labelling for intrusion classification.

Using the CICFlowmeter, data output was then further processed to add classification labels for each extracted instance, which resulted in comma-separated values (CSV) files being created which can be used for NIDS development.

The final dataset comprises a classification label and 83 statistical features such as duration of transmission, the number of packets transmitted, the number of bytes transmitter, length of packets, e.t.c, each being calculated separately for forward and reverse directions.

A.1.2 Attack Profiles

Such attack profiles used in this dataset also reflect on these attack profiles. This dataset has six attack profiles created based on a list of common attack families.

- **DoS Attack** is an attack meant to make computer service unavailable for use due to either resource constraints such as RAM hogging or CPU full utilization or Storage Disk full utilization, or simply shutting down as computer system access altogether, or corrupting files on the system rendering them inaccessible. Typically this is exploited by an attacker that overload system resources and/or prevents some or all legitimate requests fulfilment.
- **Distributed Denial of Service (*DDoS*) attack** is an advanced DoS attack that typically uses multiple computers to flood the bandwidth or resources of a victim computer network. Generally, a *Zombie* network is used by the attacker where the Zombies are computers infected with tools installed by an attacker who can then launch an attack.
- **Heart Bleed bug** is a flaw in the OpenSSL method for data encryption. This vulnerability is exploited by sending malformed requests with a small payload and large length field to the vulnerable networked computer to evoke the victim machine to respond with information in memory. The attack can be categorized as a Remote to User attack.
- **Botnet** is a group of interconnected devices used to perform various tasks on computer networks. They can sometimes be used for malicious activities such as stealing

data, transmitting spam or launching attacks such as DDoS attacks. Botnets can also be used as part of a Probing attack.

- **Brute Force Attack** is an attack used for cracking passwords and also for discovering hidden pages and content of web applications. This type of attack is based on trial and error until a victim is successfully exploited.
- **Web Attacks** are toolkits that target web applications vulnerabilities that may be existent such as SQL Injection, Cross-Site Scripting (XSS), brute force over HTTP, which are included in this dataset. The attack can fit into all categories of attacks depending on what vulnerability is being exploited.
- **Infiltration Attack** is an attack from inside a computer network that is commonly exploited from vulnerable internal software. The attacks are generally categorized as U2R or R2U attacks, leading to a further Probing attack to spread the attack surface.

A.1.3 Data Capturing

The data capturing was conducted over five days based on the attack profiles explained in Appendix A.1.2. Table A.1 shows the distribution of the attack profiles for each of the days used to generate the dataset.

Name of Files	**Day Activity**	**Attacks Found**
Monday Working ISCX.csv	Monday	Benign
Tuesday Working ISCX.csv	Tuesday	Benign, FTP Patator, SSH Patator
Wednesday Working ISCX.csv	Wednesday	Benign, DoS Golden Eye, DoS hulk, DoS Slow httptest, DoS slowloris, Heartbleed
Thursday Working Morning WebAttacks.pcap_ISCX.csv	Thursday	Benign, Web Attack - Brute Force, Web Attack- SQL Injection, Web Attack XSS
Thursday Working Afternoon Infiltration ISCX.csv	Thursday	Benign, Infiltration
Friday Working Morning ISCX.csv	Friday	Benign, Bot

Table A.1: CICIDS 2017 dataset captured files and profiles

A.1.4 Dataset Criteria

Figure A.1 shows the testbed architecture used to create the CICIDS2017 dataset.

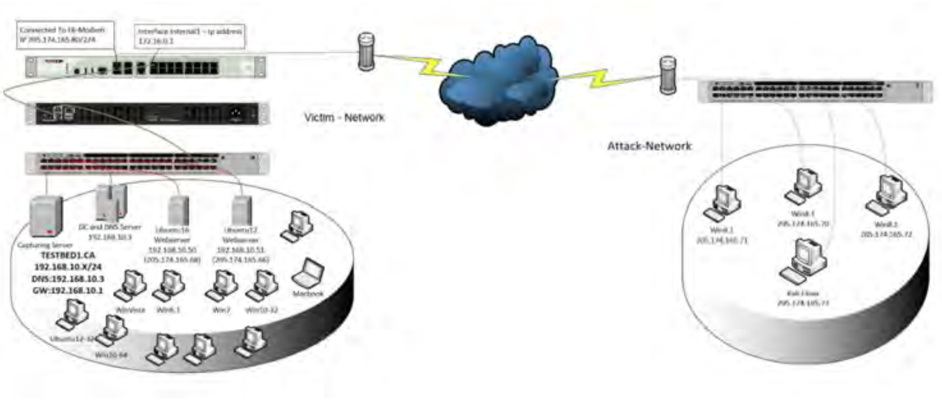

Figure A.1: CICIDS2017 Test Bed Architecture

The architecture of Figure A.1 can fulfil the 11 critical criteria for a quality dataset as follows:

- complete network configuration - architecture of dataset generation fulfils this by including the use of various operating systems such as Windows and Ubuntu and equipment such as a modem, firewall, switches and routers.
- total traffic - architecture of dataset generation fulfils this by simulation of attack network that has a user profiling agent and 12 different machines in Victim Network.
- labelling in the dataset is fulfilled by the event windows, which allowed the dataset to be labelled.
- complete interaction - this was fulfilled by having communications between an internal LAN and two different networks and Internet communication.
- complete capture - this is fulfilled through the generation of mirror ports used for all traffic to be captured and recorded
- available protocols - this was fulfilled by the availability of a range of supported protocols such as FTP, SSH and HTTP in the dataset.
- attack diversity - this was fulfilled by the use of a varying range of attacks that were described in section A.1.2.
- heterogeneity - this was achieved by capturing network traffic from the main Switch, memory dumps, and system calls for all victim machines
- feature set - this was achieved through the presence of 83 network features created by the CICFlowmeter that is used to process the network PCAPs
- metadata - the details of the dataset achieved this provided, such as time, attacks, flows, and label provided with the dataset.

Anonymity was the only criteria that was not explained how it would fully complete the crucial criteria.

A.2 Feature Engineering

A.2.1 Feature Scaling and Transformation

Standard scaler increases feature space the most but only uses a fraction of it. Pros is that it does not transform the data, so outliers remain outliers.

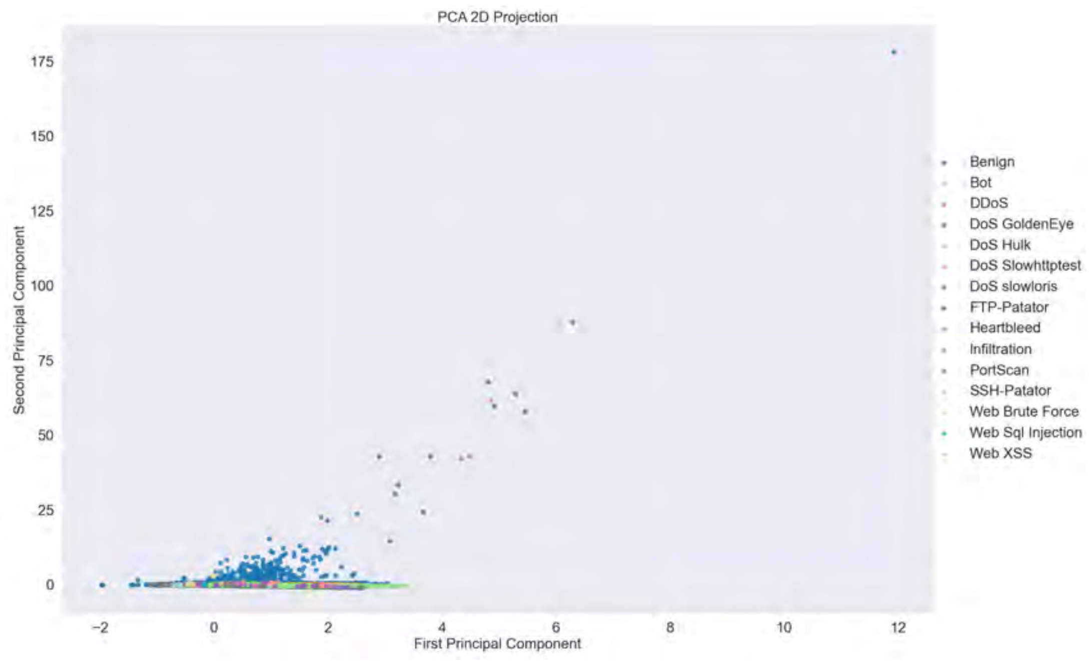

Figure A.2: Standard Scaling

Quantile Scaler provides non-linear transformations in which distances between marginal outliers and inliers are shrunk using a uniform transformation.

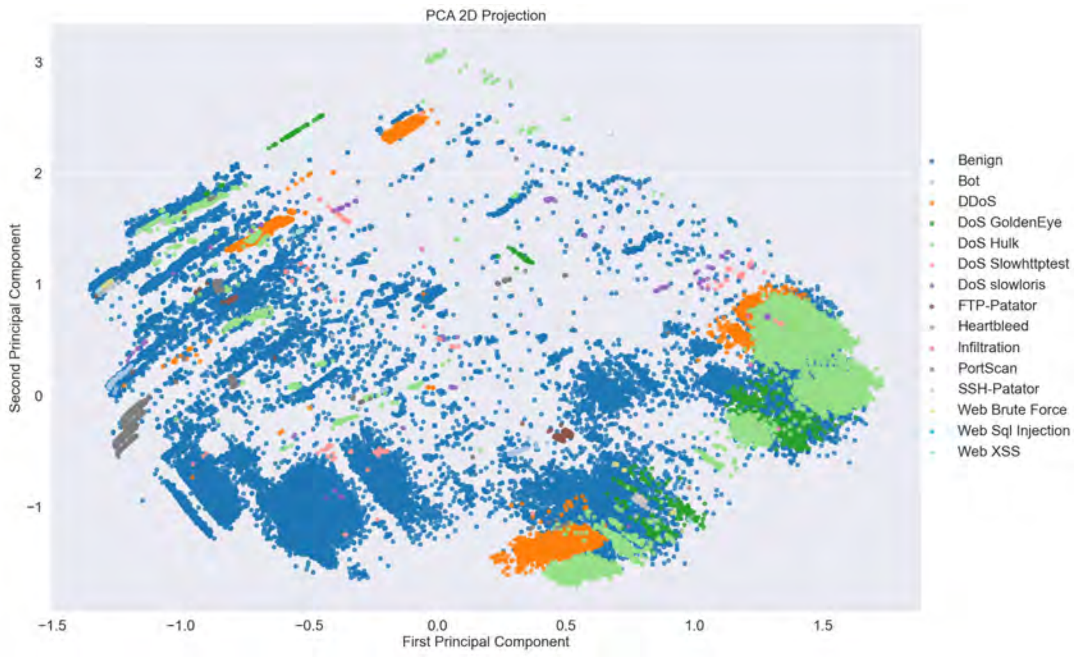

Figure A.3: Quantile Transformer

Power scaler applies a power transformation to each feature to make the data more Gaussian-like using Yeo-Johnson transformation. Similar to Quantile, it provides non-linear transformations in which data is mapped to a normal distribution to stabilize variance and minimize skewness.

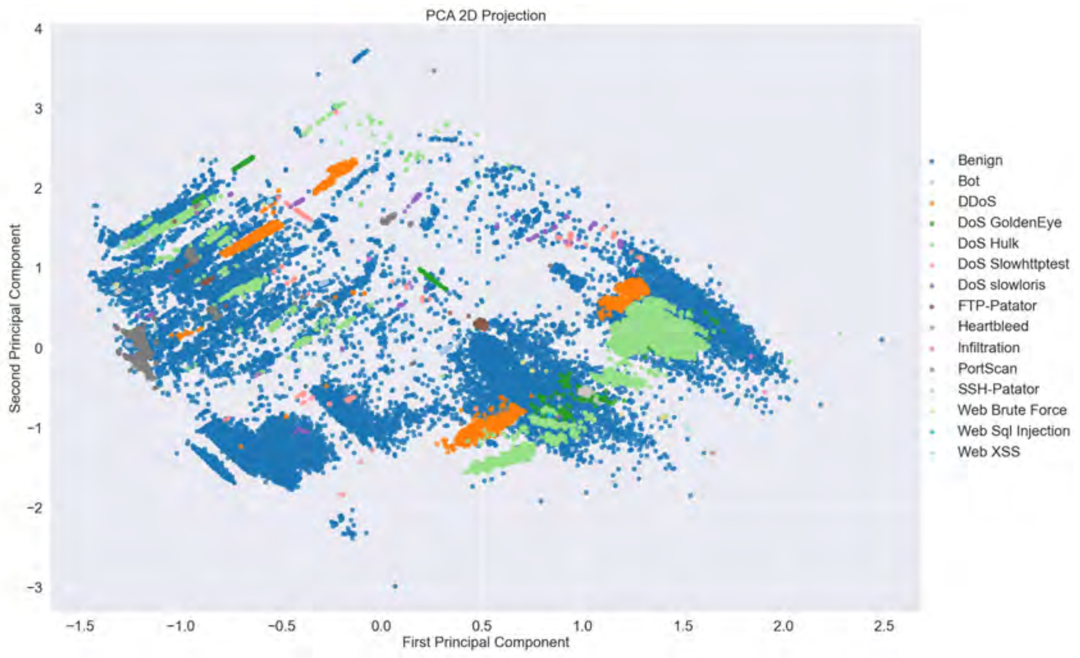

Figure A.4: Power Transformer

A.3 Dimensionality Reduction

A.3.1 Feature Ranking

RF Feature Rank

Figure A.5 shows that the *Dst Port* – destination port – feature contributes the most discriminative information to a base **RF** classifier by a significant margin. Since this is a classification problem, it implies that, overall, the input from *Dst Port* carries the most weight in distinguishing between different classes. Given that *Dst Port* is typically an entry point for any network activity – Benign or Malicious – its high importance ranking is thus expected. On the other hand, none of the next seven features is significantly better and falls within the same quantile range. The *Init Bwd Byts* is the second most significant feature for **RF**. The remainder of the top five features include *Bwd Seg Size Avg*, *Fwd IAT Min* and *Tot Len Pkts*. Moreover, this shows that each feature ranking

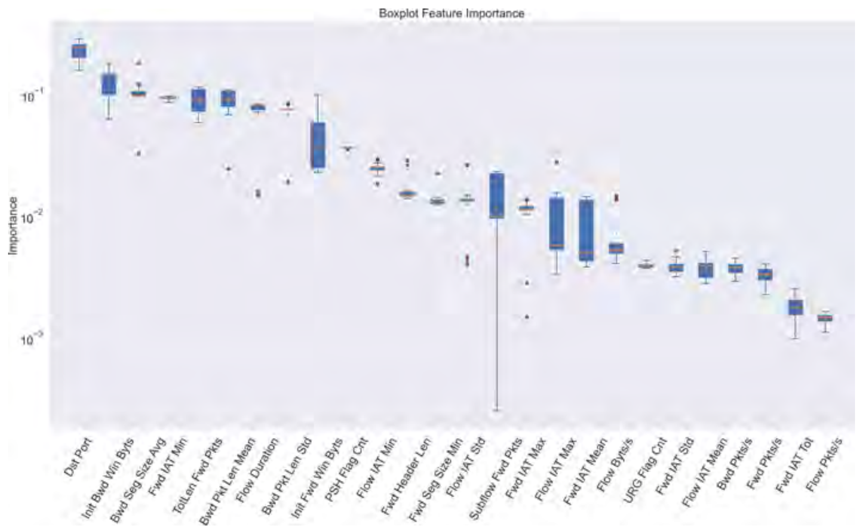

Figure A.5: RF: Boxplot of permutation importance per feature.

has differing quantile ranges and extreme lower values such as *Subflow Fwd Pkts* when applied to **RF**. In permutation importance, the shorter the range of the box, the higher the compatibility with other combinations of features in terms of achieving optimal classification performance (Altmann *et al.*, 2010) – henceforth referred to as stability. In this case, the alternating variance suggests instability in the model from the feature selection process.

KNN Feature Rank

Figure A.6 shows that the base **KNN** model prefers the *Dst Port* feature by a highly significant margin. Similar to **RF** Feature Rank, given a destination port is an entry point for network activities, its high importance ranking is thus expected. The *Flow IAT Min* is the second most significant feature for **KNN**. It suggests that the minimum time between two network flows contributes to a reduction in intraclass distance. The remainder of the top five features then include *Fwd IAT Min*, *Bwd Header Len* and *Init Bwd Win Byts*. The variance in most features is significant in combination with relatively

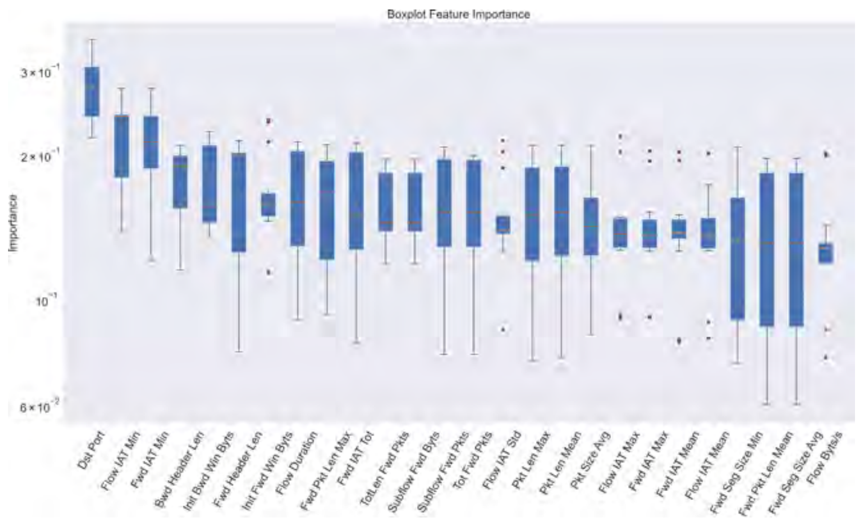

Figure A.6: KNN: Boxplot of permutation importances per feature.

high importance scores, indicating that the ranking order is significant. In the case of **KNN**, this means that the model is volatile, with slight changes potentially resulting in a significant shift in model performance, which was subsequently confirmed by reversing the order of the last ten features (ranked 17–26).

MLP Feature Rank

Figure A.7 shows that base **MLP** prefers *Dst Port* – destination port – by a highly significant margin but with one outlier, that may represent a class that is an outlier.

Similar to **RF**, this feature has consistently been the most important feature. *Fwd Act Data Pkts* – forward acknowledge data packet – is the second most significant feature for **MLP**. The remainder of the top five features then include *Fwd Pkt Len Max*, *PSH Flag Cnt* and *Init Fwd Win Byts*. The rest of the features are unstable but have equal

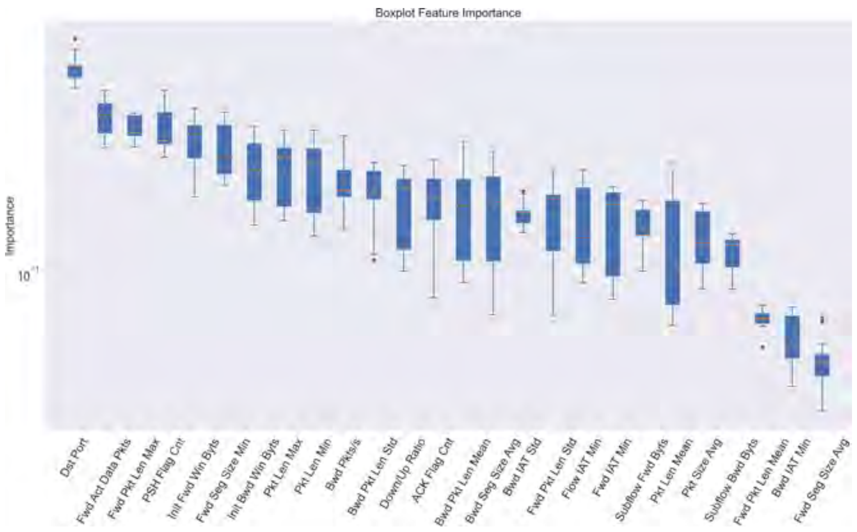

Figure A.7: MLP: Boxplot of permutation importances per feature.

importance and only five cases of outliers.

SVM Feature Rank

Figure A.8 shows that the base **SVM** prefers *Bwd Pkt Len Std* – Backward packet standard length – and *Dst Port* – destination port – features as the most significant features. However, *Bwd Pkt Len Std* shows outliers which may be classes that could not fit well with the feature. The top two features have very similar importance ranking and this suggests that both features are crucial to **SVM**. The remainder of the top five features include *PSH Flag Cnt*, *Fwd IAT Std* and *Fwd Pkt Len Max*. The top features shown in the Figure concurs with those from Aksu *et al.* (2018a)'s findings on the 'important' features that **SVM** adopts. The general distribution of the features, their small quantile range, and the count of outliers suggest that feature ranking does not significantly affect the classifier. There are many features with a narrower range of the box, suggesting higher compatibility with other combinations of features in achieving optimal classification performance.

Other Feature Rankings

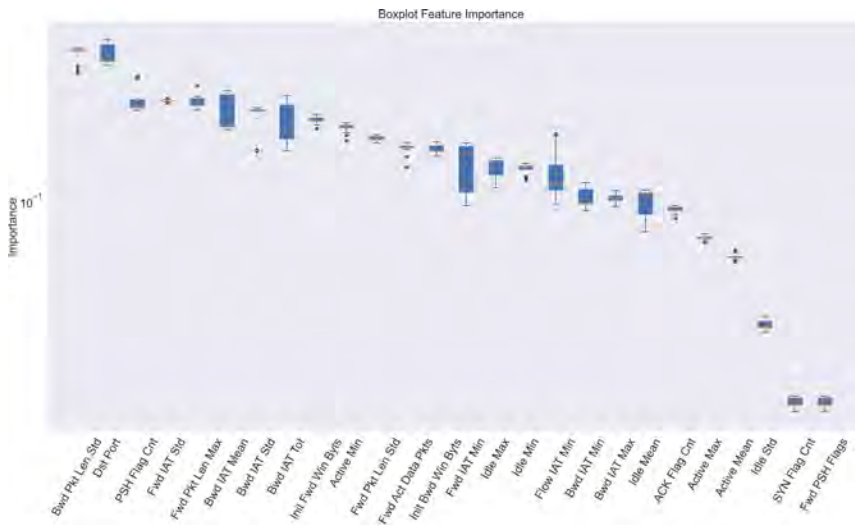

Figure A.8: SVM: Boxplot of permutation importances per feature.

DT findings show that classifier prefers *Fwd Pkt Len Max* – Forward Packet Length Max – by a highly significant margin and suggests that the length of the maximum packet sent to a network contributes to an increase in information gain. The effects of Permutation Importance on **DT** showed that the majority of features are stable, thus suggesting that feature ranking may not be necessary for DT, corresponding to what Reis *et al.* (2019) have observed.

LR findings show that the model prefers *PSH Flag Cnt* – Push flag count – as the most significant feature. *Dst Port*, *Bwd Pkt Len Min*, *Fwd Pkt Len Max* and *Fwd Pkt Len Std* are the remainder of the top five most important features. The variance in most features is stable, but most features are significantly different in importance, indicating that rank is important.

In general, the findings show that feature selection has varying effects on each ML model.

A.4 Model Evaluation

A.4.1 Tuning of Experimental Model Parameters

Figures A.9 through A.13 show validation curves resulting from most influential hyper-parameters for the best models per class for RF, MLP, KNN, DT and LR classifiers.

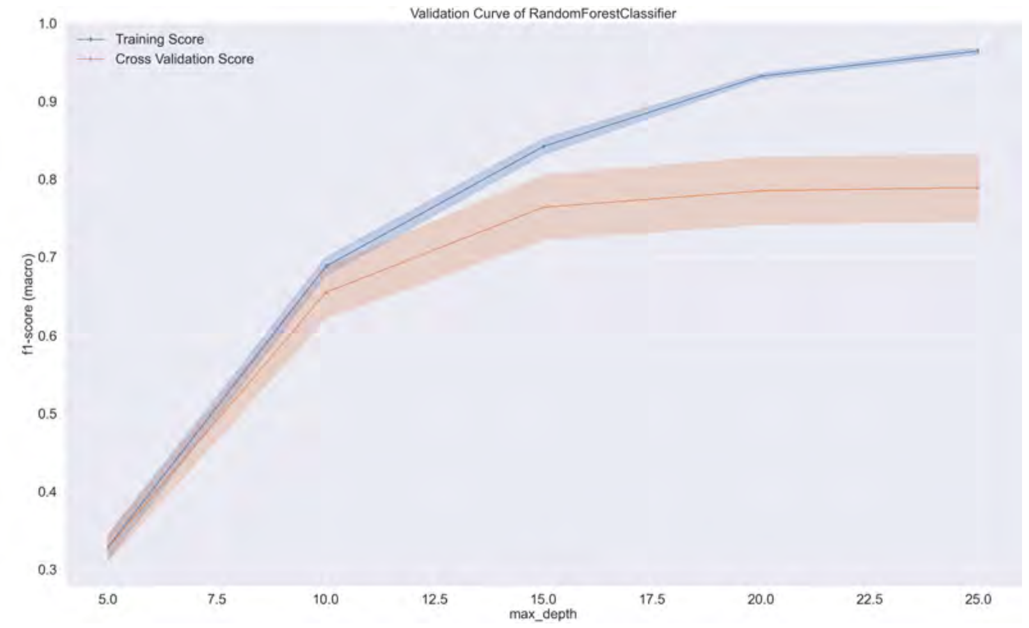

Figure A.9: 26RF-Standard: The most influential parameter's effect on the macro f1-score.

Figure A.10 shows the effects of the most influential parameters that affect the macro f1-score for MLP. The nature of the graph in Figure A.10 suggests that for training scores, multiples of 200 hidden layers may produce a similar f1-score.

A.5 Other Results

Table A.2 shows all results from the CICIDS 2017 dataset and Table A.3 shows all results against the CICIDS 2018 dataset.

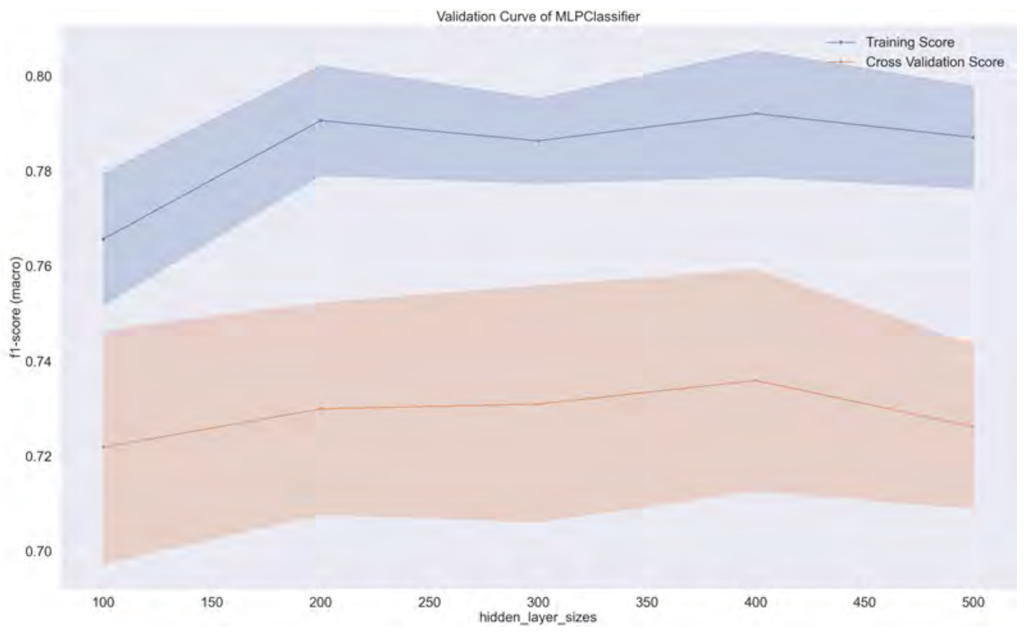

Figure A.10: 26MLP-Quantile: The most influential parameter's effect on the macro f1-score.

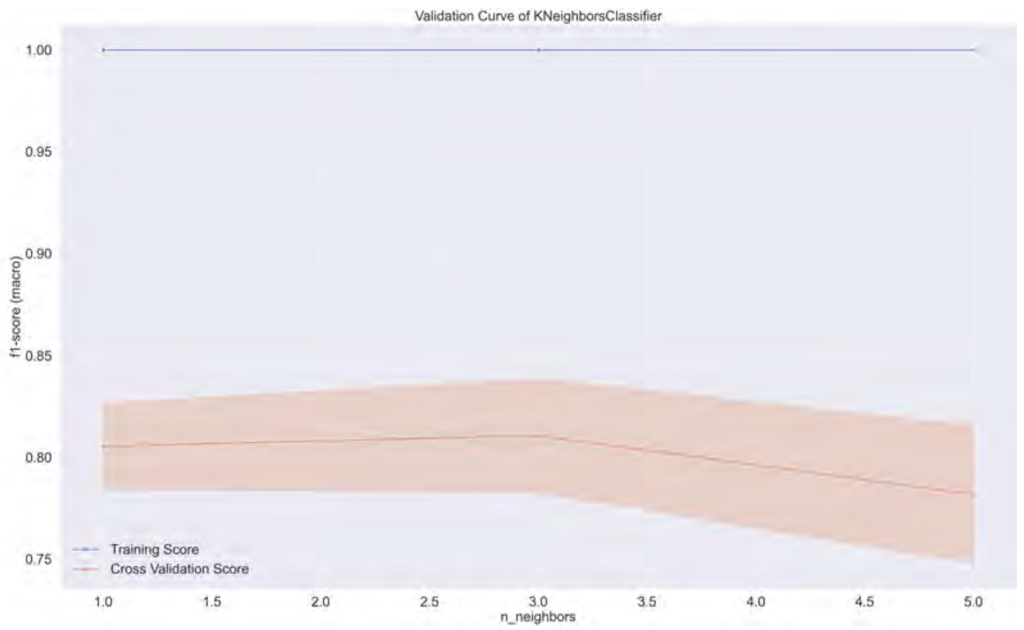

Figure A.11: 26KNN-Quantile: The most influential parameter's effect on the macro f1-score.

Continued on next page

Table A.2 – continued from previous page

Table A.2: Dataset 2017 Modelled

Processing	Classifier	Scaler	Accuracy	Precision	Recall	F1-score	Split
26PI	SVM	Quantile	1	0.82	0.84	0.81	0.9
36PI	SVM	Quantile	1	0.84	0.83	0.81	0.9
26PI	SVM	Power	1	0.84	0.83	0.8	0.7
69PI	SVM	Power	1	0.83	0.8	0.79	0.7
36PI	SVM	Standard	0.98	0.75	0.77	0.72	0.9
26PI	SVM	Standard	0.97	0.79	0.79	0.72	0.7
10PI	DT	Quantile	1	0.69	0.84	0.73	0.9
16PI	DT	Quantile	1	0.72	0.82	0.75	0.9
16PI	DT	Power	1	0.74	0.82	0.76	0.9
16PI	DT	-	1	0.78	0.83	0.76	0.9
26PI	DT	-	1	0.7	0.81	0.73	0.9
36PI	LR	Quantile	0.99	0.68	0.78	0.7	0.9
26PI	LR	Quantile	0.98	0.59	0.78	0.64	0.7
16PI	LR	Power	0.96	0.52	0.69	0.56	0.9
36PI	LR	Power	0.99	0.67	0.79	0.71	0.9
69PI	LR	Standard	0.96	0.58	0.65	0.6	0.9
36PI	LR	Standard	0.94	0.46	0.65	0.52	0.7
36PI	RF	Standard	1	0.86	0.9	0.88	0.5
10PI	RF	Quantile	1	0.88	0.86	0.87	0.7
26PI	RF	Standard	1	0.9	0.86	0.87	0.9
36PI	KNN	Power	1	0.79	0.87	0.82	0.9
10PI	MLP	Quantile	0.99	0.75	0.75	0.74	0.9
69PI	MLP	Quantile	1	0.77	0.87	0.8	0.7
10PI	MLP	Power	0.99	0.69	0.77	0.71	0.9
10PI	MLP	Power	0.99	0.72	0.79	0.72	0.7

Continued on next page

Table A.2 – continued from previous page

Processing	Classifier	Scaler	Accuracy	Precision	Recall	F1-score	Split
36PI	MLP	Standard	0.98	0.7	0.81	0.73	0.9
19PI	MLP	Standard	0.98	0.61	0.78	0.6	0.5
10PI	KNN	Quantile	1	0.77	0.84	0.79	0.9
16PI	KNN	Quantile	1	0.78	0.84	0.81	0.9
10PI	KNN	Power	0.99	0.71	0.84	0.75	0.9
26PCA	RF	Quantile	1	0.84	0.87	0.84	0.3
26PCA	RF	Power	1	0.86	0.88	0.86	0.5
26PCA	RF	Standard	1	0.76	0.85	0.77	0.5
26PCA	SVM	Quantile	1	0.81	0.88	0.82	0.3
26PI	KNN	Standard	0.99	0.74	0.87	0.78	0.9
36PI	KNN	Standard	0.98	0.64	0.89	0.7	0.5
69LDA	KNN	Robust	0.99	0.72	0.78	0.74	0.9
36PI	RF	Power	1	0.9	0.85	0.86	0.9
69LDA	MLP	Robust	0.99	0.68	0.87	0.69	0.3
69LDA	KNN	Power	1	0.72	0.82	0.74	0.9
69LDA	RF	Power	1	0.8	0.91	0.84	0.3
69LDA	MLP	Power	1	0.73	0.82	0.74	0.9
69LDA	KNN	Quantile	1	0.71	0.87	0.75	0.7
16PI	RF	Quantile	1	0.87	0.84	0.85	0.9
69LDA	MLP	Quantile	1	0.73	0.88	0.75	0.5
69LDA	KNN	Standard	0.99	0.67	0.88	0.71	0.5
69LDA	RF	Quantile	1	0.81	0.88	0.83	0.5
69LDA	MLP	Standard	0.99	0.7	0.87	0.74	0.5
26PCA	KNN	Robust	0.99	0.72	0.78	0.74	0.9
69LDA	RF	Robust	1	0.8	0.87	0.82	0.5
26PCA	MLP	Robust	0.99	0.68	0.87	0.69	0.3
26PCA	KNN	Power	1	0.74	0.88	0.77	0.5

Continued on next page

Table A.2 – continued from previous page

Processing	**Classifier**	**Scaler**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Split**
10PI	RF	Power	1	0.87	0.83	0.84	0.9
69PCA	MLP	Power	1	0.75	0.84	0.77	0.7
26PCA	KNN	Quantile	1	0.76	0.9	0.8	0.5
69LDA	RF	Standard	1	0.76	0.87	0.79	0.5
26PCA	MLP	Quantile	1	0.79	0.89	0.8	0.3
26PCA	KNN	Standard	0.99	0.7	0.79	0.74	0.9
26PCA	RF	Robust	1	0.8	0.87	0.82	0.9
26PCA	MLP	Standard	0.98	0.62	0.84	0.67	0.5
26PCA	SVM	Robust	0.98	0.72	0.81	0.68	0.3
69LDA	SVM	Power	1	0.8	0.85	0.8	0.5
69LDA	SVM	Quantile	1	0.83	0.83	0.81	0.7
69LDA	SVM	Standard	0.99	0.7	0.79	0.71	0.3
26PCA	SVM	Robust	0.98	0.72	0.81	0.68	0.3
26PCA	SVM	Power	1	0.84	0.78	0.8	0.9
26PI	DT	Power	0.99	0.76	0.86	0.78	0.7
26PCA	SVM	Standard	0.98	0.69	0.75	0.69	0.9
26PCA	DT	Robust	0.99	0.69	0.84	0.73	0.7
26PCA	LR	Robust	0.93	0.53	0.53	0.49	0.9
69LDA	DT	Power	0.99	0.66	0.83	0.7	0.7
69LDA	LR	Power	0.99	0.59	0.68	0.61	0.9
26PCA	LR	Quantile	0.99	0.69	0.78	0.73	0.9
69LDA	DT	Quantile	0.99	0.7	0.82	0.72	0.7
69LDA	LR	Quantile	0.99	0.6	0.78	0.64	0.9
69LDA	DT	Standard	0.99	0.65	0.84	0.7	0.7
69LDA	LR	Standard	0.95	0.48	0.58	0.51	0.9
26PCA	DT	Robust	0.99	0.69	0.84	0.73	0.7
26PCA	LR	Robust	0.93	0.53	0.53	0.49	0.9

Continued on next page

Table A.2 – continued from previous page

Processing	**Classifier**	**Scaler**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Split**
26PCA	DT	Power	0.99	0.65	0.86	0.71	0.5
26PCA	LR	Power	0.99	0.64	0.77	0.68	0.9
26PCA	DT	Quantile	0.99	0.72	0.77	0.74	0.9
69PI	RNN	Quantile	1	0.74	0.82	0.73	0.9
26PCA	DT	-	0.99	0.7	0.73	0.69	0.9
26PCA	LR	Standard	0.95	0.56	0.64	0.58	0.9

Table A.3: All 2018 Dataset Findings

Modeled?	**Model**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Split**
Yes	Q-69PI-RNN	1	0.74	0.82	0.73	0.9
Yes	Q-26PI-RF	0.78	0.7	0.81	0.72	0.5
Yes	P-69PI-RNN	0.95	0.71	0.78	0.71	0.9
Yes	Q-69PI-RNN	0.94	0.66	0.77	0.69	0.9
Yes	Q-26PI-SVM	0.73	0.66	0.83	0.68	0.5
Yes	P-26PI-RF	0.79	0.66	0.82	0.68	0.5
Yes	26PI-RF	0.78	0.66	0.83	0.68	0.5
Yes	26PCA-RF	0.8	0.66	0.79	0.68	0.5
Yes	P-26PI-SVM	0.74	0.65	0.83	0.67	0.5
Yes	P-26PCA-RF	0.79	0.65	0.8	0.67	0.5
Yes	P-69PCA-MLP	0.74	0.65	0.84	0.67	0.5
Yes	Q-26PCA-RF	0.79	0.64	0.81	0.67	0.5
Yes	Q-26PCA-MLP	0.78	0.65	0.84	0.67	0.5
Yes	Q-26PI-KNN	0.77	0.64	0.81	0.66	0.5
Yes	S-26PI-KNN	0.77	0.64	0.82	0.66	0.5
Yes	Q-26PI-MLP	0.76	0.63	0.83	0.65	0.5

Continued on next page

Table A.3 – continued from previous page

Modeled?	**Model**	**Accuracy**	**Precision**	**Recall**	**F1-score**	**Split**
Yes	P-26PI-MLP	0.74	0.63	0.82	0.65	0.5
Yes	P-26PCA-KNN	0.78	0.62	0.8	0.65	0.5
Yes	Q-26PCA-KNN	0.79	0.62	0.8	0.65	0.5
Yes	26PI-DT	0.77	0.62	0.8	0.64	0.5
Yes	S-69PI-RNN	0.79	0.61	0.81	0.63	0.7
Yes	P-26PI-KNN	0.77	0.61	0.81	0.63	0.5
Yes	Q-26PI-DT	0.78	0.59	0.83	0.63	0.5
Yes	S-26PCA-KNN	0.79	0.6	0.8	0.63	0.5
Yes	S-26PCA-MLP	0.74	0.6	0.81	0.62	0.5
Yes	P-26PCA-DT	0.78	0.6	0.77	0.62	0.5
Yes	P-26PCA-LR	0.7	0.6	0.8	0.62	0.5
Yes	Q-26PCA-DT	0.79	0.59	0.78	0.62	0.5
Yes	26PCA-DT	0.78	0.59	0.78	0.62	0.5
Yes	S-26PI-SVM	0.66	0.62	0.77	0.61	0.5
Yes	S-26PI-MLP	0.69	0.6	0.79	0.61	0.5
Yes	P-26PI-DT	0.77	0.59	0.82	0.61	0.5
Yes	Q-26PCA-LR	0.68	0.58	0.77	0.61	0.5
Yes	S-26PCA-LR	0.63	0.57	0.73	0.57	0.5
Yes	P-26PI-LR	0.6	0.5	0.72	0.53	0.5
Yes	S-26PI-LR	0.58	0.47	0.68	0.5	0.5
Yes	Q-26PI-LR	0.57	0.47	0.66	0.49	0.5
No	P-26PI-MLP	0.77	0.25	0.28	0.22	0.9
No	Q-26PI-SVM	0.89	0.31	0.19	0.2	0.9
No	Q-26PI-KNN	0.65	0.25	0.22	0.19	0.9
No	P-26PI-KNN	0.74	0.22	0.25	0.17	0.9
No	Q-26PI-RF	0.76	0.26	0.16	0.17	0.9
No	Q-26PI-MLP	0.81	0.18	0.21	0.17	0.9

Continued on next page

Table A.3 – continued from previous page

Modeled?	Model	Accuracy	Precision	Recall	F1-score	Split
No	Q-26PI-LR	0.74	0.21	0.22	0.15	0.9
No	Q-26PI-LR	0.74	0.13	0.15	0.13	0.9
No	26PI-DT	0.44	0.15	0.17	0.13	0.9
No	S-26PI-MLP	0.7	0.18	0.14	0.12	0.9
No	P-26PI-RF	0.82	0.13	0.1	0.1	0.9
No	S-26PI-LR	0.48	0.12	0.1	0.1	0.9
No	S-26PI-KNN	0.75	0.09	0.12	0.09	0.9
No	P-26PI-DT	0.74	0.09	0.1	0.08	0.9
No	Q-26PI-DT	0.56	0.08	0.1	0.08	0.9
No	26PI-RF	0.8	0.07	0.07	0.07	0.9
No	P-26PI-SVM	0.87	0.07	0.08	0.07	0.9
No	S-26PI-SVM	0.87	0.07	0.08	0.07	0.9
No	P-26PCA-RF	0.85	0.07	0.08	0.07	0.9
No	Q-26PCA-RF 0.87	0.07	0.08	0.07	0.9	
No	Q-26PCA-MLP	0.69	0.07	0.07	0.07	0.9
No	S-26PCA-KNN	0.77	0.07	0.07	0.07	0.9
No	26PCA-RF	0.87	0.07	0.08	0.07	0.9
No	P-26PCA-SVM	0.87	0.07	0.08	0.07	0.9
No	Q-26PCA-SVM	0.87	0.07	0.08	0.07	0.9
No	S-26PCA-SVM	0.52	0.08	0.06	0.07	0.9
No	26PCA-DT	0.58	0.07	0.11	0.07	0.9
No	P-26PCA-KNN	0.39	0.08	0.08	0.06	0.9
No	P-26PCA-MLP	0.67	0.07	0.06	0.06	0.9
No	Q-26PCA-KNN	0.56	0.07	0.05	0.06	0.9
No	P-26PCA-DT	0.7	0.07	0.06	0.06	0.9
No	Q-26PCA-DT	0.6	0.06	0.09	0.06	0.9
No	S-26PCA-MLP	0.47	0.06	0.06	0.05	0.9

Continued on next page

Table A.3 – continued from previous page

Modeled?	Model	Accuracy	Precision	Recall	F1-score	Split
No	P-26PCA-LR	0.41	0.06	0.05	0.05	0.9
No	Q-26PCA-LR	0.49	0.07	0.05	0.05	0.9
No	S-26PCA-LR	0.43	0.07	0.05	0.05	0.9

A.6 Detailed classifier performance per class

The following section shows the precision, recall and f1-score per class for each of the best models evaluated against the CICIDS 2017 dataset.

Class	Precision	Recall	f1-score
Benign	1	1	1
Bot	0.7	0.89	0.78
DDoS	1	1	1
DoS GoldenEye	0.98	0.99	0.99
DoS Hulk	1	1	1
DoS Slowhttptest	0.98	0.99	0.99
DoS slowloris	0.99	0.99	0.99
FTP-Patator	1	1	1
Heartbleed	1	1	1
Infiltration	0.94	0.52	0.67
PortScan	0.99	1	0.99
SSH-Patator	0.99	0.99	0.99
Web Brute Force	0.72	0.68	0.7
Web Sql Injection	0.71	0.29	0.42
Web XSS	0.38	0.42	0.4

Table A.4: 26PI-RF: Results for the best performing model with limited data.

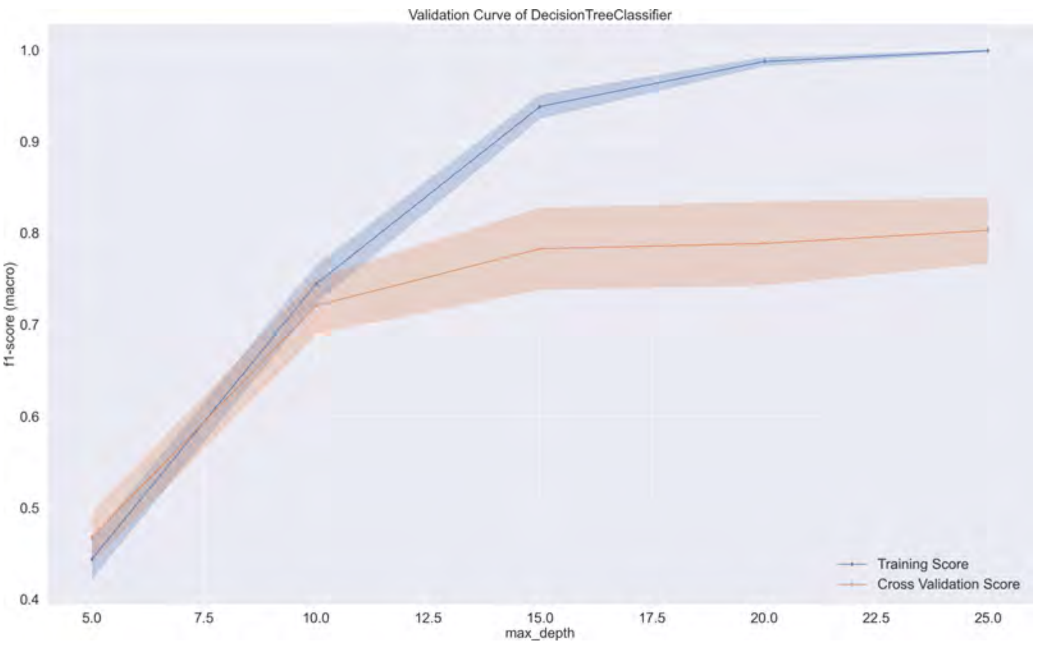

Figure A.12: 16DT-Standard: The most influential parameter's effect on the macro f1-score.

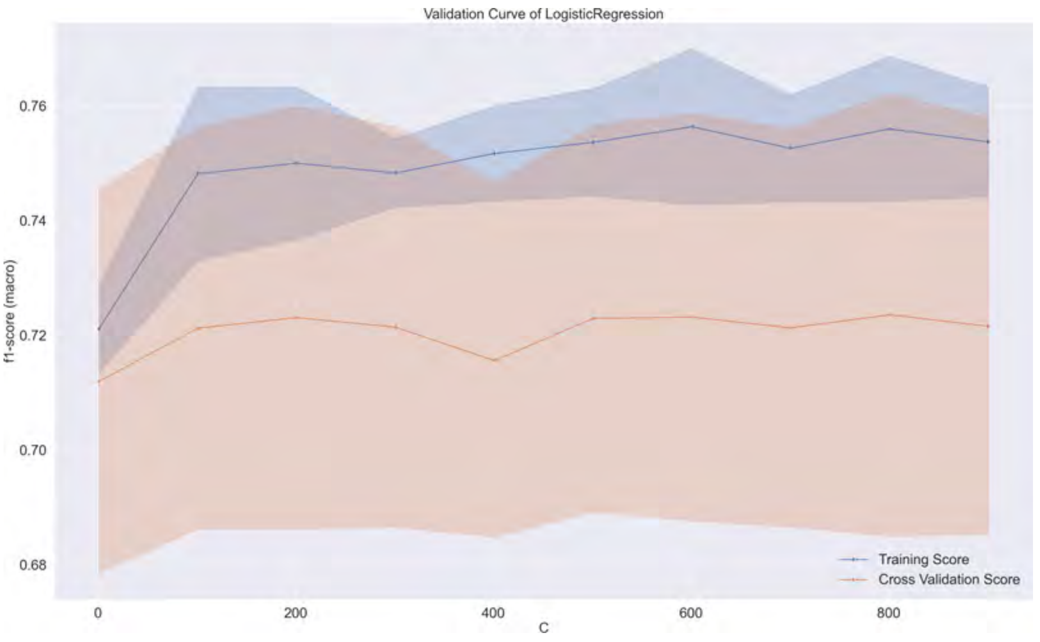

Figure A.13: 36LR-Power: The most influential parameter's effect on the macro f1-score.

Class	**Precision**	**Recall**	**f1-score**
Benign	1	1	1
Bot	0.48	0.94	0.63
DDoS	1	1	1
DoS GoldenEye	0.98	0.99	0.98
DoS Hulk	0.99	0.99	0.99
DoS Slowhttptest	0.97	0.98	0.98
DoS slowloris	0.99	0.98	0.98
FTP-Patator	1	0.99	1
Heartbleed	1	0.89	0.94
Infiltration	0.71	0.3	0.43
PortScan	0.99	1	0.99
SSH-Patator	0.97	0.98	0.98
Web Brute Force	0.69	0.59	0.64
Web Sql Injection	0.12	0.35	0.18
Web XSS	0.4	0.66	0.5

Table A.5: 26PI-SVM: Results for the best performing model with limited data.

Class	**Precision**	**Recall**	**f1-score**
Benign	1	1	1
Bot	0.54	0.89	0.67
DDoS	1	1	1
DoS GoldenEye	0.93	0.99	0.96
DoS Hulk	0.99	1	0.99
DoS Slowhttptest	0.97	0.96	0.97
DoS slowloris	0.95	0.99	0.97
FTP-Patator	0.99	1	0.99
Heartbleed	1	0.89	0.94
Infiltration	1	0.67	0.8
PortScan	0.99	1	0.99
SSH-Patator	0.97	0.99	0.98
Web Brute Force	0.63	0.98	0.77
Web Sql Injection	0.02	0.35	0.04
Web XSS	0.41	0.04	0.08

Table A.6: 36PI-MLP: Results for the best performing model with limited data.

A.7 Class performance per model against CICIDS 2017

A.7.1 RF Results

Figure A.14 shows a ROC curve providing the class representation and how the model performs in relation to FPR and TPR. The AUC for S-26PI-RF shows that the discrimi-

Class	**Precision**	**Recall**	**f1-score**
Benign	1	1	1
Bot	0.49	0.9	0.63
DDoS	1	1	1
DoS GoldenEye	0.92	1	0.95
DoS Hulk	0.99	1	1
DoS Slowhttptest	0.95	0.98	0.97
DoS slowloris	0.97	0.98	0.98
FTP-Patator	0.99	1	1
Heartbleed	1	1	1
Infiltration	0.52	0.48	0.5
PortScan	0.99	1	0.99
SSH-Patator	0.98	0.99	0.99
Web Brute Force	0.68	0.65	0.66
Web Sql Injection	0.15	0.41	0.22
Web XSS	0.35	0.45	0.4

Table A.7: 26PI-KNN: Results for the best performing model with limited data.

Class	**Precision**	**Recall**	**f1-score**
Benign	1	1	1
Bot	0.58	0.87	0.7
DDoS	1	1	1
DoS GoldenEye	0.94	0.98	0.96
DoS Hulk	0.99	1	1
DoS Slowhttptest	0.93	0.97	0.95
DoS slowloris	0.97	0.98	0.98
FTP-Patator	0.99	0.99	0.99
Heartbleed	1	0.56	0.71
Infiltration	0.11	0.36	0.17
PortScan	0.98	1	0.99
SSH-Patator	0.99	0.99	0.99
Web Brute Force	0.64	0.94	0.76
Web Sql Injection	0.07	0.76	0.13
Web XSS	0.5	0.02	0.03

Table A.8: 16PI-DT: Results for the best performing model with limited data.

native ability of the model tests across all classes is good except for *Web SQL Injection*. This curve is not visible on the line because of the FPR close to 1.

The ROC curve shows the model performing consistently high for *FTP-Patator*, *Web Brute Force* and *DoS Hulk* tests above the rest of the classes. With a TPR greater than

Class	**Precision**	**Recall**	**f1-score**
Benign	1	1	1
Bot	0.58	0.87	0.7
DDoS	0.99	1	1
DoS GoldenEye	0.93	0.98	0.95
DoS Hulk	0.99	1	0.99
DoS Slowhttptest	0.87	0.97	0.92
DoS slowloris	0.98	0.97	0.97
FTP-Patator	0.98	0.99	0.99
Heartbleed	0.05	0.78	0.09
Infiltration	0.57	0.52	0.54
PortScan	0.98	1	0.99
SSH-Patator	0.93	0.98	0.95
Web Brute Force	0.42	0.64	0.51
Web Sql Injection	0.17	0.18	0.17
Web XSS	0.31	0.37	0.34

Table A.9: 36PI-LR: Results for the best performing model with limited data.

Class	**Precision**	**Recall**	**f1-score**
Benign	1	1	1
Bot	0.59	0.59	0.59
DDoS	1	1	1
DoS GoldenEye	0.9	0.98	0.94
DoS Hulk	0.98	1	0.99
DoS Slowhttptest	0.9	0.98	0.94
DoS slowloris	0.98	0.95	0.96
FTP-Patator	0.99	1	1
Heartbleed	1	0.9	0.95
Infiltration	0.08	0.72	0.14
PortScan	0.99	1	0.99
SSH-Patator	0.89	0.99	0.94
Web Brute Force	0.67	0.16	0.25
Web Sql Injection	0.01	0.11	0.02
Web XSS	0.18	0.92	0.3

Table A.10: 69PI-RNN: Results for the best performing model with limited data.

0.999 and FPR less than 0.05, the classes *Benign*, *Bot*, *DDoS*, *DoS GoldenEye*, *SSH-Patator*, *Heartbleed* and *Port Scan* all yield good confidence levels for a NIDS model.

DoS Slowloris and *DoS Slowhttptest* also achieve an acceptable TPR confidence level between 0.995 and 0.997 and a FPR less than 0.05. However, *Web XSS*, *Infiltration* and

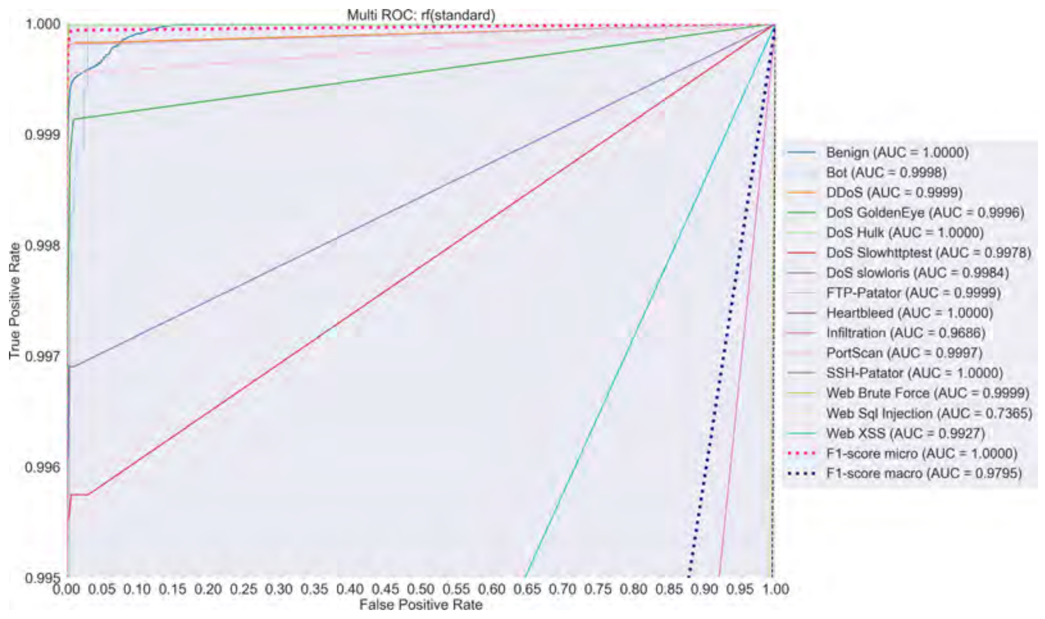

Figure A.14: 26PI-RF: ROC curve of the best performing RF model on limited data.

Web Sql Injection have a high FPR which places the classes at an unacceptable confidence level.

Upon inspection of individual classes, it was found that *Infiltration* and *Web Sql Injection* are mostly incorrectly classified as *Benign*.

A.7.2 KNN Results

Figure A.15 shows the ROC curve analysis for Q-26PI-KNN, where it is evident that the primary classes can significantly perform better than the underrepresented instances. The AUC for all classes is at an acceptable value over 0.95 except for *Web XSS* and *Web SQL Injection* that can be classified as failed tests and, as such, are unreliable.

DDoS, *DoS Hulk* and *FTP-Patator* have exceptionally better reliability for this model with an increase of between 0 and 0.05 FPR only at an interval of 0.999 and 1 TPR. *Port Scan* has a slightly lower performance than that of the top-performing classes with a TPR between 0.998 and 0.999 and FPR less than 0.05, thus making it reliable.

Generally, the models perform weaker across all classes when compared to the RF models

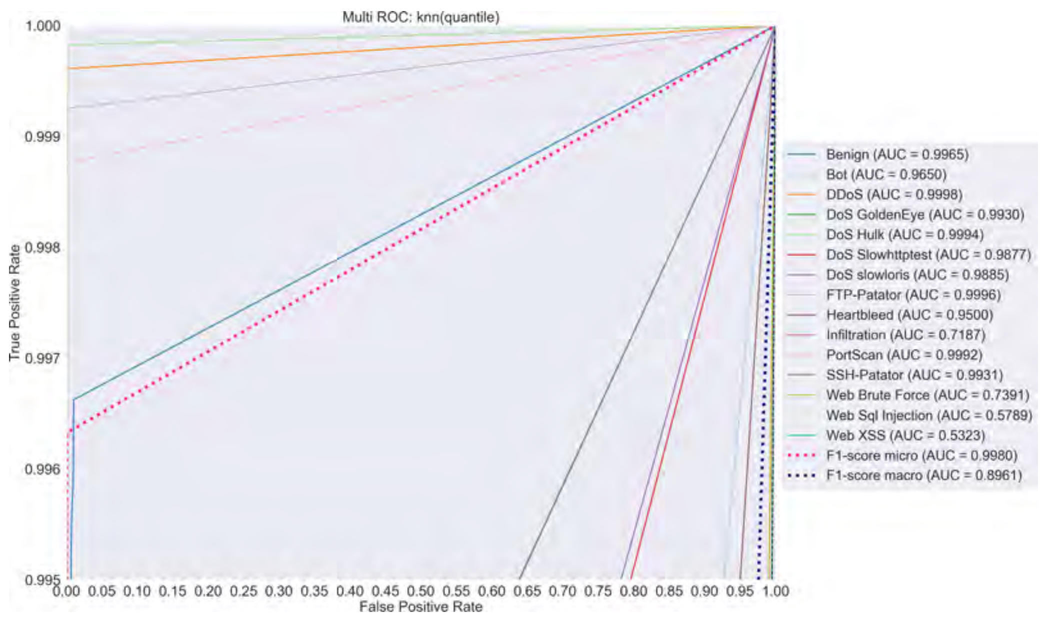

Figure A.15: Q-26PI-KNN: ROC curve of the best performing KNN model on limited data.

discussed. The finding aligns with Figure A.6 26PI-KNN box plot outliers and variance representation, suggesting there is minimal feature correlation.

A.7.3 MLP Results

Figure A.16 shows that the Q-36PI-MLP model achieves a high AUC of over 0.90 across all classes, suggesting that the model tests are better when compared to those from the RF and KNN models, which had exceptions. Generally, the Q-36PI-MLP ROC curve shows low reliability for classifying *Heartbleed* and *Infiltration* as the FPR is above 0.5.

The model performs best in classifying *FTP-Patator*, *DoS Hulk* and *DoS GoldenEye* with the FPR being very close to 0 and TPR nearly at 1. *DDoS*, *Port Scan*, *SSH-Patator* and *Benign* and *Web Brute Force* also perform fairly well with an ideal evaluation position of TPR greater than 0.999 and FPR between 0 and 0.05. However, minimal instances of incorrect classification with possible overlaps can be expected due to the FPR.

Additional classes such as *Bot* and *DoS Slowhttptest* also perform well with acceptable points of TPR greater than 0.999 and FPR between 0.05 and 0.10. *DoS Slowloris* and

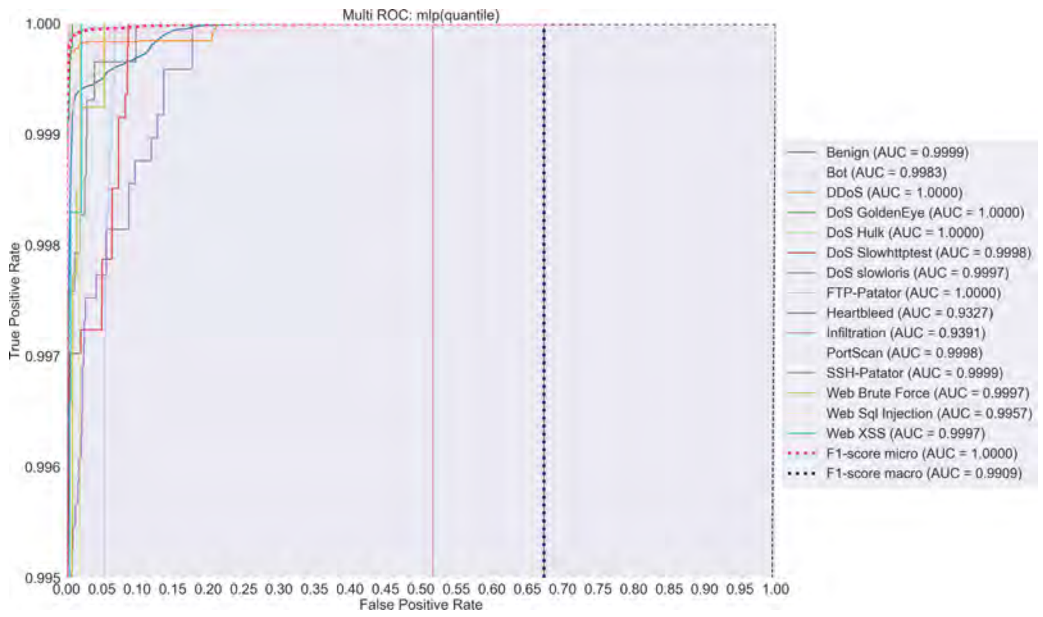

Figure A.16: Q-36PI-MLP: ROC curve of the best performing MLP model on limited data.

Web XSS show high FPR of over 0.10 or low TPR of under 0.999 which lowers their reliability for this model. Upon inspection of individual classes, it was found that *Infiltration*, *Web Sql Injection* and *Web XSS* are consistently the worst performers being incorrectly classified as *Benign*.

A.7.4 SVM Results

Figure A.17 shows a ROC curve for the Q-26PI-SVM where *Infiltration* is the worst performing test of an AUC less than 0.80. The model shows a high AUC of over 0.95 across all the other classes, suggesting the most reliable tests.

The ROC curve shows that the model performs best in classifying *DDoS*, *DoS GoldenEye* and *Port Scan* with an FPR less than 0.05 and TPR between 0.999 and 1. *DoS Hulk* and *FTP-Patator* also perform fairly well with a TPR greater than 0.998 and FPR under 0.05. FPR for the *Bot* class is consistently below 0.05 with a minimal overlap between 0.05 and 0.10. When TPR is above 0.997, the FPR starts to increase thus degrading the model. *DoS Slowhttptest*, *DoS Slowloris* and *Web Brute Force* perform very low, inconsistently

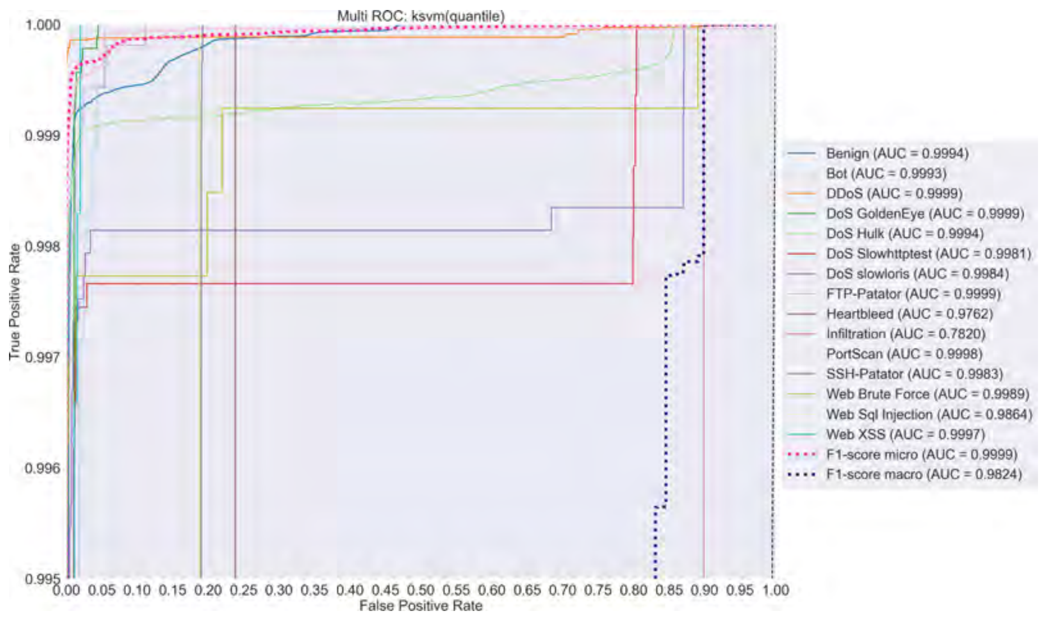

Figure A.17: Q-26PI-SVM: ROC curve of the best performing SVM model on limited data.

and unreliable to use. Minimal instances of incorrect classification with possible overlaps can be expected due to the FPR.

Additional classes such as *Bot* and *DoS Slowhttptest* also perform well with a TPR greater than 0.999 and FPR between 0.05 and 0.10. Further analysis confirms that Q-26PI-SVM is mostly unreliable for *Web XSS* and *Web SQL Injection*. The results show the model often incorrectly classifies *Infiltration* as *Benign*. The misclassifications follow the same trend observed for RF and MLP, albeit to a far greater extent.

A.7.5 DT Results

Figure A.18 shows a ROC curve for the 16PI-DT where *Infiltration* is the worst performing test of an AUC less that 0.80. The model shows a high AUC of over 0.90 across all the other classes suggesting the tests are much more reliable.

Generally, DT shows poor performance across all classes evaluated when compared against the other models discussed. The results also show that classes with minor representation in the dataset during the training phase have poor performance when evaluated in the model.

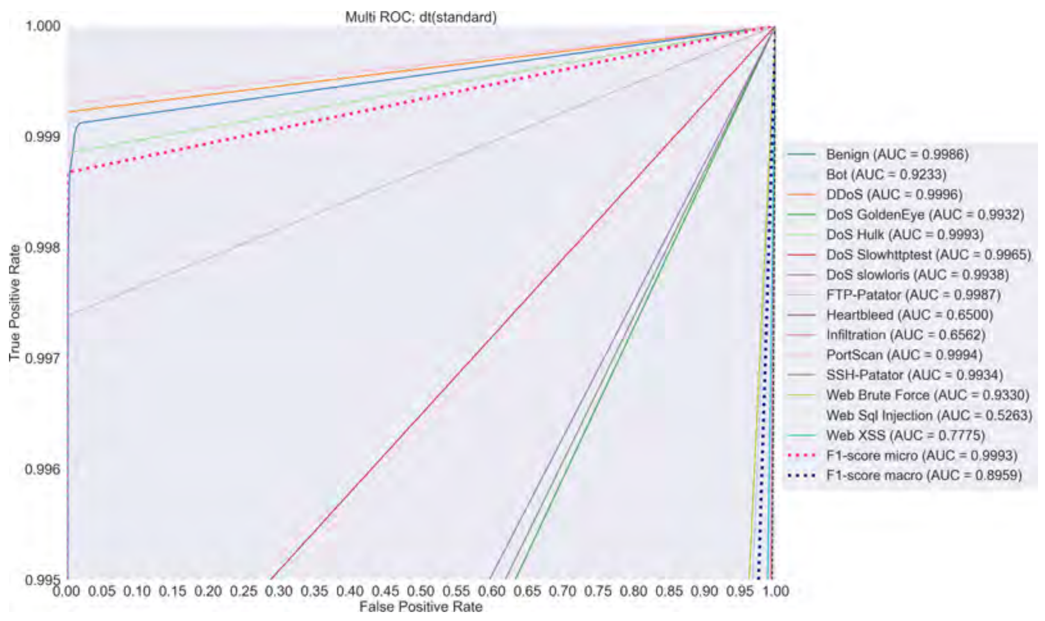

Figure A.18: 16PI-DT: ROC curve of the best performing DT model on limited data.

Upon further testing, it was discovered that adding more training samples improved the model to be on par with other classifiers like RF. However, this finding is impractical to compare, as there are very few test data samples remaining when performing testing this way.

The AUC for each class on Figure A.18 for S-16PI-DT shows the ability of the model tests across all classes are in an acceptable range with the exception of *Web Sql Injection*, *Heartbleed*, *Infiltration* and *Web XSS*. The ROC curve shows the model performing highly for *Port Scan*, *DDoS* and *Benign* tests above the rest of the classes with a TFP greater than 0.999 TPR and FPR less than 0.005.

The high TPR (range of 0.998 to 0.999) for classes *DoS Hulk* and *FTP-Patator* with an FPR less than 0.05 place the model as more reliable for these classes similar to S-26PI-RF. However, the rest of the classes the model performs poorly with a TPR less than 0.995. *Web XSS*, *Infiltration*, *Web Brute Force*, *Web Sql Injection* and *Bot* have a high FPR that places the classes at a more unacceptable confidence level.

Generally, for the performance of the models against class representation, the S-16PI-DT model shows bias towards majority classes in the dataset except for the *Bot* class.

A.7.6 LR Results

Figure A.19 shows the AUC for Q-26PCA-LR where all the tests using the AUC are reliable with the areas above 0.95 with the exception of *Benign*, *Bot* and *DoS GoldenEye* that have fair areas between 0.90 and 0.95.

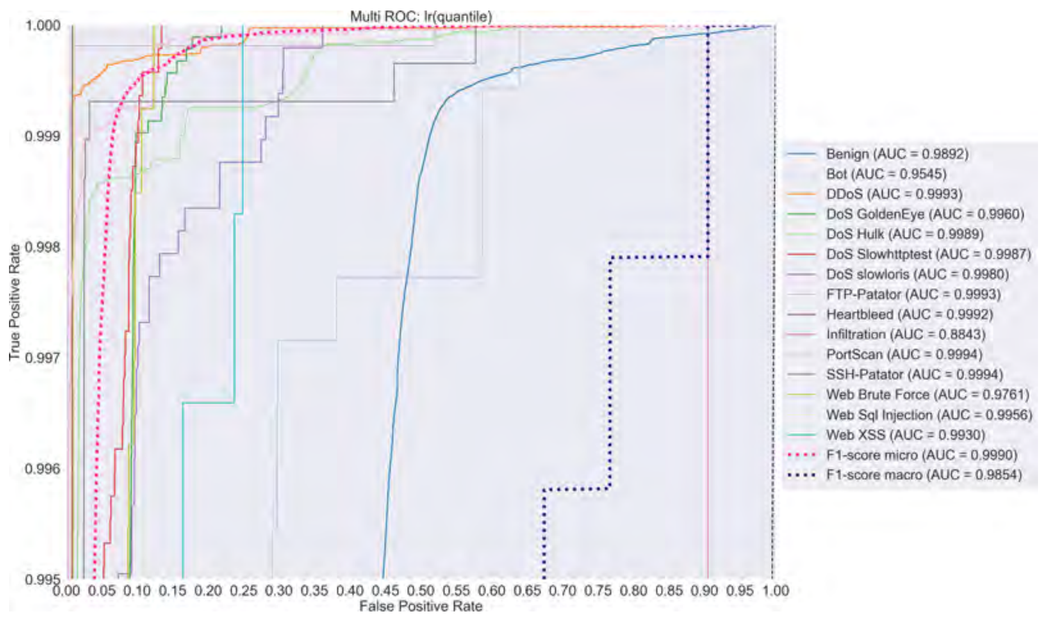

Figure A.19: Q-26PCA-LR: ROC curve of the best performing LR model on limited data.

The ROC curve in Figure A.19 shows *DDoS* as the best performing class for the model with a TPR greater than 0.999 and an FPR less than 0.05. However, this best performance is much lower than the other models presented. *Port Scan* also performs well with an FPR lower than 0.05, but the TPR is also lower than most models. *Web Brute Force* performs on average with a FPR consistently less than 0.10. The rest of the classes perform poorly for the model, with the FPR being greater than 0.10, with the worst being *Infiltration* and *Bot* classes that have an FPR greater than 0.50.

A.7.7 RNN Results

Figure A.20 shows a ROC curve for the Q-69PI-RNN used to identify and evaluate how each class performs. An AUC greater than 0.95 is shown across the classes with the exception of *Infiltration* and *Web SQL Injection* that had 0.9476 and 0.8834, respectively. The result positions the model with fairly reliable tests across all classes.

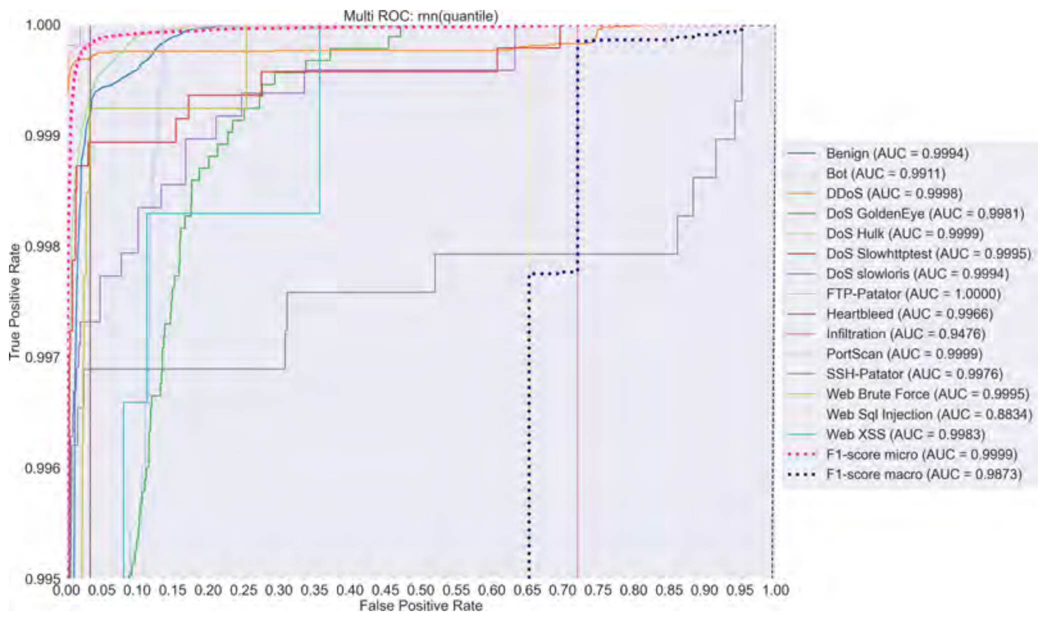

Figure A.20: Q-69PI-RNN: ROC curve of the best performing RF model on limited data.

FTP-Patator, *Port Scan* and *DDoS* are the best performing classes against the Q-69PI-RNN model with a TPR greater than 0.999 and FPR less than 0.05. Although there is a lower TRP range from 0.998 to 0.999 and FPR less than 0.05, the model performs well for *DoS Hulk*, *Benign*, *DoS Slowhttptest*, *Web Brute Force* and *Heartbleed*. The rest of the classes have a higher FPR between 0.05 and 0.10 with the exception of *Web Sql Injection* and *Infiltration* significantly under perform with an FPR greater than 0.5.

Similar to MLP, further analysis into the model shows that five classes are perfectly predicted, namely; *Benign*, *DDoS*, *DoS Hulk*, *FTP-Patator* and *Port Scan*. *Web SQL Injection* is never correctly classified and for 47% of the prediction attempts, it is incorrectly classified as *Benign*. 93% of the prediction attempts *Web XSS* have led to an incorrect classification as *Web Brute Force*. *Bot* and *Infiltration* are incorrectly classified and Benign for 63% and 66% respectively.

A.7.8 More Training Data

As discussed in the proposed system in Chapter 3, each model is evaluated with three different train split ratios: 0.1, 0.3 and 0.5 of each dataset. The split demonstrates that the proposed methodology enables classifiers, including the deep learning RNN, to require substantially less training data. Table A.11 shows how each model performs when presented with more training data.

Model	**Split**	**Accuracy (2017)**	**f1-score (2017)**
26PI-RF	0.1	1.00	0.87
26PI-RF	0.3	1.00	0.87
26PI-RF	0.5	1.00	0.90
Q-26PI-KNN	0.1	1.00	0.82
Q-26PI-KNN	0.3	1.00	0.77
Q-26PI-KNN	0.5	1.00	0.79
Q-36PI-MLP	0.1	1.00	0.81
Q-36PI-MLP	0.3	1.00	0.78
Q-36PI-MLP	0.5	1.00	0.78
Q-26PI-SVM	0.1	1.00	0.81
Q-26PI-SVM	0.3	1.00	0.79
Q-26PI-SVM	0.5	1.00	0.79
16PI-DT	0.1	1.00	0.76
16PI-DT	0.3	1.00	0.72
16PI-DT	0.5	0.99	0.75
Q-26PCA-LR	0.1	0.99	0.73
Q-26PCA-LR	0.3	0.98	0.66
Q-26PCA-LR	0.5	0.98	0.60
Q-69PI-RNN	0.1	1.00	0.73
Q-69PI-RNN	0.3	0.99	0.63
Q-69PI-RNN	0.5	0.99	0.62

Table A.11: Effects of train and test data split.

In general, the benefits of adding more test data were not consistently observed except for RF based models. The other classifiers either experienced degradation of performance or alternating performance.