UNSUPERVISED FEATURE SELECTION FOR ANOMALY-BASED NETWORK INTRUSION DETECTION USING CLUSTER VALIDITY INDICES

by

Tyrone Naidoo (208500170)

BScEng Computer Engineering

A dissertation submitted in fulfilment of the requirements for the degree of Master of Science in Engineering (Computer Engineering)

in the

College of Agriculture, Engineering and Science,

University of Kwa-Zulu Natal

Supervisor:

Prof. J. R. Tapamo

Co-Supervisor:

Mr. A. M. McDonald

April 2016

Preface

The research discussed in this dissertation was carried out in the College of Agriculture, Engineering and Science of the University of Kwa-Zulu Natal, Durban from January 2014 until November 2015 by Tyrone Naidoo under the supervision of Professor Jules-Raymond Tapamo and co-supervision of Mr. Andre Martin McDonald.

As the candidate's supervisor, I, Jules-Raymond Tapamo, agree / do not agree to the submission of this dissertation.

Signed:____________________ Date:____________________

As the candidate's co-supervisor, I, Andre Martin McDonald, agree / do not agree to the submission of this dissertation.

Signed:____________________ Date:____________________

I, Tyrone Naidoo, hereby declare that all the materials incorporated in this dissertation are my own original work, except where acknowledgement is made by name or in the form of a reference. The work contained herein has not been submitted in any form for any degree or diploma to any other institution.

Signed:____________________ Date:____________________

Declaration 1 - Plagiarism

I, ________________________________, declare that

1. The research reported in this dissertation, except where otherwise indicated, is my original research.
2. This dissertation has not been submitted for any degree or examination at any other university.
3. This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
   (a) Their words have been re-written but the general information attributed to them has been referenced,
   (b) Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.
5. This dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References sections.

Signed:________________________________

Declaration 2 - Publications

I, ________________________________, declare that the following publications were produced as a result of the work reported in this dissertation.

1. T. Naidoo, A. M. McDonald, and J. R. Tapamo, "Feature selection for anomaly-based network intrusion detection using cluster validity indices." in *Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2015*, 6-9 Sept 2015, Arabella, Hermanus, Western Cape, South Africa, pp. 145-150, Sept 2015.

I declare that the experimental work contained in these publications was performed as a joint effort by me and Mr. A. M. McDonald, and the writing of each publication was done by me with assistance from Mr. A. M. McDonald and Prof. J. R. Tapamo, and the work contained in these publications has been reproduced in this dissertation.

Signed:________________________________

Acknowledgements

I would like to thank my supervisors Prof. J. R. Tapamo and Mr. A. M. McDonald for their support, dedication, patience and endless insight. You have made a valuable impact to my research, and to the completion of this dissertation. A feat that would not have been possible without your assistance. I would also like to acknowledge the CSIR Modelling and Digital Sciences, Information Security (MDSIS) research group for your support and advice which aided in the successful completion of this dissertation. Lastly I would like to thank my family and friends, for your emotional support and endless motivation, which drove me to persevere to the end.

Unsupervised Feature Selection for Anomaly-Based Network Intrusion Detection Using Cluster Validity Indices

Abstract

In recent years, there has been a rapid increase in Internet usage, which has in turn led to a rise in malicious network activity. Network Intrusion Detection Systems (NIDS) are tools that monitor network traffic with the purpose of rapidly and accurately detecting malicious activity. These systems provide a time window for responding to emerging threats and attacks aimed at exploiting vulnerabilities that arise from issues such as misconfigured firewalls and outdated software.

Anomaly-based network intrusion detection systems construct a profile of legitimate or normal traffic patterns using machine learning techniques, and monitor network traffic for deviations from the profile, which are subsequently classified as threats or intrusions. Due to the richness of information contained in network traffic, it is possible to define large feature vectors from network packets. This often leads to redundant or irrelevant features being used in network intrusion detection systems, which typically reduces the detection performance of the system.

The purpose of feature selection is to remove unnecessary or redundant features in a feature space, thereby improving the performance of learning algorithms and as a result the classification accuracy. Previous approaches have performed feature selection via optimization techniques, using the classification accuracy of the NIDS on a subset of the data as an objective function. While this approach has been shown to improve the performance of the system, it is unrealistic to assume that labelled training data is available in operational networks, which precludes the use of classification accuracy as an objective function in a practical system.

This research proposes a method for feature selection in network intrusion detection that does not require any access to labelled data. The algorithm uses normalized cluster validity indices as an objective function that is optimized over the search space of candidate feature subsets via a genetic algorithm. Feature subsets produced by the algorithm are shown to improve the classification performance of an anomaly–based network intrusion detection system over the NSL-KDD dataset. Despite not requiring access to labelled data, the classification performance of the proposed system approaches that of effective feature subsets that were derived using labelled training data.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

BN	Bayesian Network
CART	Classification and Regression Tree
cat	Categories
CH	Calinski-Harabasz
CSTNET	Chinese Science and Technology NETwork
CVI	Cluster Validity Index
DARPA	Defence Advanced Research Projects Agency
DB	Davies-Bouldin
DBOD	Distance-Based Outlier Detection
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DDoS	Distributed Denial-of-Service
DNS	Domain Name System
DoS	Denial-of-Service
EAC	Evidence Accumulation Clustering
ELM	Extreme Learning Machines
EM	Expectation Maximization
FFSA	Forward Feature Selection Algorithm
FN	False Negative
FP	False Positive
FRM	Feature Removal Method
FTP	File Transfer Protocol
GA	Genetic Algorithm
GFR	Gradual Feature Removal

GMM	Gaussian Mixture Model
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IT	Information Theoretic
kDBOD	k-means with Distance-Based Outlier Detection
KDD	Knowledge Discovery and Data mining
kNN	k Nearest Neighbour
LCFS	Linear Correlation-based Feature Selection
LDBSCAN	Local Density-Based Spatial Clustering of Applications with Noise
LOF	Local Outlier Factor
LSSVM	Least Squares Support Vector Machine
MAC	Media Access Control
MCC	Matthew's Correlation Coefficient
MIT	Massachusetts Institute of Technology
MMIFS	Modified Mutual Information-based Feature Selection
MOGA	Multi-Objective Genetic Algorithm
NADO	Network Anomaly Detection using Outliers
NB	Naïve Bayes
NDB	Normalized Davies-Bouldin
NID	Network Intrusion Detection
NIDS	Network Intrusion Detection System
NSGA	Non-dominated Sorting Genetic Algorithm
ORC	Outlier Removal Clustering
PCA	Principal Component Analysis
R2L	Root-to-Local
RBF	Radial Basis Function
RFC	Request For Comments
ROC	Receiver Operating Characteristic
RODD	Reference-based Outlier Detection for large Datasets

SFM	Sole Feature Method
SMR	Sparse Matrix Regression
SOGA	Single-Objective Genetic Algorithm
SPADE	Statistical Packet Anomaly Detection Engine
SSC	Sub-Space Clustering
SVM	Support Vector Machine
TASVM	Triangle Area-based Support Vector Machine
TCM-kNN	Transductive Confidence Machines for *k* Nearest Neighbour
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
U2R	User-to-Root
UDP	User Datagram Protocol
UNIDS	Unsupervised Network Intrusion Detection System
VPN	Virtual Private Network
WEKA	Waikato Environment for Knowledge Analysis
XSS	Cross Site Script

Symbols

$\mathbf{C} = \{\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_K}\}$	Feature vectors representing cluster centres or medoids
$Cat_{j,f}$	Category j of categorical feature f
$Cat'_{j,f}$	Encoded category j of categorical feature f
C_R	Number of repetitions to convergence, of a clustering algorithm using different initial parameter values
$d_{i,j}$	Measure of separation between clusters i and j
d_{max}	References the candidate feature subset $F_s(d_{max})$ that minimizes the NDB index value
d_r	Defines the radius surrounding a data sample
$\mathbf{e}$	Expectation scores for each candidate feature subset
$\mathbf{F_{card}}$	Cardinality of each candidate feature subset
$\mathbf{F_{cross}}$	Candidate feature subsets produced from crossover
$\mathbf{F_{elite}}$	Candidate feature subsets produced from elitism
$\mathbf{F_{mut}}$	Candidate feature subsets produced from mutation
$\mathbf{F_s} = \{F_s(d),\ d = 1, 2, \ldots, M\}$	Collection of M candidate feature subsets representing a generation of the GA
$\mathbf{F_{sel,cross}}$	Candidate feature subsets selected for crossover
$\mathbf{F_{sel,elite}}$	Candidate feature subsets selected for elitism
$\mathbf{F_{sel,mut}}$	Candidate feature subsets selected for mutation
$\mathbf{front}$	Assignments of each candidate feature subset to a Pareto front
g_i	Measure of compactness of cluster i
$\mathbf{H} = \{\mathbf{H_1}, \mathbf{H_2}, \ldots, \mathbf{H_K}\}$	All data samples assigned to cluster k

$\mathbf{I}$	Identity matrix
k	References a particular cluster or component
K	Total number of clusters or components
$\mathbf{L} = \{l(\mathbf{x_a}) \mid a = 1, 2, \ldots, N\}$	Cluster assignments for each data sample in $\mathbf{X}$
M	Number of candidate feature subsets in a generation of the GA (i.e. the population size of the GA)
N	Total number of data samples in $\mathbf{X}$
NDB_{avg}	Average normalized DB index value
N_F	Dimensionality of full feature set
N_c	Number of candidate feature subsets in generation $t+1$ obtained via crossover
N_{cat}	Number of categories in a categorical feature f
N_d	Number of candidate feature subsets that dominate a particular feature subset in the objective space
N_e	Number of candidate feature subsets in generation $t+1$ obtained via elitism
N_{front}	Number of Pareto fronts
N_g	Maximum number of generations
N_m	Number of candidate feature subsets in generation $t+1$ obtained via mutation
N_n	Number of nearest neighbours that defines the neighbourhood of a data sample
N_o	Number of objective functions
N_{out}	Number of data samples to be identified as outliers
N_r	Number of clusters labelled as legitimate
$\mathbf{n_f} = \{n_1, n_2, \ldots, n_{N_{front}}\}$	Number of candidate feature subsets assigned to each Pareto front
p_m	Mutation probability for a feature
$R_{i,j}$	Measure of similarity between clusters i and j
r_e	Percentage of candidates to produce via elitism for generation $t+1$

r_c	Percentage of candidates to produce via crossover for generation $t+1$
r_{out}	Percentage of data samples to be identified as outliers
$\mathbf{s}$	Segments / intervals associated with each candidate feature subset for stochastic uniform selection
T_{log}	Threshold for log-likelihood
t	Current generation number
$\mathbf{W} = \{\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_{N_o}}\}$	Collection of values for each objective function for each candidate feature subset
$\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$	Collection of data samples or feature vectors
$\mathbf{X_{distance}}$	Crowding distances for a collection of data samples
$\mathbf{X_o}$	Collection of data samples that are labelled as outliers
$x_{i,f}$	Value of feature f of data sample i
$x'_{i,f}$	Normalized value of feature f of data sample i
$\hat{\mathbf{Z}} = \{z_1, z_2, \ldots, z_K\}$	Latent unobserved variable
$\boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_K}\}$	Feature vectors representing the mean of each of the K Gaussian components over the feature space
μ_f	Mean of the values in feature f
$\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_K\}$	Mixing coefficients of each of the K Gaussian components
$\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_K}\}$	Covariance matrix of each of the K Gaussian components
σ_f	Standard deviation of the values in feature f

Chapter 1

Introduction

This chapter provides an overview of the current state of security for networked computer environments, which includes their weaknesses and security threats that these environments are susceptible to. Mechanisms that are used to protect against such threats, which includes network intrusion detection systems, and the problem that is intended to be solved are discussed. The proposed research that will be carried out is outlined in this chapter as well.

1.1 Background

In modern day society, networked computer systems play a significant role in the completion of everyday tasks and activities. These tasks include daily financial transactions, often dealing with exorbitant sums of money, the execution of vital government services or the day to day social interactions between billions of people. These activities are typically communicated over the Internet, which is a universal network of millions of interconnected computer systems and smaller networks [2].

Networked computer systems have a large number of application areas, with vast amounts of valuable and confidential information being transmitted between them. This has led to networked computer systems becoming an attractive target for cybercrime. Cybercrime can be described as any act involving a computer and a network that aims to intentionally harm the reputation, or cause physical, mental, emotional or financial harm or loss to an individual or group of individuals [8]. These acts are frequently carried out over telecommunication networks such as the Internet, through the illegitimate use of, or access to network services such as chat rooms, notice boards, file servers, web servers, mail servers, network routers, etc, or mobile phone network services such as SMS and MMS [8]. Typical cybercrimes can include the theft of large sums of money, identity theft, fraud and copyright infringements. The total cost of cybercrime based on 24 major countries around the world, including South Africa, was 113 Billion US Dollars in 2013 [9]. Russia, China and South Africa were the top three in the number of victims of cybercrime, with 85%, 77% and 73% of all adults surveyed, respectively.

Cybercrime often involves some form of network intrusion. Network intrusions constitute any unauthorised activity on a computer network that leads to the loss of confidentiality and integrity of data transmitted or accessed via a network, the denial of network resources, or the unauthorised use of network resources. An example of network intrusions that may concern a system administrator include; the illegitimate usage of existing user accounts by unauthorised individuals, which provides unauthorised access to confidential system or user information [10]. This may lead to:

1. The unauthorised alteration of a user's confidential files or information, or
2. the unauthorised editing of system information in network components, such as altering router tables to deny a user access to the network (denial of services).

In general, network vulnerabilities can be defined as flaws in the design, implementation and management of a networked system and its security mechanisms. Their existence creates weaknesses in networked systems as these vulnerabilities can be exploited by various types of network attacks, leading to a loss in confidentiality and integrity of information, the denial of resources, or unauthorised use of resources [2]. An example of a vulnerability involves the use of outdated software which may have weaknesses in its code that were fixed in newer versions. Due to a lack of updating and patching, these weaknesses create vulnerabilities that an attacker may exploit. Bhattacharyya et al. [2] identify several vulnerabilities that weakens the security of networked computer systems.

- Network configuration vulnerabilities — These vulnerabilities occur due to networks and security mechanisms being improperly set up and managed by an inexperienced administrator, or due to human error. These vulnerabilities are caused by [2]:
    1. incorrect configuration of security equipment, such as firewalls,
    2. unprotected password transmission, and
    3. use of the same passwords or weak passwords for extended periods of time.
- Network perimeter vulnerabilities — This occurs due to a lack of network perimeter security and appropriate access control mechanisms. If the perimeter of the network is not well defined, the network may be susceptible to unauthorized access by illegitimate users [2].
- Communication vulnerabilities — A system can be rendered vulnerable if the communication between devices on a network is not properly secured. Examples of communication vulnerabilities are, the data transferred between network devices are not properly encrypted, or devices are not properly authenticated prior to data transmission. Well documented security protocols need to be adhered to in order to ensure that connections between nodes are secure. Additionally, proper authentication measures and data integrity checks need to be followed to ensure that users, data and devices are legitimate [2].
- Wireless communication vulnerabilities — These occur due to a lack in proper user authentication methods, as well as poor encryption policies for the data transmission over the wireless network [2]. Additionally, the use of weak passwords increases the risk of unauthorized access to a networked system.

A network attack is a sequence of operations or actions executed on a networked system that leads to the occurrence of a network intrusion. Network attacks are typically accomplished by exploiting some vulnerability in the networked system. A taxonomy of network attacks is provided in what follows [2]. Examples of these attacks are provided in table 1.1.

1. Infection — The aim of these attacks is to infect a computer system by tampering with computer software, or installing malicious programs on the target system.
2. Exploding — These attacks add flaws to the target system with the intent of overflowing the system, thereby rendering it unusable.
3. Probe — This involves the use of various tools to gather vital information about the target system or network. Typically the primary goal is to identify networked systems and services that possess vulnerabilities that can be exploited.
4. Cheat — These are attacks that involve the use of fake identities in order to access private information on the target system.
5. Traverse — This is an attempt to access a protected target system by trying a list of commonly occurring "keys" until the correct "key" is found which allows access to the target system. Typically, the "keys" are user login passwords.
6. Concurrency — The aim of these attacks is to flood a system that provides a service with a vast amount of requests. This exhausts the capacity of the system to respond to additional requests by consuming all available system memory or bandwidth. The result of this attack is to render the system or service unusable and unavailable to legitimate requests.
7. Others — These attacks make direct use of system weaknesses, as may be present in outdated software, in order to infect the target system.

Table 1.1 shows several subcategories of attacks that networked systems are susceptible to.

TABLE 1.1: *Taxonomy of network attacks (from [2]).*

Main Category	**Subcategory**
Infection	Viruses, Worms, Trojans
Exploding	Buffer overflow
Probe	Sniffing, port mapping security scanning
Cheat	IP spoofing, MAC spoofing, DNS spoofing, session hijacking, XSS (Cross Site Script) attacks, hidden area operation, and input parameter cheating
Traverse	Brute force, dictionary attacks, doorknob attacks
Concurrency	Flooding, DDoS (Distributed Denial-of-Service)

1.1.1 Conventional security mechanisms

The RFC2828 standard defines a security service as, *"a processing or communication service that is provided by a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms"* [4]. A list of five broad categories of security services, as defined by the X.800 standard is provided in what follows [4].

1. Authentication — A service that guarantees that a communicating source is the source that it claims to be.

2. Access control — This service prevents the use of computing resources by unauthorised entities. Access control determines who is allowed to access a resource, the degree to which they may use a resource, and the conditions under which a resource may be accessed.

3. Data confidentiality — A service that aims to protect confidential data from disclosure to unauthorised entities.

4. Data integrity — This service guarantees that data received from an authorised communicating source is received without being modified or altered during the transmission process.

5. Nonrepudiation — A service that aims to protect against entities denying participation in a communication that said entity was apart of.

Several security mechanisms that are used to implement security services include [4].

- Encipherment — This mechanism involves the application of mathematical algorithms to data, with the aim of encoding the data in such a way that it is incomprehensible when viewed using traditional means. The application of a decoding algorithm is required to revert the data back to its original state.
- Digital signature — A digital signature allows a recipient to verify that the data received is in fact from the originating source, and confirms that the data has not been modified or altered. A digital signature is a segment of data added to the unit of data intended for transmission, or a specific form of cryptographic transformation applied to the data prior to transmission.
- Access control — There are multiple mechanisms that can be used to control access to resources, such as user login procedures for computer systems.
- Data integrity — Several security mechanisms may be applied to assure that data is unaltered during transmission.
- Authentication exchange — This mechanism uses information exchange to verify the identity of a communicating entity.
- Traffic padding — Extra information in the form of bits are added to network packet data in order to encode the data such that an unauthorised entity is unable to comprehend the data.
- Routing control — A mechanism that selects secure routes for data transmission or alters the route of data currently being transmitted when a breach of security is detected.
- Notarization — This involves the use of a third-party which manages the data transmission between two entities.

Table 1.2 maps each security mechanism to the security service that they provide.

TABLE 1.2: *Security mechanisms and the security services that they provide (from [4]).*

Security Service	**Security Mechanism**							
	Encipherment	**Digital signature**	**Access control**	**Data integrity**	**Authentication exchange**	**Traffic padding**	**Routing control**	**Notarization**
Authentication	Y	Y			Y			
Access control			Y					
Data confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y	Y			Y

Examples of conventional security mechanisms that are used to provide authentication and access control security services include firewalls, anti-virus software and Virtual Private Networks (VPNs).

Firewalls, deployed at the perimeter of a network, contain a set of rules that describe the types of network activity (malicious or legitimate) that are allowed to traverse the firewall [4]. Firewalls employ a form of access control for data transmitted to a network using either blacklisting or whitelisting. Blacklisting allows all network activity to pass into the network or networked system except the network activity mentioned in the rule set, whereas whitelisting blocks all network activity from accessing the network or networked system except for the network activities that are specified in the rule set.

Anti-virus software typically monitors a host for activity that is already known to be malicious activity. The malicious activity is captured in the form of a signature, where the anti-virus software compares the signature to the observed activity of the monitored system. An example of malicious activity that anti-virus software may detect involves, the execution of a malicious executable file on the windows operating system which may initiate a sequence of events that results in the collection of confidential information for an attacker to exploit. This sequence of events may be known to the anti-virus software if the signature is available as an intrusion, thus, the anti-virus software will stop these events in mid-sequence.

VPNs are typically used by businesses to securely connect several offices. Communication between offices and between offices and the Internet are encrypted allowing for safer communication.

Although these security mechanisms do prove effective in protecting against casual intrusion attempts, it has been demonstrated that these preventative measures are no longer sufficient [2, 11]. Attackers have found means to create fake identities or use existing identities in order to masquerade as legitimate users on a network. User login details can be acquired through brute force attacks, or through social engineering. Firewalls and VPNs can often be misconfigured due to inexperience which creates vulnerabilities, allowing traffic from malicious sources to penetrate the firewall or VPN. The lack of an updated anti-virus software implies that the anti-virus will not recognize a sequence of events that describes new or novel types of malicious activities. Though VPNs attempt to safeguard networks from malicious content found on the Internet using encryption, they are susceptible to malicious content that may be found on the devices (such as laptops) of legitimate users that connect to the VPN. Once connected to the VPN, the malicious content on a users device may have access to the VPNs resources and confidential information.

1.1.2 Intrusion detection systems

The inability of conventional network security measures to reliably safeguard networks and networked systems against network intrusions, led to the creation of an additional layer of security. This layer of security, known as intrusion detection and prevention, was introduced by James P. Anderson in the early 1980s [12–15]. Mukherjee et al. [10] state that, "*The goal of intrusion detection is to identify, preferably in real-time, unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators.*"

Intrusion detection functionality is carried out by Intrusion Detection Systems (IDSs), which consists of hardware and software elements for automatically detecting intrusions. IDSs monitor systems for illegitimate usage patterns which are typically different from legitimate usage patterns. The detection of intrusions can be used to prevent further harm caused by an intrusion, for instance, denying access to unauthorised users, effectively limits the scope of malicious activity and harm.

It is important to note that IDSs are not intended to replace conventional security mechanisms, but rather act as an additional line of defence, with the intention of strengthening the security of information communication systems [1]. For instance, IDSs can be used to detect various types of malicious activities that a firewall may inadvertently allow to pass through into the network. In addition to this, an IDS can act as a tool for post mortem analysis, namely to determine the extent of the harm caused by an attack, and to potentially track down the source of the attack [1, 11, 13, 16, 17].

Intrusion detection systems can be classified as either host-based or network-based. Host-based IDSs monitor the events on a single computer system or host, where events may be operating system based or application based. For instance, it can detect whether log files, security policies, or other important information on that host has been modified in any way by an attacker. Host-based IDSs may impose high processing overheads on hosts, depending on the resources available to the host, however, these systems are traditionally simpler to implement and manage, as they are contained within a single host.

A network-based IDS monitors traffic patterns between network routers, network servers or network switches, which are ingress and egress points of a network [18]. The network-based IDS does not have access to information that is specific to individual hosts, which does not allow for the granular protection that the host-based IDS has. However, the network-based IDS is able to detect attacks that the host-based IDS may not be able to. For instance, probing attacks that span several hosts are not detectable on a single host, but would be detected as an intrusion over the entire network. Network-based systems are often more complex to implement and manage owing to the need to monitor multiple hosts, but they are able to detect attacks that span multiple hosts [19, 20]. Both host-based and network-based IDSs are complementary to each other in the sense that both are used to detect intrusions in different contexts.

IDSs may also be classified based on the detection approach that they utilize. Typically, the two main approaches are misuse-based detection and anomaly-based detection [1].

1.1.2.1 Misuse-based detection

Misuse-based detection approaches compares network traffic features to predefined patterns or signatures of known intrusions. Signatures represent a set of rules that describes the sequence or set of network activities that characterise an intrusion. Misuse-based approaches typically have lower false positive percentages owing to the fact that it detects intrusions that are known. The shortcoming is that misuse-based systems are unable to detect unknown intrusions or variants of existing intrusions. [1, 10, 11, 13, 14, 16, 20–22].

1.1.2.2 Anomaly-based detection

Anomaly-based detection approaches assumes that the feature values of malicious network traffic is vastly different from that of legitimate network traffic. Anomaly-based approaches involves the construction of models that represents the expected behaviour of legitimate network traffic. Incoming network traffic is compared against these models and any deviation from these models is regarded as an anomalous activity, which may indicate that malicious network traffic has penetrated the system being monitored. Anomaly-based detection systems are generally able to detect novel intrusions, however, they often have high false positive percentages [1, 10, 11, 13, 14, 16, 20–22].

While all types of IDSs have their own strengths and weaknesses, this research focusses on anomaly-based Network Intrusion Detection Systems (NIDS), owing to their ability to detect novel intrusions. Anomaly-based network intrusion detection systems typically follow a basic structure, which is provided in figure 1.1.

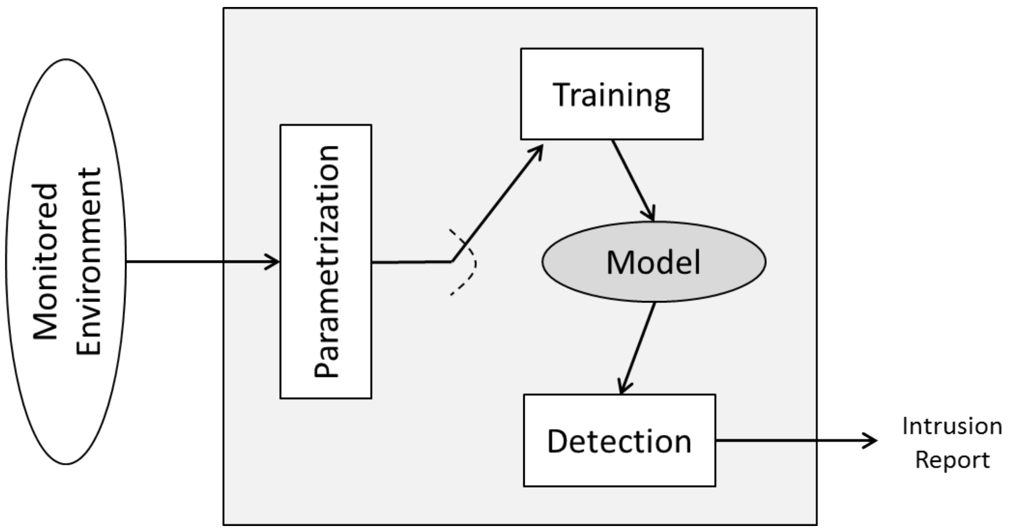


FIGURE 1.1: *Anomaly-based NIDS structure (from [1]).*

Parameterization involves representing the data observed in the monitored environment in a form that is suitable for the application of analysis techniques to the data. The training stage subsequently characterises and models the legitimate behaviour of the monitored system. Network traffic observed on the monitored system is parameterized and subsequently compared against the model in the detection stage, any deviation from this model results in the anomaly-based NIDS raising an alarm [1].

Data mining involves the analysis of data with the aim of uncovering valuable information which may be transformed into knowledge [23]. Anomaly-based network intrusion detection systems essentially perform data mining tasks on network data in order to gain knowledge of the behaviour of a system. In anomaly-based NIDS, there exists a variety of techniques that may be used to accomplish such data mining tasks, for example, machine learning techniques have been used previously in this regard [1]. Machine learning is a branch of computer programming that focusses on developing computer algorithms that build models based on input data which allows them to make predictions or decisions on new input data. Machine learning methods typically belong to one of two categories: supervised machine learning or unsupervised machine learning. The former uses labelled data to learn a general rule on how to map input data to output data; the latter does not require labelled data, it learns patterns from the input data itself, using these learned patterns to make predictions based on new input data [2].

Machine learning based anomaly-based NIDS typically constructs models of legitimate network traffic based on patterns observed in the network packets of network traffic. These patterns are observed through the analysis of specific features and their values, embedded in network packets. These feature values characterise or define various types of network activity occurring on a networked computer system.

1.2 Research problem statement

The effectiveness of various techniques used to implement network intrusion detection systems rely heavily on the set of features used. A feature set should provide a good distinction among the classes present in the data, allowing for quick and accurate classification of these classes [2]. Network traffic is rich with contextual information and as a result large feature vectors can be constructed. While larger feature vectors may hold more information, it has been shown, particularly for the KDD Cup 1999 dataset [24], and the NSL-KDD dataset [25], that features within a feature vector can be irrelevant in detecting certain types of attacks or that certain features are redundant [7, 26–34].

The consequence of performing network intrusion detection on redundant and irrelevant features is that it leads to a reduction in classification accuracy in certain cases [26, 31, 35], as well as introduces a high computational complexity which increases the detection delay. Irrelevant features do not contribute to the detection of certain types of attacks, as they are unable to capture significant characteristics that are required for the detection of these attacks [36, 37]. Redundant features do not add any new knowledge with regards to the detection of attacks [36]. A study performed in [7] revealed that several features of a well-known network intrusion detection dataset provided little to no information on any of the classes present in the dataset.

A candidate solution to this problem is feature selection, which is a procedure commonly used to remove redundant or irrelevant features that may be present within feature vectors [2, 23, 38]. Feature selection has been applied successfully in various application areas [7, 26–33]. Applying feature selection techniques to the data prior to analysis by a NIDS has the potential to improve the performance of network intrusion detection systems. It has already been demonstrated that a reduced set of more significant features can lead to an improved detection accuracy and false positive percentage as compared to the original set of features when detecting certain attacks [7, 26–33]. Additionally, the reduced feature set may consume fewer resources and take less time to process. Higher dimensional data makes the detection of attacks more difficult due to complex relationships between features, this increases the processing time, which increases the detection delay.

Feature selection, as applied in the context of network intrusion detection, typically involves the use of supervised methods that require access to labelled network data [30–33, 35, 39, 40]. Labelled data consists of network traffic samples (feature vectors) that have been previously classified by experts in the field as legitimate or associated with malicious activity. While these implementations demonstrate the effectiveness of feature selection, they are impractical in the sense that labelled network data is not available in practical environments, and difficult to construct without the aid of experts in the field. Constructing features from network packet data is a non-trivial task; it involves an in-depth analysis of network packet headers and the content of network packet payloads, which is a time consuming task [38]. Given the lack of availability of labelled network data, supervised methods cannot always be used to perform feature selection, thus an unsupervised feature selection algorithm which does not rely on the availability of labelled data would provide a solution to this problem. To the best of the author's knowledge, an unsupervised feature selection algorithm using cluster validity indices, has not yet been implemented for use in network intrusion detection.

This research introduces an unsupervised feature selection algorithm for use in network intrusion detection. The algorithm makes use of unsupervised machine learning techniques. It does not require labelled data in its execution, making the technique practical, and suitable for deployment in an operational environment in which labelled data is unavailable. This research focusses specifically on utilizing clustering algorithms, cluster validity indices, and evolutionary algorithms in the implementation of the proposed feature selection algorithm.

1.3 Research objectives

The objectives of this research are.

- To investigate and compare the performance, in terms of true positive and false positive percentages, of several clustering algorithms in the context of network intrusion detection using the NSL-KDD dataset [25].
- To compare the performance, in terms of true positive and false positive percentages, of performing network intrusion detection using several candidate feature subsets of the KDD Cup 1999 dataset [24] and the NSL-KDD dataset [25], that were proposed in the literature [31, 35].

- To design and implement a feature selection algorithm for network intrusion detection that does not rely on labelled network data, and to compare the overall anomaly detection system to that of other anomaly detection systems that use supervised feature selection algorithms.

1.4 Delineations and limitations

This section defines and limits the scope of the investigations performed in this research.

- This study is limited to the use of the NSL-KDD dataset [25]. This dataset has original candidate features that are already defined. Constructing network traffic related features involves an in-depth analysis of network packet headers and payloads, which requires significant domain knowledge and is time consuming. Thus, feature construction is outside the scope of this research.
- The use of the NSL-KDD dataset limits the study to the investigation of 41 standard network features.
- Feature selection is considered in this research, whereas feature construction and feature extraction are not.
- Unsupervised anomaly detection techniques are considered, specifically: the k-means, k-medoids, expectation-maximization, and distance-based outlier detection algorithms.
- Only the Davies-Bouldin cluster validation index is considered as a metric in the feature selection algorithm.
- Only genetic algorithms are considered for the purpose of optimisation.
- Performance is measured based on true positive and false positive percentages using both training and testing sets.

1.5 Motivation

This section motivates the need for the work performed in this research by providing a brief overview of the current work that has been done with regards to feature selection in network intrusion detection. This is provided in table 1.3.

TABLE 1.3: *Techniques used for feature selection.*

Reference	**Class**	**Technique**
[26]	Filter	Correlation and mutual information
[27]	Filter	Bayesian networks and classification and regression trees
[7, 29]	Filter	Information gain
[28]	Filter	Degree of dependency
[35]	Filter	Information gain and correlation
[30]	Filter	Correlation
[31]	Wrapper	Support Vector Machine (SVM) with Matthews correlation coefficient
[32]	Wrapper	Genetic Algorithm (GA) and SVM with true and false positive percentages
[33]	Wrapper	Bayesian network with classification accuracy

Table 1.3 shows that the works that implement either filter-based or wrapper-based feature selection algorithms in the context of network intrusion detection, requires the use of labelled data, which is not always available in practical environments. Creating a labelled network dataset is a non-trivial task. It requires the collection of vast amounts of raw network data, which then requires a further in-depth analysis of network packets and network packet payloads. Unsupervised feature selection in the context of network intrusion detection would not be constrained by the need for labelled network data.

1.6 Contribution

This section highlights the contributions made by this research.

- This research provides a comparison between several clustering algorithms used to perform classification in the context of network intrusion detection on the NSL-KDD dataset [25].

- This research provides a comparison of the performance achieved when performing classification on the NSL-KDD dataset [25] over several feature subsets.
- In this research the author designs and implements a feature selection algorithm for network intrusion detection that does not rely on labelled network data, and provides a comparison between the performance obtained when using feature subsets obtained through the proposed feature selection algorithm and other supervised feature selection algorithms.

1.7 Definitions of terms and concepts

Cybercrime: Cybercrime is defined as an act that is intended to cause harm to an individual, group of individuals or an organisation through the illegitimate use of, or access to network services [8]. These acts may negatively affect the victims reputation, or may cause physical, mental, emotional or financial harm to the victim. Cybercrime may be detrimental to a nation's security or financial health. The consequence of cybercrime can include the theft of large sums of money, the theft of vital information, rendering of services unusable, or copyright infringements.

Network intrusion: A network intrusion constitutes any unauthorised activity on a computer network that causes the loss of confidentiality and integrity of data, the denial of network resources, or the unauthorised use of network resources [10].

Network attacks: A network attack is a sequence of operations executed on a network that leads to the occurrence of a network intrusion. Network attacks are typically accomplished by exploiting some vulnerability in the system [2].

Network vulnerabilities: Network vulnerabilities can be described as flaws in the design, implementation and management of a network. Their existence creates weaknesses in networked systems as these vulnerabilities can be exploited by various types of network attacks [2].

Network anomalies: Any network activity that is considered as being out of the ordinary when compared to expected network activity. Anomalies may be associated with legitimate or malicious network activities [2].

Malicious network activity: Features of network traffic may be used to characterize the patterns of network activity, which may be associated with legitimate or malicious activity. Malicious network activity includes those traffic patterns that are associated with network attacks [41].

Network event or activity: In this research, a network event is described as any change in network feature values such as the change in traffic volume, change in IP addresses and service ports [1].

1.8 Thesis overview

This dissertation is structured as follows; chapter 2 consists of a review of the work that has been performed in the anomaly-based network intrusion detection field, and a review of feature selection in the context of network intrusion detection systems. Chapter 3 introduces an unsupervised anomaly-based classifier for network intrusion detection and presents the experimental results obtained from the comparison of several clustering algorithms, as well as several feature subsets using the proposed classifier. Chapter 4 presents an unsupervised cluster validity-based feature selection algorithm, and provides the experimental results obtained from a comparison of the proposed feature selection algorithm to feature selection algorithms found in the literature. Chapter 5 presents the conclusions that are drawn for the work performed in this research and Chapter 6 provides future work that is to be conducted.

Chapter 2

Literature Review

This chapter provides an overview of network intrusion detection systems, as well as a review of literature related to anomaly-based network intrusion detection using unsupervised machine learning techniques. A review of literature related to the application of feature selection in network intrusion detection is also provided.

2.1 Introduction

NIDS are typically classified as anomaly-based or misuse-based systems. In recent years, research efforts focus on the improvement of anomaly-based NIDS, as they show promise in detecting novel or unobserved intrusions [1]. Anomaly-based NIDS have been implemented using machine learning techniques, which are divided into supervised and unsupervised methods. Supervised methods require labelled data, whereas unsupervised methods do not [2]. Both supervised and unsupervised methods have been used to implement anomaly-based NIDS [42]. In general, it is unrealistic to assume that labelled data is available in practical network environments. Many researchers have overcome this constraint by focussing on using unsupervised methods in the implementation of anomaly-based NIDS [43–51]. Specifically, researchers have used clustering and outlier mining techniques to implement anomaly-based NIDS. Clustering techniques typically involves the use of a clustering algorithm such as k–means to cluster network data, prior to labelling the data in an unsupervised manner [43–46]. Outlier mining techniques typically use distance or density measures to identify unusual instances of network traffic, which are subsequently labelled as intrusions [48–51]. Both of these unsupervised techniques have been demonstrated to produce promising results in detecting network intrusions.

It has been shown that using irrelevant and redundant features in the application of these machine learning techniques in the context of network intrusion detection, can lead to a reduction in classification performance [7, 26–33, 35]. Thus, a number of researchers have applied feature selection algorithms to select relevant features that allow for the accurate classification of intrusions in network intrusion detection. One approach involves the use of statistical measures such as mutual information to measure feature relevancy [7, 26–30, 35], while the other approach uses the performance of a machine learning method, typically supervised methods such as support vector machines, to identify relevant features [31–33]. The design and implementation of a feature selection algorithm that does not rely on labelled data and is able to achieve a gain in classification performance is a priority.

2.2 Network intrusion detection systems

This section discusses a typical NIDS structure, and provides a more detailed description of the two major detection approaches, namely misuse-based detection and anomaly-based detection.

2.2.1 Structure

Despite various detection methodologies, network intrusion detection system designs typically adhere to a general structure in terms of functionality. Figure 2.1 shows a block diagram of the general structure of intrusion detection systems [1]. The monitored environment in figure 2.1 consists of the network traffic flowing between network routers and/or networked computer systems. Each block of figure 2.1 is discussed in what follows.

- E-boxes (Event-boxes) — Event boxes are composed of sensors or nodes that observe the monitored environment gathering information on events[1] that may require further analysis.
- D-boxes (Database-boxes) — These boxes store the information collected by the E-boxes. The stored information is passed on to the A-boxes for analysis.
- A-boxes (Analysis-boxes) — These process the event information collected by the E-boxes, with the aim of detecting malicious or potentially malicious activity occurring within the monitored environment.
- R-boxes (Response-boxes) — If malicious activity is detected, the response boxes will raise an alert and may attempt to either stop the potential intrusion, or prevent a confirmed intrusion from spreading or progressing.

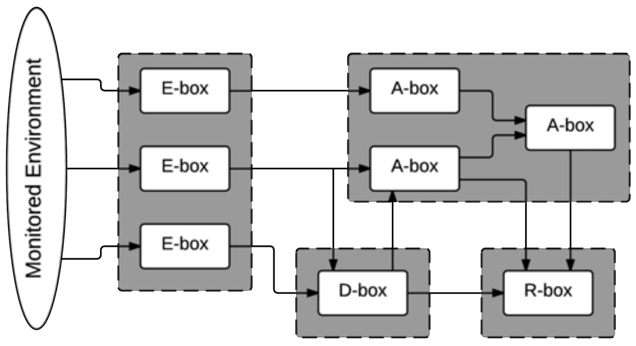


FIGURE 2.1: *Structure of a typical intrusion detection system (from [1]).*

2.2.2 Detection approaches

This section provides a more detailed description of the two major approaches toward detection of intrusions in computer networks.

[1]Events on a host-based IDS include process identifiers and system calls related to operating system information, while events on a NIDS includes traffic volumes, protocol usage, service ports, IP addresses, etc. [1]

2.2.2.1 Misuse-based detection

Misuse-based detection compares network traffic features, collected on an event basis, to predefined patterns or signatures of known attacks. Signatures may be defined using a rule-based approach, where a set of rules describes a specific network attack. For example, a set of keystrokes executed in a certain sequence may be used to define a specific attack [14]. Masri et al. [52] create signatures based on the sequence of events that occur during an attack; this sequence of events defines a specific network attack.

Misuse-based NIDS are often more accurate and have lower false positive percentages than anomaly-based detection approaches, but at the cost of not being able to detect unknown attacks or previously unobserved variants of existing attacks. Signatures require constant updating in order to keep up with new and emerging threats. They are typically created manually by administrators or security experts, which places an extra workload on them [16]. Thus, constant updating of signatures can be a time consuming exercise, where a new threat could cause a significant amount of harm to a system long before a security expert creates a signature for it [10, 11, 13, 14, 16, 20–22].

Automated techniques for generating signatures for intrusion detection systems are found in [52, 53]. The misuse-based approach is commonly used in commercial NIDS and network intrusion prevention systems owing to their lower false positive percentages as compared to anomaly-based approaches. A common misuse-based open source NIDS is Snort [54] and a common commercial misuse-based NIDS is Bro [55].

2.2.2.2 Anomaly-based detection

Anomaly-based detection techniques operate based on the assumption that the observed statistics of key features of malicious traffic deviates from that associated with legitimate traffic [10, 11, 13, 14, 16, 20–22, 56]. These techniques typically involve the construction of explicit or implicit models of legitimate network traffic. The statistics of observed network traffic features are subsequently compared to the model, where any deviation from the model is classified as anomalous, which may indicate that malicious network traffic has infiltrated the networked computer system.

The advantage of anomaly-based detection schemes is that they are able to detect those novel attacks which have not been observed previously [1, 10, 11, 13, 14, 16, 20–22]. Anomaly-based systems do not attempt to identify specific attack sequences, but rather identifies unusual network traffic, thus, attacks can be identified regardless of whether they have been observed previously. Anomaly-based systems often suffer from high false positive percentages owing to the fact that every deviation from the legitimate traffic model will be classified as anomalous, which may not necessarily be caused by malicious network traffic. The performance of anomaly-based systems relies on how accurately the legitimate traffic models represent legitimate traffic. Creating models of legitimate traffic is not a trivial task in dynamic environments with non-stationary network traffic with constantly changing statistical profiles, or cyclic trends. Often there is a discrepancy between the models and the statistics of true legitimate traffic. This increases the need to constantly update legitimate traffic models or construct models that take cyclic trends into account, which can be computationally expensive.

In recent years, commercial anomaly-based NIDS have become more prominent [1, 57]. Commercial NIDS that utilize anomaly-based methods include: SPADE [58] and Prelude [59].

2.3 Unsupervised anomaly-based NIDS

Two commonly used unsupervised machine learning techniques for anomaly-based network intrusion detection are clustering and outlier mining. Clustering techniques group similar data samples together to discover patterns that may be present within the dataset, while outlier mining techniques discover those data samples that are different from the majority of the data over the feature space considered [2]. Data samples that are similar with regards to feature values over a feature space, and are in the majority, are considered to be legitimate data samples. This is essentially an implicit model of legitimate network traffic. Those data samples that are dissimilar to the legitimate samples with regards to feature values over a feature space, and are in the minority (outliers), are considered to be attack data samples. The advantage of creating models in this manner is that labelled data is not required during the classification stage.

Both clustering and outlier mining techniques have been successfully applied in anomaly-based NIDS [43–51, 60, 61]. In what follows, NIDS that utilize these techniques are reviewed.

2.3.1 Clustering based methods

Clustering is an unsupervised learning method commonly used for data mining tasks. It partitions a dataset into groups (clusters), where those data samples that are similar to one another with respect to feature values, will be clustered together. While data samples belonging to different clusters are highly dissimilar to one another, with respect to feature values. Clustering allows for the identification of patterns and interesting distributions in the underlying data which in turn allows for the derivation of useful conclusions [2, 62]. In the context of NIDS, this is the detection of network intrusions. In various fields, clustering is found under different names: unsupervised learning in pattern recognition, numerical taxonomy in biology and ecology, typology in social sciences and partitioning in graph theory [63].

Anomaly-based NIDS that use clustering techniques [43–46] typically follow a common structure, this structure or functional block diagram is provided in figure 2.2.

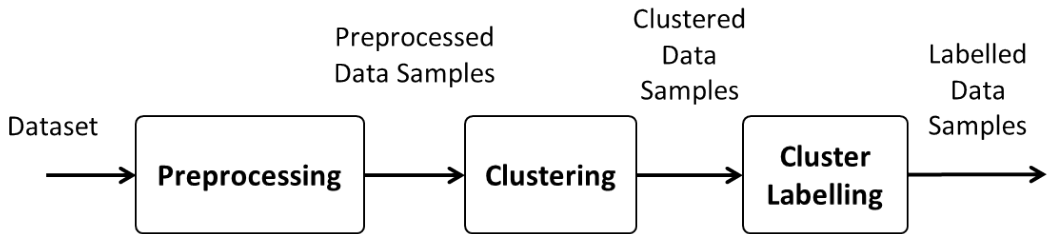


FIGURE 2.2: *Anomaly-based NIDS functional block diagram.*

Preprocessing involves the application of various algorithms to a dataset in order to represent the data in a form that is suitable for the machine learning algorithm that the data will be applied to. Normalization and dimensionality reduction are two examples of preprocessing methods [38]. Clustering involves the application of a clustering algorithm to the preprocessed data samples, and cluster labelling involves the assignment of a class to each cluster. A class or label may correspond to legitimate or malicious network traffic.

Labels are assigned based on the assumption that legitimate network traffic vastly outnumbers malicious network traffic. Thus, data samples belonging to smaller clusters, with respect to the number of data samples assigned to the cluster, are labelled as malicious traffic samples.

Training and testing is used in supervised machine learning algorithms, where labelled training data is used to create an inferred function, which is called a classifier. Testing is subsequently performed by introducing new/previously unseen data samples to the classifier, which classifies or labels the data based on the knowledge inferred from the training data [64]. Given that clustering does not require labelled data, the equivalent of training lies in the clustering of data samples and the labelling of cluster centres and data samples assigned to those cluster centres. The labelled cluster centres form the basis for the classifier in this case. "Testing" is typically carried out by assigning to new/previously unseen data samples, the label of the cluster centre that they are closest to with regards to distances computed over a feature space. In what follows, a review of anomaly-based NIDS that make use of clustering techniques is provided.

Portnoy et al. [43] introduced an anomaly-based intrusion detection system that does not require labelled data in its execution. The proposed system makes use of a single-linkage clustering technique and an innovative cluster labelling scheme that is utilized by researchers in more recent works as well. The proposed system was applied to the KDD Cup 1999 dataset [24], and follows the processing sequence illustrated in figure 2.2. Preprocessing involves the normalization of the numeric data of the KDD Cup 1999 dataset, using the statistical normalization technique (refer to section 3.3.1). A variant of the single-linkage clustering algorithm [65] is used to cluster the normalized data. The algorithm first selects one random data sample from the dataset to be used as the initial cluster centre. Subsequent data samples are assigned to the cluster centre that they are closest to, based on the Euclidean distance metric, or assigned as new cluster centres if the distance to their closest cluster centre is greater than a threshold value. Clusters are labelled based on the assumption that legitimate network traffic samples vastly outnumber malicious network traffic samples. Additionally, it is assumed that legitimate network traffic will be clustered together. Based on these assumptions the algorithm labels a predefined percentage of the largest clusters, in terms of the number of data samples belonging to the cluster, as legitimate clusters. The remaining smaller clusters are labelled as malicious or attack clusters. Training and testing sets are created by dividing the KDD Cup 1999 dataset into ten components, to be used for cross-validation testing. However, only four were used for training and testing owing to the remaining components not being representative of various types of attacks. Testing sets are normalized, and performance is evaluated by assigning each data sample within the testing sets to the cluster centres that they are closest to. The data samples are assigned the label of the cluster centre to which they are assigned to. Performance was evaluated in terms of true positive and false positive percentages. Cross-validation testing revealed that the classifier obtained true positive percentages between 18% and 57%, and false positive percentages between 0.3% and 12%, where a majority of the false positive percentages were approximately 1%. The results varied depending on which training and test set combinations were used.

Syarif et al. [44] proposed an anomaly-based detection algorithm that follows the structure of figure 2.2 using five different clustering techniques to detect network attacks in the NSL-KDD dataset [25]. The anomaly-based detection clustering techniques include: k–means, improved k–means, k–medoids, Expectation-Maximization (EM) with Gaussian Mixture Models (GMM), and distance-based outlier detection. Note that all algorithms, excluding the improved k–means clustering algorithm are discussed in greater detail in chapter 3, section 3.4. The anomaly-based detection module consists of four steps. Preprocessing consists of the application of feature extraction and normalization to the NSL-KDD dataset [25]. Feature extraction was used to create an optimized feature set, though the exact algorithm or technique was not specified. Normalization was performed on the numeric data using the statistical normalization technique (refer to section 3.3.1). The preprocessed data is subsequently clustered using each of the five clustering algorithms namely: k–means, improved k–means, k–medoids, Expectation-Maximization (EM) clustering, and distance-based outlier detection. The cluster labelling algorithm is unspecified. Unobserved data samples are subsequently classified by assigning to each sample the class of the cluster centre it is closest to. The authors compared the performance of the five clustering techniques used for anomaly-based detection on the NSL-KDD dataset, against each other and against four misuse-based classifiers that make use of machine learning algorithms. The misuse-based algorithms were applied to training and testing data in the supervised sense of training and testing. Based on classification accuracy and false positive percentage, the results showed that the misuse-based methods were unable to detect novel attacks that are within the NSL-KDD testing set, with the highest accuracy being 63.97% with a false positive percentage of 17.90%. The distance based outlier detection algorithm of the anomaly-based system was shown to have the highest classification accuracy of 80.15%, however, the false positive percentages were above 20% for all anomaly-based techniques.

Wang [45] proposed an improved k–means clustering algorithm for use in anomaly detection applied to the KDD Cup 1999 dataset [24]. The algorithm overcomes the sensitivity in performance to the selection of initial cluster centres experienced in k–means. The difference between the two algorithms is that improved k–means computes the initial centres differently; k–means chooses the initial centres randomly, while improved k–means selects the most decentralized samples as the initial cluster centres by iteratively selecting the samples that are furthest from each other with regards to the feature space, as cluster centres. The algorithm consists of a training and a testing phase, where training consists of preprocessing, clustering and labelling of the data samples and cluster centres, as depicted in figure 2.2. Clusters are labelled based on the assumption that legitimate data samples vastly outnumbers attack data samples. Testing introduces new/unobserved data samples from a testing set into the classifier produced in the training phase, and assigns to these data samples the label of the cluster centre to which they are closest to, based on Euclidean distance with respect to the feature space, which is similar to [43]. Improved k–means is shown to outperform ordinary k–means when applied to the KDD Cup 1999 dataset [24]. This minor change to the original algorithm results in an increase in true positive percentage, and a decrease in false positive percentage. Both algorithms were demonstrated to be capable of detecting unknown attacks.

Papalexakis et al. [46] proposed two co-clustering methods to perform network intrusion detection when applied to the KDD Cup 1999 dataset [24]. Co-clustering is essentially a combination of clustering and feature selection, where the algorithm clusters data samples together over different feature subspaces rather than the entire feature space. Two forms of co-clustering is presented in reference [46]: hard and soft co-clustering. Hard co-clustering involves the assignment of data samples to a single cluster, with 100% membership. The application of hard co-clustering involves the use of an Information Theoretic (IT) approach that utilizes Bregman divergences [66]. Soft co-clustering involves the assignment of data samples to clusters based on fuzzy membership, in which data samples belong to each cluster to a certain degree. It is implemented using a Sparse Matrix Regression (SMR) technique [67]. SMR co-clustering is applied to the dataset, where binary clustering is performed over ten iterations. Two clusters were obtained through SMR co-clustering where one contained 99.36% attack samples, and the other 73.21% legitimate samples, on average. This implies that SMR co-clustering is capable of distinguishing between legitimate and attack data samples. SMR performed clustering over a subset of features that it selects and it was discovered that on each execution the same seven features were used to distinguish the attack cluster. IT clustering was performed with five clusters and using only the features identified in the SMR clustering stage. By combining SMR's ability to identify features that can distinguish between attack and legitimate data samples, and IT's ability to run over the entire dataset, authors achieve results that are argued to be comparable to the winning entry of the KDD Cup 1999 competition [68] and that is not tailored for the KDD Cup 1999 dataset only [24].

2.3.2 Outlier mining methods

Outlier mining uses the distances between data samples and their neighbours or the densities surrounding a data sample, determined over a feature space, to discover data samples that are significantly different to the rest of the data samples. These are referred to as outliers. Hence, outliers are those singular data samples that possess a large distance to their neighbouring data samples, or those data samples that occur in less dense regions of the feature space [2].

Given that distance and density are used to identify outliers, outlier mining techniques will generally not be able to identify those small clusters of attack data samples that clustering techniques can identify. This is owing to the fact that samples within these small clusters are likely to be close to one another based on their distributions in the feature space. Thus, clustering techniques identify those attack data samples that occur in small compact groups within the feature space, while outlier mining identifies those attack samples that are disjoint from the rest of the data with respect to their distribution over the feature space considered.

For these reasons, it is common for outlier mining techniques to be combined with a clustering algorithm [47–51, 69], which allows for the detection of both attack clusters, as well as singular attack data samples that are disjoint from the rest of the data samples with regards to their distribution over the feature space. Training in this regard is similar to the clustering based "training", where smaller clusters are labelled as those that consist of attack data samples. The difference lies in the fact that outliers are also labelled as attack data samples, resulting in more compact clusters, though they are not utilized in the testing phase. Thus, the testing phase is the same as with the clustering techniques, where new/unobserved data samples are assigned the labels of the cluster centres to which they are closest to.

The authors of [47, 70, 71] proposed an Unsupervised Network Intrusion Detection System (UNIDS) that is able to detect novel network intrusions without the use of signatures or labelled traffic data. This is achieved by applying a novel outlier detection technique. The proposed system operates in four steps to detect network intrusions, as shown in figure 2.3.

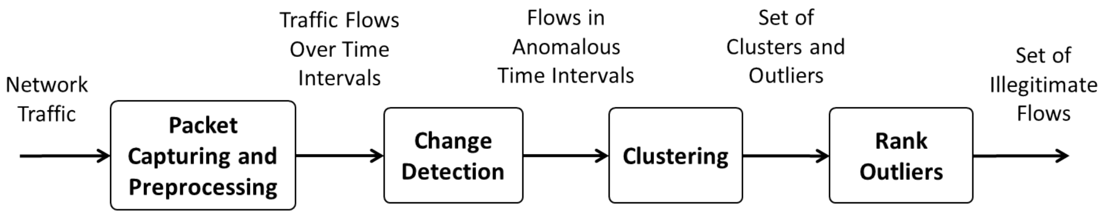

Network Traffic
Packet Capturing and Preprocessing
Traffic Flows Over Time Intervals
Change Detection
Flows in Anomalous Time Intervals
Clustering
Set of Clusters and Outliers
Rank Outliers
Set of Illegitimate Flows

Figure 2.3: *UNIDS functional block diagram*

The UNIDS captures network traffic data and aggregates it into traffic flows, which are subsequently divided into a series of intervals containing traffic flows which occurred within a certain time period. A standard time series change-detection technique, presented in [72], is applied to the aggregated traffic flows over each time interval, to detect unusual changes in feature values. Three volume based features are analysed: the number of bytes, the number of packets, and the number of flows within a specific time interval. Intervals which possess abrupt changes in these feature values are flagged as anomalous.

UNIDS [47, 70, 71] applies a multi-clustering algorithm to the traffic flows contained within each anomalous time interval, over the feature space of the three considered features. An outlier score is subsequently assigned to each interval. The multi-clustering algorithm consists of Sub-Space Clustering (SSC) [73], Density-based Spacial Clustering of Applications with Noise (DBSCAN) [74] and Evidence Accumulation Clustering (EAC) [75]. SSC seeks to find clusters of data samples within different feature subspaces of a multi-dimensional dataset [73]. SSC divides the dataset into multiple n-dimensional feature subspaces, each comprising of a distinct combination of n features from the dataset. DBSCAN is subsequently applied to cluster the data samples within each feature subspace. DBSCAN clusters data samples based on density, using a nearest neighbour approach. Data samples with a large number of data samples that are within close proximity of each other with regards to distances over the feature space are considered to be in a highly dense region in the feature space, and are clustered together. Data samples with a small number of neighbours within close proximity regarding their distances over the feature space, are considered to be in a region of low density in the feature space, and are labelled as outliers. The output of this step is thus a set of clusters and outliers for each feature subspace. EAC assigns a score to each outlier in each subspace which indicates the degree of abnormality of each outlier. The outlier score is computed within each subspace as the distance between an outlier and the centroid of the largest cluster within that subspace. A larger score indicates that the outlier has a high degree of abnormality as it has a larger distance from the majority of the data samples in the subspace. The result is a set of scores for each outlier in each subspace. All outliers from all subspaces are subsequently ranked based on their scores in descending order. All outliers that possess a value above a threshold are classified as attack data samples.

UNIDS [47, 70, 71] takes advantage of the observation that DBSCAN performs more accurately when applied to lower dimensional data [47]. The advantage of clustering multiple low dimensional subspaces is that a finer-grained analysis is performed, allowing for the detection of low intensity anomalies that are hidden within high dimensional network traffic, as demonstrated via experimentation in [47].

In [48], an anomaly detection approach known as NADO (Network Anomaly Detection using Outliers) was introduced. It is effectively a combination of outlier mining and clustering, where the outlier mining technique is the Reference-based Outlier Detection technique for large Datasets (RODD), as described in [76], which uses the density of each samples neighbourhood to identify outliers.

NADO is a two stage algorithm. It first uses a k–means clustering technique to partition the data samples of the KDD Cup 1999 dataset [24]. A reference point from each cluster is then calculated and mean-based profiles are constructed for each cluster. A profile is a vector containing the mean of each feature over the data samples belonging to a cluster, which is essentially the cluster centre of a cluster. Each cluster centre is used as a reference point to compute the degree of neighbourhood density of each data sample, computed as the sum of the absolute difference in distance between a sample and a cluster centre, and the samples neighbour and the cluster centre, for each neighbour. This is computed for each data sample, over each corresponding cluster centre. The minimum value computed over each reference point is assigned to the data sample as its outlier score. Samples with scores greater than a user specified threshold are labelled as attack data samples.

Songma et al. [49] proposed a two-phase classification method for performing intrusion detection on the KDD Cup 1999 dataset [24]. Preprocessing consists of the removal of redundant data samples from the dataset, and the conversion of categorical features to numerical values prior to normalization. The first phase of classification applies k–means clustering to the data and the second phase utilises a distance-based technique to identify outliers. Outliers are defined as those data samples whose k nearest neighbours are a distance d_r from it, where k is a user defined fraction of the entire dataset, and d_r is a user specified threshold. Thereafter, each sample of the dataset is assigned a class label. Authors demonstrate that their method outperforms SVMs and rough-set fuzzy SVMs [77] in terms of true positive and false positive percentages and overall accuracy.

Said et al. [78] performed intrusion detection using several preprocessing techniques and distance-based outlier detection on the KDD Cup 1999 dataset [24]. Their goal was to compare the performance of various attribute normalization techniques, distance metrics, and PCA threshold values, in terms of true positive percentages when applied to distance based outlier detection. The numeric data of the KDD Cup 1999 dataset was first normalized using either, Z-Score (statistical normalization), log normalization, or min-max normalization. PCA was subsequently applied to the normalized numeric data using seven different threshold values. Authors identified outliers based on the definition that outliers are the top N_{out} data samples that possess the largest sum of distances to their k nearest neighbours in the feature space. The distances for the numeric data are computed using either the Euclidean distance metric or the Mahalanobis distance metric, while the distances for the categorical data are computed using the Hamming distance metric. The hamming distance is computed by counting the number of categorical features which do not have the same category. All samples identified as outliers are labelled as attack samples. Experimentation was performed with and without applying PCA to the numeric data. The outcome of the study indicated that, overall, the best result was obtained using the log normalization technique in conjunction with the Euclidean distance metric, while applying PCA to the numeric data.

Chawla et al. [50] performed intrusion detection by combining the k–means clustering algorithm with a distance-based outlier detection technique that utilizes the Euclidean distance metric. This algorithm is a modified version of k–means, and is referred to as k–means--. First, k initial centres are randomly chosen, and data samples are assigned to the cluster centres to which they are closest to. All samples are ranked in descending order based on the distance between a sample and its corresponding cluster centre. The top N_{out} samples are selected as outliers and removed from the dataset. New cluster centres are computed over the feature space as the mean of the remaining data samples belonging to each cluster. The process was repeated using the new cluster centres as the initial cluster centres. This continues until convergence is achieved. Cluster labelling was not required for performance evaluation. Experiments were performed using a subset of the KDD Cup 1999 dataset [24], where the three largest classes, in terms of number of data samples, of the dataset (*smurf, neptune* and *normal*), were considered as non-outliers, and the remaining classes were considered as outliers. The performance was measured based on the algorithms ability to identify these samples as outliers. Using precision and purity as performance measures, the k–means-- algorithm, was compared against the k nearest neighbour (kNN) approach for distance-based outlier detection, in which outliers are defined as the N_{out} samples with the largest distance to their k^{th} nearest neighbour. The difference between the two methods is that the kNN approach does not perform clustering, outliers are removed from the entire dataset in one iteration. The k–means-- algorithm was demonstrated to outperform the kNN approach.

Authors of [51] proposed a cluster-based outlier detection technique, which classifies both small clusters and single data samples as outliers. The algorithm LDBSCAN [79] is used to find both the Local Outlier Factors (LOFs) [80] for each data sample and to assign data samples to clusters. The LOF is a density-based outlier detection method that identifies outliers based on their k^{th} nearest neighbour distances. The LOF is local in the sense that it only considers a certain number of samples as the neighbourhood of a sample in its computation of outlier scores for each sample in the dataset. The algorithm was applied to the backbone network anomaly detection system of CSTNET, which monitors the input and output throughput of CSTNET, which is the internet service provider for all institutes of the Chinese academy of science. The network administrators of CSTNET report that the proposed algorithm produced fewer alerts when compared to the LOF technique alone, where the LOF technique raised many false positives.

2.4 Feature selection

In general, feature selection is an important preprocessing step for data used in machine learning applications [38]. Network traffic is rich with contextual information that are characteristic of various types of network activities, which may be legitimate or malicious. Due to this richness, large feature vectors may be constructed from network packet data for the purpose of network intrusion detection, which may result in redundant and irrelevant features being used for classification. This may inadvertently reduce the classification performance of the machine learning algorithms used in the design of NIDS (as discussed in section 1.2). In sections 2.3.1 and 2.3.2, many of the works did not perform feature selection prior to the application of machine learning techniques

Several works [7, 26–33, 35, 39, 81] improve the classification performance of NIDS by performing feature selection in the context of network intrusion detection. These feature selection methods are typically applied to either the KDD Cup 1999 dataset [24] or the NSL-KDD dataset [25].

The following section is divided into filter and wrapper feature selection methods [82]. A filter method uses a statistical measure to assign scores to each feature. Features are subsequently ranked according to the scores produced by these statistical measures, where the top n features are selected. The degree of correlation, and information gain are commonly used statistical measures [7, 27, 29, 30, 35]. The degree of correlation is measured between features and between features and classes. A significant feature is one that is highly correlated with a class, while showing little or no correlation to other features. Information gain measures the level in which a particular feature is able to discriminate between classes.

Filter methods require the class labels in order to determine the significance of features and thus require labelled data. Filter methods are independent of the classification algorithm, and are typically less computationally intense than wrapper methods [83]. However, applying data over the resultant feature subset to a classifier may not produce classification accuracies as high as that obtained when applying the same data over a feature subset produced by a wrapper method, which optimizes feature subsets for a particular classifier.

Wrapper methods rank features based on the results obtained from performing classification on a dataset over various feature subsets. An example of this approach is the use of a Support Vector Machine (SVM) to perform classification on a dataset, in which the predicative accuracy of the SVM is used to indicate the significance of a feature subset or of a single feature [31]. This approach requires labelled data to compute the predictive accuracy of the SVM.

Wrapper methods are dependent on the classifier, and can be more computationally intensive than filter methods, if the classification algorithm is computationally intense [83]. Wrapper methods generally produce feature subsets that produce results that are better than that obtained using feature subsets produced by filter methods, provided that the same classifier is used as was used during feature selection. Thus, wrapper methods produce feature subsets that are tailored to specific classification algorithms [2, 31, 82].

2.4.1 Filter methods

Amiri et al. [26] proposed two feature selection algorithms for network intrusion detection over the KDD Cup 1999 dataset [24]. The algorithms are the Linear Correlation-based Feature Selection (LCFS) algorithm, and the Modified Mutual Information-based Feature Selection (MMIFS) algorithm. LCFS ranks features using the linear correlation coefficient between features and classes and between pairs of features. The first feature selected is the feature that possesses the highest correlation with the class. The remaining features are selected iteratively based on the greedy search algorithm. The next feature is selected as the one that maximizes the difference between the correlation to the class and the sum of correlations to each previously selected feature. The process is repeated until the desired number of features are selected. The MMIFS algorithm functions in a similar fashion to LCFS, where instead, features are ranked using the mutual information between features and classes and between features and selected features. Both proposed algorithms are compared against the Forward Feature Selection Algorithm (FFSA). This method utilizes mutual information to rank features, but differs from MMIFS by only computing the mutual information between features and classes. Classification is done using a least squares variant of a support vector machine (LSSVM). The LSSVM method solves a set of linear equations in the optimization stage, rather than convex quadratic equations as with traditional SVMs. Utilizing the LSSVM method with a Radial Basis Function (RBF) kernel was demonstrated to reduce computational costs [84]. The LSSVM is used to perform binary classification on the KDD Cup 1999 dataset. The dataset is divided into five broad classes, thus five LSSVMs were implemented where each LSSVM is a binary classifier for a different class. Using true positive and false positive percentages, over each LSSVM classifier, the results showed that MMIFS was the most effective in producing feature subsets that were able to detect probe and Root-to-Local (R2L) attacks, while both FFSA and MMIFS performed comparably in producing feature subsets that were able to detect User-to-Root (U2R) and Denial-of-Service (DoS) attacks, and legitimate traffic samples.

Chebrolu et al. [27] proposed two filter-based feature selection algorithms based on Bayesian Networks (BN) and Classification and Regression Trees (CART). The Bayesian network consists of a set of nodes and directed edges; in this case, the nodes represent features and the edges represent the conditional dependency, or conditional probability between the features that the edge connects. Thus, the edges indicate the level of dependency between pairs of features. The Markov blanket of a Bayesian network consists of those nodes that are conditionally independent of one another. The Markov blanket of a particular class C is defined as the set of features that are conditionally independent of C. Thus, defining a feature subset of features that are independent of one another, that is, they are uncorrelated to one another, which implies that the feature subset will contain features that are not redundant. A CART is a binary recursive tree, where each node is split exactly two times. The CART is constructed by using features as the nodes, where the Gini rule is used to split each node, essentially creating a path of features leading to a leaf node. The tree is complete if nodes cannot be split further, these final nodes are leaves and represent the classes of the dataset. Feature subsets are then constructed based on the maximal tree that leads to the identification of the various classes. Classification was done using a combination of the Bayesian network and CART, applied to the KDD Cup 1999 dataset [24], the feature subsets produced by the Bayesian network were demonstrated to return higher classification accuracies when compared to using the full feature set.

Kayacik et al. [7] used information gain to perform an analysis of each feature of the KDD Cup 1999 dataset [24], with respect to its ability to discriminate between the classes in the dataset. Information gain was computed between each feature and each class, which allowed the authors to construct a list that maps each class to the feature that best discriminates it. It was found that *normal*, *smurf*, and *neptune* classes were easily distinguishable owing to the fact that a large number of features possessed a large information gain for those classes. It was also found that a number of features had very little information gain for all classes in the dataset, implying that some features of the KDD Cup 1999 dataset are irrelevant for network intrusion detection. A list of class labels and the feature that is most relevant in discriminating the class is provided.

Tang et al. [29] performed feature selection on the KDD Cup 1999 dataset [24] using the information gain between features and classes, which is a similar approach to [7]. Authors construct a list that maps a class to the feature with the highest information gain to that class. All features that appear within this list are selected as the featute subset. Authors apply a Triangle Area-based Support Vector Machine (TASVM) to perform classification. TASVM is a combination of the k-means clustering algorithm and the SVM learning algorithm.

Olusola et al. [28] performed feature selection on the KDD Cup 1999 dataset [24] using the rough set degree of dependency and dependency ratio to measure the relevance of each feature in distinguishing each class. Feature selection is conducted using two approaches. The first approach computes degree of dependency for each class based on the available number of data samples belonging to a class. This signifies how well the feature can distinguish one class from other classes. The second approach maps each class label to others for each feature. Authors produce a list that highlights which features are the most relevant for detecting each class present in the KDD Cup 1999 dataset.

Zargari et al. [35] reviewed the findings of [7, 26–29], providing a table that lists the features of the KDD Cup 1999 dataset [24] along with those attack classes that are strongly correlated with each feature. The authors proposed two feature selection algorithms: in the first, feature subsets are selected based on their degree of correlation, where better feature subsets are those that consist of features that exhibit a high correlation with the classes and a low correlation with each other [85]. Features are selected in a similar manner to the LCFS algorithm in [26], using a greedy approach. The second algorithm uses information gain as a measure of feature relevance, and selects the top features as the feature subsets based on the ranking of individual features. Intrusion detection performance is computed over these feature subsets in the dataset via a random forest algorithm. The feature subsets produced by the two proposed algorithms were reported to produce results that outperform the results produced when performing intrusion detection over a feature subset constructed through a majority vote of the feature sets from [7, 26–29].

Eid et al. [30] performed feature selection on the NSL-KDD dataset [25] using a linear correlation-based feature selection approach. The proposed method is based on the analysis of Pearson correlation coefficients. The approach consists of two phases, the correlations between pairs of features are first computed, and constructed into an N_F-by-N_F matrix, where N_F is the total number of features in the feature space considered. The pairs of features with the highest correlations to one another are found, and one of the features are discarded, if the coefficient is greater than a threshold, if not, neither feature is discarded. The second phase involves the computation of the correlations between the features selected in the first phase and the classes, where features that are highly correlated to the classes are selected as the final feature subset. Classification is done using a decision tree (C4.5) [86] implementation. The proposed method is shown to produce a feature subset that achieves a higher classification accuracy when compared to using all the features within the dataset. The feature subset produced by the proposed algorithm is also shown to achieve higher classification accuracies when compared to using feature subsets produced by two commonly used feature selection methods, gain ratio and information gain, as well as a widely used feature extraction method, Principal Component Analysis (PCA) [87]. Both the feature selection and classification stages of the proposed algorithm require labelled data in their execution.

2.4.2 Wrapper methods

Li et al. [31] proposed a Gradual Feature Removal (GFR) method that was applied to the KDD Cup 1999 dataset [24] for network intrusion detection. The classification algorithm output, the average Matthews Correlation Coefficient (MCC), as computed over a feature subset of labelled data is used as a measure of the fitness of a candidate feature subset.

The GFR method begins with the full feature set and iteratively removes the least significant feature, until only one feature remains. More specifically, given N_F features in the feature space, an iteration involves the computation of the average MCC over N_F candidate feature subsets. Each subset consists of $N_F - 1$ features, where a different feature is excluded from each subset.The subset that produces the highest average MCC value reveals the least significant feature as the one that was excluded from that subset; the feature is then removed from the candidate feature space. The process is subsequently repeated with $N_F - 1$, $N_F - 2$, $N_F - 3$ features and so on, until only one feature remains. The final result is a list of all the excluded features ranked from most to least significant. Using an SVM, the best accuracy was obtained when the 19 most significant features were used. The resultant feature subset was compared against feature subsets produced by three similar feature selection algorithms, namely the Feature Removal Method (FRM), the Sole Feature Method (SFM) and a hybrid of these two methods. The FRM is performed in a similar manner to the GFR method in that features are ranked based only on the first iteration of the GFR method. The SFM computes the average MCC for one feature at a time, where higher average MCC values shows a higher level of significance for a feature. The authors demonstrated that the GFR method produced a feature subset that was able to attain classification accuracies that were higher than the feature subsets produced by the other three feature selection algorithms. It is noteworthy that all the feature selection algorithms considered in [31], require the use of labelled data.

Dastanpour et al. [32] proposed the use of a genetic algorithm and an SVM to perform feature selection on the KDD Cup 1999 dataset [24] for network intrusion detection. A feature subset is selected based on the true positive percentage of the SVM, which is produced by performing classification on the dataset over a candidate feature subspace selected by the GA. The proposed method is compared against two filter-based feature selection methods, namely the linear correlation-based feature selection method, and the forward feature selection algorithm. Both these algorithms were utilized by Amiri et al. [26], as reviewed in section 2.4.1. Applying the KDD Cup 1999 dataset to the SVM over the feature subsets selected by all three feature selection algorithms demonstrated that the proposed method was able to produce a feature subset that attained higher true positive percentages, and lower false positive percentages than the feature subsets produced by the two filter-based feature selection methods.

Zhang et al. [33] proposed the use of a Bayesian network classifier to perform feature selection and classification on the NSL-KDD dataset [25]. The classification accuracy of the BN classifier is used to measure the relevance of features in the NSL-KDD dataset. The classification accuracy when utilizing the full feature set is first computed as a benchmark for ranking features. Ranking consists of an iterative process which involves the removal of one feature and the recalculation of the classification accuracy over the reduced feature space, during each iteration. If the classification accuracy produced by performing classification over the reduced feature subset is less than the classification accuracy produced over the full feature set, then the removed feature is significant. The process is repeated until the classification accuracies of all features that are deemed significant is lower than the classification accuracy attained over the full feature set. The proposed method was compared against four filter-based feature selection methods, namely information gain [88], gain ratio, ReliefF [89] and ChiSquare [90]. It was demonstrated that the proposed method produced a feature subset of the NSL-KDD dataset, that attained a higher classification accuracy than the feature subsets produced by the four filter-based methods. Additionally, the method produced the candidate feature subset in less time than the filter-based methods.

2.5 Conclusion

This chapter reviewed several works that design and implement anomaly-based network intrusion detection systems through the application of two unsupervised machine learning techniques. These include clustering and outlier mining techniques. The literature reveals that anomaly-based NIDS typically follow a common structure, which involves data preprocessing, clustering, and cluster labelling. While outlier mining techniques are typically implemented in conjunction with a clustering algorithm. It was found that cluster labelling is commonly achieved by making the assumption that legitimate network traffic vastly outnumbers malicious network traffic. The literature reviewed, demonstrates that anomaly-based NIDS using unsupervised machine learning techniques produces promising results with regards to classification accuracy. These methods also provide the added benefit of not requiring labelled data. Thus, unsupervised machine learning techniques are considered in this research for the design and implementation of an anomaly-based network intrusion detection system.

Additionally, this chapter reviews several works that design and implement feature selection algorithms for use in network intrusion detection. Feature selection algorithms are divided into filter-based and wrapper-based methods. It was found that filter-based methods commonly used in the literature include the use of statistical measures such as: information gain, mutual information and degree of correlation. From the literature reviewed, commonly used wrapper-based methods includes the use of classification accuracy as a measure of the quality of features or feature subsets. The literature demonstrates that both methods are capable of producing feature subsets that produce classification accuracies that are higher or comparable to the classification accuracy produced when performing classification over the full feature set. The shortcoming of the feature selection methods reviewed in this chapter is that all techniques require labelled data to determine feature relevance. Labelled data is not always available in practical environments, thus, this research focusses on designing and implementing a feature selection algorithm that does not require labelled data.

Chapter 3

An Unsupervised Classifier for Anomaly-based Network Intrusion Detection

3.1 Introduction

The proposed classifier for anomaly-based NID is presented in this chapter. The classifier uses an unsupervised machine learning algorithm which involves the preprocessing and clustering of data samples, and the labelling of clusters. The algorithm takes in as input a set of feature vectors derived from network traffic, and performs binary classification on each vector, which assigns the label *legitimate* or *attack*, to each feature vector. The proposed classifier does not require labelled data samples.

The functional block diagram of the proposed classifier is presented in figure 3.1.

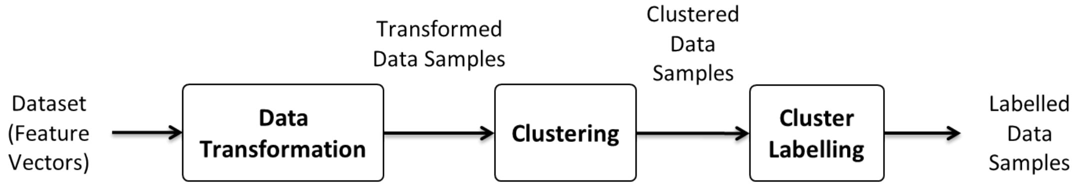


FIGURE 3.1: *Functional block diagram of the proposed classifier.*

In what follows, a description of each block of figure 3.1, as well as the dataset, are provided.

3.2 Dataset

In this research the NSL-KDD dataset [25] is used. It is widely used as a benchmark dataset for anomaly-based NIDS. The use of this dataset allows for the comparison of results with existing systems proposed in the literature. The NSL-KDD dataset was derived from the KDD Cup 1999 dataset [24]; both datasets are discussed in what follows.

3.2.1 KDD Cup 1999 dataset

In 1998, DARPA (Defence Advanced Research Projects Agency) and the Lincoln laboratory at the Massachusetts Institute of Technology (MIT) executed an intrusion detection system evaluation programme that involved the collection of network traffic data over a simulated network environment of an air force base [91]. The environment consisted of an intranet of the air force base and an external network representing the Internet. The intranet consisted of several physical UNIX machines and a gateway to thousands of emulated workstations that utilize a variety of network applications and services to generate network traffic. This traffic consisted of typical user activities such as sending and receiving emails, browsing websites, transferring files using the File Transfer Protocol (FTP), and using telnet to log into remote computers. The external network consisted of a sniffer to capture network traffic, a gateway to hundreds of emulated workstations, and an additional gateway to thousands of emulated web servers [91]. Malicious or attack traffic was automatically simulated, or simulated by actual users if the attack was too complex to be automated. The generated legitimate traffic and the simulated attack traffic in the network environment, were captured/recorded by the sniffer. Over a period of seven weeks, the sniffer captured 4 gigabytes of compressed, raw network packets (`TCPdump` data).

3.2.1.1 Attacks

The attacks that were simulated during the DARPA IDS evaluation programme all belong to one of four broader categories of network attacks, as proposed by DARPA [91]:

- **Denial-of-Service (DoS)** — This form of attack occurs when an attacker overloads or exhausts a system resource (such as available memory or bandwidth) through service requests. This renders the system unable to accommodate new requests from legitimate users. An example is the SYN flood (Neptune) attack which involves the transmission of a large number of SYN packets to a host system. A SYN packet initialises a TCP connection between the attacker and the server, preparing the server for data transmission. The server replies with a SYN/ACK packet and waits for an ACK packet from the attacker in order to complete the new connection. However, the attacker does not reply with an ACK packet leaving the connection half open. A server can only accommodate a finite number of half open connections, and once that finite number is reached, new legitimate connections cannot be established.
- **User to Root (U2R)** — In this category of attack, an attacker is assumed to already have user-level access to a target system in the network or intranet. This user-level access is obtained through legitimate or illegitimate means, such as a brute-force attack on user passwords. The attack consists of the attacker escalating his/her access level to gain root access (administrator-level access), by exploiting some vulnerability within the system.
- **Remote to Local (R2L)** — This occurs when an attacker remotely gains access to a local system within a network or intranet, that he/she is not entitled to access, by exploiting some vulnerability within the system. This is achieved by sending packets to the local machine over a network from a remote location.
- **Probing** — This involves an attacker attempting to gather private information about a network with the aim of finding a security flaw or vulnerability in networked systems. This includes finding target hosts, and specific hosts with open ports, which reveal network services that can be exploited. It is generally carried out as an initial step, prior to executing an attack from one of the remaining categories.

3.2.1.2 Features

Stolfo et al. [92] and Lee et al. [93, 94] specified an initial set of features obtained from information extracted from the raw data. The initial set of features were constructed for each connection record. A connection is any form of communication attempt between two hosts, a source and a destination. Each connection record is therefore represented as a vector consisting of feature values that describe a specific connection of interest. Certain features associated with specific connections of interest, were extracted from packet headers and payloads. Other features involve statistics calculated over multiple time frames and multiple connections related to the connection of interest. An example includes the number of connections to the same host or service as the current connection of interest. Stolfo et al. [92] and Lee et al. [93, 94] divided the initial set of features into four categories.

Basic features of individual TCP connections.

These represent the general features of TCP connections that are derived from packet headers, such as the amount of data transmitted during a connection, the duration of a connection, the ports used during a connection, etc. These features are not specifically derived for intrusion detection, but are commonly used for general network analysis [93].

Content features within a connection derived from the payload.

Content-based features are derived from the payloads of packets. They characterise the actions of a user, such as, the number of failed login attempts, the number of files created, the number of “root” accesses, etc. Content-based features allow for the identification of R2L and U2R attacks. R2L and U2R attacks generally occur over one single connection and are embedded in the packet payloads, unlike DoS and probe attacks which occur over multiple connections.

Traffic features computed using a two-second time window.

The motivation for these features is that attacks frequently exhibit patterns over different time frames, which can be used to detect certain attacks. For example, DoS and certain probe attacks involve the transmission of a large number of packets to a host in a very short period of time. These features are derived from statistics over a two second time frame, related to the connection of interest, such as, the number of connections to the same host or service as the current connection of interest.

Traffic features computed using the last 100 connections.

Certain attacks involve the transmission of data over extended periods of time. These attacks may in some cases be identified using features that are derived over multiple connections related to a connection of interest. These features are derived from statistics that are calculated over 100 connections related to a connection of interest, such as, the number of connections to the same host or service as the current connection of interest.

Table 3.1 summarises the attack categories of those attacks that may be detected by each feature category, as demonstrated in [92–94].

Table 3.1: *Attack categories detectable by the four feature categories.*

Feature categories	**Attack categories**
Basic and traffic features	DoS and fast probing attacks
Basic and connection-based traffic features	Slow probing attacks
Basic and content-based features	R2L and U2R attacks

Table 3.2 provides a list of all 41 features of the KDD Cup 1999 dataset [24], along with a description and the type of each feature. The 41 features consists of 32 numeric features (16 with continuous values and 16 with discrete values), and 9 categorical features (6 with two categories and 3 with several categories).

TABLE 3.2: *List of features contained in the KDD Cup 1999 dataset (from [5–7]).*

#	Feature Name	Description	Type
		Basic features of individual TCP connections	
1	duration	length (number of seconds) of the connection	Numeric (Continuous)
2	protocol_type	type of the protocol, e.g. TCP, UDP, etc.	Categorical (3-cat.)
3	service	network service on the destination, e.g., HTTP, telnet, etc.	Categorical (11-cat.)
4	flag	normal or error status of the connection	Categorical (64-cat.)
5	src_bytes	number of data bytes from source to destination	Numeric (Discrete)
6	dst_bytes	number of data bytes from destination to source	Numeric (Discrete)
7	land	1 if connection is from/to the same host/port; 0 otherwise	Categorical (Binary)
8	wrong_fragment	number of "wrong" fragments	Numeric (Discrete)
9	urgent	number of urgent packets	Numeric (Discrete)
		Content features within a connection derived from the payload	
10	hot	number of "hot" indicators	Numeric (Discrete)
11	num_failed_logins	number of failed login attempts	Numeric (Discrete)
12	logged_in	1 if successfully logged in; 0 otherwise	Categorical (Binary)
13	num_compromised	number of "compromised" conditions	Numeric (Discrete)
14	root_shell	1 if root shell is obtained; 0 otherwise	Categorical (Binary)
15	su_attempted	1 if "su root" command attempted; 0 otherwise	Categorical (Binary)
16	num_root	number of "root" accesses	Numeric (Discrete)
17	num_file_creations	number of file creation operations	Numeric (Discrete)
18	num_shells	number of shell prompts	Numeric (Discrete)
19	num_access_files	number of operations on access control files	Numeric (Discrete)
20	num_outbound_cmds	number of outbound commands in an ftp session	Numeric (Discrete)
21	is_host_login	1 if the login belongs to the "host" list; 0 otherwise	Categorical (Binary)
22	is_guest_login	1 if the login is a "guest" login; 0 otherwise	Categorical (Binary)
		Traffic features computed using a two-second time window	
23	count	number of connections to the same host as the current connection	Numeric (Discrete)
24	srv_count	number of connections to the same service as the current connection	Numeric (Discrete)
25	serror_rate	% of connections that have "SYN" errors in the count feature	Numeric (Continuous)
26	srv_serror_rate	% of connections that have "SYN" errors in the srv count feature	Numeric (Continuous)
27	rerror_rate	% of connections that have "REJ" errors in the count feature	Numeric (Continuous)
28	srv_rerror_rate	% of connections that have "REJ" errors in the srv count feature	Numeric (Continuous)
29	same_srv_rate	% of connections to the same service in the count feature	Numeric (Continuous)
30	diff_srv_rate	% of connections to different services in the count feature	Numeric (Continuous)
31	srv_diff_host_rate	% of connections to different hosts in the srv count feature	Numeric (Continuous)
		Traffic features computed using the last 100 connections	
32	dst_host_count	number of connections to the same host as the current connection	Numeric (Discrete)
33	dst_host_srv_count	number of connections to the same service as the current connection	Numeric (Discrete)
34	dst_host_same_srv_rate	% of connections to the same service in the dst_host_count feature	Numeric (Continuous)
35	dst_host_diff_srv_rate	% of connections to different services in the dst_host_count feature	Numeric (Continuous)
36	dst_host_same_src_port_rate	% of connections whose source port is the same to that of the current connection in the dst_host_count feature	Numeric (Continuous)
37	dst_host_srv_diff_host_rate	% of connections to different hosts in the dst_host_srv_count feature	Numeric (Continuous)
38	dst_host_serror_rate	% of connections that have "SYN" errors in the dst_host_count feature	Numeric (Continuous)
39	dst_host_srv_serror_rate	% of connections that have "SYN" errors in the dst_host_srv_count feature	Numeric (Continuous)
40	dst_host_rerror_rate	% of connections that have "REJ" errors in the dst_host_count feature	Numeric (Continuous)
41	dst_host_srv_rerror_rate	% of connections that have "REJ" errors in the dst_host_srv_count feature	Numeric (Continuous)

The KDD Cup 1999 dataset [24] consists of training and testing subsets; specifically, the dataset consists of,

- a training set with 5 million records,
- a 10% subset of the training set, randomly selected, with 500 000 records, and
- a test set with 2 million records.

A detailed overview of the composition of these subsets are not provided, as the NSL-KDD dataset [25], which was derived from the KDD Cup 1999 dataset, was used in this research.

3.2.1.3 Shortcomings of the KDD Cup 1999 dataset

The KDD Cup 1999 dataset [24] is a widely used dataset for evaluating the performance of anomaly-based network intrusion detection systems [43, 45, 46, 48–50, 78]. However, the dataset has several characteristics which detract from its fitness as a benchmark dataset for performance evaluation. Tavallaee et al. [95] performed a detailed analysis on the KDD Cup 1999 dataset in order to identify and address the shortcomings of the dataset for the purpose of serving as a benchmark dataset. The study points out two main shortcomings of the KDD Cup 1999 dataset.

The first is the presence of redundant records, where the full training and testing sets contain approximately 80% and 75% duplicated records, respectively [95]. These duplicates occur due to the nature of certain attacks such as Denial-of-Service (DoS) and Probing attacks [34]. While the existence of duplicates is representative of real network traffic, their presence introduces a bias towards those attack classes when the dataset is applied to machine learning algorithms. If a significantly large proportion of the dataset consists of duplicates and all duplicates are classified correctly, this will result in a high classification accuracy, whereas if they are classified incorrectly, this results in a low classification accuracy. Both scenarios may be misleading as they may not indicate the classification or detection capabilities of certain classifiers with regards to detecting less frequent attacks, such as R2L and U2R attacks [34, 95].

The second shortcoming is that the majority of the connection records are trivial to classify, this implies that the results reported in certain articles would typically result in high classification rates in certain classifiers [95]. This shortcoming detracts from the detection of more sophisticated attacks, which reduces the ability of the KDD Cup 1999 dataset [24] to serve as a benchmark dataset for intrusion detection. In order to demonstrate the second shortcoming, Tavallaee et al. [95] performed experiments using seven supervised machine learning algorithms that were applied to the KDD Cup 1999 dataset. Experiments were run using the WEKA software package [96] using the following algorithms:

- J48 decision tree [97],
- Naive Bayes (NB) [98],
- NBTree [99],
- random forest [100],
- random tree [101],
- multilayer perceptron [102], and
- support vector machines [103].

Tavallaee et al. [95] trained each of the seven supervised machine learning algorithms three times using three different training sets. The three training sets were randomly selected subsets of the full KDD training dataset, each consisting of 50 000 connection records. This process produced 21 classifiers, which were applied to classify each of the remaining connection records of the full KDD training and testing sets as either a legitimate connection, or a specific attack (from one of the classes defined in section 3.2.2, in table 3.4). The results indicated that each of the 21 classifiers were able to correctly classify 98% and 87% of all records for both the training and test sets, respectively. This demonstrates that the majority of the attacks are elementary attacks, which are trivial to detect when using the implementation in [95].

3.2.2 NSL-KDD dataset

To address the shortcomings of the KDD Cup 1999 dataset [24], Tavallaee et al. [95] proposed a new dataset known as the NSL-KDD dataset [25], which is a subset of the KDD Cup 1999 dataset. The NSL-KDD dataset resolves the shortcomings of the KDD Cup 1999 dataset by eliminating duplicated records and retaining only certain connection records in order to obtain a distribution that consists of a larger proportion of sophisticated attacks, making the dataset more challenging for intrusion detection tasks, and thus more appropriate for use as a benchmark dataset for intrusion detection [95]. In this research, the NSL-KDD dataset is utilized in all simulations.

The NSL-KDD dataset was created in two steps:

1. The initial step consisted of the removal of all duplicated records, retaining only one copy of each distinct connection record. This reduced the full KDD training and testing sets to around 20% and 25% of their original size, respectively.

2. Tavallaee et al. [95] grouped the connection records based on the number of classifiers (as specified in section 3.2.1) that were able to correctly classify each record. The groups were as follows, those records that were correctly classified by: 0-5 classifiers, 6-10 classifiers, 11-15 classifiers, 16-20 classifiers, and 21 classifiers, were grouped together. To obtain the NSL-KDD dataset [25], Tavallaee et al. [95], randomly selected a proportion of connection records from each group, where each proportion consisted of a number of connection records that constitute an inversely proportional percentage of connection records present in the KDD Cup 1999 training and test sets, after duplicates were removed. For example, after the removal of duplicates, the KDD Cup 1999 dataset [24] consisted of 0.04% of those connection records belonging to group 0-5. Thus, 99.96% of those records belonging to group 0-5, were randomly selected for the NSL-KDD dataset. Thus, the NSL-KDD dataset constitutes a larger proportion of those attacks that are harder to correctly classify, as compared to the KDD Cup 1999 dataset.

The NSL-KDD dataset consists of two training sets and two test sets:

- **Training set 1**. This training set was derived by applying steps 1 and 2, as set out above, to the full KDD Cup 1999 training set. It has 125 973 connection records.
- **Training set 2**. This training set was created by randomly selecting 20% of the connection records of NSL-KDD training set 1. It has 25 192 records.
- **Test set 1**. This test set was derived by applying steps 1 and 2, as set out above, to the full KDD Cup 1999 test set. It has 22 544 connection records.
- **Test set 2**. This test set is a subset of NSL-KDD test set 1, created by removing **all** connection records with a rank of 21. It has 11 850 records. The records of test set 2 are considered more difficult to classify correctly than the records in test set 1.

Table 3.3 summarises the contents of the training and test sets of the NSL-KDD dataset [25], detailing the number of connection records as well as the distribution of records that are associated with legitimate network traffic, and the four network attack categories (as defined in section 3.2.1.1).

Table 3.3: *Contents of the training and testing sets of the NSL-KDD Dataset.*

	Training set 1		**Training set 2**		**Test set 1**		**Test set 2**	
DoS	45 927	36.46%	9 234	36.65%	7 475	33.16%	4 359	36.78%
Probe	11 656	9.25%	2 289	9.09%	2 421	10.74%	2 402	20.27%
R2L	995	0.79%	209	0.83%	2 870	12.73%	2 870	24.22%
U2R	52	0.04%	11	0.04%	67	0.30%	67	0.57%
Normal	67 343	53.46%	13 449	53.39%	9 711	43.08%	2 152	18.16%
Total	**125 973**	**100%**	**25 192**	**100%**	**22 544**	**100%**	**11 850**	**100%**

A comprehensive list of all the attacks in the NSL-KDD training and testing sets is provided in table 3.4. Each attack is grouped into one of the four broader attack categories (as specified in section 3.2.1.1), and the percentage of records within the dataset belonging to each attack is provided. The information in table 3.4 was gathered from the KDD Cup 1999 dataset website [5], from an analysis of the class labels and from references [104, 105], which provide a list of attacks. The interested reader can refer to [106] for a description of each attack and the steps involved in their execution.

Table 3.4: *List of attacks contained in the NSL-KDD dataset, and percentage of connection records belonging to each attack.*

Training Set 1		**Training Set 2**		**Test Set 1**		**Test Set 2**		**Attack Category**
back	0.76%	back	0.78%	back	1.59%	back	3.03%	DoS
land	<0.1%	land	<0.1%	land	<0.1%	land	<0.1%	DoS
neptune	32.72%	neptune	32.88%	neptune	20.66%	neptune	13.32%	DoS
pod	0.16%	pod	0.15%	pod	0.18%	pod	0.35%	DoS
smurf	2.10%	smurf	2.10%	smurf	2.95%	smurf	5.29%	DoS
teardrop	0.71%	teardrop	0.77%	teardrop	<0.1%	teardrop	0.10%	DoS
ipsweep	2.86%	ipsweep	2.82%	ipsweep	0.63%	ipsweep	1.19%	Probe
nmap	1.19%	nmap	1.19%	nmap	0.32%	nmap	0.62%	Probe
portsweep	2.33%	portsweep	2.33%	portsweep	0.70%	portsweep	1.32%	Probe
satan	2.88%	satan	2.74%	satan	3.26%	satan	6.14%	Probe
ftp_write	<0.1%	ftp_write	<0.1%	ftp_write	<0.1%	ftp_write	<0.1%	R2L
guess_passwd	<0.1%	guess_passwd	<0.1%	guess_passwd	5.46%	guess_passwd	10.39%	R2L
imap	<0.1%	imap	<0.1%	imap	<0.1%	imap	<0.1%	R2L
multihop	<0.1%	multihop	<0.1%	multihop	<0.1%	multihop	0.15%	R2L
phf	<0.1%	phf	<0.1%	phf	<0.1%	phf	<0.1%	R2L
spy	<0.1%	spy	<0.1%	—	—	—	—	R2L
warezclient	0.71%	warezclient	0.72%	—	—	—	—	R2L
warezmaster	<0.1%	warezmaster	<0.1%	warezmaster	4.19%	warezmaster	7.97%	R2L
buffer_overflow	<0.1%	buffer_overflow	<0.1%	buffer_overflow	<0.1%	buffer_overflow	0.17%	U2R
loadmodule	<0.1%	loadmodule	<0.1%	loadmodule	<0.1%	loadmodule	<0.1%	U2R
perl	<0.1%	—	—	perl	<0.1%	perl	<0.1%	U2R
rootkit	<0.1%	rootkit	<0.1%	rootkit	<0.1%	rootkit	0.11%	U2R
				apache2	3.27%	apache2	6.22%	DoS
				mailbomb	1.30%	mailbomb	2.47%	DoS
				named	<0.1%	named	0.14%	DoS
				processtable	3.04%	processtable	5.78%	DoS
				udpstorm	<0.1%	udpstorm	<0.1%	DoS
				mscan	4.42%	mscan	8.41%	Probe
				saint	1.42%	saint	2.61%	Probe
				httptunnel	0.59%	httptunnel	1.12%	R2L
				sendmail	<0.1%	sendmail	0.12%	R2L
				snmpgetattack	0.79%	snmpgetattack	1.50%	R2L
				snmpguess	1.47%	snmpguess	2.79%	R2L
				worm	<0.1%	worm	<0.1%	R2L
				xlock	<0.1%	xlock	<0.1%	R2L
				xsnoop	<0.1%	xsnoop	<0.1%	R2L
				ps	<0.1%	ps	0.13%	U2R
				sqlattack	<0.1%	sqlattack	<0.1%	U2R
				xterm	<0.1%	xterm	0.11%	U2R

Table 3.4 indicates that training set 1 has 22 attacks, whereas training set 2, the 20% subset of training set 1, has 21 attacks, with the absence of *perl* records. Both test sets have 37 attacks, in which 20 attacks are found in the training sets, and 17 are additional attacks.

Each feature vector of all training and testing sets consists of the same 41 features as defined in the KDD Cup 1999 dataset [24], as well as a class label for each record in the dataset. Each record is labelled as either one specific attack, as set out in table 3.4, or *normal*, which represents legitimate network traffic. Note that the class labels are only used to evaluate performance after classification, and not used as a feature during classification.

3.3 Data transformation

In general, data transformation involves converting the values of data samples into a form that is suitable for machine learning tasks [23]. It involves the scaling of numeric feature values to fall within a desired range (normalization), as well as the conversion of categorical feature values into numeric values, as may be required by certain machine learning tasks (encoding). Several articles have reviewed various methods for normalizing and encoding the data of the features of the NSL-KDD dataset [25], and their impact on the classification performance of several anomaly-based NIDS [107, 108]. This section provides an overview of the procedure used in this research to transform the NSL-KDD dataset into a format that is suitable for clustering feature values associated with each data sample.

3.3.1 Normalization

The NSL-KDD dataset [25] contains numeric features that are defined over ranges with different extent. This disparity in magnitude may lead to the emergence of a bias towards certain features, during for example, distance calculations in clustering algorithms. Normalization is a technique that is employed to ensure that the features are defined over a common range, with no bias towards certain features, as a result of disparity of scale [38].

Several normalization techniques that can be used to normalize numerical features include [23, 107, 108]:

- mean range normalization,
- frequency normalization,
- maximize normalization,
- rational normalization,
- statistical normalization (standardisation),
- ordinal normalization, and
- decimal scaling normalization.

Wang et al. [108] performed a comparison of the performance impact of several normalization techniques as applied to network intrusion detection over the KDD Cup 1999 dataset [24]. Three supervised learning methods, namely, k-nearest neighbour (kNN), Principal Component Analysis (PCA) and Support Vector Machines (SVMs) were trained and tested on subsets of the KDD Cup 1999 dataset. The training set consisted of only legitimate network traffic to build "normal" models, and the testing set consisted of both legitimate and attack traffic, in which attacks were detected as those samples that deviated from the "normal" models. PCA was implemented as a classifier by projecting each test sample onto the subspace found by the PCA algorithm applied to the training set, that represents normal behaviour. The test sample is labelled as legitimate if the distance between the test sample and its reconstruction onto the subspace is below a threshold. Prior to applying machine learning, the numeric features of the KDD Cup 1999 dataset were normalized using one of the following normalization techniques: mean range, statistical, ordinal and frequency normalization. The categorical features were not included in the experiment. The authors suggested that statistical and mean range normalization should be used if distance-based computations are required, and in situations involving large datasets, statistical normalization is recommended, as it provides the best overall performance.

Statistical normalization was successfully applied in several anomaly-based NIDS that involve distance-based calculations [43, 44]. Both Syarif et al. [44] and Portnoy et al. [43] used clustering techniques to perform network intrusion detection. Both authors applied statistical normalization on the numeric features of the NSL-KDD dataset [25] and KDD Cup 1999 dataset [24], respectively. Chapter 2, section 2.3.1 describes these systems in greater detail.

In this research the statistical normalization technique is applied to the numeric features of the NSL-KDD dataset [25]. The technique rescales each numeric feature such that it has a mean value of zero and a unity standard deviation when calculated over all samples of the dataset. The value $x_{i,f}$ of feature f of data sample i is scaled as,

$$x'_{i,f} = \frac{x_{i,f} - \mu_f}{\sigma_f}, \quad i = 1 \ldots N \tag{3.1}$$

where $x'_{i,f}$ is the scaled value, N is the number of data samples and μ_f and σ_f are the mean and standard deviation of the values of feature f, respectively. Both the mean and standard deviation are calculated as,

$$\mu_f = \frac{1}{N}\sum_{i=1}^{N} x_{i,f} \tag{3.2}$$

$$\sigma_f = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_{i,f} - \mu_f)^2} \tag{3.3}$$

3.3.2 Encoding

This section describes the technique that was used to encode the categories of the categorical features present in the NSL-KDD dataset [25]. Encoding consists of the mapping of categories to corresponding numerical values. Several encoding techniques have been proposed in the literature. Table 3.5 lists several of the techniques that have been successfully applied in the context of anomaly-based network intrusion detection, together with classifiers used with each technique, and the normalization technique that may have been applied to the encoded categories of the categorical features.

Table 3.5: *Summary of encoding techniques.*

Encoding Technique	**Reference**	**Classifier**	**Normalization**
Ordinal	[26]	Least Squares SVM (Chapter 2, section 2.4)	Maximize
	[48]	NADO (Chapter 2, section 2.3.2)	None
	[29]	TASVM (Chapter 2, section 2.4)	Maximize
	[109]	SVM	Mean Range
Binary	[110]	Extreme learning machines (ELM)	None
	[111]	Decision tree, kNN, Multi-layer perceptron, regularized discriminant analysis, Fisher linear discriminant, k-means, single linkage clustering, quarter-sphere SVM, γ-algorithm	Statistical
	[112]	Parzen-window estimators with Gaussian kernels	None
Frequency	[107]	Random forest, Bayes net, naive Bayes (NB), NB tree and decision trees.	None
	[113, 114]	Transductive Confidence Machines k-Nearest Neighbour (TCM-kNN)	None

To illustrate each technique suppose that a categorical feature has N_{cat} possible categories.

Ordinal encoding

In ordinal encoding, each of the N_{cat} categories are mapped to a distinct integer in the set $0, 1, \ldots, N_{cat} - 1$. Amiri et al. [26] encode the categories of the categorical features of the KDD Cup 1999 dataset [24] as follows; the *protocol type* feature (for instance) consists of three categories: *TCP*, *UDP* and *ICMP*; authors assign integer values to these categories by setting the category *TCP* to 1, *UDP* to 2, and *ICMP* to 3. The transformed categorical data as well as the numerical data of the KDD Cup 1999 dataset, are subsequently normalized using the maximize normalization technique.

The ordinal encoding technique is inappropriate as the clustering techniques of interest use distances over the feature space. If ordinal encoding is used together with a distance metric, an implicit measure of dissimilarity is assigned to categories that may not be accurate or appropriate. The implication is that (for instance) categories mapped to 1 and 3 are more dissimilar than categories mapped to 1 and 2.

Binary encoding

In binary encoding, each of the N_{cat} categories are mapped to a distinct N_{cat}-bit number, where a single bit is nonzero. Authors of [110–112] encode the categories of the categorical features of the KDD Cup 1999 dataset [24] using binary encoding, where the categories of the *protocol type* feature (for instance) are encoded as follows; *TCP*, *UDP* and *ICMP* are encoded as $(0, 0, 1)$, $(0, 1, 0)$, and $(1, 0, 0)$, respectively.

The drawback of this encoding technique is that it increases the dimensionality of the dataset. Essentially, one feature becomes N_{cat} features. Encoding the categories of the categorical features of the NSL-KDD dataset [25] to binary numbers would drastically increase the dimensionality of the dataset, which enlarges the search space for feature selection.

Frequency encoding

In frequency encoding, each category is mapped to a real number between 0 and 1 that represents the fraction of occurrences of the category in the dataset. A category of categorical feature f, is encoded as,

$$Cat'_{j,f} = \frac{\sum_{i=1}^{N} \mathbb{1}(x_{i,f} = Cat_{j,f})}{N}, \quad j = [1, 2, \ldots, N_{cat}] \tag{3.4}$$

where $Cat_{j,f}$ represents category j of feature f, $Cat'_{j,f}$ is the encoded value of category j of feature f, N is the number of data samples, and N_{cat} is the number of categories in feature f.

Li et al. [113, 114] encoded the categories of the categorical features of the KDD Cup 1999 dataset [24], using equation 3.4, to perform network intrusion detection.

Due to the disadvantages of the ordinal and binary encoding methods, in this research, the frequency-based encoding method is used to encode the data of the non-binary categorical features of the NSL-KDD dataset [25].

It is noteworthy that some anomaly-based NIDS proposed in the literature exclude the binary-valued categorical features of the KDD Cup 1999 dataset [24] and NSL-KDD dataset [25]. This is not considered as an option in this research as it has been demonstrated previously that several of the binary-valued categorical features have value in distinguishing between the attack categories present in both datasets. In particular references [7, 28, 29] suggest that the *land* feature is valuable for identifying *DoS* attacks while references [26, 28] suggest that the *root shell* feature is valuable for identifying *U2R* attacks. Li et al. [31] showed that, according to their ranking algorithm, the *land* feature was the third most significant feature out of all 41 features of the KDD Cup 1999 dataset.

In this research, the binary-valued categorical features were encoded using the ordinal encoding technique, where values were encoded as either a 0 or a 1. Further normalization of the encoded binary-valued categorical features was not considered as the feature values are within the same range as that of the encoded non-binary categorical features of the NSL-KDD dataset [25]. This is the same approach as used in references [48, 107]

3.4 Clustering

This section describes each of the clustering algorithms that were used in this research. The goal of clustering is to find hidden structures or regularities among the data samples in the feature space considered. Clustering algorithms assign data samples in the feature space to groups or clusters, such that data samples that are in a common region of the feature space belong to the same cluster. In applying clustering in the context of classification, the assumption is that data samples belonging to the same cluster will belong to the same class. Clustering may be done based on (among others), the distances between data samples, the relative density of data samples, or according to distributional models such as a Gaussian mixture model, all computed over the feature space considered [2, 63].

The clustering algorithms described in this section are all sensitive to the choice of initial cluster centres, where different initial centres produce different results. In the implementation of the classifier, the clustering algorithm is repeatedly applied to the data, with a different choice of initial representative points for each cluster on each iteration. A total of C_R repetitions is performed, and during each repetition, the clustering algorithm is executed until it converges. Each of the C_R clustering results obtained, are subsequently labelled (refer to figure 3.1).

3.4.1 Centroid-based clustering

Centroid-based clustering involves the use of cluster centres or centroids to represent each cluster [23]. A cluster centre is a feature vector which defines the centre of a cluster in the feature space considered. A cluster centre may or may not correspond to an actual data sample of the dataset. Data samples are assigned to the cluster centre that they are closest to, based on some measure of proximity that is calculated over the feature space considered. Typically the Euclidean distance is used as the proximity measure, however, other measures that can be used include the Manhattan distance, Minkowski distance, and Chebyshev distance [23].

3.4.1.1 K–means

K–means is a widely used centroid-based clustering technique for grouping data samples together based on distances over the feature space, while maintaining a separation between samples that are relatively distant from one another with regards to the feature space. The name k–means was first introduced by James MacQueen in 1967 [115], but the algorithm itself was derived from the work of Hugo Steinhaus in 1956 [116].

The pseudocode for the k–means algorithm is provided in algorithm 3.1.

Algorithm 3.1 K–means clustering algorithm

```
    Function $[\mathbf{C},\mathbf{L}] = kmeans(\mathbf{X}, K, \mathbf{F_s})$
    Inputs:
        $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$,    ▷ A set of feature vectors to be clustered
        $K$,    ▷ The number of clusters
        $\mathbf{F_s}$,    ▷ A feature subset to perform clustering over
    Outputs:
        $\mathbf{C} = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_K}\}$,    ▷ Set of feature vectors representing cluster centres
        $\mathbf{L} = \{l(\mathbf{x_a}) | a = 1, 2, \ldots, N\}$,    ▷ Set of cluster assignments for X
    ▷ Initialization
 1: $\mathbf{C} \leftarrow K$ random samples drawn uniformly from dataset $\mathbf{X}$, without replacement
    ▷ Assign data samples in $\mathbf{X}$ to cluster centres in $\mathbf{C}$
 2: for $(\mathbf{x_i} \in \mathbf{X})$ do
 3:     $l(\mathbf{x_i}) \leftarrow \arg\min_j [Distance(\mathbf{x_i}, \mathbf{c_j})]$ ;    ▷ where $j = \{1, \ldots, K\}$
 4: end for
 5: repeat
 6:     $changed \leftarrow false$ ;
 7:     for all $(\mathbf{c_i} \in \mathbf{C})$ do
 8:         $\mathbf{c_i} \leftarrow mean(\mathbf{x} \in \mathbf{X} \mid l(\mathbf{x}) = i)$ ;    ▷ Recompute cluster centres
 9:     end for
    ▷ Reassign data samples to closest cluster centre
10:     for all $(\mathbf{x_i} \in \mathbf{X})$ do
11:         $minDist\_index \leftarrow \arg\min_j [Distance(\mathbf{x_i}, \mathbf{c_j})]$ ;    ▷ where $j = \{1, \ldots, K\}$
12:         if $minDist\_index \neq l(\mathbf{x_i})$ then
13:             $l(\mathbf{x_i}) \leftarrow minDist\_index$ ;
14:             $changed \leftarrow true$ ;
15:         end if
16:     end for
17: until $changed = false$
18: Return $\mathbf{C}, \mathbf{L}$
```

The algorithm partitions the data into K clusters, where K is a user-specified parameter. First, K random data samples are chosen from the dataset, as the initial cluster centres. Each data sample is then assigned to the cluster centre that it is closest to based on the squared Euclidean distance from the data sample to each cluster centre. The k–means algorithm iteratively recalculates the K cluster centres as the mean of all the samples belonging to the k^{th} cluster, where the mean is computed over each feature in the feature space. The data samples are subsequently reassigned to the new cluster centres, and the process is repeated until the cluster centres remain unchanged.

The k–means algorithm is computationally efficient and scalable for application to relatively large datasets. It often converges to a local optimum and is not guaranteed to converge to a global optimum [23]. It is also able to find spherical or convex shaped clusters [2]. The disadvantage of k–means clustering is that it is sensitive to outliers in the dataset. Outliers affect the positioning of cluster centres in the feature space as they possess large feature values that disproportionately affect the movement of cluster centres in the feature space, during clustering [2, 23].

3.4.1.2 K–medoids

The k–medoids algorithm is a centroid-based clustering algorithm and is similar to the k–means algorithm. The difference lies in the recalculation of cluster centres during each iteration [2, 23, 63]. K–medoids selects cluster centres as those data samples belonging to a cluster, that possesses the minimum summed distance to all remaining samples belonging to the same cluster. The samples that are selected as cluster centres are known as *medoids*. The k–means algorithm, however, recalculates cluster centres as the mean of all the data samples assigned to the same cluster centre. The pseudocode for the k–medoids clustering algorithm is provided in algorithm 3.2.

The algorithm first initializes K medoids and subsequently iterates through several assignment and recalculation steps until convergence. The initial cluster medoids (centres) are selected as K random samples from the dataset. The remaining data samples are subsequently assigned to the cluster medoids to which they are closest to based on the squared Euclidean distance computed over the feature space. New cluster medoids are found by

Algorithm 3.2 K–medoids clustering algorithm

```
Function $[\mathbf{C}, \mathbf{L}] = kmedoids(\mathbf{X}, K)$
Inputs:
  $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$,   ▷ A set of feature vectors to be clustered
  $K$,   ▷ The number of clusters
Outputs:
  $\mathbf{C} = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_K}\}$,   ▷ Set of feature vectors representing cluster centres/medoids
  $\mathbf{L} = \{l(\mathbf{x_a}) | a = 1, 2, \ldots, N\}$   ▷ Set of cluster assignments for $\boldsymbol{X}$

1: $\mathbf{C} \leftarrow K$ random samples drawn uniformly from dataset $\mathbf{X}$, without replacement ;
2: repeat
3:   $changed \leftarrow false$ ;
4:   for all $(\mathbf{x_i} \in \mathbf{X})$ do   ▷ Assign data samples to the closest medoid
5:     $l(\mathbf{x_i}) \leftarrow \arg\min_j [Distance(\mathbf{x_i}, \mathbf{c_j})]$ ;   ▷ where $j = \{1, \ldots, K\}$
6:   end for
7:   for $i = 1$ to $K$ do
8:     $\mathbf{H_i} \leftarrow (\mathbf{x} \in \mathbf{X} \mid l(\mathbf{x}) = i)$ ;   ▷ Data samples belonging to cluster $i$
9:     for all $(\mathbf{h} \in \mathbf{H_i})$ do
10:      $sum\_dist \leftarrow 0$ ;
11:      for all $(\mathbf{x} \in \mathbf{H_i})$ do   ▷ $\mathbf{x} \neq \mathbf{h}$
12:        $sum\_dist \leftarrow sum\_dist + dist(\mathbf{x}, \mathbf{h})$ ;
13:      end for
14:      $sum(\mathbf{h}) \leftarrow sum\_dist$ ;
15:    end for
16:    $index \leftarrow \arg\min_{\mathbf{h} \in \mathbf{H_i}} [sum(\mathbf{h})]$ ;   ▷ Data sample with the smallest summed
17:    $\mathbf{c_{tmp}} \leftarrow \mathbf{H_i}(index)$ ;   ▷ distance to all samples in cluster $i$
18:    if $(\mathbf{c_{tmp}} \neq \mathbf{c_i})$ then   ▷ If the candidate medoid $c_{tmp}$, is different from the
19:      $\mathbf{c_i} \leftarrow \mathbf{c_{tmp}}$ ;   ▷ the previous medoid $c_i$, then the candidate medoid
20:      $changed \leftarrow true$ ;   ▷ becomes the new cluster medoid
21:    end if
22:  end for
23: until $changed = false$

24: Return $\mathbf{C}, \mathbf{L}$
```

computing the sum of the distances from each sample belonging to a cluster to each remaining sample belonging to the same cluster. The sample that possesses the minimum summed distance to all remaining samples belonging to the same cluster, is selected as the new cluster medoid. The process is repeated until the cluster medoids do not change after an iteration.

The disadvantage of the k–medoids clustering algorithm is that it has a greater average computational complexity than the k–means clustering algorithm during cluster centre or medoid reselection [2, 23, 63]. The k–medoids algorithm requires the pairwise squared Euclidean distances between all data samples belonging to a cluster to be computed on each iteration. The advantage is that it is more resilient to outliers than k–means. In the k–means clustering algorithm, outliers disproportionately affects the location of cluster centres in the feature space, during clustering, given that cluster centres are computed as the mean of all data samples belonging to the same cluster. The location of cluster centres or medoids with regards to the feature space, of the k–medoids algorithm, are not as severely influenced by outliers. The algorithm will reject the use of an outlier as a medoid, as an outlier lies in a region of the feature space that is distant from the remaining samples with regards to the feature space. This will result in a large summed distance between an outlier and all remaining samples belonging to the same cluster as the outlier.

3.4.1.3 K–means with distance-based outlier detection

Outlier detection is a technique that is used to find samples that are significantly different from the majority of the samples within a dataset, for instance based on proximity measures such as distance or density in the feature space. These samples are known as outliers [23]. In anomaly-based network intrusion detection, anomalies are described as traffic patterns that are vastly different from the traffic patterns of what is assumed to be legitimate network activity, with regards to the statistics of features and feature values computed over the feature space. These anomalies may be the result of malicious activity and may indicate that an intrusion has occurred on the network. Anomalies can be considered as outliers in network traffic, and thus identifying outliers is of interest [2]. There exists a number of ways in which outlier detection may be incorporated in anomaly-based

NIDS. Outlier detection can be combined with clustering algorithms, which may improve the performance of the clustering algorithm [44]. For example, the k–means clustering algorithm is known to be sensitive to outliers, in the sense that they may disproportionately shift the location of cluster centres in the feature space. By removing outliers prior to clustering, this effect no longer occurs.

The two widely used proximity-based outlier detection techniques are density-based and distance-based methods [23]. Density-based detection methods estimate the density of data samples surrounding a particular data sample, and the density of data samples surrounding its neighbours. An outlier is identified as a data sample with a relatively lower density of surrounding data samples than that of the density of data samples surrounding its neighbours. Distance-based detection methods identify outliers as those data samples whose set of neighbours, defined by a given radius, are distant from it, where the radius is defined based on a distance measure computed over the feature space [23]. For instance, Ramaswamy et al. [117] defines outliers as the N_{out} data samples with the largest distance to their k^{th} nearest neighbour.

In what follows, the algorithm for detecting outliers is provided, followed by the algorithm in which it is combined with clustering. The pseudocode for identifying outliers using Ramaswamy's k^{th} nearest neighbour distance-based outlier detection approach is provided in algorithm 3.3, further details can be found in [118]. In order to avoid confusion with previous algorithms, where the number of clusters is represented by the symbol K; the symbol N_n is used in algorithm 3.3 to represent k in the k^{th} nearest neighbour approach, where N_n represents the number of nearest neighbours to consider.

The technique implemented in this research involves the combination of outlier detection with clustering, and was derived from [69], where the difference lies in the manner in which outliers are identified. The implementation in this research utilizes algorithm 3.3 to identify outliers, whereas the implementation in [69] utilizes distances to cluster centres to identify outliers.

Hautamaki et al. [69] proposed an Outlier Removal Clustering (ORC) technique that is applied to several synthetic datasets. The algorithm clusters the dataset until convergence, removes outliers, and re-clusters the reduced dataset. The process is subsequently repeated

Algorithm 3.3 Distance-based outlier detection algorithm

```
    Function $[\mathbf{X_o}] = DBOD(\mathbf{X}, N_n, r_{out})$
    Inputs:
      $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$,    ▷ A set of feature vectors
      $N_n$,    ▷ Number of nearest neighbours that defines the neighbourhood of a sample
      $r_{out}$    ▷ The percentage of data samples to identify as outliers
    Output:
      $\mathbf{X_o} = \{\mathbf{X_{o_1}}, \mathbf{X_{o_2}}, \ldots, \mathbf{X_{o_{N_{out}}}}\}$    ▷ Set of feature vectors identified as outliers

 1: $N_{out} \leftarrow |\mathbf{X}| \times r_{out}$ ;    ▷ Number of data samples to identify as outliers
    ▷ Compute square matrix consisting of the distances between all data samples in $\mathbf{X}$
 2: for $i = 1$ to $N$ do
 3:     for $j = 1$ to $N$ do
 4:         $\mathbf{P\_Dists}(i, j) \leftarrow Distance(\mathbf{x_i}, \mathbf{x_j})$ ;    ▷ $j \neq i$
 5:     end for
 6: end for
    ▷ Sort the columns of $\mathbf{P\_Dist}$ in ascending order
 7: $\mathbf{Sorted\_PDists} \leftarrow Sort(\mathbf{P\_Dists}, ascending)$ ;

    ▷ Extract $N_n^{th}$-nearest neighbour distances. Select the $(N_n^{th} + 1)$ row of $\mathbf{Sorted\_PDist}$
    ▷ The first row represents a samples distance to itself (i.e. 0)
 8: $\mathbf{N_n^{th}\_NN\_Dists} \leftarrow \mathbf{Sorted\_PDist}(N_n + 1)$ ;    ▷ Vector of distances from each sample
                                                              ▷ to its' $N_n^{th}$ nearest neighbour
    ▷ Sort the vector $\mathbf{N_n^{th}\_NN\_Dists}$ in descending order, and store
    ▷ the sorted indices of all samples
    ▷ (i.e. $\mathbf{N_n^{th}\_NN\_Dists}(\mathbf{Sorted\_N_n^{th}\_NN\_Indices}(j)) \leftarrow \mathbf{Sorted\_N_n^{th}\_NN\_Dists}(j)$)
 9: $[\mathbf{Sorted\_N_n^{th}\_NN\_Dists},\ \mathbf{Sorted\_N_n^{th}\_NN\_Indices}]$
                    $\leftarrow Sort(\mathbf{N_n^{th}\_NN\_Dists}, descending)$ ;

    ▷ Select the top $N_{out}$ samples from $\mathbf{Sorted\_N_n^{th}\_NN\_Indices}$
10: for $j = 1$ to $N_{out}$ do
11:     $\mathbf{X_o}(j) \leftarrow \mathbf{X}(\mathbf{Sorted\_N_n^{th}\_NN\_Indices}(j))$ ;
12: end for
13: Return $\mathbf{X_o}$
```

for a user specified number of iterations. Initially the entire dataset is clustered using the k–means clustering algorithm until convergence. Each data sample is subsequently assigned an outlier score, which is computed as a data sample's distance to the cluster centre to which it is assigned to. Each score is normalized by the largest outlier score amongst all data samples, which scales all outlier scores to within the range [0,1]. The samples with scores that are larger than a threshold are removed from the dataset. The reduced dataset is re-clustered with the initial centres set as the final cluster centres obtained from the previous iteration.

The pseudocode used for the combination of the distance-based outlier detection approach provided in algorithm 3.3, and the k–means clustering algorithm provided in algorithm 3.1, is provided in algorithm 3.4.

Algorithm 3.4 K–means clustering with DBOD algorithm

```
Function $[\mathbf{C}',\ \mathbf{L}'] = kDBOD(\mathbf{X}, K, N_n, r_{out})$
Inputs:
  $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_N}\}$,   ▷ A set of feature vectors to be clustered
  $K$,   ▷ The number of clusters
  $N_n$   ▷ Number of nearest neighbours that defines the neighbourhood of a sample
  $r_{out}$   ▷ The percentage of data samples in each cluster to identify as outliers
Outputs:
  $\mathbf{C}' = \{\mathbf{c_1}', \mathbf{c_2}', \dots, \mathbf{c_K}'\}$,   ▷ Set of feature vectors representing cluster centres
  $\mathbf{L}' = \{l'(\mathbf{x_a}) | a = 1, 2, \dots, N\}$,   ▷ Set of cluster assignments for data samples in X

1: $(\mathbf{C}, \mathbf{L}) \leftarrow kmeans(\mathbf{X},\ K,\ \mathbf{F_s})$ ;   ▷ Algorithm 3.1 on entire dataset

2: for $i = 1$ to $K$ do   ▷ Remove outliers from each cluster
3:     $\mathbf{H_i} \leftarrow (\mathbf{x} \in \mathbf{X} \mid l(\mathbf{x}) = i)$ ;   ▷ Data samples belonging to cluster i
4:     if $(|\mathbf{H_i}| \leq N_n)$ then   ▷ If a cluster consists of fewer samples than
5:         continue ;   ▷ the number of neighbours $N_n$ to consider,
6:     end if   ▷ then leave cluster as is
7:     $\mathbf{H_{i_o}} \leftarrow DBOD(\mathbf{H_i}, N_n, r_{out})$ ;   ▷ Algorithm 3.3
8:     $\mathbf{X_o}(|\mathbf{X_o}| + 1, \dots, |\mathbf{X_o}| + |\mathbf{H_{i_o}}|) \leftarrow \mathbf{H_{i_o}}$ ;   ▷ Collection of outliers from all clusters
```

```
 9: end for

    ▷ Remove all data samples in $\mathbf{X}$ identified as outliers
10: for all ($\mathbf{x_i} \in \mathbf{X}$) do
11:     $\mathbf{X}' \leftarrow \mathbf{x_i} \in \mathbf{X}, \quad where\ \mathbf{x_i} \notin \mathbf{X_o}$ ;
12: end for
13: $(\mathbf{C}', \mathbf{L}') \leftarrow kmeans(\mathbf{X}',\ K,\ \mathbf{F_s})$ ;   ▷ Cluster data samples in $\mathbf{X}'$ using algorithm 3.1
                                                          ▷ with $\mathbf{C}$ as initial centres
    ▷ Assign each outlier to the cluster centre in $\mathbf{C}'$ that it is closest to
14: for all ($\mathbf{x_i} \in \mathbf{X_o}$) do
15:     $l'(\mathbf{x_i}) \leftarrow \arg\min_j[Distance(\mathbf{x_i}, \mathbf{c_j}')]$ ;        ▷ $j = \{1, \ldots, K\}$
16: end for

17: Return $(\mathbf{C}', \mathbf{L}')$
```

The algorithm clusters the dataset until convergence, removes a percentage r_{out} of outliers from each cluster, and re-clusters the remaining data samples. Initially, k–means clustering is applied to the entire dataset until convergence, obtaining a set of cluster centres $\mathbf{C}$ and cluster assignments $\mathbf{L}$ for each sample in the dataset. Outliers are subsequently identified amongst those samples that belong to the same cluster based on the N_n^{th} nearest neighbour algorithm (algorithm 3.3) proposed by Ramaswamy et al. [117]. All data samples that were identified as outliers $\mathbf{X_o}$ are removed from the dataset. K–means clustering is applied to the remaining data samples $\mathbf{X}'$ using the cluster centres obtained from the previous clustering result $\mathbf{C}$, as the initial cluster centres. This produces new cluster centres $\mathbf{C}'$ and cluster assignments $\mathbf{L}'$. Cluster assignments $\mathbf{L}'$ is updated by assigning each outlier in $\mathbf{X_o}$ to the cluster centre in $\mathbf{C}'$ that it is closest to, based on the squared Euclidean distance computed over the feature space. The algorithm subsequently returns the final cluster centres and cluster assignments $\mathbf{C}'$ and $\mathbf{L}'$, respectively.

3.4.2 Distribution-based clustering

Distribution-based clustering is carried out by fitting a parametric distribution to the data samples. A distribution has multiple components, an example is the Gaussian mixture model. The posterior probabilities of components conditioned on a specific data sample, is used as a metric to assign data samples to a component. Those data samples that belong to the same component constitute a cluster [119].

Distribution-based clustering has been demonstrated as being able to accurately capture the correlation and dependence between features. However, a possible problem that may arise when representing data samples using a multi-component parametric distribution, is that of over-fitting [119]. Over-fitting occurs when the number of components are excessive, and the distribution model captures variations of individual samples / noise, in addition to trends in the dataset itself. This problem may be solved by specifying a conservative number of components K to fit the data to. This number K may be calculated by incorporating model complexity into the performance metric during fitting. Distribution-based techniques typically relies on the assumption that data samples can be accurately represented using the selected distribution.

Gaussian Mixture Model (GMM)

A single Gaussian component may not always be sufficient to model real datasets, as real datasets may be multi-modal, where in some cases a mixture of multiple Gaussian components would fit the dataset more effectively [119]. A Gaussian mixture model is a linear superposition of several Gaussian components, which can be defined as [119]:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k}) \tag{3.5}$$

where $p(\mathbf{x})$ represents the marginal probability of data sample $\mathbf{x}$, $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})$ represents a Gaussian component of the mixture model (that is, the multivariate Gaussian probability density function), with mean $\boldsymbol{\mu_k}$ and covariance matrix $\boldsymbol{\Sigma_k}$. The parameter π_k represents the mixing coefficients of the k^{th} Gaussian component, and K is the number of Gaussian components used to fit the dataset [119].

For an N_F-dimensional vector $\mathbf{x}$ the multivariate Gaussian probability density function is defined as [119],

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{N_F}{2}}}\frac{1}{|\boldsymbol{\Sigma}|^{\frac{1}{2}}}\exp\left\{\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \tag{3.6}$$

where $\boldsymbol{\mu}$ is an N_F-dimensional mean vector, $\boldsymbol{\Sigma}$ is an $N_F \times N_F$ covariance matrix, and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$.

Maximum likelihood

The parameters of the GMM may be estimated by searching for values that maximize the likelihood of the observed data, conditioned on the parameters of interest, thereby fitting the distribution to the data samples. The maximum likelihood is a way of estimating the parameters of a statistical model (such as the Gaussian distribution model) that maximizes the likelihood function [119]. From equation 3.5, the log of the likelihood function can be defined as [119]:

$$\ln p(\mathbf{X}|\pi,\boldsymbol{\mu},\boldsymbol{\Sigma}) = \sum_{n=1}^{N}\ln\left(\sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}|\boldsymbol{\mu_k},\boldsymbol{\Sigma_k})\right) \tag{3.7}$$

A problem associated with applying the maximum likelihood framework to Gaussian mixture models is the potential for certain components to degenerate into singularities [119]. Suppose the covariance matrix $\mathbf{\Sigma_k}$ is given by $\boldsymbol{\Sigma_k} = \sigma_k^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. If the mean of the j^{th} component is exactly equal to one of the data samples in the dataset such that $\boldsymbol{\mu_j} = \mathbf{x_n}$, then this sample will contribute a term to the likelihood function, proportional to,

$$\mathcal{N}(\mathbf{x_n}|\mathbf{x_n},\sigma_j^2\mathbf{I}) = \frac{1}{(2\pi)^{\frac{1}{2}}}\frac{1}{\sigma_{\mathbf{j}}} \tag{3.8}$$

In the limit as σ_j tends to zero, the likelihood contribution by component k will tend to infinity, and the log-likelihood function will also tend to infinity [119]. These singularities are characteristic of over-fitting of the data, thus the models will not fit the data effectively.

3.4.2.1 Expectation-Maximization (EM) clustering

A method that may be used to find maximum likelihood parameter estimates in the case of models with latent variables (see reference [119] for an overview of latent variable models) is the expectation-maximization algorithm. Thus, the expectation-maximization algorithm can be used to estimate the parameters $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_K\}, \boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_K}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_K}\}$ that produces a Gaussian mixture model that effectively fits the data.

Let a GMM be represented as a model with latent variables. In sampling from the GMM, first draw a component $\{1, 2, \ldots, K\}$ according to the prior probabilities $\{\pi_1, \pi_2, \ldots, \pi_K\}$. Assign a latent (unobserved) variable $\hat{\mathbf{Z}} = \{z_1, z_2, \ldots, z_K\}$. If component b was drawn, $z_b = 1$ and $z_a = 0$, for all $a \neq b$, $a = 1, 2, \ldots, K$.

A latent variable $z_{n,k}(n = 1, 2, \ldots, N, k = 1, 2, \ldots, K)$ is assigned to each observation (data sample) of the dataset, which corresponds to the component from which that data sample originated.

Let $\hat{\mathbf{Z}}_\mathbf{i} = \{z_{i,1}, z_{i,2}, \ldots, z_{i,K}\}$ denote the latent variable for data sample i. The parameters $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_K\}, \boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_K}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_K}\}$ are estimated using the posterior probability of components, conditioned on data samples $\mathbf{x}$. It is shown in [119] that the posterior distribution can be calculated as:

$$\gamma(z_{i,k}) \equiv p(z_{i,k} = 1|\mathbf{x_i}) = \frac{\pi_k \mathcal{N}(\mathbf{x_i}|\boldsymbol{\mu_k}, \boldsymbol{\Sigma_k})}{\sum_l \pi_l \mathcal{N}(\mathbf{x_i}|\boldsymbol{\mu_l}, \boldsymbol{\Sigma_l})} \tag{3.9}$$

The mean $\boldsymbol{\mu_k}$ of the Gaussian components is computed by setting the derivative of equation 3.7 with respect to $\boldsymbol{\mu_k}$ of the Gaussian components to zero [119]. $\boldsymbol{\mu_k}$ is then derived as:

$$\boldsymbol{\mu_k} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{n,k})\mathbf{x_n} \tag{3.10}$$

where N represents the total number of data samples in the dataset, $\mathbf{x_n}$ represents a sample in the dataset, and N_k is defined as,

$$N_k = \sum_{n=1}^{N}\gamma(z_{n,k}) \tag{3.11}$$

Similarly, the covariance matrix $\boldsymbol{\Sigma_k}$ of the Gaussian components is computed by setting the derivative of equation 3.7 with respect to $\boldsymbol{\Sigma_k}$ of the Gaussian components to zero [119]. $\boldsymbol{\Sigma_k}$ is then derived as:

$$\boldsymbol{\Sigma_k} = \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{n,k})(\mathbf{x_n} - \boldsymbol{\mu_k})(\mathbf{x_n} - \boldsymbol{\mu_k})^T \tag{3.12}$$

The mixing coefficients π_k are obtained by maximizing $\ln p(\mathbf{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to π_k [119].

The mixing coefficients $\pi_k(k = 1, 2, \ldots, K)$ is required to sum to one, in order to satisfy this constraint a Lagrange multiplier is used (refer to [119] for further details). Thus, π_k is derived as:

$$\pi_k = \frac{N_k}{N} \tag{3.13}$$

Further details regarding these derivations can be found in [119]. The expectation maximization algorithm as applied to Gaussian mixture models to perform clustering is provided in algorithm 3.5.

The EM algorithm is divided into two successive steps, the Expectation (E-step) and Maximization (M-step), which are performed iteratively until convergence. Initial values are first chosen for $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_k\}$, $\boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_k}\}$. For the component means $\boldsymbol{\mu}$, K random data samples are chosen from the dataset.

Algorithm 3.5 Expectation-Maximization clustering with GMMs

```
Function $[\mathbf{L}] = EM(\mathbf{X}, K, T_{log})$
Inputs:
    $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$,        ▷ A set of feature vectors
    $K$,                                       ▷ Number of Gaussian components
    $T_{log}$                                    ▷ Threshold for log likelihood
Outputs:
    $\mathbf{L} = \{l(\mathbf{x_a}) | a = 1, 2, \ldots, N\}$          ▷ Set of cluster assignments for X

▷ Initialization
1: $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_k\}, \boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_k}\}$ ;
2: $log\_likelihood \leftarrow 0$ ;
3: $convergence \leftarrow false$ ;
4: repeat
▷ E-Step
5:     Compute $\gamma(z_{n,k}) \equiv p(k|\mathbf{X})$ ;        ▷ Posterior probabilities (equation 3.9)
▷ M-Step
▷ Using equations (3.10), (3.12), and (3.13)
6:     Recompute $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_k\}, \boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_k}\}$
▷ Log-likelihood
7:     Compute $new\_log\_likelihood \leftarrow \ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ ;        ▷ Using equation 3.7
▷ Convergence test
8:     if $(|new\_log\_likelihood - log\_likelihood| < T_{log})$ then
9:         $convergence \leftarrow true$ ;
10:    else
11:        $log\_likelihood \leftarrow new\_log\_likelihood$ ;
12:    end if
13: until $(convergence = true)$
14: Recompute $\gamma(z_{n,k}) \equiv p(k|\mathbf{x})$ ;        ▷ Posterior probabilities (equation 3.9)
▷ Cluster Assignments
15: for all $(\mathbf{x} \in \mathbf{X})$ do
16:     $l(\mathbf{x}) \leftarrow \arg\max_k [p(k|\mathbf{x})]$ ;
17: end for
18: Return $\mathbf{L}$
```

The mixing coefficients are uniform $\frac{1}{K}$, and the covariance matrices for all components are diagonal, where element j on the diagonal is the variance of the data samples in $\mathbf{X}$ along feature j. The E-step computes the posterior probabilities (responsibilities) for which the component k takes for fitting the data sample $\mathbf{x}$, defined by $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_k\}, \boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_k}\}$. The M-step uses the posterior probabilities to calculate new values for $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_k\}, \boldsymbol{\mu} = \{\boldsymbol{\mu_1}, \boldsymbol{\mu_2}, \ldots, \boldsymbol{\mu_k}\}$, and $\boldsymbol{\Sigma} = \{\boldsymbol{\Sigma_1}, \boldsymbol{\Sigma_2}, \ldots, \boldsymbol{\Sigma_k}\}$. E and M steps are iteratively carried out until convergence. The algorithm converges when the change in the log-likelihood is below a threshold [119]. Data samples are subsequently assigned to components based on the posterior probability that a sample belongs to a given component k.

3.5 Cluster labelling

Cluster labelling is the task of assigning class labels to each cluster. Data samples belonging to each cluster are subsequently assigned the class label of the cluster. In this research, binary classification is carried out, where clusters are labelled as *legitimate* or *attack*. Cluster labelling is typically carried out in unsupervised network anomaly detection under the following assumptions [2, 43]:

1. The amount of legitimate network traffic outnumbers the amount of malicious network traffic in the dataset considered.
2. In the feature space considered, malicious network traffic is found in a different region of the feature space as that of legitimate network traffic.
3. Certain groups of malicious network traffic have a higher similarity than groups of legitimate network traffic.

An important note is that certain Denial-of-Service (*DoS*) attacks are volumetric in nature. If network traffic data is collected over a short period of time, then it is possible that the network traffic produced by a *DoS* attack may constitute a majority of the network traffic collected over this time period. In this case, the first assumption does not hold. The

NSL-KDD dataset [25] was created from network traffic collected over a significantly long period of time, resulting in legitimate network traffic constituting the majority of the dataset. Thus, the first assumption holds.

References [43, 45, 48, 120–124] perform cluster labelling after the application of the k–means, the single-linkage, or a variant of the k–means clustering algorithm, based on the assumptions mentioned previously.

A number of authors label a fraction of the largest clusters as *legitimate.* The largest cluster is defined as the cluster with the largest number of data samples belonging to the cluster. Clusters are ranked according to the number of data samples belonging to each cluster. A fraction of the top ranking clusters are labelled as *legitimate*, while the remaining clusters are subsequently labelled as *attacks* [43, 48, 120, 122]. A variant of this technique involves the use of a threshold, where a cluster is assigned the label *legitimate* if the number of data samples associated with a cluster exceeds the threshold. Han et al. [121] and Guan et al [123] label clusters with more than T data samples as *legitimate*, while clusters with less than T data samples are labelled as *attacks.* Wang [45] also use a threshold, but instead labels clusters consisting of less than T data samples as *attacks*, while the remaining clusters are labelled as *legitimate.* The threshold is calculated as a percentage of the data samples of the entire dataset. Zhong et al. [124] label a percentage of the data samples of the entire dataset that are closest to the centre of the largest cluster, as *legitimate.*

The labelling scheme implemented in this research is based on the assumptions mentioned previously and follows a similar labelling scheme to [43, 48, 120, 122]. The N_r largest clusters are labelled as *legitimate*, while the remaining clusters are labelled as *attacks.* The value of N_r is varied from 1 to $K-1$, where K is the number of clusters, to obtain different operating characteristics, each corresponding to a certain false positive and true positive percentage.

3.6 Experimental results

In this section, the experimental results obtained by applying the proposed classifier to the NSL-KDD dataset [25] are presented, compared and discussed. The purpose of the experimental work was to evaluate the classification performance attained by

- applying the four clustering algorithms presented in sections 3.4.1.1 to 3.4.2.1, to the proposed classifier over the full feature set, and
- to evaluate the classification performance attained by applying the four clustering algorithms to the proposed classifier over different feature subsets obtained from the literature [31, 35].

The false positive and true positive percentages were used as measures of classification performance in each case.

3.6.1 Experimental setup

This section describes the setup of the proposed classifier, as used to carry out the experimental work. The parameters for each block of the proposed classifier's functional block diagram (figure 3.1), which includes the four clustering algorithms, are provided. Each of the feature subsets used for classification, as well as the classification performance measures are presented.

3.6.1.1 Classifier parameters

The parameters of the proposed classifier are presented in table 3.6.

3.6.1.2 Feature subsets

The list of feature sets that were selected for the experimental work of this chapter are presented in table 3.7. The table presents the name assigned to each feature subset, the

Table 3.6: *Experimental setup of the proposed classifier for the comparison of clustering algorithms.*

Dataset (Feature vectors)		
Dataset selected	NSL-KDD training set 2	Section 3.2.2. Table 3.3 provides the dataset distribution.
Data transformation		
Normalization of numeric features	Statistical normalization	Section 3.3.1
Encoding of non-binary categorical features	Frequency encoding	Section 3.3.2
Encoding of binary categorical features	Ordinal encoding	Section 3.3.2
Clustering		
Clustering algorithm 1: k–means		Section 3.4.1.1
Initialization	Initial cluster centres - K randomly selected data samples	—
Distance metric	Euclidean distance	—
Number of clusters (K)	2 - 10	—
Number of repetitions with different cluster centres (C_R)	100	—
Clustering algorithm 2: k–medoids		Section 3.4.1.2
Initialization	Initial cluster centres - K randomly selected data samples	—
Distance metric	Euclidean distance	—
Number of clusters (K)	2 - 10	—
Number of repetitions with different cluster centres (C_R)	100	—
Clustering algorithm 3: k–means with distance-based outlier detection		Section 3.4.1.3
Initialization	Initial cluster centres - K randomly selected data samples	—
Distance metric	Euclidean distance	—
Number of clusters (K)	2 - 10	—
Number of repetitions with different cluster centres (C_R)	100	—
Number of nearest neighbours considered (N_n)	10	—
Percentage of outliers to remove from each cluster (r_{out})	10%	—
Clustering algorithm 4: Expectation Maximization (EM) clustering		Section 3.4.2.1
Initialization	Component means - K randomly selected data samples, Mixing coefficients - uniform (1/K), Covariance matrices - diagonal; element j on the diagonal is the variance of the data samples in the dataset along feature j	—
Distance metric	Euclidean distance	—
Number of components (K)	2 - 10	—
Number of repetitions with different cluster centres (C_R)	100	—
Regularization value	0.005	—
Cluster labelling		
Labels	Binary (intrusion / legitimate)	Section 3.5
Labelling algorithm	Largest N_r clusters labelled as legitimate, remaining clusters labelled as intrusions	—
Number of legitimate clusters (N_r)	1 to K-1	—

features present in each subset, and the method that was used to derive the feature subset[1]. The feature numbers presented in table 3.7 corresponds to the full list of 41 features as presented in table 3.2.

Note that feature 20, the number of outbound commands, was excluded from all feature subsets during experimentation. This was due to the fact that the feature had a value of zero among all data samples within the NSL-KDD dataset [25]. This implies that the feature is irrelevant (i.e. it does not provide significant information for detecting intrusions).

In addition to the full, original feature set (FS1), a total of 10 subsets (FS2 – FS11) of the full feature set were used in this research. It is reported in the literature that, of the 10 feature subsets, 4 feature subsets (FS2 – FS5) were derived using wrapper-based feature selection techniques [31] applied to the KDD Cup 1999 dataset [24]. The remaining 6 feature subsets (FS6 – FS11) were derived using filter-based feature selection techniques [35] applied to the KDD Cup 1999 and NSL-KDD datasets [25]. All 10 feature subsets were used directly in the experimental work. A detailed summary of the relevant filter and wrapper-based feature selection techniques is provided in chapter 2, section 2.4.

Table 3.7: *Feature set descriptions of those feature subsets found in the literature.*

Name	**# Features**	**Feature List**	**Method**	**Method Type**	**Reference**
FS1	40	All, excl. feature '20'	Full feature set, excluding feature '20'	–	–
FS2	10	8,10,14,31,32,33,35,36,37,40	Feature Removal Method (FRM)	Wrapper	[31]
FS3	10	6,7,23,24,25,29,30,31,32,38	Sole Feature Method (SFM)	Wrapper	[31]
FS4	10	10,14,23,24,25,31,32,33,36,38	Hybrid of FRM and SFM	Wrapper	[31]
FS5	19	2,4,8,10,14,15,19,25,27,29,31, 32,33,34,35,36,37,38,40	Gradual FRM	Wrapper	[31]
FS6	4	3,5,6,39	Best of articles [7, 26–29]	Filter	[35]
FS7	10	3,4,5,6,14,16,27,28,37,39	Best of articles [7, 26–29]	Filter	[35]
FS8	10	2,3,4,5,6,8,23,30,34,36	Degree of Correlation + Greedy Stepwise	Filter	[35]
FS9	10	2,3,5,6,23,24,33,34,35,36	Information Gain + Ranker	Filter	[35]
FS10	4	2,3,5,6	Degree of Correlation + Greedy Stepwise	Filter	[35]
FS11	4	3,5,23,24	Information Gain + Ranker	Filter	[35]

[1]The feature subsets considered in this chapter do not include those derived using the proposed feature selection algorithm; results for these feature subsets are presented in chapter 4

3.6.1.3 Performance metrics

The two-class or binary classification of a data sample leads to four possible classification outcomes, depending on whether the data sample was classified correctly or incorrectly. The four classification outcomes for a data sample are referred to as True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) [2].

- True Positive (TP) - A data sample representing an attack is correctly classified as an attack.
- True Negative (TN) - A data sample representing legitimate network traffic is correctly classified as legitimate or normal.
- False Positive (FP) - A data sample representing legitimate network traffic is incorrectly classified as an attack.
- False Negative (FN) - A data sample representing an attack is incorrectly classified as legitimate or normal.

The four possible classification outcomes are illustrated in table 3.8 [44].

TABLE 3.8: *Four classification scenarios.*

		Actual result	
		Attack	Legitimate
Predicted result	Attack	TP	FP
	Legitimate	FN	TN

In this research, the percentage of true positives and false positives are used to evaluate the performance of the proposed classifier. The true positive percentage is a measure of a classifier's ability to correctly identify data samples representing attacks. It is defined as the percentage of the total number of attack samples in the dataset that were correctly classified as attack data samples. The true positive percentage is computed as:

$$\frac{TP}{TP + FN} \times 100 \tag{3.14}$$

The false positive percentage is a measure of a classifier's tendency to misidentify legitimate data samples as attack data samples. It is defined as the percentage of the total number of legitimate data samples in the dataset that were incorrectly classified as attack data samples. The false positive percentage is computed as:

$$\frac{FP}{FP+TN} \times 100 \tag{3.15}$$

Receiver Operating Characteristic (ROC) curves provide a visual representation of the trade-off between the true positive and false positive percentages of a classifier, which is brought about by varying the detection threshold[2] of the classifier. ROC curves are useful for analysing the behaviour of a classifier under different operating conditions, and for comparing the performance of multiple classifiers. Figure 3.2 is an example of a ROC curve, where the x-axis and y-axis of the graph represents the FP and TP percentages, respectively. Improved classification performance corresponds to curves that lie towards the top left corner of the graph. A curve that lies on top of, or to the left of another curve is indicative of superior performance over the range of TP percentages or FP percentages, respectively. The line $y = x$ corresponds to a classifier that randomly assigns one of the two classes to each data sample with equal probability.

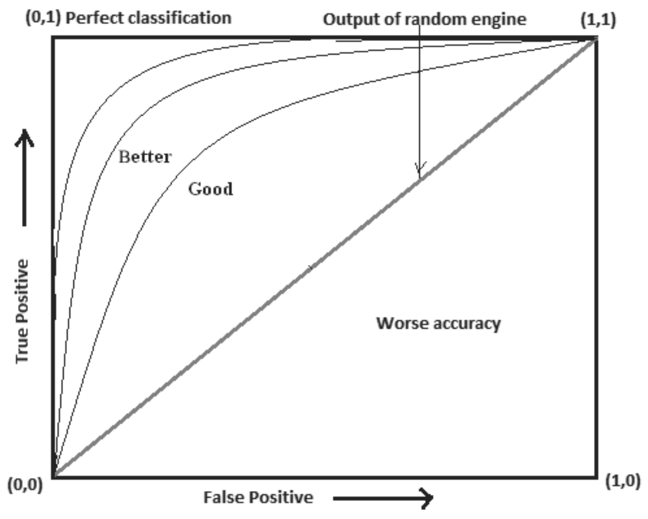


FIGURE 3.2: *Examples of ROC curves (from [2]).*

[2]The detection threshold is the number of clusters labelled as legitimate (N_r), where N_r is varied in the experimental work

3.6.1.3.a Construction of ROC curves of the proposed classifier

This section provides a detailed description of the method used to construct the ROC curves of the proposed classifier. In the classifier configuration presented in this chapter, one pair of average FP and TP percentages is obtained for each pair of parameter values K and N_r of the clustering algorithm, where K is the number of clusters and N_r is the number of largest clusters to label as legitimate (and where the average is calculated over the C_R repetitions of the clustering algorithm with randomly selected initial cluster centres). A ROC curve is constructed for each value of K; i.e., each ROC curve represents clustering with a fixed number of clusters. The points on the curve correspond to different values of N_r; in this approach, the number of clusters N_r to label as legitimate controls the threshold of the classifier.

To illustrate the construction of the ROC curve, consider the TP and FP percentages obtained during k–means clustering over the transformed NSL-KDD dataset [25], with a fixed value of $K = 5$, and for $N_r = 1, 2, \ldots 4$, as indicated in table 3.9.

Table 3.9: *TP and FP percentages obtained during k–means clustering with $K = 5$.*

Clusters labelled as legitimate (N_r)	**TP percentage**	**FP percentage**
1 Largest cluster	88.29	19.61
2 Largest clusters	39.47	15.90
3 Largest clusters	17.55	9.35
4 Largest clusters	5.40	3.37

These values were used to construct the ROC curve indicated in figure 3.3, where the corresponding value of N_r is indicated next to each point of the ROC curve. The point on the ROC curve most distant from the origin typically represents the labelling scheme where only the largest cluster is labelled as legitimate, with subsequent points closer to the origin corresponding to successively larger values of N_r

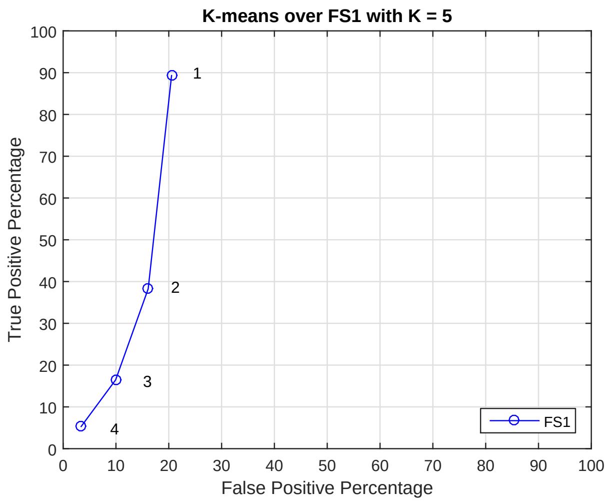


Figure 3.3: *The plot points for a typical ROC curve obtained using the cluster labelling scheme described in section 3.5, and the TP and FP percentages.*

In order to ease the comparison between multiple classifiers, where each classifier is associated with multiple ROC curves, the concept of a composite ROC curve is introduced. A single composite ROC curve is associated with each classifier, and represents the best performance that can be obtained over the multiple ROC curves of the classifier. The composite ROC curve is constructed by first sorting all the FP and TP percentage pairs obtained over the full range of $K = 2, \ldots, 10$ and $N_r = 1, \ldots, K-1$ for the classifier according to increasing FP percentages. The point with the lowest FP percentage is assigned to the composite ROC curve. Points with successively higher FP percentages are iteratively added to the composite ROC curve only if the TP percentage of the corresponding point is larger than or equal to the TP percentage of the previously added point of the composite ROC curve. In this manner, points with higher FP percentages are only considered if it affords a higher TP percentage. This process is illustrated in figure 3.4, where the composite ROC curve is illustrated with the blue dashed line.

Figure 3.4 illustrates the TP and FP percentages obtained during k–means clustering over the transformed NSL-KDD dataset [25], with $K = 2, \ldots, 10$. The blue solid lines on the figure represent the ROC curves for each corresponding value of $K = 2, \ldots, 10$, where each point represents a pair of TP and FP percentages that were obtained for each value of $N_r = 1, \ldots, K-1$. The blue dashed line represents the composite ROC curve, which consists of the best TP and FP percentages.

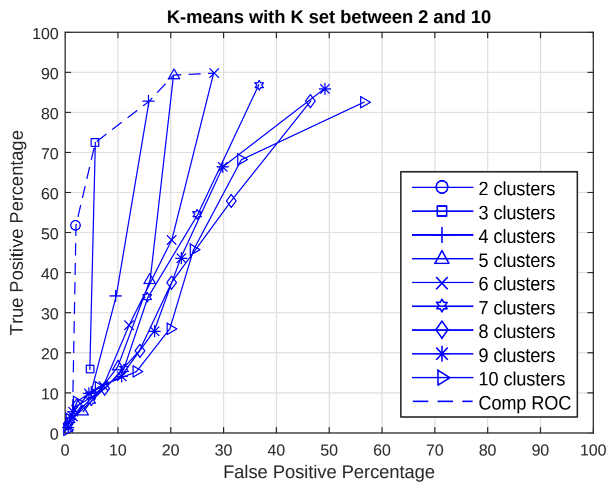


FIGURE 3.4: *Illustration of the construction of a composite ROC curve.*

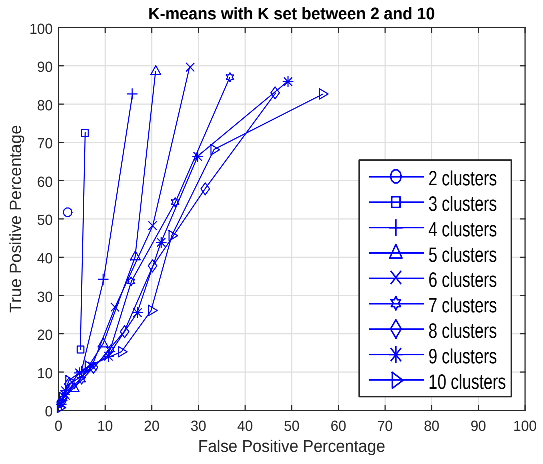
3.6.2 Results and analysis

In this section, the performance results of the proposed classifier are presented and discussed. The results are divided into subsections; the first subsection investigates the impact of the number of clusters on classification performance. The next subsection compares the classification performance of the clustering algorithms, and the final subsection compares the classification performance that was attained by each clustering algorithm when applied over the full feature set and several other feature subsets. All results are presented as ROC curves or composite ROC curves.

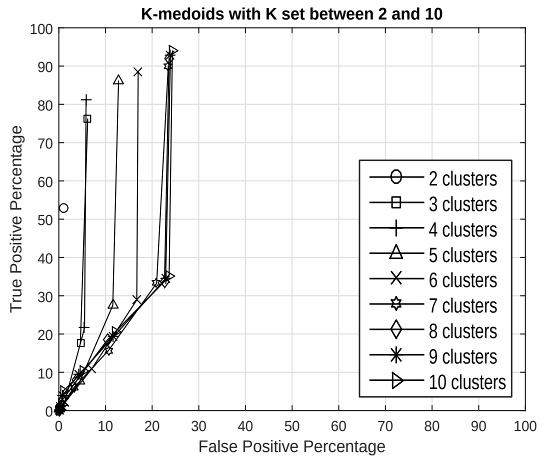
3.6.2.1 Impact of the number of clusters K

The performance impact of varying the number of clusters K was investigated by applying the proposed classifier to the full feature set (FS1). Each of the four clustering algorithms were applied over the range of values $K \in \{2, \ldots, 10\}$, where each distinct value of K corresponds to a single ROC curve. The ROC curves corresponding to each of the four clustering algorithms are presented in figures 3.5a to 3.5d.

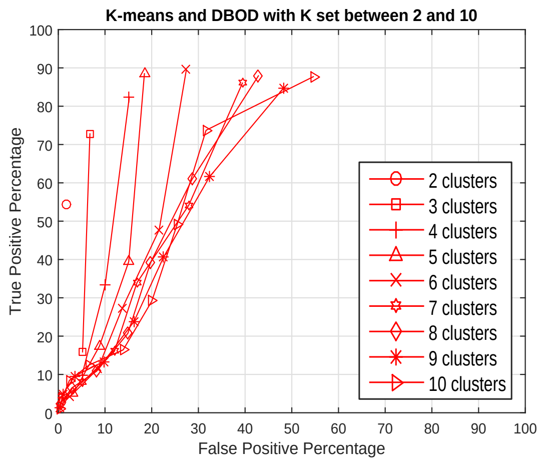
The figures reveal that there is no single value for K that provides both a superior TP and FP percentage than the remaining values of K, for each of the clustering algorithms.



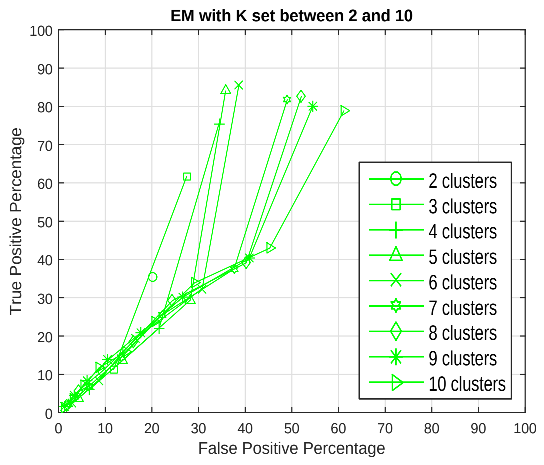
(a) K–means



(b) K-medoids



(c) K–means with DBOD



(d) Expectation-Maximization

FIGURE 3.5: *Performance of all four clustering algorithms over feature set FS1, with K varied between 2 and 10.*

This implies that the choice of a suitable value of K is dependant on the desired classification performance. For instance, if one seeks to maximize the TP percentage, a value of $K = 6$ for the k–means, k–means with DBOD and the EM clustering algorithms would produce the maximum TP percentages, which are 90%, 90% and 86%, respectively, with corresponding FP percentages of 28%, 28% and 38%, respectively. A value of $K = 10$ for the k–medoids clustering algorithm would produce the maximum TP percentage at 95%, with a corresponding FP percentage of 24%. However, if one seeks to minimize the FP percentage, a value of $K = 2$ would produce the minimum FP percentages over all clustering algorithms at less than 2% for the k–means, k–means with DBOD and k–medoids algorithms, with corresponding TP percentages of approximately 51%, 55% and 53%, respectively. The EM clustering algorithm attains a minimum FP percentage of 20% with a corresponding TP percentage of 35% at $K = 2$.

A noticeable trend is that the FP percentages increase as the number of clusters K increases. This is accompanied by an increase in the TP percentage when all clusters except the largest, are labelled as attacks, up to a maximum value of K for each clustering algorithm. The k–medoids algorithm attains its maximum TP percentage at $K = 10$. While the TP percentages for the k–means algorithm, the k–means algorithm in combination with the DBOD technique, and the EM clustering algorithm, reduces at $K > 6$. An increase in FP percentages implies that an increasing number of legitimate data samples are grouped into smaller clusters pertaining to attack clusters, while a decrease in TP percentages implies that a greater number of attack data samples are grouped into the largest cluster.

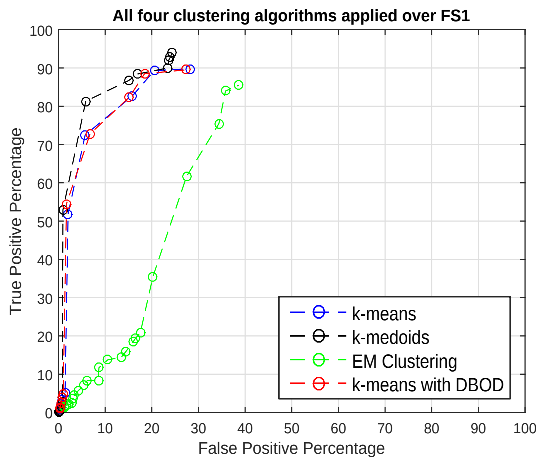
3.6.2.2 Comparison of clustering algorithms

A comparison between all four clustering algorithms over each feature subset in table 3.7 is presented in this subsection. Figures 3.6a to 3.6k contain the composite ROC curves which were obtained by applying the classifier to each feature subset, with $K \in \{2, \ldots, 10\}$. Each composite ROC curve corresponds to a different clustering algorithm.

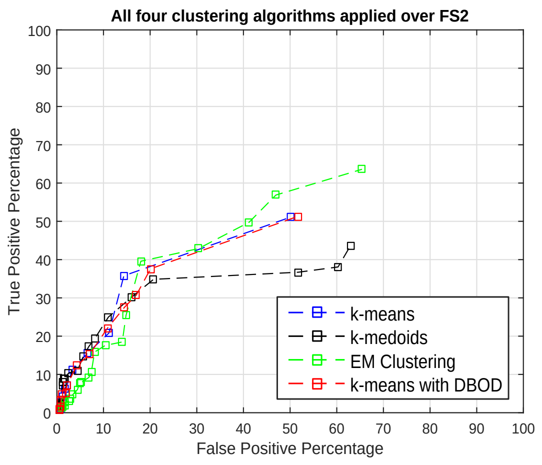
Figure 3.6 reveals that there is no individual clustering algorithm, from the four considered in this research, that outperforms the remaining algorithms over all feature subsets considered. The performance of each clustering algorithm varies depending on the feature subset used. For instance, feature subset FS9 (figure 3.6i) enables the k–means clustering algorithm to attain a TP percentage of approximately 66%, with a corresponding FP percentage of 14%, while at the same FP percentage (14%) the k–medoids clustering algorithm applied over FS9, only attains a TP percentage of 23%. However, when applying the classifier over feature set FS1 (figure 3.6a), it is observed that at a FP percentage of 6%, the k–medoids clustering algorithm attains a 9% increase in TP percentage when compared to the k–means clustering algorithm, at the same FP percentage.

The k–means, k–medoids and k–means with DBOD algorithms provide similar performance over the majority of feature subsets (FS1, FS3, FS4, FS5, FS7, FS8, FS11), and this performance is in general superior to that of the EM clustering algorithm.

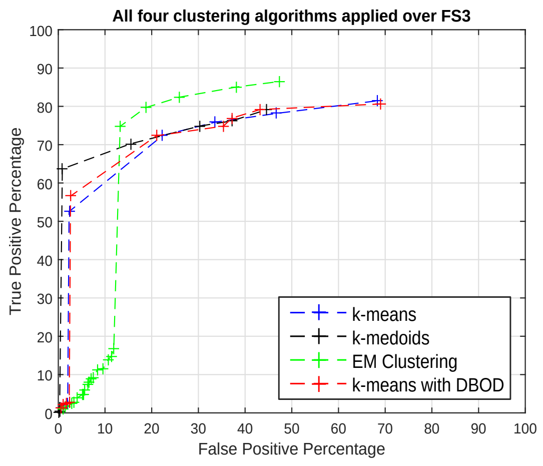
It was observed that the EM clustering algorithm was found to produce comparable classification results over FS2, FS3, FS7 and FS11. In the case of FS3 (figure 3.6c), for FP percentages greater than 12%, the EM clustering algorithm produces up to a 10% improvement in TP percentages when compared to the remaining three clustering algorithms.



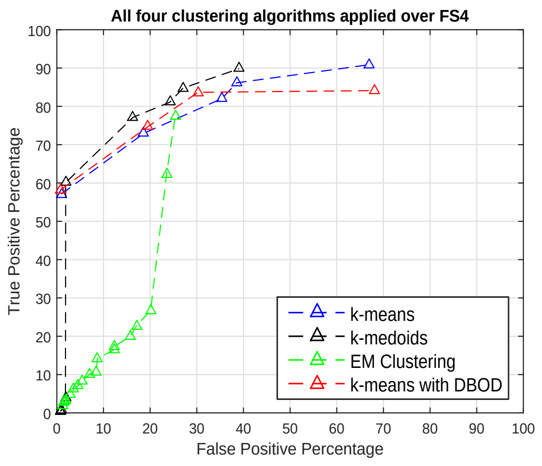
(a) FS1



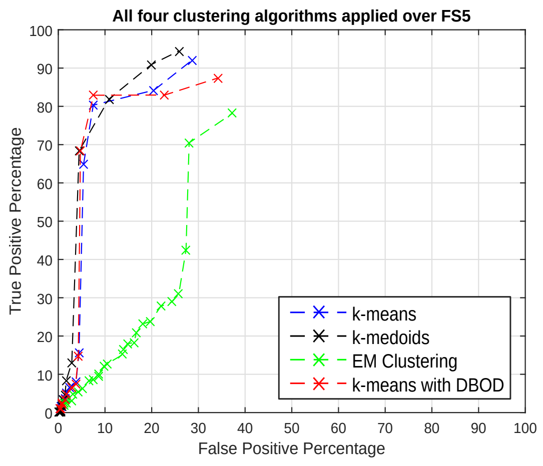
(b) FS2



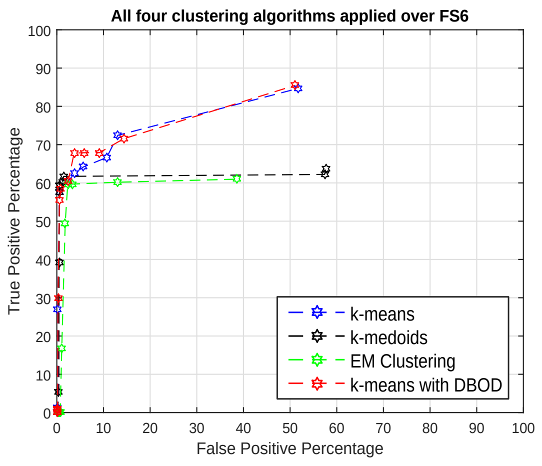
(c) FS3



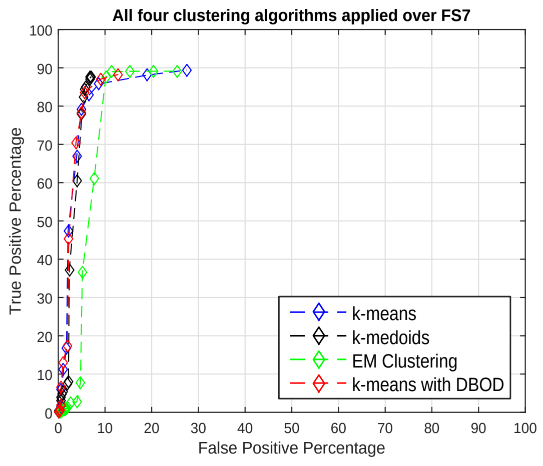
(d) FS4



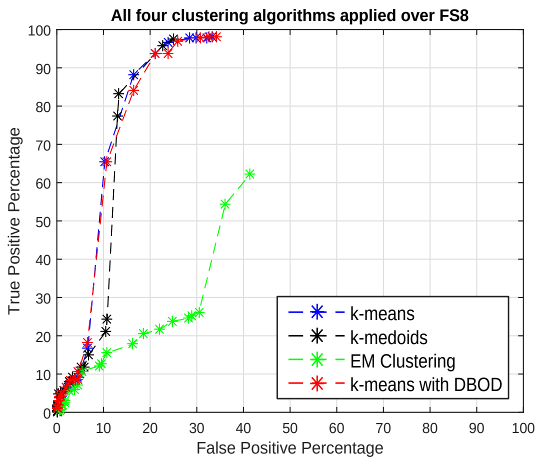
(e) FS5



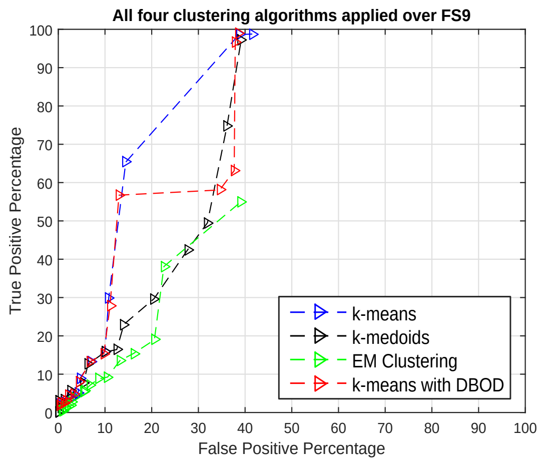
(f) FS6



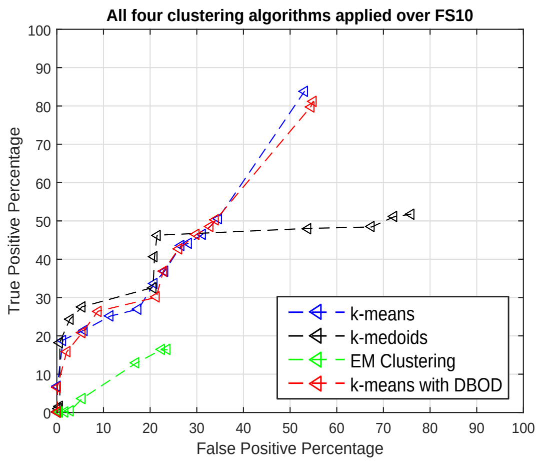
(g) FS7



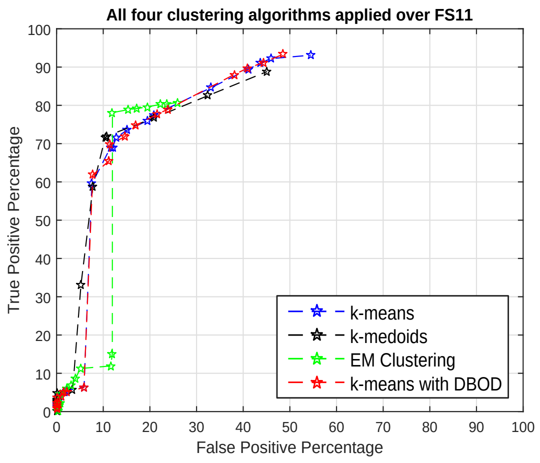
(h) FS8



(i) FS9



(j) FS10



(k) FS11

FIGURE 3.6: *Performance of all four clustering algorithms over each feature subset*

The performance of the EM algorithm was found to be more sensitive to the number of clusters K (see also figure 3.5d), with the algorithm producing empty clusters over FS6 for $K > 7$ and FS10 for $K > 5$. Table 3.10 provides the contents of each cluster found by the EM clustering algorithm when applied over feature subset FS10, with $K = 5$. The contents of each cluster is presented as the number of data samples belonging to a cluster, that correspond to each of the four broad classes of network attacks described in section 3.2.1.1, or correspond to the "normal" class, which represents legitimate network traffic.

TABLE 3.10: *Cluster contents obtained by applying the EM clustering algorithm over FS10 with* $K = 5$

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**
DoS	755	0	0	8 479	0
Probe	1 142	0	1	1 145	1
R2L	0	7	13	189	0
U2R	1	0	0	10	0
Normal	2 768	33	49	10 342	257
Total	**4 666**	**40**	**63**	**20 165**	**258**

Table 3.10 demonstrates that at $K = 5$ over feature subset FS10, the EM clustering algorithm groups approximately 80% of all data samples in the dataset into one cluster,

as shown in cluster 4, which consists of 84% of all attack data samples, and 77% of all legitimate data samples in the entire dataset. This implies that 84% of the attack samples are incorrectly classified as legitimate data samples for $N_r = 1$, which severely decreases the TP percentage, as demonstrated in figure 3.6j. The FP percentage remains at 23% in figure 3.6j as a majority of the legitimate data samples are grouped into the largest cluster, and thus correctly classified as legitimate data samples. It is also observed that clusters 2, 3 and 5 consists of a small number of data samples. When $K > 5$, these clusters become empty, as the algorithm is only able to fit the data to a finite number of clusters, when EM clustering is performed over the features present in subset FS10.

3.6.2.3 Comparison of feature subsets

A comparison of the performance obtained by applying each individual clustering algorithm over all 11 feature subsets is presented in this section. The classification results for each clustering algorithm is presented in figures 3.7 to 3.10. Each figure contains 11 composite ROC curves, where each composite ROC curve corresponds to a different feature subset.

3.6.2.3.a K–means

Figure 3.7 provides the composite ROC curves obtained by applying the proposed classifier to the dataset using the k–means clustering algorithm over each of the 11 feature subsets. The figure reveals that the performance of the k–means clustering algorithm varies significantly when applied over the different feature subsets.

The feature subsets can be divided into three groups based on the classification performance produced by each feature subset (as indicated by the red ellipses in figure 3.7). The first group consists of feature subsets FS1, FS5, FS7 and FS8; these subsets produce a superior classification performance with higher TP and lower FP percentages, as compared to the remaining feature subsets. FS7 produces TP percentages between 79% and 87%, with corresponding FP percentages between 5% and 10%. FS8 produces TP percentages

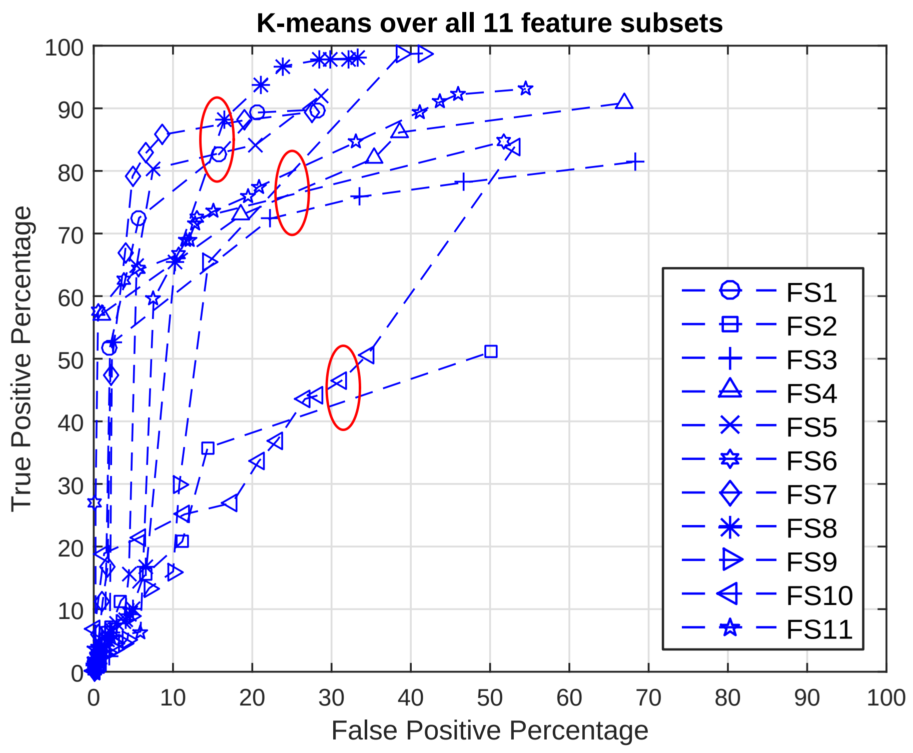


FIGURE 3.7: *A comparison of the composite ROC curves obtained by applying the proposed classifier to the dataset using the k–means clustering algorithm applied over each feature subset.*

of almost 100% with corresponding FP percentages within the range 28% and 34%, which is the highest FP percentage produced by those feature subsets belonging to the first group. FS7 produces superior classification performance amongst all feature subsets applied over the k–means clustering algorithm, where points that lie towards the upper left corner of the graph are indicative of better performance.

The second group consists of feature subsets FS3, FS4, FS6, FS9 and FS11, where FS4 and FS6 produce FP percentages of almost 0%, but with a TP percentage of approximately 57%. FS9 produces a TP percentage of almost 100%, similar to the performance produced by FS8, however, with a higher corresponding FP percentage of approximately 40%.

The third group consists of feature subsets FS2 and FS10, which produce inferior classification performances to all remaining feature subsets.

It is observed that the k–means clustering algorithm when applied over feature subsets FS5, FS7 and FS8 outperforms the same clustering algorithm when applied over the full feature set FS1. An interesting observation is that feature subset FS5 was produced by a wrapper-based feature selection method, while both feature subsets FS7 and FS8 were produced by filter-based feature selection methods.

3.6.2.3.b K–medoids

Figure 3.8 provides the composite ROC curves obtained by applying the proposed classifier to the dataset using the k–medoids clustering algorithm over each of the 11 feature subsets. The figure reveals that the performance of the k–medoids clustering algorithm varies significantly when applied over the different feature subsets.

The feature subsets can be divided into similar groups, as with the k–means clustering algorithm, based on the classification performance produced by each feature subset (as indicated by the red ellipses in figure 3.8). The exception is feature subset FS9 which lies in two of the three groups.

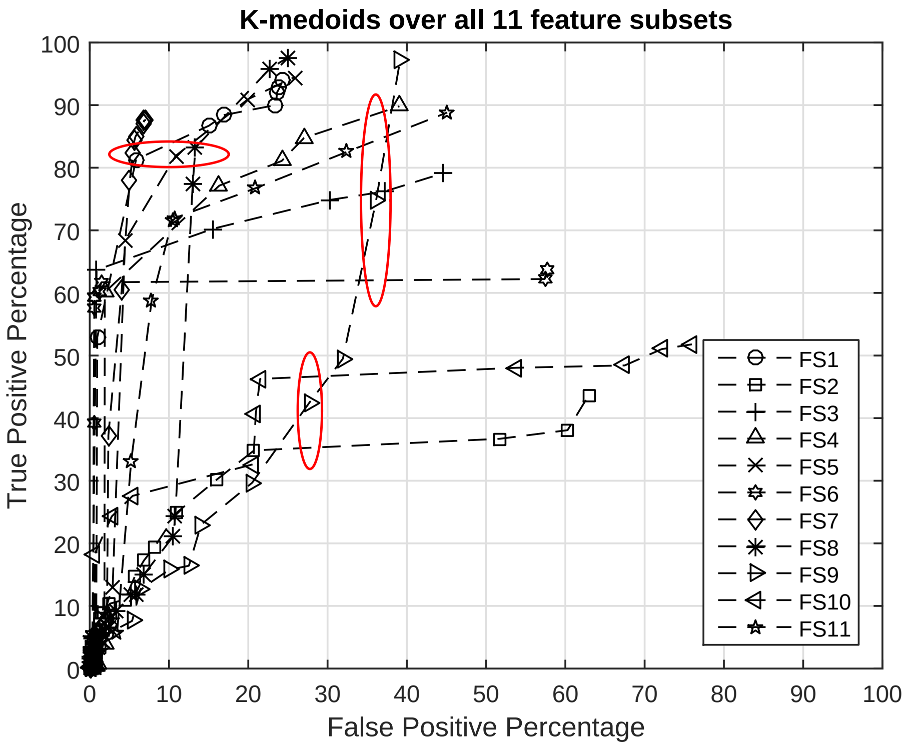


FIGURE 3.8: *A comparison of the composite ROC curves obtained by applying the proposed classifier to the dataset using the k–medoids clustering algorithm applied over each feature subset.*

The first group consists of feature subsets FS1, FS5, FS7 and FS8; these subsets produce a superior classification performance with higher TP and lower FP percentages, as compared to the remaining feature subsets. FS7 produces TP percentages between 81% and 88%, with corresponding FP percentages between 5% and 7%. FS8 produces the highest TP percentage of 97% amongst all remaining feature subsets, with a corresponding FP percentage of 25%. FS7 produces superior classification performance amongst all feature subsets applied over the k–medoids clustering algorithm, where points that lie towards the upper left corner of the graph are indicative of better performance.

The second group consists of feature subsets FS3, FS4, FS6, FS9 and FS11, where FS3, FS4 and FS6 produce FP percentages of less than 2%, but with TP percentages between 57% and 64%. The performance of FS9 severely reduces in comparison to that obtained using the k–means clustering algorithm, at lower FP percentages, however, at higher FP percentages, specifically at 39%, FS9 produces a TP percentage of 97%, similar to the performance produced by FS8, however, with a higher corresponding FP percentage.

The third group consists of feature subsets FS2, FS9 and FS10. FS9 is included in the third group as well, as it produces similar inferior classification performances to FS2 and FS10, at FP percentages less than 30%. The inferior classification performance of both FS2 and FS10, which both exhibit significantly high FP percentages (up to 75%), with low TP percentages (up to 52%) is a result of the k–medoids algorithm grouping a large number of both legitimate and attack data samples into a single cluster. Table 3.11 provides the contents of each cluster found by applying the k–medoids algorithm over feature subset FS2, with $K = 5$. The TP and FP percentages for this particular clustering result is approximately 33% and 47% respectively, which lies in close proximity to the composite ROC curve provided in figure 3.8. The contents of each cluster is presented as the number of data samples belonging to a cluster, that correspond to each of the four broad classes of network attacks described in section 3.2.1.1, or correspond to the "normal" class, which represents legitimate network traffic.

Table 3.11 reveals that the largest cluster consists of a large number of both legitimate data samples, and DoS data samples, which constitutes 64% of all the attack data samples present in the dataset. This accounts for the low TP percentages observed in figure 3.8

for feature subset FS2. The high FP percentages is attributed to the fact that 43% of all legitimate data samples are grouped into the second largest cluster, thus for $N_r = 1$, these data samples are all misclassified as attacks.

It is observed that the k–medoids clustering algorithm when applied over feature subsets FS5, FS7 and FS8 outperforms the same clustering algorithm when applied over the full feature set FS1.

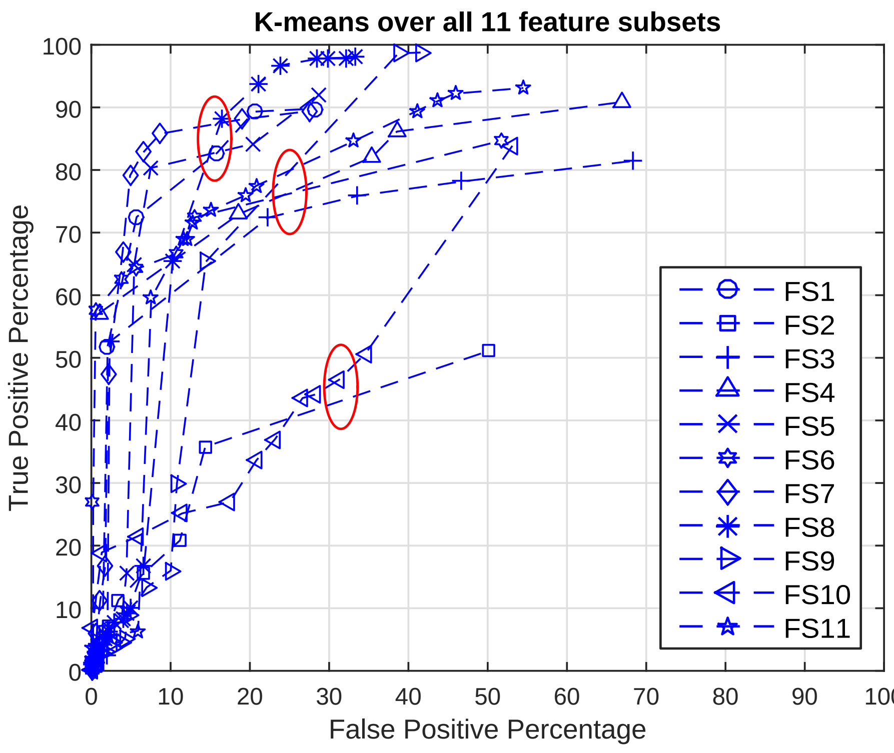
TABLE 3.11: *Cluster contents obtained by applying the k–medoids clustering algorithm over FS2 with $K = 5$*

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**
DoS	9	139	1 397	7 502	187
Probe	810	80	1 187	212	0
R2L	5	126	11	67	0
U2R	1	7	0	3	0
Normal	97	5 737	617	6 998	0
Total	**922**	**6 089**	**3 212**	**14 782**	**187**

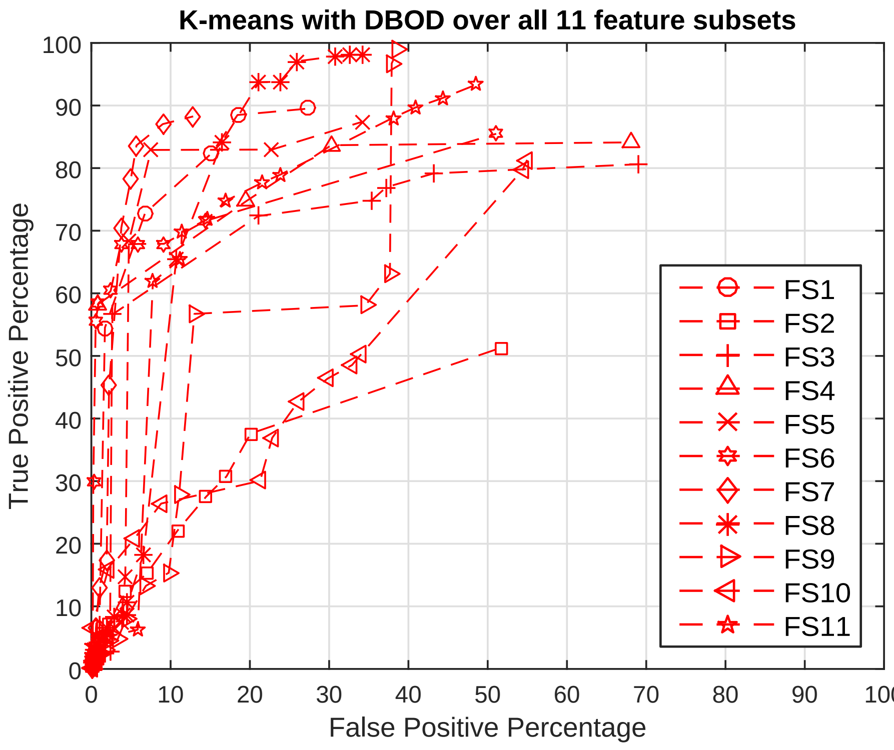
3.6.2.3.c K–means clustering with distance-based outlier detection

Figures 3.9a and 3.9b present the composite ROC curves that were obtained by applying the proposed classifier to the dataset using the k–means clustering algorithm and the k–means clustering algorithm in combination with the distance-based outlier detection algorithm, over each of the 11 feature subsets, respectively.

The figures reveal that both algorithms produce similar results over the majority of feature subsets. The distance-based outlier detection method with k–means clustering (shown in figure 3.9b) typically produces marginal changes in both the TP and FP percentages obtained over a number of feature subsets. Table 3.12 provides the TP and FP percentages that correspond to the composite ROC curves presented in figures 3.9a and 3.9b, which were produced by applying both the k–means and k–means with distance-based outlier detection algorithms over feature set FS1. The table also includes the change in TP and FP percentages between both composite ROC curves.



(a) K–means



(b) K–means with DBOD

FIGURE 3.9: *(a) Composite ROC curves obtained by applying the proposed classifier using k–means clustering over all 11 feature subsets, (b) Composite ROC curves obtained by applying the proposed classifier using k–means clustering in combination with DBOD over all 11 feature subsets*

Table 3.12: *TP and FP percentages of the composite ROC curves produced over FS1, provided in figures 3.9a and 3.9b*

k–means over FS1		**k–means with DBOD over FS1**		**Change from k–means to k–means with DBOD**	
TP (%)	**FP (%)**	**TP (%)**	**FP (%)**	**Δ TP (%)**	**Δ FP (%)**
1.01	0.26	0.81	0.25	-0.20	-0.01
1.20	0.33	1.46	0.48	0.26	0.15
2.48	0.53	2.14	0.53	-0.34	0.00
3.82	0.86	3.39	0.89	-0.43	0.04
4.93	1.00	5.24	1.41	0.31	0.40
54.51	1.71	51.87	2.06	-2.64	0.34
72.79	6.77	72.49	5.72	-0.30	-1.05
82.40	15.21	82.75	15.88	0.35	0.67
88.48	18.44	89.32	20.53	0.83	2.09
89.51	27.42	89.80	28.12	0.28	0.70

Table 3.12 illustrates the marginal changes in TP and FP percentages over feature subset FS1. This trend is similar amongst several of the remaining feature subsets. However, feature subset FS9 exhibits a significant difference in both TP and FP percentages. In figure 3.9b, FS9 exhibits a drastic increase in FP percentage from 13% to 34% with only a 1% increase in TP percentage. However, in figure 3.9a, FS9 does not exhibit this drastic increase at an FP percentage of 13%. This can also be seen in figure 3.6i. An increase in FP percentage, implies that a clustering algorithm divides a larger number of legitimate data samples amongst smaller clusters pertaining to attack clusters, thus these are misclassified as attack data samples.

Owing to the similarity between the classification performance obtained by the k–means clustering algorithm and the k–means with DBOD algorithm, the same observation can be made as with the k–means clustering algorithm, where feature subsets FS5, FS7 and FS8 provide improved classification performance over all remaining feature subsets including the full feature set FS1.

3.6.2.3.d Expectation Maximization using GMMs

Figure 3.10 provides the composite ROC curves obtained by applying the proposed classifier to the dataset using the EM clustering algorithm with GMMs over each of the 11 feature subsets. The figure reveals that the performance of the algorithm varies significantly when applied over the different feature subsets.

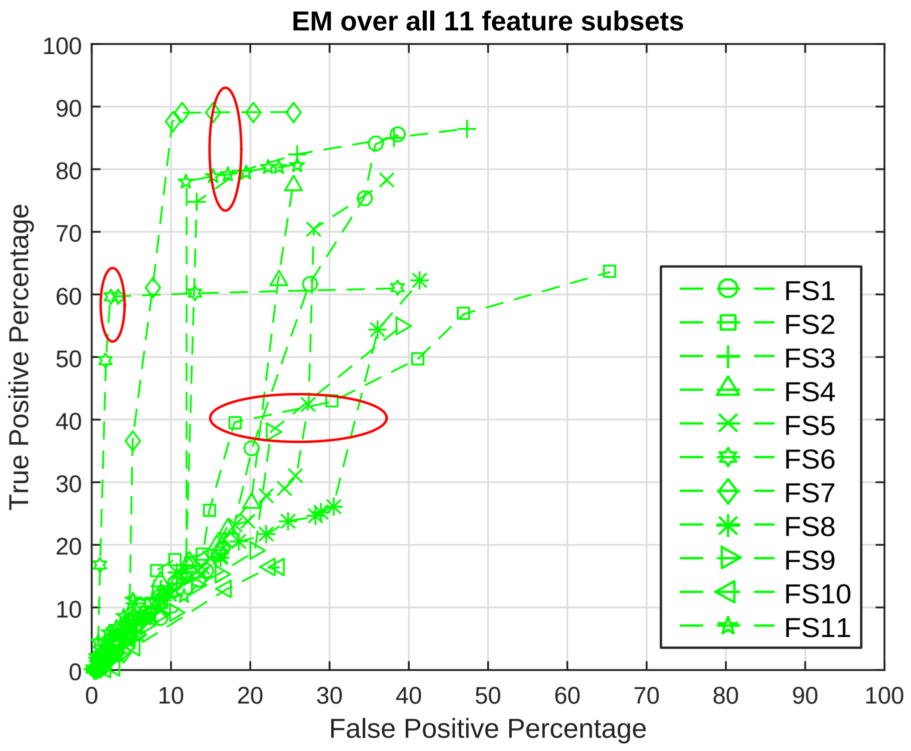


FIGURE 3.10: *A comparison of the composite ROC curves obtained by applying the proposed classifier to the dataset using the expectation maximization clustering algorithm with Gaussian mixture models, applied over each feature subset.*

The feature subsets can be divided into three groups, based on the classification performance produced by each feature subset (as indicated by the red ellipses in figure 3.10).

The first group consists of feature subsets FS3, FS7 and FS11 which provide superior classification performance over all remaining feature subsets. Feature subset FS7 provides the best performance over all feature subsets with TP percentages between 87% and 89%, which is an improvement of 11% when compared to the performance of feature subsets FS3 and FS11, at FP percentages between 10% and 12%.

The second group consists of only feature subset FS6, as this is the only feature subset to produce FP percentages that are less than 4%, at a reasonable TP percentage of 60%.

The third group consists of all remaining feature subsets, FS1, FS2, FS4, FS5, FS8, FS9 and FS10. These feature subsets, in general, provide inferior classification performance when compared to all other feature subsets. Exceptions in this group include feature subsets FS1, FS4 and FS5, which produce relatively high TP percentages of 84%, 77% and 78%, respectively, but in most cases the classification performance of these subsets is inferior.

The EM clustering algorithm tends to be sensitive to the features that are present in each feature subset. For instance, the three groups of feature subsets presented in figure 3.10, are dissimilar to the groups presented in figures 3.7 and 3.8. There is also no direct correlation between the dimensionality of a feature subset and the classification performance that the EM clustering algorithm produces over the subset. For example, feature subsets FS6, FS10 and FS11 consists of four features, but it is observed that the classification performance produced by each feature subset greatly differs. In figure 3.10, it is observed that FS6, FS10 and FS11 produce an FP percentage of 24%, however, FS11, FS6 and FS10 produce corresponding TP percentages of 16%, 60% and 81%, respectively.

Additionally, feature subset FS10 produces empty clusters when $K > 5$. An interesting observation is that FS6 and FS10 contain three of the same feature, and FS6, FS10 and FS11 contain two of the same features. This demonstrates that the EM clustering algorithm is sensitive to the actual features and feature values present in the dataset, rather than the dimensionality of the feature subset.

An important note is that the EM clustering algorithm, when applied over feature subsets FS3, FS4, FS5, FS6, FS7 and FS11 outperforms the same clustering algorithm when applied over the full feature set FS1.

3.7 Conclusion

In this chapter an unsupervised classifier for anomaly-based network intrusion detection was presented. A functional block diagram of the proposed classifier was provided and a detailed description of each block was carried out. These descriptions include the dataset considered, the data transformation methods utilized, the clustering algorithms considered, and the cluster labelling scheme used in this research. The proposed classifier was used to carry out three experiments, that is, the investigation of the impact of the number of clusters K, a comparison of the performance of the four clustering algorithms, and a comparison of the performance attained by the four clustering algorithms when applied to various feature subsets found in the literature.

The experimental results revealed that the number of clusters K has a significant impact on the classification performance attained by each clustering algorithm. A typical trend is that the FP percentages of each clustering algorithm increases as the value of K increases, while the TP percentage increases up to a maximum for a specific value of K, and decreases for subsequent values of K. The k–means, k–means with distance-based outlier detection and EM clustering algorithms all reach a maximum TP percentage at $K = 6$, with lower TP percentages at $K > 6$. The k–medoids clustering algorithm reaches a maximum TP percentage at $K = 10$.

The comparison of the clustering algorithms revealed that one single clustering algorithm, for example k–means, does not outperform all other clustering algorithms for all feature subsets. This is evident when applying the clustering algorithms over feature subsets FS9 and FS1, where the former enables the k–means algorithm to outperform all others, while the latter enables the k–medoids algorithm to outperform all others. It was also discovered that the EM clustering algorithm can be sensitive to the number of clusters depending on the feature subsets used, as was evident with the application of the EM clustering algorithm over feature subsets FS6 and FS10.

The comparison of the feature subsets revealed that the classification performance of the clustering algorithms varies significantly over different feature subsets. In all clustering algorithms, the feature subsets were grouped together based on classification performance, where certain feature subsets enabled the clustering algorithms to produce higher TP percentages at lower corresponding FP percentages. Amongst the k–means, k–medoids and k–means with distance-based outlier detection clustering algorithms, feature subsets FS5, FS7 and FS8 provided superior classification performance over all remaining feature subsets. For the EM clustering algorithm, feature subsets FS3, FS7 and FS11 provided superior performance. All these feature subsets provide classification performances that are superior to that produced by the full feature set (FS1), this further motivates the need for feature selection. An important observation is that there is no direct correlation between the dimensionality of a feature subset and the classification performance produced by the feature subset, when applied to the clustering algorithms of the proposed classifier. This is evident in the fact that FS6, FS10 and FS11 all contain four features, however, the TP and FP percentages produced by these feature subsets over all clustering algorithms, varies significantly.

Chapter 4

A Cluster Validity-based Feature Selection Algorithm

4.1 Introduction

The proposed unsupervised feature selection algorithm is presented in this chapter. The algorithm is a wrapper-based technique that makes use of the k–means clustering algorithm. It uses normalized Cluster Validity Indices (CVIs) as an objective function to be minimized over the search space of candidate feature subsets via a genetic algorithm. The concept behind the proposed feature selection algorithm is that better[1] cluster validity indices, as individually computed after clustering of a dataset over a generation of candidate feature subsets, are indicative of more relevant candidate feature subsets. In other words, a fit candidate feature subset is expected to produce an improved clustering result, with clusters that are more compact and with greater separation between clusters. In turn, these candidate feature subsets translate into improved classifier performance.

The advantage of this proposed approach to feature selection is that the algorithm is not dependent on labelled data. This is due to the fact that the learning algorithm used in the proposed wrapper-based feature selection algorithm is unsupervised, and that the computation of cluster validity indices does not involve the use of labelled data. To the

[1]The term “better” is used in this context, as for some CVIs such as the DB index [125], smaller values are associated with improved clustering results, while the opposite holds for other CVIs

best of the author's knowledge, the use of CVIs to construct an unsupervised feature selection algorithm is novel in the context of anomaly-based network intrusion detection.

The proposed wrapper-based feature selection algorithm has been successfully used in application areas other than network intrusion detection [126] (refer to [83] for a summary of this technique, and related unsupervised feature selection algorithms). In [126], the proposed algorithm was applied as a precursor to the recognition of handwritten text using a supervised algorithm for classification. It was demonstrated that the proposed algorithm produced smaller feature subsets that resulted in classification performance comparable to that obtained using the full feature set, but with reduced execution time.

Figure 4.1 contains the functional block diagram of the proposed feature selection algorithm implemented in this research. Each block of this diagram is summarised in what follows, with the exception of the dataset description and the data transformation block. The description of the NSL-KDD dataset [25] can be found in section 3.2.2, and the description of the data transformation block can be found in section 3.3.

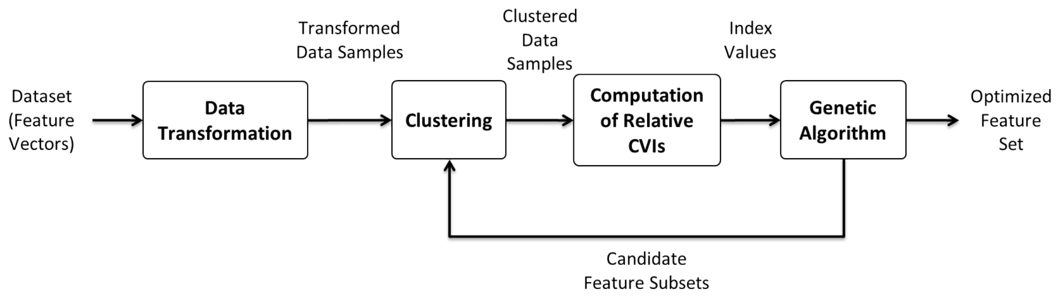


FIGURE 4.1: *Functional block diagram of the proposed feature selection algorithm.*

4.2 Clustering

The k–means clustering algorithm (algorithm 3.1) is applied to the transformed dataset over each candidate feature subset in the current generation produced by the genetic algorithm. Owing to the fact that the k–means algorithm is sensitive to the choice in initial cluster centres, the proposed implementation repeats k–means clustering C_R times for each candidate feature subset, with initial cluster centres selected randomly from the samples of the dataset at the start of each execution.

A full summary of the k–means clustering algorithm is provided in section 3.4.1.1. The pseudocode for the k–means clustering algorithm, as implemented in the proposed feature selection algorithm, is provided in algorithm 4.1.

Algorithm 4.1 K–means for each candidate feature subset over C_R repetitions

```
Function $[\mathbf{C}, \mathbf{L}] = repeated\_kmeans(\mathbf{X}, K, \mathbf{F_s}, C_R)$
Inputs:
  $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$,        ▷ A set of N feature vectors, corresponding
                                                  ▷ to the N dataset samples
  $K$,                                            ▷ The number of clusters
  $\mathbf{F_s} = \{\mathbf{F_s}(d),\ d = 1, 2, \ldots, M\}$,   ▷ Set of M candidate feature subsets in current
                                                  ▷ generation returned by GA
  $C_R$        ▷ Number of repetitions of k–means with random initial cluster centres
Outputs:
  $\mathbf{C} = \{\mathbf{C}(d, z) | d = 1, 2, \ldots, M;\ z = 1, 2, \ldots, C_R\}$, where
    $\mathbf{C}(d, z) = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_K}\}$,   ▷ The K cluster centres for each clustering result
  $\mathbf{L} = \{\mathbf{L}(d, z) | d = 1, 2, \ldots, M;\ z = 1, 2, \ldots, C_R\}$, where
    $\mathbf{L}(d, z) = \{l(\mathbf{x_a}) | a = 1, 2, \ldots, N\}$   ▷ Cluster assignments for all data samples
                                                  ▷ for each clustering result

▷ Cluster $C_R$ times for each candidate feature subset
1: for $d = 1$ to $M$ do
2:     for $z = 1$ to $C_R$ do
3:         $[\mathbf{C}(d, z),\ \mathbf{L}(d, z)] \leftarrow kmeans(\mathbf{X},\ K,\ \mathbf{F_s}(d))$ ;   ▷ Algorithm 3.1
4:     end for
5: end for

6: Return $(\mathbf{C}, \mathbf{L})$
```

The input of the algorithm includes the transformed dataset $\mathbf{X}$, the number of clusters K, and the number of repetitions C_R of the k–means clustering algorithm to perform, as in algorithm 3.1. An additional input to the algorithm is the population of M candidate feature subsets $\mathbf{F_s}(d),\ d = 1, 2, \ldots, M$, that is produced by the GA, over which clustering is to be applied. The algorithm subsequently performs k–means clustering C_R times over each feature subset $\mathbf{F_s}(d)$, each with a randomly selected set of initial cluster centres. This produces a total of $M \times C_R$ clustering results, represented by cluster centres $\mathbf{C}(d, z)$ and sample assignments $\mathbf{L}(d, z)$, where $d = 1, 2, \ldots, M$, and $z = 1, 2, \ldots, C_R$.

It should be noted that the value of the number of clusters K remained fixed during feature selection. Feature selection was repeated with different values of K to obtain different feature subsets, which are compared in section 4.6.2.2.

4.3 Relative cluster validity indices

Cluster Validity Indices (CVIs) were used as a measure of clustering quality after applying the k–means clustering algorithm to the transformed dataset over each candidate subset of features. Relative CVI scores are computed as a function of the degree of *compactness* and *separation* of the resultant clusters in the feature space, as defined in what follows [62].

- Compactness requires that the samples belonging to a cluster be as close to each other as possible with regards to distances in the feature space.
- Separation requires that distinct clusters be as far apart as possible. Three measures that may be used to compute the separation between two clusters are [62]:

    1. Single linkage, which involves the distance between the two closest samples of two clusters.
    2. Complete linkage, which involves the distance between the two furthest samples of two clusters.
    3. Comparison of centroids, which involves the distance between the cluster centres.

Several relative cluster validity indices that may be calculated for a specific clustering result include the Davies-Bouldin (DB) [125], Dunn [127], Silhouette [128], *S_Dbw* [129] and

Calinski-Harabasz (CH) indices [130]. These indices differ in the selection of compactness and separation measures, as well as the manner in which these measures are combined to obtain the value of the CVI.

A previous feature selection algorithm for text classification [126], that is similar to the proposed algorithm, made use of the DB index due to its linear time complexity and the fact that it is unbiased with respect to the number of clusters used [83]. In this research a normalized version of the DB index is utilized as the relative CVI in the feature selection algorithm.

The DB index was first introduced by Davies and Bouldin in 1979 [125]. To define the DB index, suppose a given clustering result consists of K clusters $\mathbf{H_1}, \mathbf{H_2}, \ldots, \mathbf{H_K}$ and corresponding cluster centres $\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_K}$, where $\mathbf{H_k}$ represents the set of data samples belonging to cluster k. The DB index is defined as a function of a similarity measure $R_{i,j}$ between pairs of clusters $\mathbf{H_i}$ and $\mathbf{H_j}$, where $i, j = 1, 2, \ldots, K$ with $i \neq j$. The measure of similarity $R_{i,j}$, is defined as a function of compactness g_i and separation $d_{i,j}$ scores.

The compactness g_i of a cluster $\mathbf{H_i}$ is defined as the average Euclidean distance between samples belonging to a cluster and its corresponding cluster centre,

$$g_i = \frac{1}{|\mathbf{H_i}|} \sum_{\mathbf{x} \in \mathbf{H_i}} d(\mathbf{x}, \mathbf{c_i}), \tag{4.1}$$

where $d(\mathbf{x}, \mathbf{c_i})$ denotes the Euclidean distance between sample $\mathbf{x}$ and its corresponding cluster centre $\mathbf{c_i}$, and $|\mathbf{H_i}|$ is the number of samples in cluster $\mathbf{H_i}$. The separation between clusters $\mathbf{H_i}$ and $\mathbf{H_j}$ is defined as the pairwise Euclidean distance between cluster centres,

$$d_{i,j} = d(\mathbf{c_i}, \mathbf{c_j}). \tag{4.2}$$

The similarity measure $R_{i,j}$ of the DB index is defined as

$$R_{i,j} = \frac{g_i + g_j}{d_{i,j}}. \tag{4.3}$$

A small value of $R_{i,j}$ indicates that clusters are compact and well-separated, which indicates that clusters $\mathbf{H_i}$ and $\mathbf{H_j}$ are highly dissimilar. It is proved in [125] that the similarity measure $R_{i,j}$ of equation 4.3 satisfies the following conditions for a real-valued function to be both a distance function and a dispersion function [125]:

1. $R_{i,j} \geq 0$
2. $R_{i,j} = R_{j,i}$
3. $R_{i,j} = 0$, iff $g_i = g_j = 0$
4. if $g_j > g_k$ and $d_{i,j} = d_{i,k}$ then $R_{i,j} > R_{i,k}$
5. if $g_j = g_k$ and $d_{i,j} < d_{i,k}$ then $R_{i,j} > R_{i,k}$

Let $R_{i,j}(d,z)$ denote the similarity measure obtained from the evaluation of equation 4.3 using cluster centres $\mathbf{C}(d,z)$ and sample assignments $\mathbf{L}(d,z)$, where d refers to the d^{th} candidate feature subset $\mathbf{F_s}(d)$, where $d = 1, 2, \ldots, M$, in the population of M candidate feature subsets produced by the GA, and z refers to the z^{th} clustering result of the k–means clustering algorithm, where $z = 1, 2, \ldots, C_R$ (refer to algorithm 4.1). The DB index corresponding to this clustering result is defined as

$$DB(d,z) = \frac{1}{K}\sum_{i=1}^{K} R_i(d,z), \tag{4.4}$$

where K represents the number of clusters. In equation 4.4, $R_i(d,z)$ is the maximum value of $R_{i,j}(d,z)$ over clusters $\mathbf{H_j}$, where $j = 1, 2, \ldots, K$; that is,

$$R_i(d,z) = \max_{j=1,\ldots,K, j\neq i} R_{i,j}(d,z), \quad i = 1, \ldots, K. \tag{4.5}$$

$DB(d,z)$ may therefore be interpreted as the average of the similarity scores between each cluster and the cluster most similar to it. Note that the DB index is defined in such a manner that smaller numerical values of the DB index corresponds to better clustering quality [129].

The DB index, as defined in equation 4.4, tends to be biased towards lower dimensional feature subsets if the index is used as a measure for comparing clustering quality over feature subsets with different dimensions [83, 126]. This is owing to the fact that the Euclidean distance between two points becomes smaller when the number of dimensions over which the calculation is performed is reduced; since smaller DB index values indicate better clustering quality, the index will favour lower dimensional feature subsets.

Handl et al. [83] investigated two possible approaches to overcoming the bias of the index by using the feature cardinality (the number of features in the subset) as an additional objective function for the genetic algorithm, thereby reformulating the problem as one of multi-objective optimization. The first approach that was investigated consisted of the minimization of the DB index value as the first objective, while simultaneously maximizing the number of features as the second objective. The motivation for this approach is that the additional objective of maximizing the number of features counteracts the bias of the DB index value towards lower dimensional feature subsets. The second approach consisted of the minimization of the DB index value that is normalized by the feature cardinality as the first objective, while simultaneously minimizing the number of features as the second objective. The normalized DB index is defined as

$$NDB(d,z) = \frac{1}{|\mathbf{F_s}(d)|} DB(d,z), \tag{4.6}$$

where $|\mathbf{F_s}(d)|$ represents the cardinality of the candidate feature subset $\mathbf{F_s}(d)$ that was used during clustering. The motivation for this approach stems from the observation in [83, 126] that the normalized DB index tends to be biased towards feature subsets with higher dimensionality. By including the minimization of the number of features as a second objective, this bias is counteracted.

Single-objective optimization, using the NDB index of equation 4.6, as well as multi-objective optimization using the NDB index (equation 4.6) and the number of features, were separately considered in this research. Specifically, for the single-objective optimization, the objective function is defined as the average of the NDB index values obtained over the C_R repetitions of the k–means algorithm, which is defined as

$$NDB_{avg}(d) = \frac{1}{C_R}\sum_{z=1}^{C_R} NDB(d,z), \tag{4.7}$$

where d represents the index of feature subset $\mathbf{F_s}(d),\ d = 1, 2, \ldots, M$, in the population of M candidate feature subsets. For the multi-objective optimization, the first objective function is the average of the NDB index values obtained over the C_R repetitions of the k–means clustering algorithm (equation 4.7), and the second objective function is the number of features in the candidate feature subset.

The pseudocode for the computation of the average NDB index value (equation 4.7), as implemented in the proposed feature selection algorithm, is provided in algorithm 4.2.

The algorithm iterates over each of the C_R clustering results obtained over each candidate feature subset and calculates the NDB index for each clustering result. The average of the index, as associated with each candidate feature subset is calculated over the C_R clustering results.

Algorithm 4.2 CVI computation for each candidate feature subset over C_R repetitions

```
Function [$\mathbf{NDB_{avg}}$] = $Avg\_CVI(\mathbf{X}, K, \mathbf{C}, \mathbf{L}, C_R)$
Inputs:
  $\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$,          ▷ A set of N feature vectors, corresponding
                                                  ▷ to the N dataset samples
  $K$,                                            ▷ The number of clusters
  $\mathbf{C} = \{\mathbf{C}(d,z) | d = 1, 2, \ldots, M;\ z = 1, 2, \ldots, C_R\}$, where
    $\mathbf{C}(d,z) = \{\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_K}\}$,   ▷ The K cluster centres for each clustering result
  $\mathbf{L} = \{\mathbf{L}(d,z) | d = 1, 2, \ldots, M;\ z = 1, 2, \ldots, C_R\}$, where
    $\mathbf{L}(d,z) = \{l(\mathbf{x_a}) | a = 1, 2, \ldots, N\}$     ▷ Cluster assignments for all data samples
                   ▷ for each clustering result over each candidate feature subset
  $C_R$     ▷ Number of repetitions of k-means with random initial cluster centres
Output:
  $\mathbf{NDB_{avg}}$,            ▷ Average NDB index value for each candidate feature
                                 ▷ subset over all $C_R$ clustering results
```

```
   ▷ Compute average NDB index over $C_R$ clustering results for
   ▷ each candidate feature subset
1: for $d = 1$ to $M$ do
2:     for $z = 1$ to $C_R$ do
3:         $\mathbf{NDB}(d, z) \leftarrow Compute\ NDB\_Index\_Value(\mathbf{C}(d, z), \mathbf{L}(d, z))$ ;   ▷ Equation 4.6
4:     end for
5:     $\mathbf{NDB_{avg}}(d) \leftarrow mean(\mathbf{NDB}(d))$ ;   ▷ Equation 4.7
6: end for

7: Return [$\mathbf{NDB_{avg}}$]
```

4.4 Genetic algorithm

Genetic algorithms are heuristic search algorithms that are based on principles of evolution and natural selection [2]. These algorithms are commonly used in optimization problems, where a search is performed over a space of candidate solutions for a solution that is considered to be optimal according to a specified measure. Genetic algorithms are considered in this research as they are robust and can be applied to a wide range of problem areas [131, 132]. GAs also explore a relatively large region of the solution space and are insensitive to reasonable amounts of noise, where noise is created from the variance of fitness values, noisy selection methods and the variance of genetic operations [133]. However, GAs can become computationally expensive as the size of the dataset and its number of dimensions increase [131].

GAs operate by converting problems into a framework that uses a data structure referred to as a chromosome, which represents a candidate solution to the problem [2]. Chromosomes, or individuals, are represented as character strings in the GA (this is analogous to the chromosomes found in DNA). Chromosomes are evolved through multiple generations through operations such as **selection**, **crossover** and **mutation**, which facilitate the traversal of the feature space.

The objective function is used to measure how well a chromosome solves the problem (that is, it measures the fitness of each chromosome). Chromosomes that provide good (or "fit") solutions to the problem are selected and crossed with one another, in an attempt to find more fit chromosomes. Mutation involves changes to random characters of the character string, and is used to maintain diversity from one generation to the next. It forces the genetic algorithm to explore more distant regions of the search space, allowing the GA to potentially find solutions that are external to the area surrounding a local optimum [131]. This approach towards optimization is inspired by Charles Darwin's theory of 'survival of the fittest', where only the 'fittest' individuals are selected for crossover and mutation while the 'unfit' ones are removed from the system, thereby improving the average fitness of all candidate solutions over successive generations [131]. The final output of the GA is the chromosome with the best fitness score from those chromosomes produced in the last generation of the GA, as returned by the objective (or fitness) function [131].

In this research, each chromosome represents a candidate feature subset of the feature space considered, which is a candidate solution to the problem of optimising the CVI. A chromosome in the GA of the feature selection algorithm proposed in this research is encoded as a bit string, where each bit represents one of the original 41 candidate features of the dataset. A value of 1 indicates that the corresponding feature is present in the feature subset, whereas a value of 0 indicates that the corresponding feature is not present in the feature subset.

Two types of genetic algorithms were applied in this research. The first corresponds to the single-objective optimization problem, which produces a candidate feature subset to minimize the NDB index. The second type corresponds to the multi-objective optimization problem, which involves the minimization of the NDB index, and the minimization of the number of features in the candidate feature subset.

4.4.1 Single-objective genetic algorithm

The single-objective genetic algorithm implementation attempts to minimize the NDB index as an objective function over the space of all possible candidate feature subsets. Specifically, the average value of the NDB index, as computed over multiple C_R clustering results over a feature subset (equation 4.7), is used as the objective function. The pseudocode for the single-objective genetic algorithm, as applied in this research, is provided in algorithm 4.3.

Algorithm 4.3 Single-Objective Genetic Algorithm (SOGA)

```
Function $[\mathbf{F_s}^{(t+1)}] = SOGA(\mathbf{F_s}^{(t)}, \mathbf{NDB_{avg}}^{(t)}, t)$
Inputs:
    $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M,$          ▷ Current generation (i.e. generation t) of M
                                                             ▷ candidate feature subsets
    $\mathbf{NDB_{avg}}^{(t)}(d),\ d = 1, 2, \ldots, M,$    ▷ Average NDB index value corresponding to
                                                             ▷ candidate feature subsets in $\mathbf{F_s}^{(t)}(d)$
    $t,\ t = 0, 1, 2, \ldots, N_g$                          ▷ Current generation number
Output:
    $\mathbf{F_s}^{(t+1)}(d),\ d = 1, 2, \ldots, M,$        ▷ Next generation of M candidate feature subsets
Constants:
    $M$,        ▷ Number of candidate feature subsets in each generation
    $N_F$,      ▷ Dimensionality of full, initial feature set
    $r_e$,      ▷ Percentage of top (elite) candidates to select in generation t,
                ▷ for appearance in generation $t + 1$
    $r_c$,      ▷ Percentage of candidates to produce from crossover,
                ▷ for appearance in generation $t + 1$
    $p_m$,      ▷ Mutation probability for a feature
    $N_g$       ▷ Maximum number of generations

1: if $t = 0$ then                    ▷ If this is the first generation
2:     for $d = 1$ to $M$ do           ▷ Produce initial generation of candidate feature subsets

    ▷ Uniformly draw candidate feature subsets from a collection
    ▷ of all candidate feature subsets
3:         $\mathbf{F_s}^{(1)}(d) \leftarrow random\_uniform\_sample(N_F)$ ;
4:     end for
5:     Return $\mathbf{F_s}^{(1)}$ ;
```

```
 6: else if $t = N_g$ then                                  ▷ If this is the final generation
 7:     $d_{max} \leftarrow \arg\min_d(\mathbf{NDB_{avg}}^{(N_g)}(d))$ ;  ▷ Find the fittest candidate feature subset in
 8:                                      ▷ the current generation and return this candidate
 9:     $\mathbf{F_s}^{(N_g+1)} \leftarrow \mathbf{F_s}^{(N_g)}(d_{max})$ ;
10:     Return $\mathbf{F_s}^{(N_g+1)}$ ;
11: end if

12: $N_e \leftarrow ceil(M \times r_e)$ ;          ▷ Number of candidates in next generation obtained
                                                   ▷ via elitism in current generation
13: $N_c \leftarrow round((M - N_e) \times r_c)$ ;  ▷ Number of candidates in next generation obtained
                                                   ▷ via crossover in current generation
14: $N_m \leftarrow (M - N_e - N_c)$ ;             ▷ Number of candidates in next generation obtained
                                                   ▷ via mutation in current generation

    ▷ Sort average NDB index values in ascending order, and calculate ranking of each
    ▷ candidate in sorted list
    ▷ (i.e. $\mathbf{NDB_{avg}}(\mathbf{Rank}(d)) \leftarrow \mathbf{Sorted\_NDB_{avg}}(d)$, for $d = 1, 2, \ldots, M$)
15: $[\mathbf{Sorted\_NDB_{avg}}, \mathbf{Rank}] \leftarrow sort(\mathbf{NDB_{avg}}, ascending)$ ;

    ▷ Selection (Algorithm 4.4)
16: $[\mathbf{F_{sel,elite}},\ \mathbf{F_{sel,cross}},\ \mathbf{F_{sel,mut}}] \leftarrow GA\_Selection(\mathbf{F_s}^{(t)}, N_e, N_c, N_m, N_F, \mathbf{Rank})$ ;

    ▷ Elitism
    ▷ Top ranked $N_e$ candidates in generation $t$ appears unchanged in next generation
17: $[\mathbf{F_{elite}}] \leftarrow \mathbf{F_{sel,elite}}$ ;

    ▷ Scattered Crossover (Algorithm 4.5)
    ▷ Produce $N_c$ candidates for next generation through crossover
18: $[\mathbf{F_{cross}}] \leftarrow GA\_Crossover(\mathbf{F_{sel,cross}}, N_c, N_F)$ ;

    ▷ Uniform Mutation (Algorithm 4.6)
    ▷ Produce $N_m$ candidates for next generation through mutation
19: $[\mathbf{F_{mut}}] \leftarrow GA\_Mutation(\mathbf{F_{sel,mut}}, N_m, N_F, p_m)$ ;
```

```
    ▷ Create next generation (i.e. generation $t + 1$)
20: ${\mathbf{F_s}}^{(t+1)}(1, \ldots, N_e) \leftarrow \mathbf{F_{elite}}$ ;
21: ${\mathbf{F_s}}^{(t+1)}(N_e + 1, \ldots, N_e + N_c) \leftarrow \mathbf{F_{cross}}$ ;
22: ${\mathbf{F_s}}^{(t+1)}(N_e + N_c + 1, \ldots, N_e + N_c + N_m) \leftarrow \mathbf{F_{mut}}$ ;

23: Return ${\mathbf{F_s}}^{(t+1)}$
```

The genetic algorithm first determines whether the first generation ${\mathbf{F_s}}^{(1)}$ is to be constructed ($t = 0$), or whether the best candidate is to be selected from the final generation ${\mathbf{F_s}}^{(N_g)}$ (corresponding to $t = N_g$). In the former case, the GA constructs an initial population ${\mathbf{F_s}}^{(1)}(d),\ d = 1, 2, \ldots, M$, of M candidate feature subsets, where each candidate is randomly and uniformly selected from all possible candidate feature subsets. The algorithm subsequently returns this set of candidates. In the latter case, the GA selects and returns the candidate feature subset from the final generation $\mathbf{F_s}(d)$ with the best NDB index value as the fittest candidate found during optimization. If neither the first case ($t = 0$) nor the last case ($t = N_g$) applies, the GA performs the steps of selection, elitism, crossover and mutation to produce the next generation ${\mathbf{F_s}}^{(t+1)}(d),\ d = 1, 2, \ldots, M$.

The selection step consists of the selection of those candidates in the current generation to be directly included in the next generation (elitism), those to undergo crossover, and those to undergo mutation ($\mathbf{F_{sel,elite}}$, $\mathbf{F_{sel,cross}}$ and $\mathbf{F_{sel,mut}}$, respectively). New candidate feature subsets are subsequently produced through elitism, crossover and mutation ($\mathbf{F_{elite}}$, $\mathbf{F_{cross}}$ and $\mathbf{F_{mut}}$, respectively). These candidates constitute the next generation, which is returned. The selection, elitism, crossover and mutation operations are discussed in what follows.

Selection

As an initial step to selection, the GA calculates the number of candidate feature subsets in the next generation to be derived using elitism (N_e), crossover (N_c) and mutation (N_m) in the current generation. This is illustrated via an example. Consider a generation consisting of 20 (M) candidate feature subsets, with the number of candidates in the next generation to be derived using elitism (N_e), set to 10% (r_e) of M, and the number of candidates in the next generation to be derived using crossover (N_c), set to 80% (r_c) of all remaining candidates, excluding the number of candidates required to be produced from elitism (N_e). Figure 4.2 illustrates the computations.

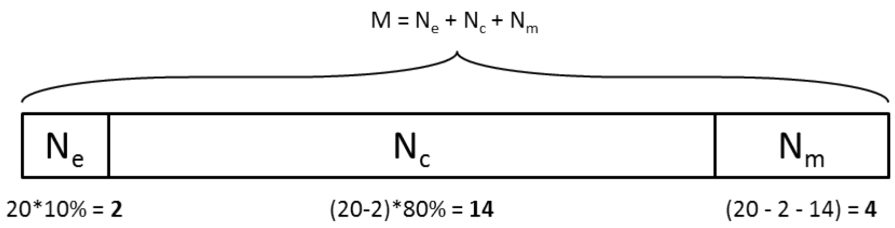


FIGURE 4.2: *Illustration of the computation of the number of candidates in the next generation to be derived from elitism (N_e), crossover (N_c) and mutation (N_m).*

The candidates for crossover and candidates for mutation are selected using the stochastic uniform selection method. The selected candidates are subsequently shuffled (i.e. randomly permuted) to randomize the order in which candidates are used in the crossover and mutation operations. The pseudocode for the selection of candidate feature subsets is provided in algorithm 4.4.

The algorithm proceeds by selecting N_e candidates in the current generation for elitism ($\mathbf{F_{sel,elite}}$), $2N_c$ candidates for crossover ($\mathbf{F_{sel,cross}}$) and N_m candidates for mutation ($\mathbf{F_{sel,mut}}$). The candidates selected for elitism constitute the top N_e ranked candidates in $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M$, according to the average NDB index value.

Stochastic uniform selection proceeds in three steps. Firstly, each candidate is assigned an expectation score $\mathbf{e}(d),\ d = 1, 2, \ldots, M$ based on their rank. Expectation scores are proportional to the likelihood of a candidate being selected. The expectation scores are calculated as

$$\mathbf{e}(d)' = \frac{\mathbf{e}(d)}{\sum_{d=1}^{M} \mathbf{e}(d)}, \tag{4.8}$$

Algorithm 4.4 Single-objective GA selection algorithm

```
Function [$\mathbf{F}_{\mathbf{sel,elite}}$, $\mathbf{F}_{\mathbf{sel,cross}}$, $\mathbf{F}_{\mathbf{sel,mut}}$] = $GA\_Selection(\mathbf{F_s}^{(t)}, N_e, N_c, N_m, N_F, Rank)$
Inputs:
    $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M,$        ▷ Current generation (i.e. generation t) of M
                                                         ▷ candidate feature subsets
    $N_e,$                                               ▷ Number of candidates in next generation obtained
                                                         ▷ via elitism in current generation
    $N_c,$                                               ▷ Number of candidates in next generation obtained
                                                         ▷ via crossover in current generation
    $N_m,$                                               ▷ Number of candidates in next generation obtained
                                                         ▷ via mutation in current generation
    $N_F,$                                               ▷ Dimensionality of full, initial feature set
    Rank        ▷ Ranking of each candidate feature subset in $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M,$
                                                         ▷ based on average DB index value
Outputs:
    $\mathbf{F}_{\mathbf{sel,elite}}(d),\ d = 1, 2, \ldots, N_e$      ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for elitism
    $\mathbf{F}_{\mathbf{sel,cross}}(d),\ d = 1, 2, \ldots, 2N_c$     ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for crossover
    $\mathbf{F}_{\mathbf{sel,mut}}(d),\ d = 1, 2, \ldots, N_m$        ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for mutation

1: $\mathbf{F}_{\mathbf{sel,elite}} \leftarrow \mathbf{F_s}^{(t)}(\mathbf{Rank}(1, \ldots, N_e))$     ▷ Select $N_e$ top ranked candidates
                                                         ▷ from $\mathbf{F_s}^{(t)}$ for elitism

   ▷ Stochastic uniform selection algorithm, consisting of three steps:
   ▷ computation of candidate expectation scores, computation of candidate segments,
   ▷ and iterative selection

   ▷ Computation of candidate expectation scores $\mathbf{e}(d),\ d = 1, 2, \ldots, M$:
   ▷ Expectation scores are inversely proportional to candidate's rank

2: for $d = 1$ to $M$ do
3:     $\mathbf{e}(d) \leftarrow [\mathbf{Rank}(d)]^{-\frac{1}{2}}$ ;
4: end for
```

```
 5: $e_{total} \leftarrow \sum_{d=1}^{M} \mathbf{e}(d)$ ;
 6: for $d = 1$ to $M$ do
 7:     $\mathbf{e}(d)' \leftarrow \frac{\mathbf{e}(d)}{e_{total}}$ ; ▷ Normalize each expectation score such that the sum equals unity,
                                                        ▷ and each score value lies within [0,1]
 8: end for

    ▷ Computation of candidate segments $\mathbf{s}(d),\ d = 1, 2, \ldots, M$:
    ▷ Numeric interval associated with each candidate, where the length of the interval
    ▷ is proportional to its likelihood of being selected
 9: $\mathbf{s}(1) \leftarrow [0, \mathbf{e}(1)')$ ;
10: for $d = 2$ to $M$ do
11:     $\mathbf{s}(d) \leftarrow [\ \sum_{j=1}^{d-1} \mathbf{e}(j)', \sum_{j=1}^{d} \mathbf{e}(j)'\ )$ ;
12: end for

    ▷ Iterative selection: construct uniformly spaced (by $\Delta_s$) sample points
    ▷ over [0,1], and select candidates based on the corresponding
    ▷ segment/interval in which each sample point falls
13: $\Delta_s \leftarrow \frac{1}{2N_c+N_m}$ ;
14: $rand\_start \leftarrow Random\ number\ drawn\ uniformly\ over\ [0,\ \Delta_s]$ ;
15: for $d = 1$ to $(2N_c + N_m)$ do
16:     $sample\_point \leftarrow rand\_start + ((d-1) \times \Delta_s)$ ;
17:     $\mathbf{F_{sel}}(d) \leftarrow \mathbf{F_s}^{(t)}(q)$ ;          ▷ where $sample\_point \in \mathbf{s}(q)$
18: end for
19: $\mathbf{F_{sel}} \leftarrow$ Random permutation($\mathbf{F_{sel}}$) ;

    ▷ Select $2N_c$ candidates for crossover from current generation
20: $\mathbf{F_{sel,cross}} \leftarrow \mathbf{F_{sel}}(1, \ldots, 2N_c)$ ;
    ▷ Select $N_m$ candidates for mutation from current generation
21: $\mathbf{F_{sel,mut}} \leftarrow \mathbf{F_{sel}}(2N_c + 1, \ldots, 2N_c + N_m)$ ;

22: Return $[\mathbf{F_{sel,elite}},\ \mathbf{F_{sel,cross}},\ \mathbf{F_{sel,mut}}]$
```

where

$$\mathbf{e}(d) = \frac{1}{\sqrt{\mathbf{Rank}(d)}}, \tag{4.9}$$

where **Rank** is the ranking of the $\mathbf{F_s}(d)$ candidates according to the average NDB index. Higher ranked candidates will have a larger expectation score implying that they are more likely to be selected. In the second step, each candidate feature subset in $\mathbf{F_s}(d)$ is assigned a subinterval $\mathbf{s}(d)$ of [0,1], where the length of the subinterval is proportional to its expectation score. Candidate d is assigned the interval $\mathbf{s}(d) = [\ \sum_{j=1}^{d-1} \mathbf{x}(j)', \sum_{j=1}^{d} \mathbf{e}(j)'\ )$ with $s(1) = [0, \mathbf{e}(1))$. In the third step of the stochastic uniform selection algorithm, candidates are selected iteratively by constructing $2N_c + N_m$ uniformly spaced sample points over [0,1], and selecting candidate features based on the corresponding segment/interval in which each sample falls. In this manner, those candidates with longer subintervals of [0,1] have a higher likelihood of being selected, or are selected more frequently.

The stochastic uniform selection process is illustrated via an example. Consider a case where each generation has 6 candidates where each successive generation requires $N_e = 1$, $N_c = 4$, and $N_m = 1$ candidates to be produced through elitism, crossover and mutation for the next generation. Thus, the number of candidates required to be selected from the current generation is 10 (i.e. $N_e + 2N_c + N_m = 10$). Of these, 9 are selected using the stochastic uniform selection method for crossover and mutation. Figure 4.3 depicts the operation of the stochastic uniform selection method using the expectation scores for this scenario.

In figure 4.3, the unit interval is depicted by the blue dotted line. On top of the unit interval, a segmented black solid line represents the segments associated with candidates, where the length of the segment is proportional to a candidate's expectation score. The fittest candidate, ranked 1, is the most likely candidate to be selected and thus possesses the longest segment on the line. Iterative selection occurs by selecting 9 equidistant points (starting with a random point P1 in $[0, \frac{1}{2N_c+N_m}]$), as indicated using the green arrows. The candidate corresponding to the segment in which each point falls is selected, in this case, the selected candidates are 1,1,2,2,3,3,4,5,6.

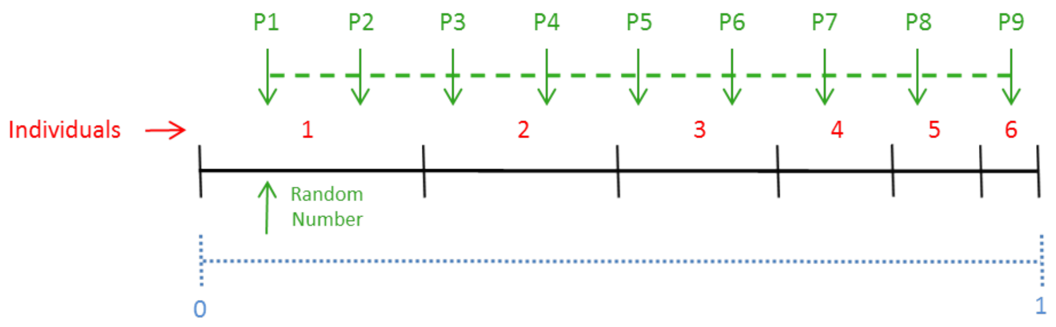


FIGURE 4.3: *Illustration of the stochastic uniform selection method.*

Elitism

Elitism is a mechanism that is incorporated into GAs for guaranteeing that the average fitness does not decrease from one generation to the next [131]. The elitism operation duplicates the N_e selected candidates $\mathbf{F_{sel,elite}}$ in the next generation $\mathbf{F_{elite}}$ (i.e. they appear unchanged in the next generation).

Crossover

Scattered crossover is subsequently performed on the first $2N_c$ selected candidates $\mathbf{F_{sel,cross}}$, which produces N_c new candidates $\mathbf{F_{cross}}$ for the next generation. The pseudocode for the scattered crossover operation is provided in algorithm 4.5.

The algorithm constructs chromosomes that represent the candidate feature subsets of the current generation. Each chromosome is a bit string, as described in section 4.4. The algorithm iteratively selects two consecutive candidates to cross, A and B from $\mathbf{F_{sel,cross}}$, and iterates through each bit of A and B. The new candidate is produced by randomly assigning the bit from either candidate A or candidate B, with equal probability, to a new chromosome. The crossover algorithm concludes by constructing the candidate feature subsets $\mathbf{F_{cross}}(d)$, $d = 1, 2, \ldots, N_c$ associated with the new chromosomes for the next generation.

Algorithm 4.5 Scattered crossover algorithm

```
Function $[\mathbf{F_{cross}}] = GA\_Crossover(\mathbf{F_{sel,cross}}, N_c, N_F)$
Inputs:
    $\mathbf{F_{sel,cross}}(d),\ d = 1, 2, \ldots, 2N_c$    ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for crossover
    $N_c$,    ▷ Number of candidates in next generation obtained
              ▷ via crossover in current generation
    $N_F$,    ▷ Dimensionality of full, initial feature set
Output:
    $\mathbf{F_{cross}}$    ▷ Candidates in next generation obtained via crossover
Variables:
    $\mathbf{B}(d)$    ▷ Chromosomes representing selected candidates from $\mathbf{F_{sel,cross}}$ for crossover
    $\mathbf{B}'(d)$    ▷ Chromosomes representing candidates obtained via crossover in $\mathbf{F_{cross}}$

▷ Create chromosomes $\mathbf{B}(d)$ for each candidate in $\mathbf{F_{sel,cross}}(d),\ d = 1, 2, \ldots, 2N_c$
▷ Binary string where a bit is set to one if the corresponding feature is present
▷ in the candidate feature subset
1:  $\mathbf{B}(d) \leftarrow [\mathbf{B}_1(d), \mathbf{B}_2(d), \ldots, \mathbf{B}_{N_F}(d)]$    ▷ where $\mathbf{B}_i(d) \leftarrow 0$, if feature $f_i \notin \mathbf{F_{sel,cross}}(d)$
                                                   ▷ and $\mathbf{B}_i(d) \leftarrow 1$, if feature $f_i \in \mathbf{F_{sel,cross}}(d)$
2:  $index \leftarrow 1$ ;
3:  for $d = 1$ to $N_c$ do
4:      for $f_{num} = 1$ to $N_F$ do
5:          $rand\_val \leftarrow$ Random number drawn uniformly from $[0, 1]$ ;
6:          if $(rand\_val < 0.5)$ then
7:              $\mathbf{B_{f_{num}}}'(d) \leftarrow \mathbf{B_{f_{num}}}(index)$ ;    ▷ Bit from candidate A
8:          else
9:              $\mathbf{B_{f_{num}}}'(d) \leftarrow \mathbf{B_{f_{num}}}(index + 1)$ ;    ▷ Bit from candidate B
10:         end if
11:     end for
12:     $index \leftarrow index + 2$ ;
13: end for
▷ Construct feature subsets $\mathbf{F_{cross}}(d)$ obtained via crossover from
▷ new chromosomes $\mathbf{B}'(d),\ d = 1, 2, \ldots, N_c$
14: $\mathbf{F_{cross}}(d) \leftarrow \{f_{num} \in \{1, 2, \ldots, N_F\} : \mathbf{B_{f_{num}}}'(d) = 1\}$ ;
15: Return $\mathbf{F_{cross}}$
```

Mutation

Mutation is subsequently applied to the $\mathbf{F_{sel,mut}}(d),\ d = 1, 2, \ldots, N_m$ candidate feature subsets, in order to obtain mutated candidates $\mathbf{F_{mut}}$ for the next generation. The pseudocode for the uniform mutation operation is provided in algorithm 4.6.

As was the case with the crossover operation, the algorithm constructs chromosomes for feature subsets in $\mathbf{F_{mut}}(d)$. The algorithm iterates through each bit of each chromosome, and randomly inverts certain bits. Each bit has a p_m percentage chance of being inverted. Candidate feature subsets $\mathbf{F_{mut}}$ are derived from the mutated chromosomes and returned.

Algorithm 4.6 Uniform mutation algorithm

```
   Function [$\mathbf{F_{mut}}$] = $GA\_Mutation(\mathbf{F_{sel,mut}}, N_m, N_F, p_m)$
   Inputs:
     $\mathbf{F_{sel,mut}}$,            ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for mutation
     $N_m$,                    ▷ Number of candidates in next generation obtained
                               ▷ via mutation in current generation
     $N_F$,                    ▷ Dimensionality of full, initial feature set
     $p_m$,                    ▷ Mutation probability for a feature
   Output:
     $\mathbf{F_{mut}}$                  ▷ Candidates in next generation obtained via mutation
   Variables:
     $\mathbf{B}(d)$   ▷ Chromosomes representing selected candidates from $\mathbf{F_{sel,mut}}$ for mutation
     $\mathbf{B}'(d)$        ▷ Chromosomes representing candidates obtained via mutation in $\mathbf{F_{mut}}$

   ▷ Create chromosomes $\mathbf{B}(d)$ for each candidate in $\mathbf{F_{sel,mut}}(d),\ d = 1, 2, \ldots, N_m$
   ▷ Binary string where a bit is set to one if the corresponding feature is present
   ▷ in the candidate feature subset
1: $\mathbf{B}(d) \leftarrow [\mathbf{B}_1(d), \mathbf{B}_2(d), \ldots, \mathbf{B}_{N_F}(d)]$   ▷ where $\mathbf{B}_i(d) \leftarrow 0$, if feature $f_i \notin \mathbf{F_{sel,mut}}(d)$
                                             ▷ and $\mathbf{B}_i(d) \leftarrow 1$, if feature $f_i \in \mathbf{F_{sel,mut}}(d)$

2: for $d = 1$ to $N_m$ do
3:     for $f_{num} = 1$ to $N_F$ do
```

```
4:         $rand\_val \leftarrow$ Random number drawn uniformly from $[0, 1]$ ;
5:         if $(rand\_val < p_m)$ then
6:             $\mathbf{B_{f_{num}}}'(d) \leftarrow 1 - \mathbf{B_{f_{num}}}(d)$ ;                ▷ Mutation
7:         else
8:             $\mathbf{B_{f_{num}}}'(d) \leftarrow \mathbf{B_{f_{num}}}(d)$ ;                    ▷ Non-mutation
9:         end if
10:     end for
11: end for

    ▷ Construct feature subsets $\mathbf{F_{mut}}(d)$ obtained via mutation from
    ▷ new chromosomes $\mathbf{B}'(d),\ d = 1, 2, \ldots, N_m$
12: $\mathbf{F_{mut}}(d) \leftarrow \{f_{num} \in \{1, 2, \ldots, N_F\}\ :\ \mathbf{B_{f_{num}}}'(d) = 1\}$ ;

13: Return $\mathbf{F_{mut}}$
```

4.4.2 Multi-objective genetic algorithm

This section describes the genetic algorithm that was applied in this research for optimization over multiple objective functions (i.e. where the fitness of each candidate solution is measured against multiple criteria). Specifically, the multi-objective GA attempts to find feature subsets that minimize the NDB index as a first objective, as well as minimize the number of features in the subset as a second objective, as discussed in section 4.3.

Multi-objective GAs are similar to single-objective GAs in the sense that selection, crossover and mutation operations are applied to chromosomes over multiple generations, in an attempt to find more "fit" solutions. The key difference between these algorithms lies in the ranking of candidates during the selection step. The single-objective GA ranks candidates according to their fitness scores obtained from the evaluation of the single objective function. In contrast, the multi-objective GA attempts to find solutions that are considered to provide a superior trade-off between the objective functions, as compared to competing solutions. These solutions are referred to as the Pareto-optimal front.

In this research, a variant[2] of the improved Non-dominated Sorting Genetic Algorithm, or NSGA-II, was used [135]. The NSGA-II applies the non-dominated sorting algorithm and the crowding distance algorithm prior to selecting candidate feature subsets for crossover and mutation. The non-dominated sorting algorithm ranks candidate feature subsets based on the trade-off between the objective functions, where higher ranked candidates provide a better trade-off than lower ranked candidates. The crowding distance algorithm maintains diversity amongst selected candidate feature subsets by measuring the spread amongst the candidates in the objective space. The pseudocode for the multi-objective genetic algorithm, as applied in this research, is provided in algorithm 4.7.

Algorithm 4.7 Multi-Objective Genetic Algorithm (MOGA)

```
Function [$\mathbf{F_s}^{(t+1)}$] = $MOGA(\mathbf{F_s}^{(t)}, \mathbf{NDB_{avg}}^{(t)}, t)$
Inputs:
    $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M,$              ▷ Current generation (i.e. generation t) of M
                                                ▷ candidate feature subsets
    $\mathbf{NDB_{avg}}^{(t)}(d),\ d = 1, 2, \ldots, M,$         ▷ Average NDB index value corresponding to
                                                ▷ candidate feature subsets in $\mathbf{F_s}^{(t)}(d)$
    $t,\ t = 0, 1, 2, \ldots, N_g$                       ▷ Current generation number
Output:
    $\mathbf{F_s}^{(t+1)}(d),\ d = 1, 2, \ldots, M,$            ▷ Next generation of M candidate feature subsets
Constants:
    $M,$                      ▷ Number of candidate feature subsets in each generation
    $N_F,$                                ▷ Dimensionality of full, initial feature set
    $r_c,$                        ▷ Percentage of candidates to produce from crossover,
                                    ▷ for possible appearance in generation $t + 1$
    $p_m,$                                     ▷ Mutation probability for a feature
    $N_g$                                      ▷ Maximum number of generations
```

[2]NSGA-II is an improvement over the ordinary NSGA algorithm in the sense that it incorporates elitism, whereas ordinary NSGA does not. It is also computationally less expensive owing to its use of a more efficient sorting algorithm [134].

```
1:  if $t = 0$ then                                      ▷ If this is the first generation
2:      for $d = 1$ to $M$ do          ▷ Produce initial generation of candidate feature subsets
    ▷ Uniformly draw candidate feature subsets from a collection
    ▷ of all candidate feature subsets
3:          $\mathbf{F_s}^{(1)}(d) \leftarrow random\_uniform\_sample(N_F)$ ;
4:      end for
5:      Return $\mathbf{F_s}^{(1)}$ ;
6:  end if

    ▷ Compute the second objective function,
    ▷ the feature cardinality for each candidate feature subset
7:  for $d = 1$ to $M$ do
8:      $\mathbf{F_{card}}^{(t)}(d) \leftarrow sum(\mathbf{F_s}^{(t)}(d))$ ;
9:  end for

    ▷ Assign each candidate to a pareto front
10: $\mathbf{Rank} \leftarrow Non\_Dominated\_Sorting(\mathbf{NDB_{avg}}^{(t)}, \mathbf{F_{card}}^{(t)})$      ▷ Algorithm 4.8

    ▷ Compute crowding distance of each candidate
11: $\mathbf{X_{distance}} \leftarrow Crowding\_Distance(\mathbf{NDB_{avg}}^{(t)}, \mathbf{F_{card}}^{(t)})$      ▷ Algorithm 4.9

12: if $t = N_g$ then                                    ▷ If this is the final generation
13:     $i \leftarrow 1$ ;
14:     for $d = 1$ to $M$ do
15:         if $(\mathbf{Rank}(d) = 1)$ then
16:             $\mathbf{front_1}(i) \leftarrow d$ ;   ▷ where $front_1$ contains the indices of those candidates
                                                   ▷ belonging to the Pareto-optimal front
17:             $i \leftarrow i + 1$ ;
18:         end if
19:     end for
20:     $\mathbf{F_s}^{(N_g+1)} \leftarrow \mathbf{F_s}^{(N_g)}(\mathbf{front_1})$ ;        ▷ Return all candidates belonging to the first
21:     Return $\mathbf{F_s}^{(N_g+1)}$                     ▷ pareto front, in the current generation
22: end if
```

```
23: $N_c \leftarrow round(M \times r_c)$ ;            ▷ Number of candidates in next generation obtained
                                                      ▷ via crossover in current generation
24: $N_m \leftarrow (M - N_c)$ ;                     ▷ Number of candidates in next generation obtained
                                                      ▷ via mutation in current generation

    ▷ Selection (Algorithm 4.10)
    ▷ Select candidates for elitism, crossover and mutation
25: $[\mathbf{F_{sel,elite}},\ \mathbf{F_{sel,cross}},\ \mathbf{F_{sel,mut}}] \leftarrow MOGA\_Selection(\mathbf{F_s}^{(t)}, N_c, N_m, N_F, Rank, \mathbf{X_{distance}})$

    ▷ Elitism
    ▷ All candidates in generation t have the possibility to appear unchanged
    ▷ in the next generation $t+1$
26: $[\mathbf{F_{elite}}] \leftarrow \mathbf{F_{sel,elite}}$ ;                       ▷ Equivalent to $\mathbf{F_s}^{(t)}$

    ▷ Scattered Crossover (Algorithm 4.5)
    ▷ Produce $N_c$ candidates for possible appearance in next generation through crossover
27: $[\mathbf{F_{cross}}] \leftarrow GA\_Crossover(\mathbf{F_{sel,cross}}, N_c, N_F)$ ;

    ▷ Uniform Mutation (Algorithm 4.6)
    ▷ Produce $N_m$ candidates for possible appearance in next generation through mutation
28: $[\mathbf{F_{mut}}] \leftarrow GA\_Mutation(\mathbf{F_{sel,mut}}, N_m, N_F, p_m)$ ;

    ▷ Collection of $2M$ pre-selected candidates for possible appearance in generation $t+1$,
    ▷ consisting of all candidates from generation t, and candidates obtained through
    ▷ the application of crossover and mutation to candidates in generation t
29: $\mathbf{F_{s,pre-sel}}(1, \ldots, M) \leftarrow \mathbf{F_{elite}}$ ;
30: $\mathbf{F_{s,pre-sel}}(M+1, \ldots, M+2N_c) \leftarrow \mathbf{F_{cross}}$ ;
31: $\mathbf{F_{s,pre-sel}}(M+2N_c+1, \ldots, 2M) \leftarrow \mathbf{F_{mut}}$ ;

    ▷ In order to select the top M candidates from $\mathbf{F_{s,pre-sel}}$ for the next generation $(t+1)$,
    ▷ the computation of the rank and crowding distances for the collection of $\mathbf{F_{s,pre-sel}}$
    ▷ candidates is required. This in turn requires the computation of the $NDB_{avg}$ and
    ▷ $F_{card}$ values for those crossed and mutated candidates
```

```
32: $\mathbf{NDB_{avg,pre-sel}}(1,\dots,M) \leftarrow \mathbf{NDB_{avg}}^t$ ;          ▷ First $M$ candidates belong
33: $\mathbf{F_{card,pre-sel}}(1,\dots,M) \leftarrow \mathbf{F_{card}}^t$ ;          ▷ to generation $t$

    ▷ Perform clustering on the candidates obtained from crossover and mutation
34: $[\mathbf{C},\ \mathbf{L}] \leftarrow repeated\_kmeans(\mathbf{X}, K, \mathbf{F_{s,pre-sel}}(M+1,\dots,2M), C_R)$ ;

    ▷ Compute the average NDB index values for those candidates obtained
    ▷ from crossover and mutation
35: $\mathbf{NDB_{avg,pre-sel}}(M+1,\dots,2M) \leftarrow Avg\_CVI(\mathbf{X}, K, \mathbf{C}, \mathbf{L}, C_R)$ ;

    ▷ Compute the second objective function, the feature cardinality for
    ▷ those candidates obtained from crossover and mutation
36: for $d = (M+1)$ to $2M$ do
37:     $\mathbf{F_{card,pre-sel}}(d) \leftarrow sum(\mathbf{F_{s,pre-sel}}(d))$ ;
38: end for

    ▷ Assign each candidate in $\mathbf{F_{s,pre-sel}}$ to a Pareto front
39: $\mathbf{Rank} \leftarrow Non\_Dominated\_Sorting(\mathbf{NDB_{avg,pre-sel}},\ \mathbf{F_{card,pre-sel}})$ ; ▷ Algorithm 4.8

    ▷ Compute crowding distance of each candidate in $\mathbf{F_{s,pre-sel}}$
40: $\mathbf{X_{distance}} \leftarrow Crowding\_Distance(\mathbf{NDB_{avg,pre-sel}},\ \mathbf{F_{card,pre-sel}})$ ;   ▷ Algorithm 4.9

41: for $j = 1$ to $max(\mathbf{Rank})$ do                    ▷ The number of pareto fronts
42:     $i \leftarrow 1$ ;
43:     for $d = 1$ to $2M$ do
44:         if $(\mathbf{Rank}(d) = j)$ then
45:             $\mathbf{front_j}(i) \leftarrow d$ ;   ▷ where $front_j$ contains the indices of those candidates
                                 ▷ in $\mathbf{F_{s,pre-sel}}$ belonging to the $j^{th}$ Pareto front
46:             $i \leftarrow i+1$ ;
47:         end if
48:     end for
49: end for
```

```
50: $j \leftarrow 1$ ;
51: repeat
        ▷ Add those candidates belonging to $front_j$ to the next generation
52:     $\mathbf{F_s}^{(t+1)}(|\mathbf{F_s}^{(t+1)}|+1,\ldots,|\mathbf{F_s}^{(t+1)}|+|\mathbf{front_j}|) \leftarrow \mathbf{F_{s,pre-sel}}(\mathbf{front_j})$ ;
53:     $j \leftarrow j+1$ ;
54: until $(|\mathbf{F_s}^{(t+1)}|+|front_j| \geq M)$        ▷ Until the next generation cannot accomodate
                                                            ▷ all candidates belonging to the next front

    ▷ Sort the candidates of the remaining front in descending order of crowding distance,
    ▷ such that $\mathbf{X_{distance}}(\mathbf{Rank}(d)) \leftarrow \mathbf{sorted\_X_{distance}}(d)$, for $d = 1,2,\ldots,M$
55: $[\mathbf{sorted\_X_{distance}}, \mathbf{Rank}] \leftarrow sort(\mathbf{X_{distance}}(\mathbf{front_j}), descending)$ ;
56: $\mathbf{F_s}^{(t+1)}(|\mathbf{F_s}^{(t+1)}|+1,\ldots,M) \leftarrow \mathbf{F_{s,pre-sel}}(\mathbf{front_j}(\mathbf{Rank}(1:(M-|\mathbf{F_s}^{(t+1)}|))))$ ;

57: Return $\mathbf{F_s}^{(t+1)}$ ;
```

The multi-objective genetic algorithm first determines whether the first generation $\mathbf{F_s}^{(1)}$ is to be constructed ($t = 0$), or whether the Pareto-optimal front is to be selected from the final generation $\mathbf{F_s}^{(N_g)}$ (corresponding to $t = N_g$). In the former case, the GA constructs an initial population $\mathbf{F_s}^{(1)}(d),\ d = 1,2,\ldots,M$, of M candidate feature subsets, where each candidate is randomly and uniformly selected from all possible candidate feature subsets. The algorithm subsequently returns this set of candidates. In the latter case, the GA selects and returns the candidate feature subsets from the final generation $\mathbf{F_s}(d)$ that belong to the Pareto-optimal front. If neither the first case ($t = 0$) nor the last case ($t = N_g$) applies, the GA performs the steps of selection, elitism, crossover and mutation to produce the next generation $\mathbf{F_s}^{(t+1)}(d),\ d = 1,2,\ldots,M$.

Candidate feature subsets of the current generation are ranked using the non-dominated sorting algorithm according to the normalized DB index and the number of features as criteria (refer to algorithm 4.8). The algorithm subsequently calculates the crowding distance of each candidate feature subset (refer to algorithm 4.9). The ranking of the candidate feature subsets and their corresponding crowding distances are used in the selection of candidates for crossover and mutation. The selection process is provided in algorithm 4.10, whereas the crossover and mutation operations are identical to those carried out for the single-objective GA (refer to algorithms 4.5 and 4.6).

Following the crossover and mutation operations, the algorithm returns the top-ranked M candidates in the union between the current generation and the candidates obtained via crossover and mutation.

Non-dominated sorting algorithm

The non-dominated sorting algorithm ranks candidates based on how well each candidate provides a trade-off between the multiple objective functions. The superiority of the trade-off provided by a candidate, as compared to other candidates, is established using the concept of domination. To define this concept, consider a generation of candidate feature subsets $\mathbf{F_s}(d),\ d = 1, 2, \ldots, M$, and the corresponding objective function values $\mathbf{W}(d) = (\mathbf{w_1}(d), \mathbf{w_2}(d) \ldots, \mathbf{w_{N_o}}(d))$ that are to be minimized as a function of the feature subset, where N_o represents the number of objective functions where $N_o \geq 2$. A candidate feature subset $\mathbf{F_s}(d_1)$ dominates another candidate feature subset $\mathbf{F_s}(d_2)$ if both of the following conditions are satisfied [135–137]:

1. $w_i(d_1) \leq w_i(d_2)$ for all objective function scores, $i = 1, 2, \ldots, N_o$
2. $w_i(d_1) < w_i(d_2)$ for at least one objective function score, $i = 1, 2, \ldots, N_o$

The pseudocode for performing non-dominated sorting of candidate feature subsets, which ranks the candidates in the current generation according to the concept of domination, is provided in algorithm 4.8 [134].

The ranking of candidates is carried out as follows. Starting with the full generation of candidate feature subsets, those candidates that are not dominated by any other candidates are assigned a rank of one, and are collectively referred to as the Pareto-optimal front. These candidates are removed from the current generation for the remainder of the non-dominated sorting algorithm. The remaining candidates that are not dominated by any other candidates in the current generation are assigned a rank of two, and constitute

Algorithm 4.8 Non-dominated sorting

```
Function [front] = $Non\_Dominated\_Sorting(\mathbf{NDB_{avg}}, \mathbf{F_{card}})$
Inputs:
  $\mathbf{NDB_{avg}}$,          ▷ Average normalized DB index value for each candidate
                                 ▷ feature subset over all $C_R$ clustering results
  $\mathbf{F_{card}}$            ▷ Number of features present in each candidate feature subset
Output:
  front                          ▷ Consists of the indices of those candidates
                                 ▷ belonging to each pareto front
Variables:
  $\mathbf{N_d}(i)$ ;            ▷ Counts the number of candidates that dominate candidate i
  $p\_front$ ;                   ▷ Contains the set of indices of those candidates assigned to front
                                 ▷ currently being constructed

1: $N_{front} \leftarrow 1$ ;    ▷ Front counter, determines which Pareto front is under construction
2: while $|\mathbf{NDB_{avg}}| \neq 0$ do
3:     $n_f \leftarrow 0$ ;      ▷ Stores the number of candidates currently assigned
                                 ▷ to pareto front $N_{front}$
4:     for $i = 1$ to $|\mathbf{NDB_{avg}}|$ do
5:         $\mathbf{N_d}(i) \leftarrow 0$ ;

6:         for $j = 1$ to $|\mathbf{NDB_{avg}}|$ do
7:             if $(j = i)$ then
8:                 continue ;
9:             end if

   ▷ Test if candidate j dominates candidate i
10:            if $(\mathbf{NDB_{avg}}(j) \leq \mathbf{NDB_{avg}}(i)) \wedge (\mathbf{F_{card}}(j) \leq \mathbf{F_{card}}(i))$ then
11:                $cond1 \leftarrow true$ ;   ▷ where $cond_1$ represents the first condition that is
12:            else                            ▷ required to be satisfied for domination
13:                $cond1 \leftarrow false$ ;
14:            end if
```

```
15:         if $(\mathbf{NDB_{avg}}(j) < \mathbf{NDB_{avg}}(i)) \vee (\mathbf{F_{card}}(j) < \mathbf{F_{card}}(i))$ then
16:             $cond2 \leftarrow true$ ;    ▷ where $cond_2$ represents the second condition that is
17:         else                             ▷ required to be satisfied for domination
18:             $cond2 \leftarrow false$ ;
19:         end if

  ▷ If both conditions are true, candidate $j$ dominates candidate $i$
20:         if $(cond1 = true) \wedge (cond2 = true)$ then
21:             $N_d(i) \leftarrow N_d(i) + 1$ ;          ▷ Count the number of candidates that
                                                          ▷ dominate candidate $i$
22:         end if
23:       end for
24:       if ( $N_d(i) = 0$ ) then        ▷ Candidate $i$ belongs to the Pareto front $N_{front}$
25:           $n_f = n_f + 1$ ;
26:           $p\_front(n_f) \leftarrow i$ ;            ▷ Add index of candidate $i$ to the Pareto
                                                          ▷ front being constructed
27:       end if
28:     end for
29:     $\mathbf{front}(N_{front}) \leftarrow p\_front$ ;     ▷ Store indices of those candidates
                                                          ▷ belonging to front $N_{front}$
30:     $N_{front} \leftarrow N_{front} + 1$ ;
31:     $\mathbf{NDB_{avg}}(p\_front) = Inf$ ;          ▷ Exclude those candidates already
32:     $\mathbf{F_{card}}(p\_front) = Inf$ ;           ▷ assigned to a pareto front
33: end while

34: Return front
```

the second Pareto front. These candidates are subsequently removed from the current generation for the remainder of the non-dominated sorting algorithm. This process is repeated until no candidates remain in the current generation, thereby producing a succession of Pareto fronts. Candidates are ranked according to the Pareto front to which they belong, with candidates in a specific Pareto front dominated by all candidates in preceding fronts. That is, candidates in Pareto front i are considered as providing a superior trade-off between the objective functions as compared to candidates in successive Pareto fronts i+1, i+2, etc.

Crowding distance algorithm

The Pareto-optimal front represents those candidate solutions that provide a superior trade-off between objective functions. These candidates may be represented by points in the N_o-dimensional space corresponding to the range of possible values of the N_o objective functions (referred to as the 'objective space'). As the solution most relevant to the researcher may be biased towards particular objective functions, it is desirable that the Pareto optimal front contain a diverse set of candidates that are located far apart in the objective space; this increases the likelihood of points appearing in the relevant region of the objective space. Should this be achieved, the researcher is provided the opportunity to select a candidate solution from the Pareto optimal front that is relevant to the problem context.

In order to maintain diversity within a set of Pareto-optimal solutions, the NSGA-II algorithm calculates what is known as a crowding distance for each candidate. The crowding distance associated with a candidate of interest is inversely proportional to the density of candidates surrounding the candidate of interest in the objective space, and serves as a measure of the diversity of the candidate solution with respect to the remaining candidates. During selection, those candidates that have a higher crowding distance are favoured.

The crowding distance is an estimate of the side length of the largest rectangle that surrounds the solution in the objective space, but which does not contain any other solutions (refer to figure 4.4). The crowding distance of candidate i is computed by considering distances over each singular dimension of the objective space. Specifically, the nearest neighbour on either side of the candidate in each dimension of the solution space is found, and the distance between these neighbours is calculated in that dimension; this distance is normalized by the distance between the two furthest points in the dimension of interest. These distances are added to obtain the crowding distance. Those candidates that are at the edges of the objective space (i.e. which have no neighbours on one of its sides in any of the dimensions of the objective space) are assigned a crowding distance of infinity, which implies that these candidates are given preference during selection.

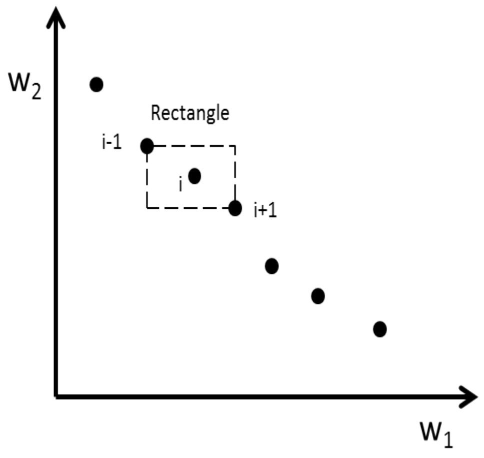


FIGURE 4.4: *Crowding distance computation (from [3]).*

The pseudocode for the computation of the crowding distance for each candidate in a Pareto-optimal front [134] is provided in algorithm 4.9.

Algorithm 4.9 Crowding distance assignment

```
Function $[\mathbf{X_{distance}}] = Crowding\_Distance(\mathbf{NDB_{avg}}, \mathbf{F_{card}})$
Inputs:
    $\mathbf{NDB_{avg}}$,        ▷ Average normalized DB index value for each candidate
                                 ▷ feature subset over all $C_R$ clustering results
    $\mathbf{F_{card}}$          ▷ Number of features present in each candidate feature subset
Output:
    $\mathbf{X_{distance}}$      ▷ The crowding distance for each candidate

1: $\mathbf{W} \leftarrow [\mathbf{NDB_{avg}},\ \mathbf{F_{card}}]$ ;     ▷ Set of objective function scores
2: for $d = 1$ to $|\mathbf{NDB_{avg}}|$ do
3:     $X_{distance}(d) \leftarrow 0$ ;
4: end for
```

```
5: for $j = 1$ to $|\mathbf{W}|$ do                                   ▷ For each objective function $w_j$
6:     $X_{distance}(1) \leftarrow X_{distance}(N) \leftarrow \infty$ ;        ▷ Boundary points always selected

   ▷ Vector “indices” contains the indices of each candidate sorted based on the
   ▷ objective function w(j) values under consideration, such that
   ▷ $w(j)(indices(j)) \leftarrow sorted\_scores(j)$, where $j = 1, 2, \ldots, |\mathbf{NDB_{avg}}|$
7:     $[sorted\_scores, indices] \leftarrow sort(w(j), ascending)$ ;

8:     for ( $d = 2$ to $(|\mathbf{NDB_{avg}}| - 1)$ ) do
9:         $X_{distance}(indices(d)) \leftarrow X_{distance}(indices(d)) + \frac{(w(j, indices(d)+1) - w(j, indices(d)-1))}{(\max(w(j)) - \min(w(j)))}$
10:    end for
11: end for
12: Return $\mathbf{X_{distance}}$
```

The algorithm uses the normalized DB index of the candidate feature subset, as well as the number of features in the subset, as objectives. It subsequently iterates over the two dimensions of the objective space. The objective scores corresponding to the dimension are sorted; the sorted indices are used to find the two neighbours on either side of each candidate. The normalized distance between the neighbours are calculated, and added to the total over all dimensions to obtain the crowding distance. A value of infinity is assigned to the candidates in each dimension with the smallest and largest objective scores.

Selection

The multi-objective GA selects M, N_c and N_m candidate feature subsets for elitism, crossover and mutation ($\mathbf{F_{sel,elite}}$, $\mathbf{F_{sel,cross}}$, $\mathbf{F_{sel,mut}}$), respectively. The selection of candidates for crossover and mutation, is carried out using the tournament selection algorithm, thereby producing feature subsets for each operation. It is worth noting that the multi-objective algorithm ranks the collection consisting of the elite candidates and the candidates obtained via crossover and mutation using the non-dominated sorting algorithm and constructs the next generation as the top M candidates, as discussed in section 4.4.2. Hence, those candidates selected for elitism, and the candidates obtained via crossover and mutation, may not necessarily form part of the next generation; only their availability for the next generation is ensured. This implies that the definition of elitism for the multi-objective GA differs from that of the single-objective GA, in that selection for elitism does not guarantee appearance in the next generation.

The pseudocode for the tournament selection algorithm is provided in algorithm 4.10. Unlike the single-objective genetic algorithm, the multi-objective genetic algorithm selects the entire current generation for elitism. The selection of the candidates for crossover and mutation involves the iterative random selection of pairs of candidate feature subsets from the current generation, which are compared in terms of their rank. The higher ranked candidate is included in the collection of features selected for crossover and mutation; if both candidates are of equal rank, the candidate with the larger crowding distance is selected. Using this approach, those candidates that have a higher rank and larger crowding distance are favoured, and are more likely to be selected / selected more frequently than other candidates.

Elitism, crossover and mutation

The multi-objective genetic algorithm ensures that the entire generation is made available as candidates for the next generation through elitism. The same algorithms for crossover (algorithm 4.5) and mutation (algorithm 4.6), as used in the single-objective genetic algorithm (algorithm 4.3), were used in the multi-objective genetic algorithm. Following the application of elitism, crossover and mutation, a collection of candidate feature subsets of size $2M$ is obtained.

The collection of $2M$ candidate feature subsets is ranked using the non-dominated sorting algorithm, and the crowding distance of each candidate is computed. The next generation $\mathbf{F_s}^{t+1}(d)$ is created by successively including the candidate feature subsets from each Pareto front, starting with the first Pareto front, or the optimal front, until there are M candidate solutions in the next generation. If the final front to be included has more than the required number of candidates, those candidates with the largest crowding distances are selected from the final considered Pareto front. The resulting collection of M candidate feature subsets are returned by the algorithm as the next generation (provided that the initial or final generation is not to be produced - refer to section 4.4.2 for these cases).

Algorithm 4.10 Multi-objective GA selection algorithm

```
Function [$\mathbf{F_{sel,elite}}$, $\mathbf{F_{sel,cross}}$, $\mathbf{F_{sel,mut}}$]
        = $MOGA\_Selection(\mathbf{F_s}^{(t)}, N_c, N_m, N_F, Rank, \mathbf{X_{distance}})$
Inputs:
    $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M,$        ▷ Current generation (i.e. generation t) of M
                                                   ▷ candidate feature subsets
    $N_c,$                                          ▷ Number of candidates in next generation obtained
                                                   ▷ via crossover in current generation
    $N_m,$                                          ▷ Number of candidates in next generation obtained
                                                   ▷ via mutation in current generation
    $Rank,$      ▷ Ranking of each candidate feature subset in $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M,$
                                                   ▷ based on the pareto front that each candidate belongs to
    $\mathbf{X_{distance}}$                         ▷ Crowding distance of each candidate feature subset in
                                                   ▷ $\mathbf{F_s}^{(t)}(d),\ d = 1, 2, \ldots, M$
Outputs:
    $\mathbf{F_{sel,elite}}(d),\ d = 1, 2, \ldots, M$       ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for elitism
    $\mathbf{F_{sel,cross}}(d),\ d = 1, 2, \ldots, 2N_c$    ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for crossover
    $\mathbf{F_{sel,mut}}(d),\ d = 1, 2, \ldots, N_m$       ▷ Feature subsets from $\mathbf{F_s}^{(t)}$ selected for mutation

1: $\mathbf{F_{sel,elite}} \leftarrow \mathbf{F_s}^{(t)}$ ;          ▷ Select all candidates from $\mathbf{F_s}^{(t)}$ for elitism

2: for $d = 1$ to $(2N_c + N_m)$ do
3:     $A \leftarrow rand(1 : M)$ ;   ▷ Random number between 1 and total number of candidates
4:     repeat
5:         $B \leftarrow rand(1 : M)$ ;          ▷ Random number between 1 and total number of
6:     until $(B \neq A)$                                                  ▷ candidates

    ▷ If $\mathbf{F_s}^{(t)}(A)$ and $\mathbf{F_s}^{(t)}(B)$ belong to the same front (i.e. have the same rank),
    ▷ select either $\mathbf{F_s}^{(t)}(A)$ or $\mathbf{F_s}^{(t)}(B)$ based on crowding distance
```

```
 7:     if $Rank(A) = Rank(B)$ then
 8:         if $X_{distance}(A) > X_{distance}(B)$ then
 9:             $\mathbf{F_{sel}}(d) \leftarrow \mathbf{F_s}^{(t)}(A)$ ;
10:             continue ;
11:         else
12:             $\mathbf{F_{sel}}(d) \leftarrow \mathbf{F_s}^{(t)}(B)$ ;
13:             continue ;
14:         end if
15:     end if

    ▷ Select the candidate with the better (lower integer value) rank
16:     if $Rank(A) < Rank(B)$ then
17:         $\mathbf{F_{sel}}(d) \leftarrow \mathbf{F_s}^{(t)}(A)$ ;
18:         continue ;
19:     else
20:         $\mathbf{F_{sel}}(d) \leftarrow \mathbf{F_s}^{(t)}(B)$ ;
21:         continue ;
22:     end if
23: end for
24: $\mathbf{F_{sel}} \leftarrow$ Random permutation($\mathbf{F_{sel}}$) ;

    ▷ Select $2N_c$ candidates for crossover from current generation
25: $\mathbf{F_{sel,cross}} \leftarrow \mathbf{F_{sel}}(1, \ldots, 2N_c)$ ;
    ▷ Select $N_m$ candidates for mutation from current generation
26: $\mathbf{F_{sel,mut}} \leftarrow \mathbf{F_{sel}}(2N_c + 1, \ldots, 2N_c + N_m)$ ;

27: Return [$\mathbf{F_{sel,elite}}$, $\mathbf{F_{sel,cross}}$, $\mathbf{F_{sel,mut}}$]
```

4.5 Flowchart of the proposed feature selection algorithm

A flowchart of the proposed feature selection algorithm is provided in figure 4.5. The inputs of the feature selection algorithm are the dataset, the number of clusters K, the maximum number of generations of the GA (N_g), the population size (M), and the number of repetitions to perform the k–means clustering algorithm over each candidate feature subset (C_R). The dataset is transformed using the normalization and encoding techniques described in sections 3.3.1 and 3.3.2, respectively. Following data transformation, the feature selection algorithm iteratively applies the k–means clustering algorithm to the dataset over generations of candidate feature subsets, computes the relative cluster validity indices corresponding to the clustering results, and selects the subsequent generation via the genetic algorithm.

During the first iteration, the initial generation of candidate feature subsets is drawn uniformly with replacement from the set of all possible candidate feature subsets (the pseudocode for this selection appears in algorithm 4.7, the multi-objective GA). In each iteration, k–means clustering is applied to the transformed dataset over each candidate feature subset a number of C_R executions, each with random starting positions for the cluster centres (refer to algorithm 4.1). This is followed by the calculation of the average normalized DB index (equation 4.7) for both the single and multi-objective GAs (refer to algorithm 4.2). The corresponding genetic algorithm (i.e. single or multi-objective) (refer to algorithms 4.3 or 4.7) is subsequently executed. If the final generation is reached, the highest ranked candidate feature subset from the final generation is returned as the optimized feature subset of the feature selection algorithm in the case of the single-objective optimization. In the case of the multi-objective optimization, those candidates with the highest rank (those belonging to the Pareto-optimal front) are returned as the optimized collection of feature subsets.

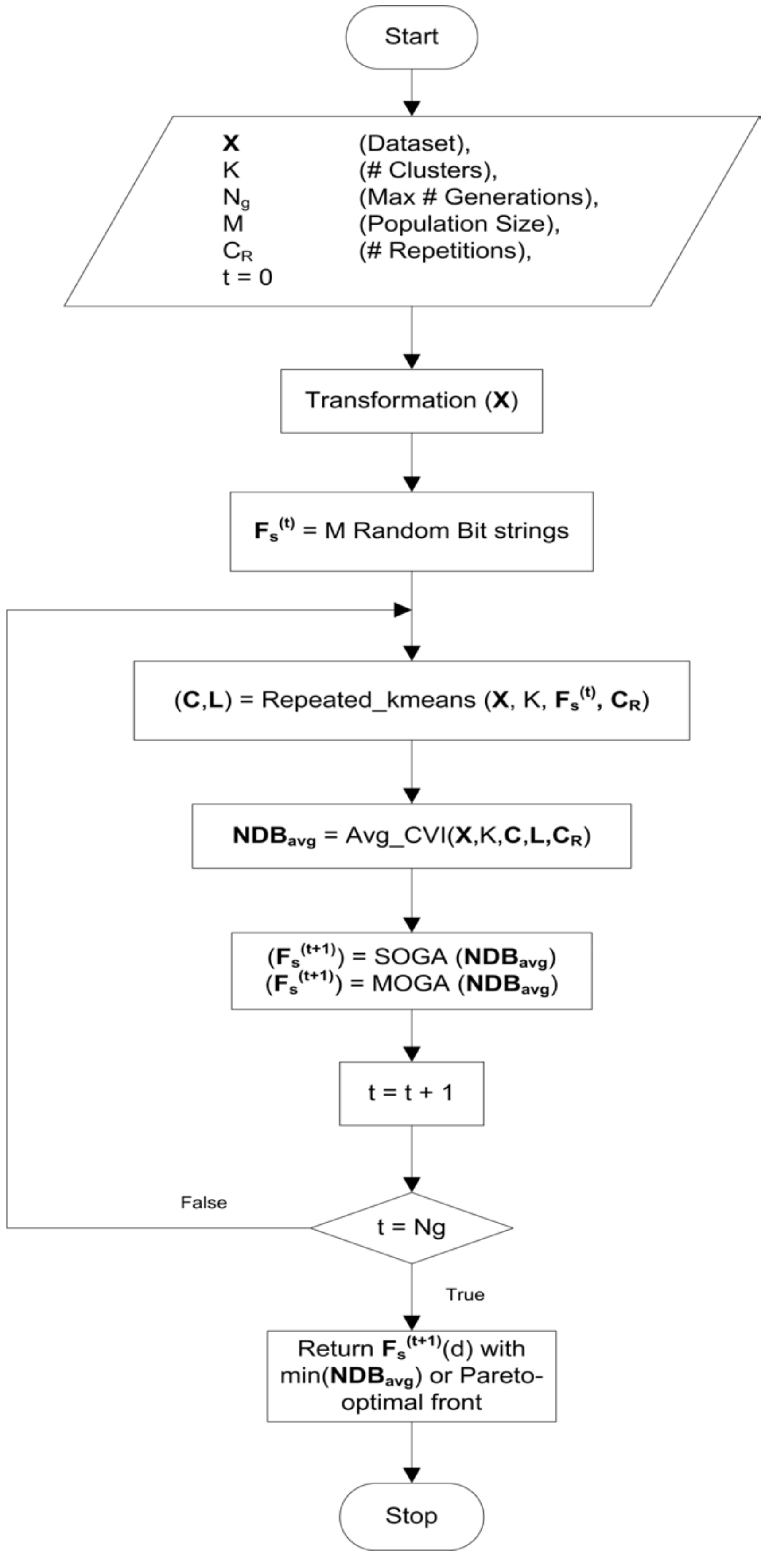

Start
$\mathbf{X}$ (Dataset),
K (# Clusters),
N_g (Max # Generations),
M (Population Size),
C_R (# Repetitions),
t = 0
Transformation ($\mathbf{X}$)
$\mathbf{F_s}^{(t)}$ = M Random Bit strings
($\mathbf{C}$,$\mathbf{L}$) = Repeated_kmeans ($\mathbf{X}$, K, $\mathbf{F_s}^{(t)}$, $\mathbf{C_R}$)
$\mathbf{NDB_{avg}}$ = Avg_CVI($\mathbf{X}$,K,$\mathbf{C}$,$\mathbf{L}$,$\mathbf{C_R}$)
($\mathbf{F_s}^{(t+1)}$) = SOGA ($\mathbf{NDB_{avg}}$)
($\mathbf{F_s}^{(t+1)}$) = MOGA ($\mathbf{NDB_{avg}}$)
t = t + 1
False
t = Ng
True
Return $\mathbf{F_s}^{(t+1)}$(d) with min($\mathbf{NDB_{avg}}$) or Pareto-optimal front
Stop

FIGURE 4.5: *Flow diagram of the proposed feature selection algorithm.*

4.6 Experimental results

In this section the results obtained from the application of the proposed unsupervised feature selection algorithm are presented, compared and discussed. The feature subsets obtained using the proposed feature selection algorithm, as well as the classification performance obtained by applying the proposed classifier in chapter 3 to the NSL-KDD dataset [25] over the optimized feature subsets produced by the GA, are presented. The classification results are compared to feature subsets produced by wrapper and filter techniques from the literature. The false positive and true positive percentages, as well as the Matthew's Correlation Coefficient (MCC), were used as measures of classification performance.

4.6.1 Experimental setup

This section provides,

1. the setup of the proposed feature selection algorithm, as used to perform feature selection on the NSL-KDD dataset [25],
2. the benchmark feature subsets derived using wrapper and filter-based techniques from the literature, and
3. the performance metrics used to evaluate the classification performance of feature subsets.

4.6.1.1 Feature selection algorithm parameters

The parameters for each block of the proposed feature selection algorithm's functional block diagram (figure 4.1) is provided.

TABLE 4.1: *Parameters of the proposed feature selection algorithm.*

Dataset (Feature vectors)		
Dataset selected	NSL-KDD training set 2	Section 3.2.2. Table 3.3 provides the dataset distribution.
Data transformation		
Normalization of numeric features	Statistical Normalization	Section 3.3.1
Encoding of non-binary categorical features	Frequency Encoding	Section 3.3.2
Encoding of binary categorical features	Ordinal Encoding	Section 3.3.2
Clustering		
Clustering algorithm	K–means	Section 3.4.1.1
Initialization	Initial cluster centres - K randomly selected data samples	—
Distance metric	Euclidean distance	—
Number of clusters (K)	5 - 7	—
Number of repetitions with different cluster centres (C_R)	100	—
Computation of relative CVIs		
Relative CVI selected	Normalized Davies-Bouldin (NDB) index	Section 4.3, equation 4.6
Single-Objective Genetic Algorithm (SOGA)		
Objective function	Average normalized DB index	Section 4.3, algorithm 4.2
Chromosome	40-bit binary string	—
Population size (M)	50	—
Max number of generations (N_g)	150	—
Ranking method	Fitness scaling - sorted NDB index values	—
Selection method	Stochastic uniform	Section 4.4.1, algorithm 4.4
Crossover method	Scattered	Section 4.4.1, algorithm 4.5
Mutation method	Uniform	Section 4.4.1, algorithm 4.6
Percentage of elite candidates (r_e)	5% of population size ($N_e = M \times r_e$)	—
Percentage of candidates created via crossover (r_c)	80% of ($M - N_e$)	—
Mutation probability (p_m)	1%	—
Stopping criteria	Max number of generations reached	—
Multi-Objective Genetic Algorithm (MOGA)		
Objective functions	Average normalized DB index, feature cardinality	Section 4.3, algorithm 4.2
Chromosome	40-bit binary string	—
Population size (M)	50	—
Max number of generations (N_g)	150	—
Ranking method	Non-dominated sorting	Section 4.4.2, algorithm 4.8
Selection method	Tournament	Section 4.4.2, algorithm 4.10
Crossover method	Scattered	Section 4.4.1 algorithm 4.5
Mutation method	Uniform	Section 4.4.1, algorithm 4.6
Percentage of candidates created via crossover (r_c)	80% of (M)	—
Mutation probability (p_m)	1%	—
Stopping criteria	Max number of generations reached	—

4.6.1.2 Feature subsets

The feature subsets obtained using the proposed feature selection algorithm were compared against two wrapper-based feature selection algorithms, obtained from [31]. The feature subsets obtained from these wrapper-based techniques were optimized for use with the k–means clustering algorithm, as part of the experimental work. Refer to section 2.4.2 and Appendix A for details on the implementation of these wrapper-based feature selection algorithms. Additionally, the feature subsets obtained using the proposed feature selection algorithm were compared against two filter-based feature selection algorithms found in the literature [35], in which the feature subsets produced by these methods, in [35], were directly used.

Table 4.2 provides the feature subsets that were compared to the feature subsets obtained from the proposed feature selection algorithm.

Note that the wrapper-based feature selection algorithms produced feature subsets by applying the training set to each of the feature selection algorithms. The feature subsets were tested by applying the training, as well as two test sets to the proposed classifier with $K = 2, \ldots, 10$, producing composite ROC curves for each of the training and test sets.

TABLE 4.2: *Feature set descriptions of the full feature set and of those feature subsets produced by several feature selection algorithms.*

Name	# Features	Feature List	Method	Reference
FS1	40	All excl. feature '20'	Full feature set, excluding feature '20'	—
Wrapper-1 (FRM)	15	1,2,3,4,12,23,25,26,27,28,31,34,35,36,39	Feature Removal Method (FRM) optimized for the k–means clustering algorithm	—
Wrapper-2 (SFM)	8	2,3,18,23,25,26,29,35	Sole Feature Method (SFM) optimized for the k–means clustering algorithm	—
Filter-1 (FS7)	10	3,4,5,6,14,16,27,28,37,39	Best of articles [7, 26–29]	[35]
Filter-2 (FS8)	10	2,3,4,5,6,8,23,30,34,36	Degree of Correlation + Greedy Stepwise	[35]

4.6.1.3 Performance metrics

As in the previous chapter, the TP and FP percentages were used as performance measures, and the performance is illustrated using the composite ROC curve as defined in section 3.6.1.3.a, and illustrated in figure 3.4.

4.6.2 Results and analysis

This section presents the feature subsets that were obtained using the proposed feature selection algorithm. The classification performance of the feature subsets produced by the proposed feature selection algorithm over different datasets, is provided and discussed. Additionally, a comparison of the classification performance of the full feature set, the filter-based feature subsets, the best performing wrapper-based feature subsets and the best performing feature subsets produced by the genetic algorithm, is provided and discussed.

4.6.2.1 Feature subsets produced by the genetic algorithm

This section provides the feature subsets that were produced by the proposed feature selection algorithm for both the single-objective and multi-objective optimization. Of these feature subsets, the best performing ones, in terms of average MCC values, are compared against the feature subsets presented in table 4.2.

4.6.2.1.a Single-objective genetic algorithm

The proposed feature selection algorithm using single-objective optimization and the training set produced three feature subsets for each value of $K = 5, \ldots, 7$. These feature subsets are provided in table 4.3. The proposed classifier was applied to the training set, as well as two test sets, over each feature subset with $K = 2, \ldots, 10$, producing composite ROC curves corresponding to each feature subset. These composite ROC curves are provided in figure 4.6.

TABLE 4.3: *Feature subsets produced by the proposed feature selection algorithm using single-objective optimization.*

Name	# Features	Feature List	Method	Reference
SOGA-1	28	2,3,4,5,7,9,11,12,13,14,15,16,17,19,21, 22,23,24,25,26,27,28,29,31,38,39,40,41	Proposed feature selection algorithm using the single-objective GA with $K = 5$	Novel
SOGA-2	20	2,3,4,7,10,11,14,15,21,22,23, 24,25,26,27,28,38,39,40,41	Proposed feature selection algorithm using the single-objective GA with $K = 6$	Novel
SOGA-3	23	2,3,4,7,8,13,14,15,16,17,21,22,23,24, 25,26,27,28,29,38,39,40,41	Proposed feature selection algorithm using the single-objective GA with $K = 7$	Novel

4.6.2.1.b Multi-objective genetic algorithm

The proposed feature selection algorithm using multi-objective optimization and the training set produced multiple feature subsets for each value of $K = 5, \ldots, 7$ (i.e. three Pareto-optimal fronts).

The proposed classifier was applied to the training set, as well as two test sets, over all feature subsets with $K = 2, \ldots, 10$, producing composite ROC curves corresponding to each feature subset in each of the three Pareto-optimal fronts. The average MCC value for each composite ROC curve was computed, and the feature subset with the largest average MCC value in each Pareto-optimal front was selected. This resulted in the selection of three feature subsets, where each represents the best performing feature subset amongst those produced using a different value of $K = 5, \ldots, 7$. The composite ROC curves of the best performing feature subsets are provided in figure 4.7.

TABLE 4.4: *Best performing feature subsets from each Pareto-optimal front, produced by the proposed feature selection algorithm using multi-objective optimization.*

Name	# Features	Feature List	Method	Reference
MOGA-1	15	4,7,8,9,14,15,18,21,22,25,26,27,28,39,41	Proposed feature selection algorithm using the multi-objective GA with $K = 5$	Novel
MOGA-2	15	4,9,10,14,15,18,21,22,25,26,27,28,38,39,41	Proposed feature selection algorithm using the multi-objective GA with $K = 6$	Novel
MOGA-3	19	2,3,4,5,7,8,9,14,15,18,21,22,23,24,27, 28,38,39,41	Proposed feature selection algorithm using the multi-objective GA with $K = 7$	Novel

4.6.2.2 Performance of feature subsets produced by the GA

This section presents the classification performance of the three feature subsets produced by the single-objective GA, and the three selected feature subsets produced by the multi-objective GA. The classification performance was measured by applying the training set, and two test sets to the proposed classifier over each of the six feature subsets (in table 4.3 and 4.4), and subsequently computing the composite ROC curves. The test sets used are described in section 3.2.2 and the distribution of the test sets can be found in table 3.3.

Figure 4.6 consists of the composite ROC curves for each dataset over each of the three feature subsets produced by the single-objective GA. Figure 4.7 consists of the composite ROC curves for each dataset over each of the three selected feature subsets produced by the multi-objective GA.

Figure 4.6 reveals that classification performed over the training set produces the best performance compared to that produced over the test sets, with higher TP percentages at the same FP percentages. For instance, at an FP percentage of 10%, the training set produces TP percentages between 80% and 85%, while test set 1 produces TP percentages between 61% and 71%. Test set 2, however, produces TP percentages between 30% and 40%.

Figure 4.7 reveals a similar trend, where classification performed over the training set produces the best performance compared to that produced over the test sets, with higher TP percentages at FP percentages greater than 2%. This figure, however, illustrates that the MOGA-1 and MOGA-2 composite ROC curves for each dataset maintain low FP percentages as the TP percentages increase. At most, these composite ROC curves attain a 7% FP percentage, with corresponding TP percentages of 80%, 63% and 54% for the training set, test set 1, and test set 2, respectively. An interesting observation is that test set 1 over MOGA-1 and MOGA-2, attains approximately 1% to 5% lower FP percentages than the training set over MOGA-1 and MOGA-2.

It is observed that all composite ROC curves for test set 2 in figure 4.6 and 4.7, typically exhibit lower TP percentages than all remaining feature subsets. This is owing to the fact that test set 2 consists of a majority of attack data samples (refer to table 3.3). Given the cluster labelling scheme used in the proposed classifier, the largest cluster will typically consist of a larger quantity of attack data samples, which are incorrectly classified as legitimate data samples, thus decreasing the TP percentage. Table 4.5 illustrates a typical clustering result obtained by applying the k–means clustering algorithm over the full feature set with $K = 6$. The table provides the contents of each cluster which shows that the largest cluster consists of a majority of attack data samples, as evident in cluster 6.

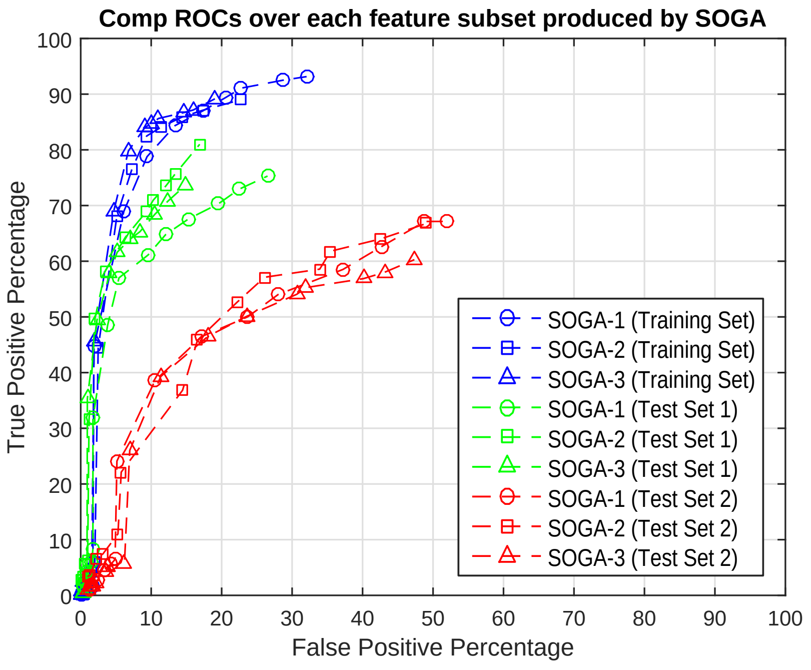


FIGURE 4.6: *Composite ROC curves obtained by applying the proposed classifier to the training and test sets over the feature subsets produced by the single-objective genetic algorithm over multiple values of K.*

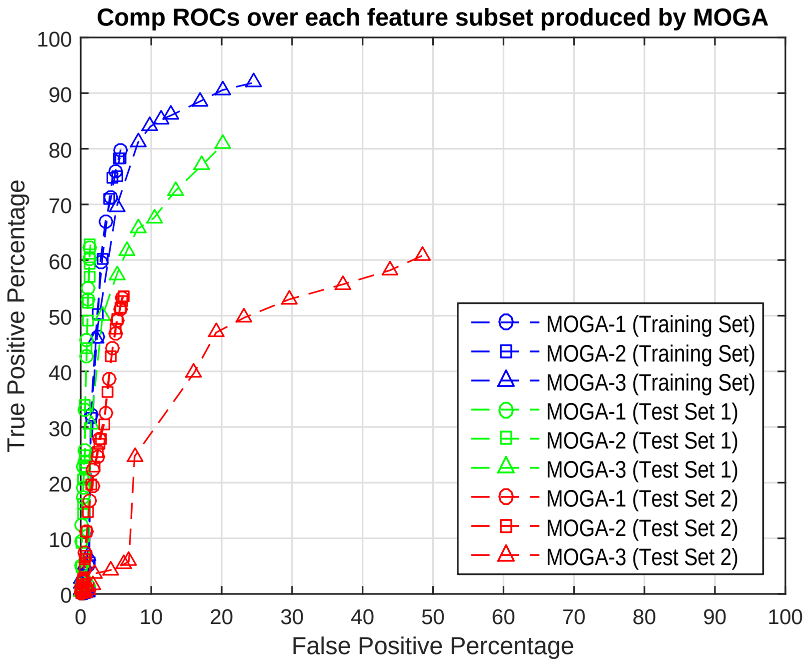


FIGURE 4.7: *Composite ROC curves obtained by applying the proposed classifier to the training and test sets over the feature subsets produced by the multi-objective genetic algorithm over multiple values of K.*

Table 4.5: *Contents of each cluster produced by applying k–means over test set 2, with* $K = 6$.

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**	**Cluster 6**
DoS	43	1 019	853	202	0	2 242
Probe	376	689	262	103	555	417
R2L	220	75	18	0	0	2 557
U2R	9	0	4	0	0	54
Normal	99	71	25	5	0	1 952
Total	**747**	**1 854**	**1 162**	**310**	**555**	**7 222**

4.6.2.3 Comparison of feature selection algorithms

This section compares the classification performance obtained by applying the proposed classifier to each of the training and test sets over each of the feature subsets of those depicted in table 4.2, and the best performing feature subsets depicted in table 4.3 and 4.4. The best performing feature subsets were selected as those feature subsets that produced the highest average MCC value as computed over the composite ROC curves produced by the application of the proposed classifier over the training set. The selected subsets are SOGA-3 and MOGA-3.

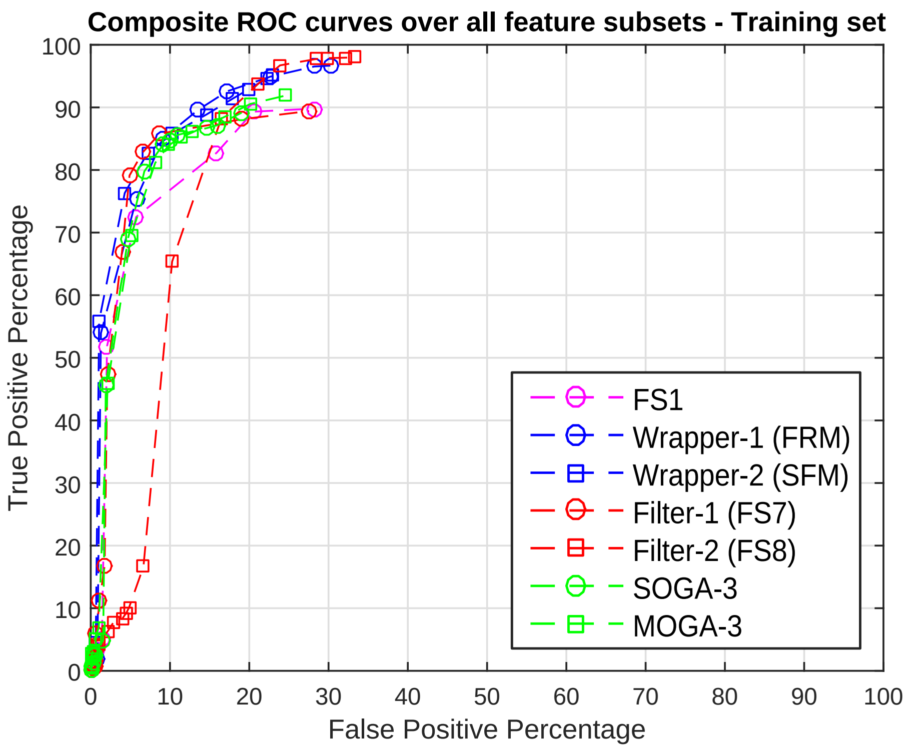
Figures 4.8, 4.9 and 4.10 consists of the composite ROC curves produced by each feature subset as well as the full feature set (FS1), by applying the proposed classifier to the training set, test set 1, and test set 2, respectively. Figures 4.8b, 4.9b and 4.10b provide a closer inspection of the composite ROC curves presented in figures 4.8a, 4.9a and 4.10a.

It is observed in figure 4.8, that a majority of the feature subsets (Wrapper-1, Wrapper-2, Filter-1, SOGA-3, MOGA-3) produce superior performance with TP percentages of 85%, at corresponding FP percentages of 10%. While at the same FP percentage, the full feature set (FS1) and Filter-2 produce TP percentages of 77% and 65%, respectively. However, Filter-2 produces the highest TP percentage amongst all remaining feature subsets of approximately 97%, but with corresponding FP percentages of more than 30%. The feature subsets produced by the proposed feature selection algorithm (SOGA-3 and MOGA-3) shows comparable performance to both Wrapper-1 and Wrapper-2, as well as Filter-1. At lower FP percentages, between 7% and 10%, SOGA-3 and MOGA-3 typically exhibit a marginal reduction in TP percentage of approximately 5%, at most. However, at FP percentages greater than 16%, both SOGA-3 and MOGA-3 produce, at most, 5% higher TP percentages than Filter-1. The marginal reduction in TP percentages produced by both SOGA-3 and MOGA-3 is acceptable given that these feature subsets were selected using an unsupervised feature selection algorithm, whereas the remaining feature subsets (excluding FS1), were selected using supervised feature selection algorithms. An important observation is that both SOGA-3 and MOGA-3 produce classification performances that are superior to that produced by the full feature set FS1.

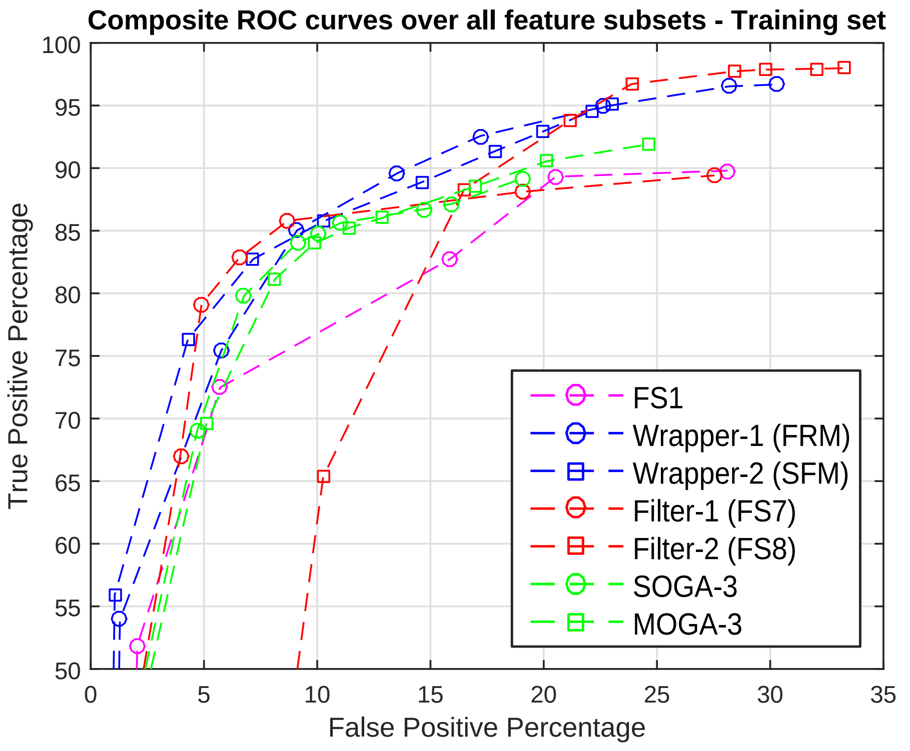
Figure 4.9 reveals that Filter-1, SOGA-3 and MOGA-3 all provide superior performance when compared to the remaining feature subsets. Filter-1 produces the lowest FP percentage of approximately 1%, however, Filter-1 only attains a maximum TP percentage of 65%, whereas all remaining feature subsets produce maximum TP percentages that are greater than 80%. Although, this only occurs at corresponding FP percentages between 20% and 49%. SOGA-3 and MOGA-3 consistently produce higher TP percentages than that produced by both wrapper-based subsets, both filter-based subsets and the full feature set, over the entire range of FP percentages. At FP percentages above 5%, both SOGA-3 and MOGA-3 provide a minimum increase of approximately 5% and a maximum increase of approximately 15%, to TP percentages, when compared to all feature subsets excluding Filter-1. Similarly, at TP percentages above 60%, both SOGA-3 and MOGA-3 provide a minimum reduction of approximately 4% and a maximum reduction of approximately 30%, to FP percentages, when compared to all feature subsets excluding Filter-1. This is an important observation as the SOGA-3 and MOGA-3 feature subsets were selected such that the clustering result of the training set would be optimized. However, when applied to a new dataset (i.e. test set 1), with data samples previously unseen by the proposed unsupervised feature selection algorithm, both feature subsets produce classification performances that outperform a majority of the classification performance produced by feature subsets selected using supervised wrapper-based and filter-based feature selection algorithms, as well as the full feature set.

Figure 4.10 reveals that the classification performance produced by Filter-1 is superior to all remaining feature subsets, with FP percentages of less than 9%, at a maximum TP percentage of 56%. At the same TP percentage (56%), all other feature subsets produce FP percentages that are between 32% and 70%, depending on the feature subset observed. The classification performance produced by both SOGA-3 and MOGA-3 is marginally improved when compared to that produced by sevearl of the feature subsets, excluding Filter-1, where both SOGA-3 and MOGA-3 produce improved TP percentages within certain ranges of FP percentages, while both produce lower TP percentages with regards to certain feature subsets, within other ranges.

In figure 4.10, at an FP percentage of 10% SOGA-3 provides the same TP percentage as the full feature set (FS1), approximately 36%, while both SOGA-3 and MOGA-3 exhibit higher TP percentages than Wrapper-1, Wrapper-2 and Filter-2; SOGA-3 produces a 7%, 16% and 19% improvement in TP percentages, respectively, and MOGA-3 produces a 2%, 10% and 13% improvement in TP percentages, respectively. At an FP percentage of 20%, however, SOGA-3 and MOGA-3 produced higher TP percentage than all remaining feature subsets, excluding Filter-1. When compared to FS1, Wrapper-1, Wrapper-2 and Filter-2, both SOGA-3 and MOGA-3 produce a 4%, 4%, 14% and 20% improvement in TP percentage, respectively. At a higher FP percentage of 40%, SOGA-3 and MOGA-3 both produce higher TP percentage than all remaining feature subsets, excluding Filter-1 and Wrapper-1. When compared to FS1, Wrapper-2 and Filter-2, both SOGA-3 and MOGA-3 produce a 8%, 12% and 17% improvement in TP percentage, respectively.

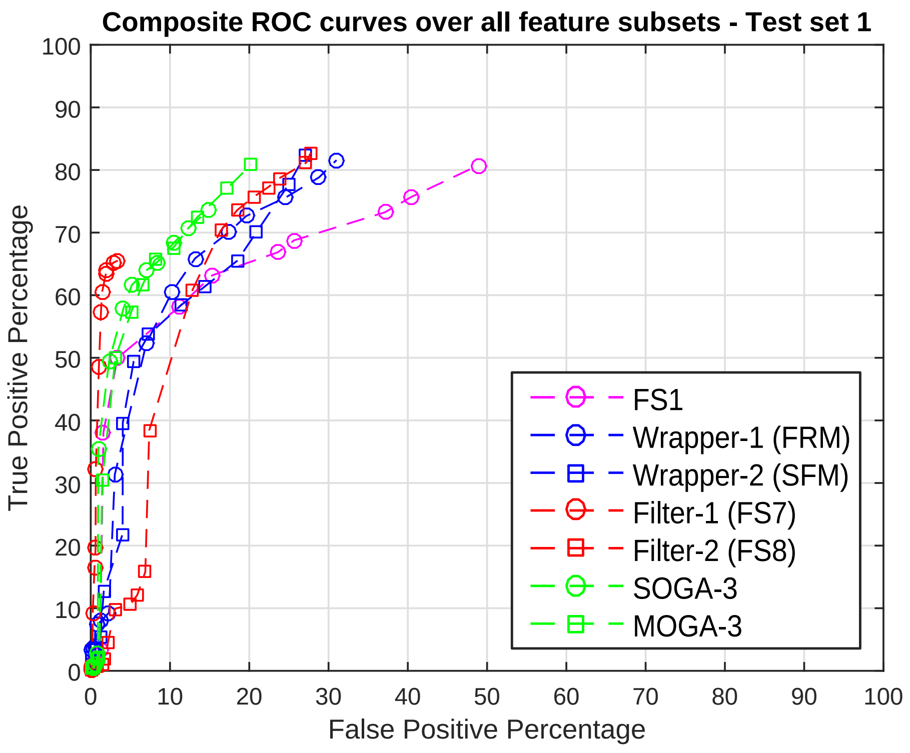


(a) Comparison over training set 2

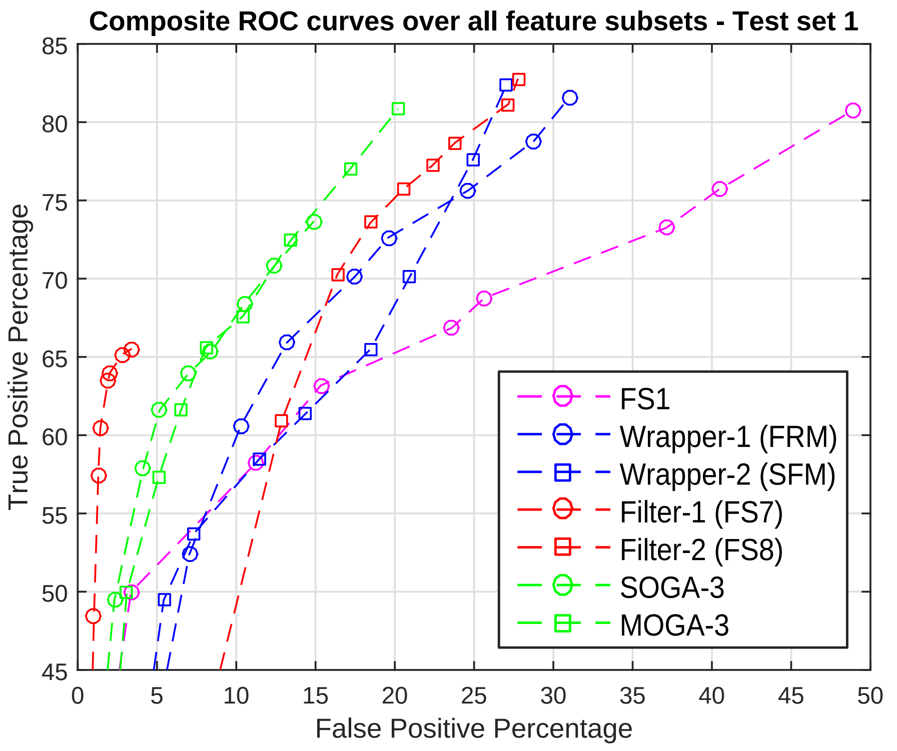


(b) Closer inspection of the comparison over training set 2

FIGURE 4.8: *Comparison of the performance attained by applying the proposed classifier to the training set over the feature subsets produced by several feature selection algorithms, which includes the proposed feature selection algorithm.*

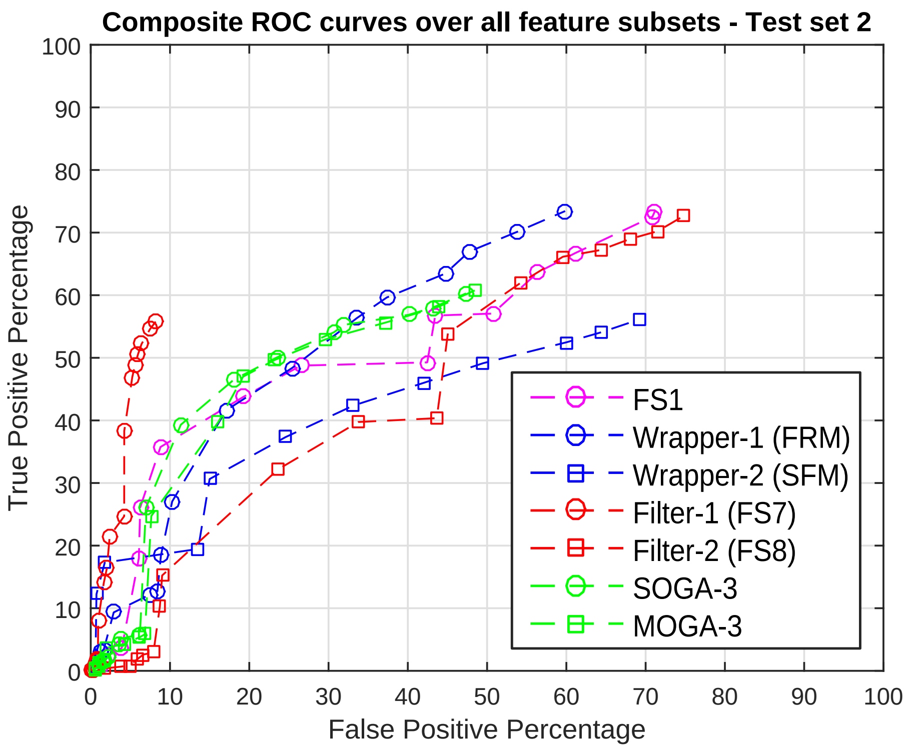


(a) Comparison over test set 1

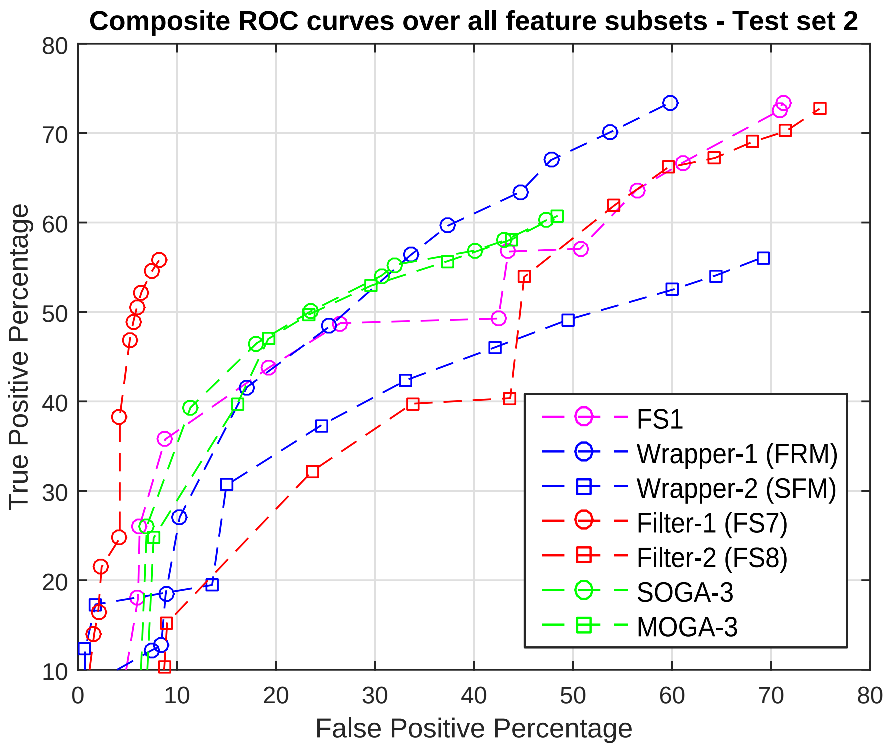


(b) Closer inspection of the comparison over test set 1

Figure 4.9: *Comparison of the performance attained by applying the proposed classifier to test set 1 over the feature subsets produced by several feature selection algorithms, which includes the proposed feature selection algorithm.*



(a) Comparison over test set 2



(b) Closer inspection of the comparison over test set 2

FIGURE 4.10: *Comparison of the performance attained by applying the proposed classifier to test set 2 over the feature subsets produced by several feature selection algorithms, which includes the proposed feature selection algorithm.*

4.6.2.4 Analysis of the convergence of the single-objective GA

This section presents an analysis of the convergence of the single-objective GA, starting from the initial randomly generated set of candidates until the final generation, in which the NDB index is minimized and the optimized feature subset is returned. Figure 4.11 shows the minimization of the NDB index value over each generation, when applying the proposed feature selection algorithm to the training set with $K = 8$. Each point corresponds to a generation, that is, each point corresponds to a set of candidate feature subsets. The blue points represent the mean of the NDB index values over the set of candidate feature subsets, and the black points represent the smallest NDB index value over the set of candidate feature subsets. The three circled points that exhibit an abrupt reduction in the minimum NDB index value in the set of candidate feature subsets will be discussed in what follows.

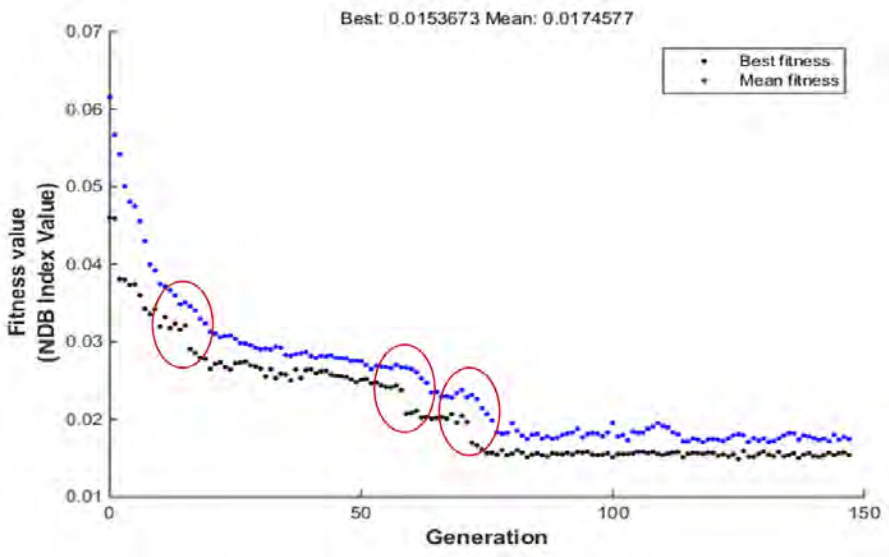


FIGURE 4.11: *The convergence of the genetic algorithm in searching for an optimized feature subset, that minimizes the average NDB index value.*

Figure 4.12 shows the step-by-step progression of the minimization of the NDB index value. On the left, the region of interest is illustrated, and on the right are several of the composite ROC curves that are produced using the sets of candidate feature subsets directly above the coloured blocks. The points above each coloured block in the figure indicates that those points (i.e. those candidate feature subsets), produced the corresponding composite ROC curves of the same colour.

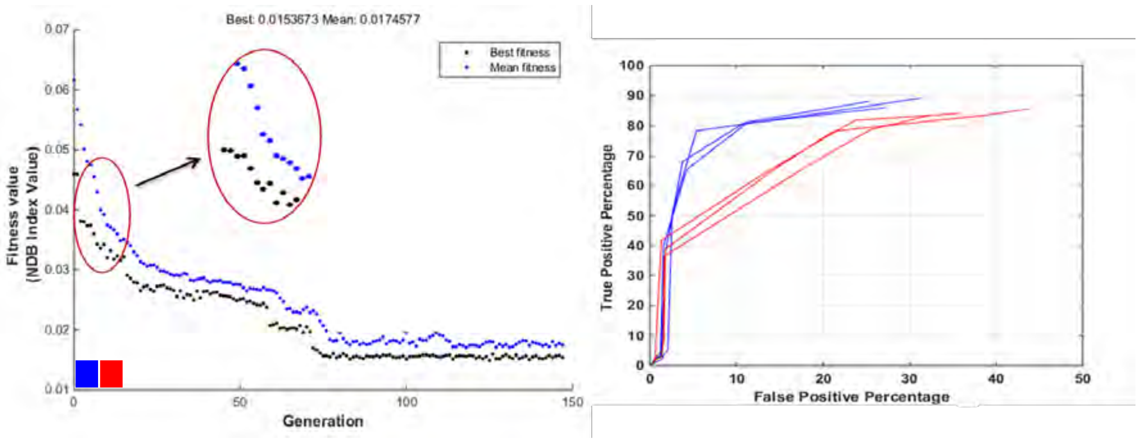
Figure 4.12a illustrates the initial stage of the genetic algorithm in which it searches for an appropriate set of candidate feature subsets which produces smaller average NDB index values. At this stage, the search space of the GA is large as it has only begun to progress to an improved feature subset. Figure 4.12a indicates that the blue and red composite ROC curves constitute a diverse or wide spread region of the TP and FP space.

Figure 4.12b illustrates that the GA locates sets of candidate feature subsets that provide classification performance that is, in general, better than the red curves (higher TP percentages), but worse than the blue curves (lower TP percentages), at the same FP percentages. However, upon closer inspection, it is observed that the green curves exhibit a marginally reduced FP percentage than the blue curves, approximately 0.5% to 1%, at TP percentages less than 40%. All points above the green block produce similar composite ROC curves until the next region of interest is reached.

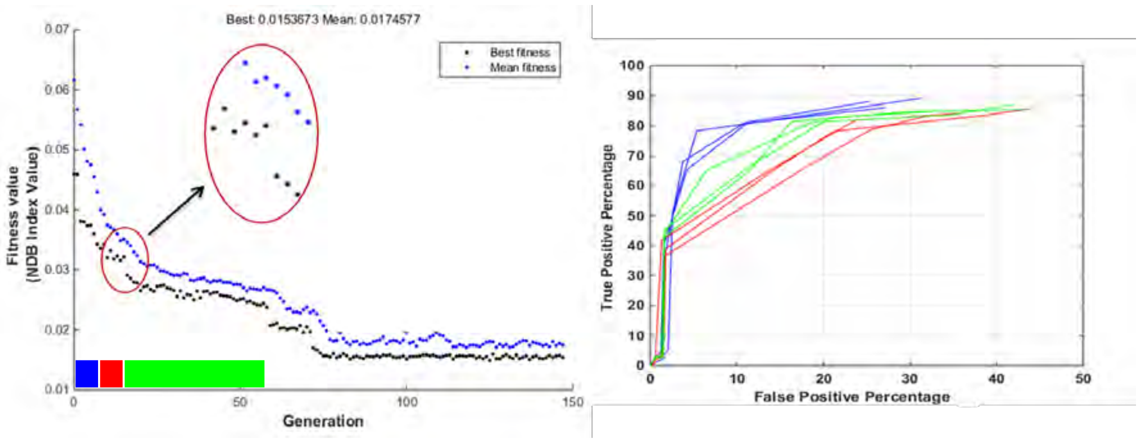
Figure 4.12c illustrates the region where the GA locates sets of candidate feature subsets that produce composite ROC curves (the black curves) that exhibit an improved level of performance over the composite ROC curves produced by the previous candidate feature subsets. The black curves exhibit a 5% to 8% improvement in TP percentages when compared to the blue curves, at FP percentages between 5% and 8%. All points above the black block produce similar composite ROC curves until the next region of interest is reached.

Figure 4.12d illustrates the final region of interest in which the GA locates sets of candidate feature subsets which further minimizes the NDB index value, however, these subsets do not increase the TP percentages when compared to the black curves. At an FP percentage of 5%, the pink composite ROC curves reach a maximum TP percentage of 80%, while the black composite ROC curves also produce an 80% TP percentage. However, the black curves reach a maximum TP percentage of 88%, at a corresponding FP percentage of 8%. At TP percentages less than 60%, the pink curves exhibit a marginal reduction in FP percentages, up to 2%, when compared to the black curves. All points above the pink block produce similar composite ROC curves for the remaining generations.

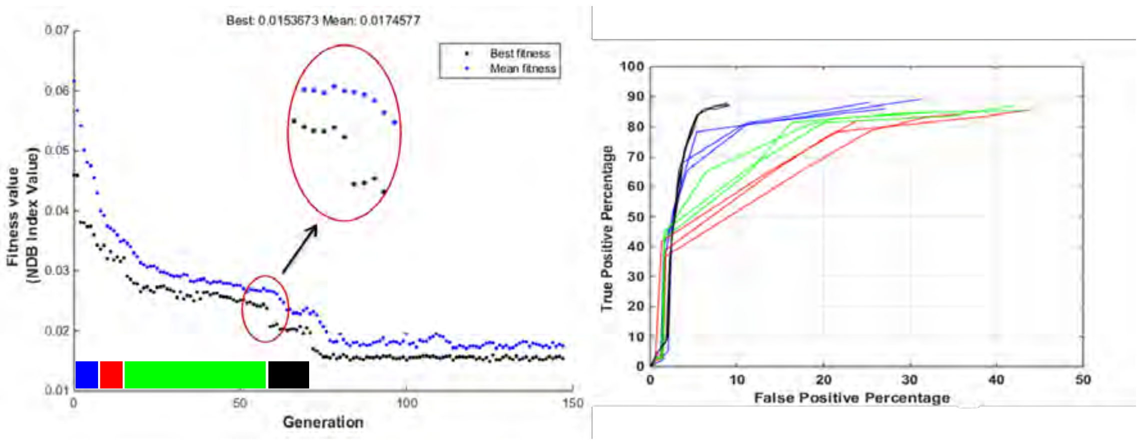
Given that the algorithm favoured the green curves over the blue curves in figure 4.12b, and the pink curves over the black curves in figure 4.12d, it is assumed that the NDB index favours lower false positive percentages over higher true positive percentages. However, this assumption requires further investigation.



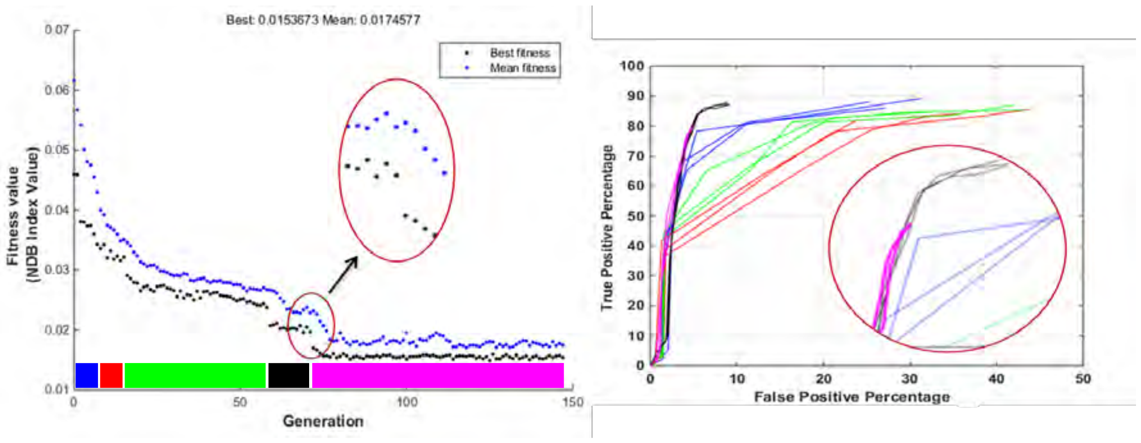
(a) Stage 1



(b) Stage 2



(c) Stage 3



(d) Stage 4

FIGURE 4.12: *Analysis of four significant stages of the progression of a particular implementation of the genetic algorithm to produce an optimized feature subset.*

4.7 Conclusion

In this chapter an unsupervised cluster validity-based feature selection algorithm for use in network intrusion detection was presented. A functional block diagram of the proposed feature selection algorithm was provided, and each block was discussed in further detail. The blocks consist of the dataset utilized, the data transformation using normalization and encoding methods, clustering of the dataset using the k–means clustering algorithm, the computation of the Davies-Bouldin cluster validity index, and the implementation of a genetic algorithm. A flowchart of the proposed feature selection algorithm was also provided.

The proposed feature selection algorithm was applied using both single-objective and multi-objective optimizations, using fixed values of K for the k–means clustering algorithm. This produced three feature subsets for the single-objective optimization, and three Pareto-optimal fronts for the multi-objective optimization.

The experimental work consisted of two experiments. The first compared the classification performance of the three subsets produced through the single-objective optimization and the three best performing subsets in each Pareto-optimal front produced through the multi-objective optimization, over different datasets. The second experiment compared the classification performance of the best performing feature subsets produced by the proposed feature selection algorithm, based on average MCC value, to the full feature set, two feature subsets obtained through filter-based methods, and two feature subsets obtained through wrapper-based methods. The classification performance was measured by applying the proposed classifier to a training set and to two test sets over each of the feature subsets, which produced composite ROC curves for each feature subset.

The comparison of the classification performance produced by the feature subsets returned by the GA over different datasets, revealed that the best performance is typically achieved when classification is performed over the same dataset that was used in the selection of the feature subsets. It was observed that the feature subsets produced by the multi-objective GA enables the k–means clustering algorithm to attain lower FP percentages when compared to that produced by the single-objective GA. Specifically, when applied to test set 1, MOGA-1 and MOGA-2 produce FP percentages of less than 1%, at a TP percentage of 63%. Additionally, it was observed that the proposed classifier suffers in performance when applied to a dataset that does not contain a majority of legitimate data samples. This is evident in the fact that test set 2 typically produces lower TP and higher FP percentages, when compared to the performance attained when using the training set and test set 1.

The comparison between the subsets produced by all feature selection algorithms considered in this chapter, revealed that the proposed feature selection algorithm is capable of producing feature subsets that perform comparably, or outperforms the remaining feature selection algorithms. For the training set, the SOGA and MOGA feature subsets perform comparably to the remaining subsets. These feature subsets exhibit a marginal reduction in TP percentages when compared to a majority of the remaining feature subsets, at similar FP percentages. For test set 1, it was discovered that the SOGA and MOGA feature subsets outperformed all remaining feature subsets, excluding feature subset Filter-1, with higher TP percentages throughout the entire range of FP percentages. For test set 2, the SOGA and MOGA feature subsets produced superior performance to all feature subsets, excluding the Filter-1 and Wrapper-1 subsets. It is important to note that the proposed feature selection algorithm did not use labelled data in the selection of the feature subsets, while all remaining feature subsets used labelled data to select features. Additionally, an important observation is that both the single-objective and multi-objective feature subsets attained improved classification performances than that produced by the full feature subset, which was attained with fewer features than the full feature set.

Chapter 5

Conclusion

This research consists of three main objectives, which includes the comparison of clustering algorithms and feature subsets and the implementation of an unsupervised feature selection algorithm, all in the context of anomaly-based network intrusion detection.

The comparison of four clustering algorithms and 10 feature subsets found in the literature, in addition to the full feature set, was achieved through the design and implementation of an unsupervised anomaly-based classifier for network intrusion detection. The proposed classifier consists of several stages, data transformation using normalization and encoding techniques, clustering and cluster labelling. The performance of the classifier was subsequently measured using true positive and false positive percentages, which were illustrated using receiver operating characteristic (ROC) curves, as well as a composite of these curves, which was introduced in this research.

The proposed classifier was applied using four different clustering algorithms: k–means, k-medoids, k–means with distance-based outlier detection and the expectation-maximization clustering algorithms. Each classifier was subsequently applied to the NSL-KDD training set [25] over the full feature set, as well as 10 different feature subsets. This allowed for the comparison of the classification performance produced by each clustering algorithm over the full feature set and different feature subsets.

The experimental results revealed that from the four clustering algorithms considered in this research, one single clustering algorithm does not outperform all other clustering algorithms over all feature subsets. For instance, when clustering is performed over the full feature set, the k–medoids algorithm outperforms all others, while clustering over feature subset FS9 reveals that the k–means algorithm outperforms all others. It was observed that performing clustering using all four clustering algorithms over feature subset FS7, produced classification performances that were superior to all other feature subsets. In addition to FS7, it was observed that several feature subsets enabled each clustering algorithm to produce classification performances that are superior to that produced by the full feature set. This was attained using fewer features than that found in the full feature set, which motivates the need for feature selection. An interesting observation is that there is no direct correlation between feature cardinality and TP and FP performance. Feature subsets with the same number of features were demonstrated to provide classification performances that are highly dissimilar to one another.

The design and implementation of the unsupervised feature selection algorithm was driven by the lack of labelled network data in practical environments. The proposed feature selection algorithm consists of several stages which include, data transformation using normalization and encoding techniques, clustering using the k–means clustering algorithm, the computation of the Davies-Bouldin cluster validity index and the application of both a single-objective and multi-objective genetic algorithm.

The feature subsets produced by the proposed feature selection algorithm were demonstrated to produce classification performances that outperformed that produced by the full feature set. When compared against two wrapper-based and two filter-based feature selection algorithms, the proposed feature sets produced TP and FP percentages that were comparable to the subsets of the remaining feature selection algorithms, and in some cases outperformed the subsets of the remaining feature selection algorithms when applied to different datasets. The proposed classifier was applied to a test set over the feature subsets produced by each feature selection algorithm. It was observed that the feature subsets of the proposed feature selection algorithm produced up to a 15% improvement in TP percentages, for FP percentages above 5%, over a majority of the remaining feature subsets. This is an important observation as it demonstrates that the proposed feature selection algorithm is capable of producing feature subsets that provide superior classification performance when applied to a dataset containing data samples that were not present during the feature selection stage.

These results demonstrate that an unsupervised feature selection algorithm for use in network intrusion detection was successfully designed and implemented. The proposed feature selection algorithm did not use labelled network data to produce feature subsets, while the benchmark feature subsets, were produced through both filter-based and wrapper-based feature selection algorithms that required labelled data to select features.

Chapter 6

Future Work

This chapter highlights several aspects of this research that, if extended upon or altered, could potentially improve the performance of the unsupervised feature selection algorithm.

The first aspect involves the use of an alternate cluster validity index. In this research, the Davies-Bouldin cluster validity index was used to evaluate the clustering quality of the clustering results returned by various clustering algorithms. This index was subsequently used as an indicator of the quality of the feature subsets returned by the genetic algorithm. It is believed that an alternate cluster validity index may better discriminate between high and low quality clustering results, which may subsequently improve the quality of the feature subsets returned by the genetic algorithm. Future work would involve the use of other cluster validity indices in the feature selection algorithm, where alternate indices were specified in section 4.3.

The normalization of the Davies-Bouldin index was performed to counter the bias that the index places towards lower dimensional spaces. This normalization was achieved by dividing the index value by the feature subset cardinality. This method of normalization may be improved upon as there are other methods that may be used to normalize the Davies-Bouldin index, which would aid in producing better quality feature subsets. Future work would involve further investigation into these methods.

Appendix A

Supervised Wrapper-based Feature Selection Algorithms

The following sections describe the derivation of the wrapper-based feature subsets provided in table 4.2, which were compared against the feature subsets produced by the proposed feature selection algorithm.

A.1 Feature Removal Method (FRM)

The feature removal method, a wrapper-based feature selection algorithm, that was summarized in section 2.4.2, was used to produce a feature subset for comparison. The FRM algorithm was implemented in this research using the k–means clustering algorithm as a classifier, thereby producing feature subsets optimized for use with k–means.

The FRM algorithm performs feature selection in two steps. The initial step involves the ranking of each feature based on a performance measure, as calculated after clustering and cluster labelling was performed. The second step involves the selection of a suitable number of the top ranked features to constitute the final feature subset produced by the FRM feature selection algorithm.

In the first step, the FRM algorithm iteratively ranks each feature present in the full feature set, as described in what follows.

- A feature is removed from the full feature set.
- The proposed classifier with k–means clustering, is applied to the dataset over the reduced feature set for $K \in \{2, \ldots, 10\}$.
- The TP and FP percentages are computed for each value of K.
- The composite ROC curve using the obtained TP and FP percentages is constructed.
- The average Matthew's Correlation Coefficient (MCC) over the TP and FP percentages of the composite ROC curve is computed.

MCC measures the quality of the classification produced by a binary classifier by taking into account the number of TPs, FPs, TNs and FNs that were produced by the classifier. It is computed as

$$MCC = \frac{(TP)(TN) - (FP)(FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}. \tag{A.1}$$

The MCC value is between [-1,1], where -1 indicates that none of the data samples were correctly classified, 1 indicates that all data samples were correctly classified, and 0 indicates that the equivalent of a random classification was performed.

The result of this iterative process is an average MCC value corresponding to each feature. The FRM algorithm associates smaller average MCC values with more significant features. That is, if the removal of a feature results in a significant reduction in the average MCC value, then the removed feature is considered significant. The output of the first step is a list of features ranked in ascending order with regards to average MCC values.

The second step of the FRM algorithm involves the construction of feature subsets produced by the FRM algorithm, and the selection of the best performing feature subset. Given the ranked list of features, the algorithm constructs N_F feature subsets, where the i^{th} feature subset consists of the top i features. That is, the first feature subset consists of only the top ranked feature; the second feature subset consists of the two top ranked features, and so on. From the candidate N_F feature subsets, the algorithm selects the best feature subset according to classification performance. The proposed classifier is applied to the dataset over each of the N_F feature subsets produced by the FRM algorithm, and the composite ROC curves corresponding to each feature subset is computed. The average MCC value for each composite ROC curve is calculated and the feature subset corresponding to the highest MCC score is selected. (i.e. Wrapper-1 in table 4.2).

A.2 Sole Feature Method (SFM)

The sole feature method, a wrapper-based feature selection algorithm, that was summarized in section 2.4.2, was used to produce a feature subset for comparison. The SFM algorithm was implemented in this research using the k–means clustering algorithm as a classifier, thereby producing feature subsets optimized for use with k–means.

The SFM algorithm performs feature selection in two steps. The initial step involves the ranking of each feature based on a performance measure, as calculated after clustering and cluster labelling was performed. The second step involves the selection of a suitable number of the top ranked features to constitute the final feature subset produced by the SFM feature selection algorithm.

In the first step, the SFM algorithm iteratively ranks each feature present in the full feature set, as described in what follows.

- A single feature is selected from the full feature set.
- The proposed classifier with k–means clustering, is applied to the dataset over the selected feature for $K \in \{2, \ldots, 10\}$.
- The TP and FP percentages are computed for each value of K.
- The composite ROC curve using the obtained TP and FP percentages is constructed.
- The average Matthew's Correlation Coefficient (MCC) over the TP and FP percentages of the composite ROC curve is computed.

The result of this iterative process is an average MCC value corresponding to each feature. The SFM algorithm associates larger average MCC values with more significant features. The output of the first step is a list of features ranked in descending order with respect to the average MCC values.

The second step of the SFM algorithm involves the construction of feature subsets produced by the SFM algorithm, and the selection of the best performing feature subset. This step is identical to that of the second step of the FRM feature selection algorithm described in section A.1. The composite ROC curve with the highest average MCC value was selected for comparison (i.e. Wrapper-2 in table 4.2).

References

[1] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

[2] D. K. Bhattacharyya and J. K. Kalita, *Network Anomaly Detection: A Machine Learning Perspective.* CRC Press, 2013.

[3] C. R. Raquel and P. C. Naval Jr, "An effective use of crowding distance in multi-objective particle swarm optimization," in *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation.* ACM, 2005, pp. 257–264.

[4] W. Stallings, *Network security essentials: applications and standards.* Pearson Education India, 2007.

[5] S. J. Stolfo. (1999, Oct.) Kdd datasets. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[6] S. J. Stolfo. (1999, Oct.) Kdd task description. [Online]. Available: https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/task.html

[7] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: a feature relevance analysis on kdd 99 intrusion detection datasets," in *Proceedings of the third annual conference on privacy, security and trust*, 2005.

[8] D. Halder, K. Jaishankar, and K. Jaishankar, *Cyber crime and the victimization of women: laws, rights and regulations.* Information Science Reference, 2012.

[9] Symantec. (2013) Norton report 2013 - south africa. [Online]. Available: http://www.symantec.com/content/en/us/about/presskits/b-norton-report-2013-south-africa.pdf

[10] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, "Network intrusion detection," *Network, IEEE*, vol. 8, no. 3, pp. 26–41, 1994.

[11] A. Sundaram, "An introduction to intrusion detection," *Crossroads*, vol. 2, no. 4, pp. 3–7, 1996.

[12] J. P. Anderson, "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[13] A. B. Mohamed, N. B. Idris, and B. Shanmugum, "A brief introduction to intrusion detection system," in *Trends in Intelligent Robotics, Automation, and Manufacturing.* Springer, 2012, pp. 263–271.

[14] J. Cannady and J. Harrell, "A comparative analysis of current intrusion detection technologies," in *Proceedings of the Fourth Technology for Information Security Conference*, vol. 96, 1996.

[15] T. F. Lunt, "Automated audit trail analysis and intrusion detection: A survey," in *In Proceedings of the 11th National Computer Security Conference.* Citeseer, 1988.

[16] E. Biermann, E. Cloete, and L. M. Venter, "A comparison of intrusion detection systems," *Computers & Security*, vol. 20, no. 8, pp. 676–683, 2001.

[17] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[18] G. Vigna and C. Kruegel, *Host-based intrusion detection.* na, 2006.

[19] P. Kabiri and A. A. Ghorbani, "Research on intrusion detection and response: A survey." *IJ Network Security*, vol. 1, no. 2, pp. 84–102, 2005.

[20] M. Ektefa, S. Memar, F. Sidi, and L. S. Affendey, "Intrusion detection using data mining techniques," in *Information Retrieval & Knowledge Management,(CAMP), 2010 International Conference on.* IEEE, 2010, pp. 200–203.

[21] R. A. Kemmerer and G. Vigna, "Intrusion detection: a brief history and overview," *Computer*, vol. 35, no. 4, pp. 27–30, 2002.

[22] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," *Proc. SIAM*, 2003.

[23] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques: concepts and techniques.* Elsevier, 2011.

[24] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[25] M. Tavallaee, "Nsl-kdd dataset," March 2009. [Online]. Available: http://nsl.cs.unb.ca/KDD/NSL-KDD.html

[26] F. Amiri, M. Rezaei Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1184–1199, 2011.

[27] S. Chebrolu, A. Abraham, and J. P. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers & Security*, vol. 24, no. 4, pp. 295–307, 2005.

[28] A. A. Olusola, A. S. Oladele, and D. O. Abosede, "Analysis of kdd'99 intrusion detection dataset for selection of relevance features," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1. Citeseer, 2010, pp. 20–22.

[29] P. Tang, R.-a. Jiang, and M. Zhao, "Feature selection and design of intrusion detection system based on k-means and triangle area support vector machine," in *Future Networks, 2010. ICFN'10. Second International Conference on.* IEEE, 2010, pp. 144–148.

[30] H. F. Eid, A. E. Hassanien, T.-h. Kim, and S. Banerjee, "Linear correlation-based feature selection for network intrusion detection model," in *Advances in Security of Information and Communication Networks.* Springer, 2013, pp. 240–248.

[31] Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai, and K. Dai, "An efficient intrusion detection system based on support vector machines and gradually feature removal method," *Expert Systems with Applications*, vol. 39, no. 1, pp. 424–430, 2012.

[32] A. Dastanpour and R. A. R. Mahmood, "Feature selection based on genetic algorithm and supportvector machine for intrusion detection system," in *The Second International Conference on Informatics Engineering & Information Science (ICIEIS2013).* The Society of Digital Information and Wireless Communication, 2013, pp. 169–181.

[33] F. Zhang and D. Wang, "An effective feature selection approach for network intrusion detection," in *Networking, Architecture and Storage (NAS), 2013 IEEE Eighth International Conference on.* IEEE, 2013, pp. 307–311.

[34] V. Engen, J. Vincent, and K. Phalp, "Exploring discrepancies in findings obtained with the kdd cup'99 data set," *Intelligent Data Analysis*, vol. 15, no. 2, pp. 251–276, 2011.

[35] S. Zargari and D. Voorhis, "Feature selection in the corrected kdd-dataset," in *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on.* IEEE, 2012, pp. 174–180.

[36] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis*, vol. 1, no. 1, pp. 131–156, 1997.

[37] H. Liu and L. Yu, "Toward integrating feature selection algorithms for classification and clustering," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 4, pp. 491–502, 2005.

[38] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, vol. 30, no. 6, pp. 353–375, 2011.

[39] W. Fan, N. Bouguila, and D. Ziou, "Unsupervised anomaly intrusion detection via localized bayesian feature selection," in *Data Mining (ICDM), 2011 IEEE 11th International Conference on.* IEEE, 2011, pp. 1032–1037.

[40] H. F. Eid, M. A. Salama, A. E. Hassanien, and T.-h. Kim, "Bi-layer behavioral-based feature selection approach for network intrusion classification," in *Security Technology.* Springer, 2011, pp. 195–203.

[41] J. Jung, "Real-time detection of malicious network activity using stochastic models," Ph.D. dissertation, Citeseer, 2006.

[42] P. Gogoi, B. Borah, and D. K. Bhattacharyya, "Anomaly detection analysis of intrusion data using supervised & unsupervised approach," *Journal of Convergence Information Technology*, vol. 5, no. 1, pp. 95–110, 2010.

[43] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001.* Citeseer, 2001.

[44] I. Syarif, A. Prugel-Bennett, and G. Wills, "Unsupervised clustering approach for network anomaly detection," in *Networked Digital Technologies.* Springer, 2012, pp. 135–145.

[45] S. Wang, "Research of intrusion detection based on an improved k-means algorithm," in *Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on.* IEEE, 2011, pp. 274–276.

[46] E. E. Papalexakis, A. Beutel, and P. Steenkiste, "Network anomaly detection using co-clustering," in *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012).* IEEE Computer Society, 2012, pp. 403–410.

[47] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[48] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "Nado: network anomaly detection using outlier approach," in *Proceedings of the 2011 International Conference on Communication, Computing & Security*. ACM, 2011, pp. 531–536.

[49] S. Songma, W. Chimphlee, K. Maichalernnukul, and P. Sanguansat, "Classification via k-means clustering and distance-based outlier detection," in *ICT and Knowledge Engineering (ICT & Knowledge Engineering), 2012 10th International Conference on*. IEEE, 2012, pp. 125–128.

[50] S. Chawla and A. Gionis, "k-means-: A unified approach to clustering and outlier detection." in *SDM*, 2013, pp. 189–197.

[51] L. Duan, L. Xu, Y. Liu, and J. Lee, "Cluster-based outlier detection," *Annals of Operations Research*, vol. 168, no. 1, pp. 151–168, 2009.

[52] W. Masri, R. Abou Assi, and M. El-Ghali, "Generating profile-based signatures for online intrusion and failure detection," *Information and Software Technology*, vol. 56, no. 2, pp. 238–251, 2014.

[53] P. Casas, J. Mazel, and P. Owezarski, "Steps towards autonomous network security: Unsupervised detection of network attacks," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE, 2011, pp. 1–5.

[54] M. Roesch. (2015) Snort. [Online]. Available: https://www.snort.org/

[55] V. Paxson, "Bro: a System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999. [Online]. Available: http://www.icir.org/vern/papers/bro-CN99.pdf

[56] K.-c. Lan, A. Hussain, and D. Dutta, "Effect of malicious traffic on the network," in *Passive and Active Measurement Workshop (PAM)*, 2003.

[57] T. Verwoerd and R. Hunt, "Intrusion detection techniques and approaches," *Computer communications*, vol. 25, no. 15, pp. 1356–1365, 2002.

[58] S. Biles, "Detecting the unknown with snort and the statistical packet anomaly detection engine (spade)," *Computer Security Online Ltd., Tech. Rep*, 2003.

[59] K. Zaraska, “Prelude ids: current state and development perspectives,” *URL http://www. prelude-ids. org/download/misc/pingwinaria/2003/paper. pdf*, 2003.

[60] S. Jungsuk, H. Takakura, Y. Okabe, and K. Yongjin, “Unsupervised anomaly detection based on clustering and multiple one-class svm,” *IEICE transactions on communications*, vol. 92, no. 6, pp. 1981–1990, 2009.

[61] J. Song, H. Takakura, Y. Okabe, and K. Nakao, “Toward a more practical unsupervised anomaly detection system,” *Information Sciences*, vol. 231, pp. 4–14, 2013.

[62] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, “On clustering validation techniques,” *Journal of Intelligent Information Systems*, vol. 17, no. 2-3, pp. 107–145, 2001.

[63] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*, 4th ed. Academic Press, 2008.

[64] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, “Supervised machine learning: A review of classification techniques,” 2007.

[65] F. Murtagh and P. Contreras, “Algorithms for hierarchical clustering: an overview,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[66] A. Banerjee, I. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha, “A generalized maximum entropy approach to bregman co-clustering and matrix approximation,” in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2004, pp. 509–514.

[67] E. E. Papalexakis and N. D. Sidiropoulos, “Co-clustering as multilinear decomposition with sparse latent factors,” in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on.* IEEE, 2011, pp. 2064–2067.

[68] B. Pfahringer, “Winning the kdd99 classification cup: bagged boosting,” *ACM SIGKDD Explorations Newsletter*, vol. 1, no. 2, pp. 65–66, 2000.

[69] V. Hautamäki, S. Cherednichenko, I. Kärkkäinen, T. Kinnunen, and P. Fränti, "Improving k-means by outlier removal," in *Image Analysis.* Springer, 2005, pp. 978–987.

[70] J. Mazel, P. Casas, Y. Labit, and P. Owezarski, "Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection," in *Proceedings of the 7th International Conference on Network and Services Management.* International Federation for Information Processing, 2011, pp. 73–80.

[71] P. Casas, J. Mazel, and P. Owezarski, "Knowledge-independent traffic monitoring: Unsupervised detection of network attacks," *Network, IEEE*, vol. 26, no. 1, pp. 13–21, 2012.

[72] G. Cormode and S. Muthukrishnan, "What's new: finding significant differences in network data streams," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 6, pp. 1219–1232, 2005.

[73] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: a review," *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 90–105, 2004.

[74] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[75] A. L. Fred and A. K. Jain, "Combining multiple clusterings using evidence accumulation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 6, pp. 835–850, 2005.

[76] M. H. Bhuyan, D. Bhattacharyya, and J. Kalita, "Rodd: An effective reference-based outlier detection technique for large datasets," in *Advanced Computing.* Springer, 2011, pp. 76–84.

[77] L. Li and K.-n. Zhao, "A new intrusion detection system based on rough set theory and fuzzy support vector machine," in *Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on.* IEEE, 2011, pp. 1–5.

[78] D. Said, L. Stirling, P. Federolf, and K. Barker, “Data preprocessing for distance-based unsupervised intrusion detection,” in *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on.* IEEE, 2011, pp. 181–188.

[79] L. Duan, L. Xu, F. Guo, J. Lee, and B. Yan, “A local-density based spatial clustering algorithm with noise,” *Information Systems*, vol. 32, no. 7, pp. 978–986, 2007.

[80] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[81] M. A. Ambusaidi, X. He, and P. Nanda, “Unsupervised feature selection method for intrusion detection system,” in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 295–301.

[82] R. Kohavi and G. H. John, “Wrappers for feature subset selection,” *Artificial intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[83] J. Handl and J. Knowles, “Feature subset selection in unsupervised learning via multiobjective optimization,” *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 217–238, 2006.

[84] J. A. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[85] M. A. Hall, “Correlation-based feature selection for machine learning,” Ph.D. dissertation, The University of Waikato, 1999.

[86] J. R. Quinlan, *C4. 5: programs for machine learning.* Elsevier, 2014.

[87] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1, pp. 37–52, 1987.

[88] J. T. Kent, “Information gain and a general measure of correlation,” *Biometrika*, vol. 70, no. 1, pp. 163–173, 1983.

[89] K. Kira and L. A. Rendell, “The feature selection problem: Traditional methods and a new algorithm,” in *AAAI*, vol. 2, 1992, pp. 129–134.

[90] M. F. Zibran, "Chi-squared test of independence," *Department of Computer Science, University of Calgary, Alberta, Canada.[online].[Cited 2010-08-12]*, 2007.

[91] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham *et al.*, "Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, vol. 2. IEEE, 2000, pp. 12–26.

[92] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the jam project," in *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, vol. 2. IEEE, 2000, pp. 130–144.

[93] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," in *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on.* IEEE, 1999, pp. 120–132.

[94] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Usenix Security*, 1998.

[95] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[96] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[97] J. R. Quinlan, *C4. 5: programs for machine learning.* Morgan kaufmann, 1993, vol. 1.

[98] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[99] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid." in *KDD*, 1996, pp. 202–207.

[100] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[101] D. Aldous, "The continuum random tree." *The Annals of Probability*, pp. 1–28, 1991.

[102] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, "The multilayer perceptron as an approximation to a bayes optimal discriminant function," *Neural Networks, IEEE Transactions on*, vol. 1, no. 4, pp. 296–298, 1990.

[103] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[104] P. Natesan and P. Balasubramanie, "Multi stage filter using enhanced adaboost for network intrusion detection," *International Journal of Network Security & Its Applications (IJNSA)*, vol. 4, no. 3, pp. 121–135, 2012.

[105] C. Thomas, V. Sharma, and N. Balakrishnan, "Usefulness of darpa dataset for intrusion detection system evaluation," in *SPIE Defense and Security Symposium.* International Society for Optics and Photonics, 2008, pp. 69 730G–69 730G.

[106] Massachusetts Institute of Technology, Lincoln Laboratory. (2015, Oct.) Intrusion detection attacks database. [Online]. Available: http://www.ll.mit.edu/ideval/docs/attackDB.html

[107] Z. Ihsan, M. Y. Idris, and A. H. Abdullah, "Attribute normalization techniques and performance of intrusion classifiers: A comparative analysis," *Life Science Journal*, vol. 10, no. 4, 2013.

[108] W. Wang, X. Zhang, S. Gombault, and S. J. Knapskog, "Attribute normalization in network intrusion detection," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on.* IEEE, 2009, pp. 448–453.

[109] Y. B. Bhavsar and K. C. Waghmare, "Intrusion detection system using data mining technique: Support vector machine," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 3, 2013.

[110] C. Cheng, W. P. Tay, and G.-B. Huang, "Extreme learning machines for intrusion detection," in *Neural Networks (IJCNN), The 2012 International Joint Conference on.* IEEE, 2012, pp. 1–8.

[111] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?" in *Image Analysis and Processing–ICIAP 2005.* Springer, 2005, pp. 50–57.

[112] D.-Y. Yeung and C. Chow, "Parzen-window network intrusion detectors," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 4. IEEE, 2002, pp. 385–388.

[113] Y. Li, B. Fang, L. Guo, and Y. Chen, "Network anomaly detection based on tcm-knn algorithm," in *Proceedings of the 2nd ACM symposium on Information, computer and communications security.* ACM, 2007, pp. 13–19.

[114] Y. Li and L. Guo, "An active learning based tcm-knn algorithm for supervised network intrusion detection," *Computers & security*, vol. 26, no. 7, pp. 459–467, 2007.

[115] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[116] H. Steinhaus, "Sur la division des corp materiels en parties," pp. 801–804, 1956.

[117] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *ACM SIGMOD Record*, vol. 29, no. 2. ACM, 2000, pp. 427–438.

[118] G. H. Orair, C. H. Teixeira, W. Meira Jr, Y. Wang, and S. Parthasarathy, "Distance-based outlier detection: Consolidation and renewed bearing," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1469–1480, 2010.

[119] C. M. Bishop, *Pattern recognition and machine learning.* springer, 2006.

[120] W. Chimphlee, A. H. Abdullah, M. Sap, and M. Noor, "Unsupervised anomaly detection with unlabeled data using clustering," 2005.

[121] L. Han, Z. Nan, and B. Lihui, "Using an improved clustering method to detect anomaly activities," *Wuhan University Journal of Natural Sciences*, vol. 11, no. 6, pp. 1814–1818, 2006.

[122] F. Weng, Q. Jiang, L. Shi, and N. Wu, "An intrusion detection system based on the clustering ensemble," in *Anti-counterfeiting, Security, Identification, 2007 IEEE International Workshop on.* IEEE, 2007, pp. 121–124.

[123] Y. Guan, A.-A. Ghorbani, and N. Belacel, "Y-means: A clustering method for intrusion detection," 2003.

[124] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Clustering-based network intrusion detection," *International Journal of reliability, Quality and safety Engineering*, vol. 14, no. 02, pp. 169–187, 2007.

[125] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 2, pp. 224–227, 1979.

[126] M. Morita, R. Sabourin, F. Bortolozzi, and C. Y. Suen, "Unsupervised feature selection using multi-objective genetic algorithms for handwritten word recognition," in *2013 12th International Conference on Document Analysis and Recognition*, vol. 2. IEEE Computer Society, 2003, pp. 666–666.

[127] J. C. Dunn†, "Well-separated clusters and optimal fuzzy partitions," *Journal of cybernetics*, vol. 4, no. 1, pp. 95–104, 1974.

[128] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[129] F. Kovács, C. Legány, and A. Babos, "Cluster validity measurement techniques," in *6th International symposium of hungarian researchers on computational intelligence.* Citeseer, 2005.

[130] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[131] D. Beasley, R. Martin, and D. Bull, “An overview of genetic algorithms: Part 1. fundamentals,” *University computing*, vol. 15, pp. 58–58, 1993.

[132] J. H. Holland, “Genetic algorithms,” *Scientific american*, vol. 267, no. 1, pp. 66–72, 1992.

[133] D. E. Goldberg, K. Deb, and J. H. Clark, “Genetic algorithms, noise, and the sizing of populations,” *Complex systems*, vol. 6, pp. 333–362, 1991.

[134] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[135] K. Deb, “Multi-objective optimisation using evolutionary algorithms: an introduction,” in *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011, pp. 3–34.

[136] K. Deb, *Multi-objective optimization using evolutionary algorithms.* John Wiley & Sons, 2001, vol. 16.

[137] N. Srinivas and K. Deb, “Muiltiobjective optimization using nondominated sorting in genetic algorithms,” *Evolutionary computation*, vol. 2, no. 3, pp. 221–248, 1994.