

2015

Intelligent network intrusion detection using an evolutionary computation approach

Samaneh Rastegari
Edith Cowan University

Follow this and additional works at: <https://ro.ecu.edu.au/theses>



Part of the [Digital Communications and Networking Commons](#), and the [Information Security Commons](#)

Recommended Citation

Rastegari, S. (2015). *Intelligent network intrusion detection using an evolutionary computation approach*.
<https://ro.ecu.edu.au/theses/1760>

This Thesis is posted at Research Online.
<https://ro.ecu.edu.au/theses/1760>

2015

Intelligent network intrusion detection using an evolutionary computation approach

Samaneh Rastegari
Edith Cowan University

Recommended Citation

Rastegari, S. (2015). *Intelligent network intrusion detection using an evolutionary computation approach*. Retrieved from <http://ro.ecu.edu.au/theses/1760>

This Thesis is posted at Research Online.
<http://ro.ecu.edu.au/theses/1760>

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

EDITH COWAN UNIVERSITY (ECU)

DOCTORAL THESIS

Intelligent Network Intrusion Detection Using an Evolutionary Computation Approach

Author:

Samaneh Rastegari

Supervisors:

Associate Professor Philip Hingston

Associate Professor C. Peng Lam

*A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy*

in the

School of Computer and Security Science
Faculty of Health, Engineering and Science

December 2015

Declaration of Authorship

I, Samaneh Rastegari, declare that this thesis titled, 'Intelligent Network Intrusion Detection Using an Evolutionary Computation Approach' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: *Samaneh Rastegari*

Date: *08/12/2015*

EDITH COWAN UNIVERSITY

Abstract

Faculty of Health, Engineering and Science
School of Computer and Security Science

Doctor of Philosophy

Intelligent Network Intrusion Detection Using an Evolutionary Computation Approach

by Samaneh RASTEGARI

With the enormous growth of users' reliance on the Internet, the need for secure and reliable computer networks also increases. Availability of effective automatic tools for carrying out different types of network attacks raises the need for effective intrusion detection systems.

Generally, a comprehensive defence mechanism consists of three phases, namely, preparation, detection and reaction. In the preparation phase, network administrators aim to find and fix security vulnerabilities (e.g., insecure protocol and vulnerable computer systems or firewalls), that can be exploited to launch attacks. Although the preparation phase increases the level of security in a network, this will never completely remove the threat of network attacks. A good security mechanism requires an Intrusion Detection System (IDS) in order to monitor security breaches when the prevention schemes in the preparation phase are bypassed. To be able to react to network attacks as fast as possible, an automatic detection system is of paramount importance. The later an attack is detected, the less time network administrators have to update their signatures and reconfigure their detection and remediation systems. An IDS is a tool for monitoring the system with the aim of detecting and alerting intrusive activities in networks. These tools are classified into two major categories of signature-based and anomaly-based. A signature-based IDS stores the signature of known attacks in a database and discovers occurrences of attacks by monitoring and comparing each communication in the network against the database of signatures. On the other hand, mechanisms that deploy anomaly detection have a model of normal behaviour of system and any significant deviation from this model is reported as anomaly.

This thesis aims at addressing the major issues in the process of developing signature based IDSs. These are: i) their dependency on experts to create signatures, ii) the complexity of their models, iii) the inflexibility of their models, and iv) their inability to adapt to the changes in the real environment and detect new attacks. To meet the requirements of a good IDS, computational intelligence methods have attracted considerable interest from the research community.

This thesis explores a solution to automatically generate compact rulesets for network intrusion detection utilising evolutionary computation techniques. The proposed framework is called ESR-NID (Evolving Statistical Rulesets for Network Intrusion Detection). Using an interval-based structure, this method can be deployed for any continuous-valued input data. Therefore, by choosing appropriate statistical measures (i.e. continuous-valued features) of network traffic as the input to ESR-NID, it can effectively detect varied types of attacks since it is not dependent on the signatures of network packets.

In ESR-NID, several innovations in the genetic algorithm were developed to keep the ruleset small. A two-stage evaluation component in the evolutionary process takes the cooperation of rules into consideration and results into very compact, easily understood rulesets. The effectiveness of this approach is evaluated against several sources of data for both detection of normal and abnormal behaviour. The results are found to be comparable to those achieved using other machine learning methods from both categories of GA-based and non-GA-based methods. One of the significant advantages of ESR-NIS is that it can be tailored to specific problem domains and the characteristics of the dataset by the use of different fitness and performance functions. This makes the system a more flexible model compared to other learning techniques. Additionally, an IDS must adapt itself to the changing environment with the least amount of configurations. ESR-NID uses an incremental learning approach as new flow of traffic become available. The incremental learning approach benefits from less required storage because it only keeps the generated rules in its database. This is in contrast to the infinitely growing size of repository of raw training data required for traditional learning.

Keywords: Network security, intrusion detection, evolutionary computation, classification, genetic based machine learning, supervised learning, rule based algorithms.

List of Publications

Journal Paper:

Rastegari, S., Hingston, P., & Lam, C. P. (2015). Evolving statistical rulesets for network intrusion detection. *Applied Soft Computing*, 33, 348-359.

Conference Paper:

Rastegari, S., Lam, C. P., & Hingston, P., (recently accepted). A Statistical Rule Learning Approach to Network Intrusion Detection. In *International Conference on IT Convergence and Security (ICITCS)*., 2015. IEEE.

Related Paper:

Rastegari, S., Hingston, P., Lam, C. P., & Brand, M. (2013, April). Testing a distributed denial of service defence mechanism using red teaming. In *Computational Intelligence for Security and Defense Applications (CISDA), 2013 IEEE Symposium on* (pp. 23-29). IEEE.

Acknowledgements

I am deeply thankful to my principal supervisor, Associate Professor Philip Hingston, for being a great inspiration throughout my candidature. His great advice and admirable patience helped me to get through the difficult stages of my thesis.

My deep appreciation goes to my co-supervisor, Associate Professor C. Peng Lam, for her useful suggestions and close engagement throughout my research. Her constant support contributed a lot to completion of this thesis.

I am forever grateful to my loving parents, Reza and Shamsi, my kind brothers, Masoud and Yousef, and my lovely sister, Sima, for their unconditional love and support. My parents have given me the best of everything in this world by their endless sacrifices for my success. I would also like to thank my husband, Mohsen, who was extremely supportive and patient while I went through an important chapter of my life. He was always there to listen to every little worry I had and helped me deal with the challenges in my research with his love and sense of humour. I certainly cannot thank him enough.

I would also like to thank my PhD labmates in the School of Computer and Security Science and my friends in Australia for their encouragement, help and support. Special thanks go to Dr Majid Rad, Mahtab, Soheila, Samad, Zahra, Ali, Ghazaleh, Iman, Samira, Mousa, Arman, Hossein, Negar, and Amir for their warm support.

Last but not least, I am also thankful to Edith Cowan University (ECU) for awarding me the postgraduate research scholarship during my PhD candidature.

Contents

Declaration of Authorship	ii
Abstract	iii
List of Publications	v
Acknowledgements	vi
Contents	vii
List of Figures	xi
List of Tables	xiii
Abbreviations	xvii
1 Introduction	1
1.1 The Role of Intrusion Detection	3
1.1.1 Essential Requirements of an IDS	4
1.2 Research Motivation and Objectives	5
1.3 Thesis Contributions	7
1.4 Thesis Structure	10
2 Background and Related Work	13
2.1 Introduction	13
2.2 Overview of Network Intrusions	13
2.3 Overview of Intrusion Detection Systems	16
2.3.1 Selection Criteria for Intrusion Detection Systems	19
2.4 Machine Learning	22
2.4.1 Non-evolutionary Rule Learning Algorithms	23
2.4.1.1 Repeated Incremental Pruning to Produce Error Reduction (RIPPER)	25

2.4.1.2	Decision Trees	25
2.4.2	k-Nearest Neighbour (kNN)	27
2.4.3	Evolutionary Rule Learning Algorithms	28
2.4.3.1	XCS	31
2.4.3.2	GASSIST	33
2.4.3.3	MPLCS	33
2.5	Rule-based nature-inspired approaches for IDSs	36
2.5.1	Artificial Immune System	36
2.5.2	GA-based Techniques	40
2.5.2.1	Adapting to Environment Changes	46
2.6	Summary of Challenges in GA-based IDSs	47
2.7	Summary	49
3	ESR-NID: A Framework for Evolving Statistical Rulesets for Network Intrusion Detection	51
3.1	Introduction	51
3.1.1	Pre-processing Stage	53
3.1.1.1	Input data	54
3.1.1.2	Data Normalization	54
3.1.2	Evolutionary Process	54
3.1.2.1	Michigan Versus Pittsburgh Approach	55
3.1.2.2	Individual Representation	56
3.1.2.3	Seed Selection and Initializer	58
3.1.2.4	Fitness Function and Performance Function	60
3.1.2.5	Reproduction of A New Generation	75
3.1.3	Post-processing	77
3.2	Summary	78
4	Performance Evaluation of ESR-NID on Synthetic Problems	81
4.1	Introduction	81
4.2	Choice of Dataset	82
4.2.1	A 3-Dimensional Medium Size Problem	83
4.2.2	A 3-Dimensional Problem with More Clusters of Hits	83
4.2.3	Problems With More Input Features	85
4.3	Evaluation Strategy Against Synthetic Datasets	87
4.4	Preliminary Experiments for Performance Evaluation	87
4.4.1	Parameter Settings	87
4.4.2	System Performance	88
4.4.3	Adaptive Elitism Mechanism	90
4.5	Algorithm Tuning	94
4.6	Parameter Tuning for Other Classifiers	98
4.7	Performance Comparison	100
4.8	Performance Evaluation of ESR-NID against a 3-Dimensional Problem with More Clusters of Hits	103

4.9	Performance Evaluation of ESR-NID against a Problem with More Input Features	109
4.10	Using a Different Performance Function	110
4.10.1	Imbalanced Dataset	114
4.11	Summary	116
5	Performance Evaluation of ESR-NID on Network Intrusion Detection Problems	117
5.1	Introduction	117
5.2	Choice of Dataset	118
5.2.1	NSL-KDD dataset	122
5.2.2	Combined DARPA/CAIDA dataset	124
5.2.3	Features	125
5.3	Experiments for Network Intrusion Detection	128
5.3.1	Parameter Settings	129
5.3.2	Experiments with NSL-KDD Dataset	129
5.3.3	Experiments with DARPA/CAIDA Dataset	134
5.4	ESR-NID for Detecting Normal Instances	135
5.5	Summary	138
6	Adaptation in a Dynamic Environment	141
6.1	Introduction	141
6.2	Incremental Learning for ESR-NID	142
6.3	Evaluation Strategy	144
6.3.1	Experiments with Synthetic Datasets	147
6.3.1.1	Parameter Settings	148
6.3.1.2	Scenario 1	149
6.3.1.3	Scenario 2	151
6.3.1.4	Scenario 3	153
6.3.2	Experiments with NSL-KDD Dataset	155
6.4	Summary	157
7	Conclusions	163
7.1	Summary of Contributions	164
7.2	Limitations	166
7.3	Future Work	167

List of Figures

1.1	Increases on the number of devices connected to the Internet over years.	1
1.2	Organization of a generalized intrusion detection system (Wu & Banzhaf, 2010).	3
2.1	Components of a comprehensive defence mechanism.	16
2.2	Layered AIS Framework (De Castro & Timmis, 2002).	37
2.3	Encoded Form of a Rule (Gong et al., 2005).	44
3.1	General view of supervised machine learning algorithms.	52
3.2	The architecture of the ESR-NID system.	53
3.3	The structure of a chromosome in the proposed system. LB_i , UB_i are real values between 0 and 1 and represent lower bound and upper bound of feature i and on/off component is a binary value used to activate or deactivate feature i in the chromosome. Thus, the system can produce variable length rules.	57
3.4	An example of a binary classification problem. One detection rule is covering a set of input instances.	61
3.5	A ruleset with three rules (corresponding to regions marked in green, yellow and purple respectively) for detection of abnormal instances.	63
3.6	Weighting on the false positives in the proposed fitness functions (fitness function (2) and (3)) to take care of the problem of the class imbalance in a dataset.	65
3.7	A sample of sorted population based on the fitness value in generation0.	72
3.8	The breeding procedure (Luke et al., 2006).	76
3.9	Two point crossover	77
3.10	Post-processing of final optimised ruleset.	79
4.1	A medium-size problem. Red (darker) data points represent “anomalous” records, and green (lighter) points are “normal”.	84
4.2	A problem to test the performance of ESR-NID in detection of more clusters of hits. This problem includes two “normal” clusters and five “anomalous” clusters.	84
4.3	Performance of ESR-NID on the problem shown in Figure 4.1 vs generations for only one fold.	89
4.4	Fixed elitism approach (population-size=20 and elite=25%).	90
4.5	Performance of the enhanced ESR-NID on the problem shown in Figure 4.1 vs generations for only one fold.	92

4.6	Adaptive elitism approach. Individuals are sorted in descending order of fitness values. $\text{Performance}(\text{Ind0}, \text{Ind2}, \text{Ind3}, \text{Ind4}) > \text{Performance}(\text{Ind0}, \text{Ind1}, \text{Ind2}, \text{Ind3}, \text{Ind4})$.	93
4.7	Comparing the performance of ESR-NID using fixed elitism and adaptive elitism mechanisms (the results are for a total of 30 runs).	94
4.8	Performance of ESR-NID using fitness function (1). Left y-axis shows the performance value and right y-axis presents the number of rules used for classification. On x-axis, different combinations of mutation probabilities and population sizes for ESR-NID are depicted. As explained before, when the population size increases, the number of generations will be decreased to keep the total number of evaluations the same in all the experiments.	96
4.9	Performance of ESR-NID using fitness function (2) and different combinations of mutation probabilities and population sizes.	97
4.10	Performance of ESR-NID using fitness function (3) and different combinations of mutation probabilities and population sizes.	97
4.11	Performance of ESR-NID using different fitness functions.	98
4.12	Performance of ESR-NID using the best configuration found compared to J48, kNN, JRip, GASSIST-ADI and MPLCS approaches.	101
4.13	Performance of different techniques on the problem shown in Figure 4.2. Six rules are produced by ESR-NID, J48 and JRip for classification of anomalous records.	104
4.14	The final ruleset evolved by ESR-NID for classifying the anomalous records in the problem shown in Figure 4.2. f1, f2 and f3 are the three features in the dataset generated by the rules in Section 4.2.1.	106
4.15	Comparing the output of J48 to the ruleset generated by ESR-NID.	108
4.16	Performance of different techniques on a problem with 6 features.	110
4.17	Performance of different techniques on a problem with 12 features.	111
4.18	Performance of different techniques on a problem with 12 features.	113
4.19	Accuracy of different techniques on a problem with 12 features.	113
5.1	Normal traffic scenario from MIT Lincoln Laboratory.	124
5.2	Low-rate DDoS attack scenario from CAIDA.	125
5.3	Entropy for a brief distributed denial of service attack (Feinstein et al., 2003)	126
6.1	Incremental learning used in ESR-NID.	143
6.2	Evaluation strategy used in the experiments in this chapter.	145
6.3	Performance of different learning approaches in the first scenario.	150
6.4	Performance of different learning approaches in the second scenario.	153
6.5	Performance of different learning approaches in the third scenario.	155
6.6	Performance of different learning approaches against the NSL-KDD dataset.	157

List of Tables

2.1	Pros and cons of signature and anomaly based IDSs.	17
2.2	Factors affecting the selection process of defence mechanisms	20
2.3	A summary of the reviewed GA-based rule learning techniques.	35
3.1	A comparison of fitness functions for different scenarios.	67
4.1	Parameters of the J48 machine learning algorithm used in optimization.	99
4.2	Parameters of the kNN machine learning algorithm used in optimization.	99
4.3	Parameters of the JRip machine learning algorithm used in optimization.	100
4.4	Parameters of GASSIST-ADI and MPLCS used in optimization.	100
4.5	Details of the results presented in Figure 4.12.	101
4.6	Comparing the final ruleset generated by ESR-NID, J48, JRip, GASSIST-ADI and MPLCS.	102
4.7	The performance of different techniques on the problem shown in Figure 4.2.	104
4.8	Comparing the final ruleset generated by ESR-NID, J48, JRip, GASSIST-ADI and MPLCS.	105
4.9	Performance of different techniques on a problem with 6 features.	110
4.10	Performance of different techniques on a problem with 12 features.	111
4.11	Confusion matrix for a two-class problem.	112
4.12	Performance of different techniques on a problem with 12 features.	112
4.13	Accuracy of different techniques on a problem with 12 features.	114
4.14	Performance of two configurations of ESR-NID on an imbalanced problem with 12 features.	115
4.15	Accuracy of two configurations of ESR-NID on an imbalanced problem with 12 features.	115
4.16	Average performance of ESR-NID against an imbalanced problem with 12 input features.	115
4.17	Average performance of customised ESR-NID against an imbalanced problem with 12 input features.	115
5.1	Attack types in KDD99 dataset and their categorisation.	120
5.2	Basic features in KDD99 dataset.	120
5.3	Traffic features using two-second time windows in KDD99 dataset.	121
5.4	Traffic features using windows of 100 connections in KDD99 dataset.	121

5.5	Connection-based content features based on domain knowledge in KDD99 dataset.	122
5.6	Statistics of redundant records in the KDD99 training set (Tavallae et al., 2009).	123
5.7	Statistics of redundant records in the KDD99 test set (Tavallae et al., 2009).	123
5.8	Traffic features and details of CAIDA DDoS attack scenario (Moore et al., 2006; Xiang et al., 2011).	125
5.9	KDD99 continuous features.	127
5.10	Eight features extracted using CFS and BestFirst.	130
5.11	Fifteen features extracted using CSE and GreedyStepwise.	131
5.12	Comparing the performance of ESR-NID on the NSL-KDD dataset with 8 features with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.	131
5.13	A ruleset generated by ESR-NID for the NSL-KDD dataset with 8 features.	132
5.14	A ruleset generated by GASSIST-ADI for the NSL-KDD dataset with 8 features.	132
5.15	A ruleset generated by MPLCS for the NSL-KDD dataset with 8 features.	132
5.16	Comparing the performance of ESR-NID on the NSL-KDD dataset with 15 features with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.	133
5.17	Comparing the performance of ESR-NID on the NSL-KDD dataset with all 32 continuous features with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.	133
5.18	Comparing the performance of ESR-NID on the combined DARPA/-CAIDA dataset with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.	135
5.19	Comparing the final ruleset generated by ESR-NID, J48, JRip, GASSIST-ADI and MPLCS.	135
5.20	Performance of ESR-NID for detecting normal instances.	137
5.21	Performance of customised ESR-NID as an anomaly detection system.	137
6.1	Average results for startup and update phases in the first scenario.	150
6.2	Comparing the final rulesets generated by ESR-NID during startup and update phases in the first scenario.	151
6.3	Average results for startup and update phases in the second scenario.	153
6.4	Comparing the final rulesets generated by ESR-NID during startup and update phases in the second scenario.	154
6.5	Average results for startup and update phases in the third scenario.	155
6.6	Comparing the final rulesets generated by ESR-NID during startup and update phases in the third scenario.	156
6.7	Average results for startup and update phases against the NSL-KDD dataset.	157

6.8	Comparing the final rulesets generated by ESR-NID during startup and update phases for the NSL-KDD dataset.	158
-----	--	-----

Abbreviations

AI	A rtificial I ntelligence
AISs	A rtificial I mmune S ystems
ANNs	A rtificial N eural N etworks
CAIDA	C enter for A ppplied I nternet D ata A nalysis
CFS	C orrelation-based F eature S election
CSE	C onsistency S ubset E valuator
CVE	C ommon V ulnerabilities and E xposure
DARPA	D efense A dvanced R esearch P rojects A gency
DDoS	D istributed D enial of S ervice
DMZ	D e M ilitarized Z one
EC	E volutionary C omputation
EC	E volutionary A lgorithm
ESR-NID	E volving S tatistical R ulesets for N etwork I ntrusion D etection
FL	F uzzy L ogic
GA	G enetic A lgorithms
GASSIST	G enetic A lgorithms based cla S Sifier sy S Tem
GBML	G enetic-based M achine L earning
GP	G enetic P rogramming
GR	G ain R atio
HIDS	H ost-based I ntrusion D etection S ystem
ICMP	I nternet C ontrol M essage P rotocol
IDS	I ntrusion D etection S ystem
IP	I nternet P rotocol
IG	I nformation G ain

KDD	K nowledge D iscovery in D atabases
kNN	k -Nearest N eighbors
LCS	L earning C lassifier S ystem
MPLCS	M emetic P ittsburgh C lassifier S ystem
MOEA	M ulti- O bjective E volutionary A lgorithm
NIDS	N etwork-based I ntrusion D etection S ystem
RIPPER	R epeated I ncremental P runing to P roduce E rror R eduction
ROC	R eciever O perating C haracteristic
SI	S warm I ntelligence
SIP	S ession I nitiation P rotocol
TCP	T ransmission C ontrol P rotocol
TN	T rue N egative
TN	T rue P ositive
UCP	U ser D atagram P rotocol
UCS	S Upervised C lassifier S ystem
WEKA	W aikato E nvironment for K nowledge A nalysis
XCS	E Xtended C lassifier S ystem

*To my mother, my brothers, my sister and to my
soulemate, Mohsen...*

*To my father, who did not live long enough to witness
this moment...*

Chapter 1

Introduction

Computers and networks have been under threat from viruses, worms and attacks from hackers since they were first used. In 2008, the number of devices connected to the Internet exceeded the number of human beings and this increasing trend will see about 50 billion devices by 2020 (see Figure 1.1) ([Evans, 2011](#)). Securing these devices and the data passing between them is a challenging task because the number of intrusions is also increasing sharply year by year.

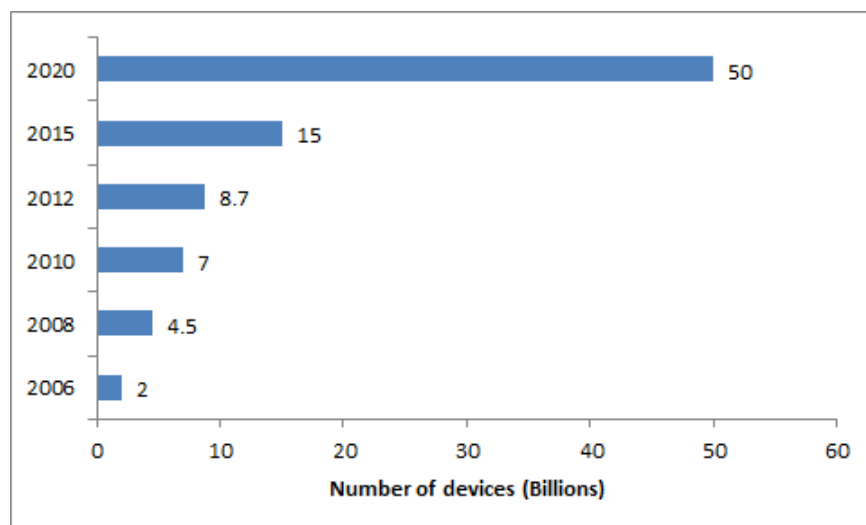


FIGURE 1.1: Increases on the number of devices connected to the Internet over years.

To address this issue, a large number of defences against network attacks have been proposed in the literature. Despite all the efforts made by researchers in the community over the last two decades, the network security problem is not completely solved. One reason for that is the rapid growth in computational power and available resources to attackers, which enables them to launch complex attacks ([Wu et al., 2010](#)). This can be considered a two-player game, where an attacker attempts to find the most effective strategy to disrupt normal operations in a network and the defender's challenge is to determine optimal defensive solutions and block illegitimate access to the network.

In general, defence against network attacks consists of preparation, detection and reaction phases. A risk analysis process is usually conducted by security engineers during the preparation phase to understand the environment and the assets they are trying to protect in that environment. This process is very crucial because it helps the engineers to understand how attacks can take place and how they affect the network ([Santos, 2007](#)). The preparation phase also includes identification of infrastructure vulnerabilities, development of security strategies and plans and installation of required security devices based upon analysis of the information gathered ([Hamdi & Boudriga, 2005](#); [Mölsä, 2005](#)). Another key element of network security is a detection system. An intrusion detection system (IDS) usually complements a firewall to form an effective cyber security solution. One security motto is “Prevention is ideal but detection is a must” ([Cole, 2011](#)). Fast detection of attacks is required to be able to react rapidly. Thus, an automatic detection phase is of paramount importance. Finally, handling detected intrusions in a network is done during the reaction phase. A traffic blocking method is an example of a mitigation mechanism used in the reaction phase.

One of the main challenges in securing networks is the appropriate design and use of an intrusion detection system, that can monitor network traffic and effectively identify network intrusions. The role of this key element in an effective defence system will be explained in the next section.

1.1 The Role of Intrusion Detection

A large number of defence approaches have been proposed in the literature to provide different functions in various environments. The core element of a good defence system is an Intrusion Detection System (IDS), which provides proper attack detection before any reaction. An IDS aims to detect intrusions before they seriously damage the network. The term *intrusion* refers to any unauthorised attempt to access the elements of a network with the aim of making the system unreliable. Figure 1.2 depicts the organization of a generalized IDS. The solid lines show the data/control flow and the dashed lines indicate the responses to the intrusions.

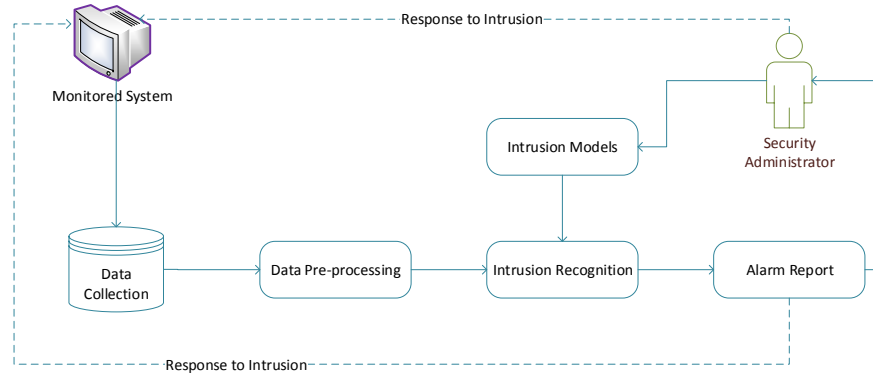


FIGURE 1.2: Organization of a generalized intrusion detection system (Wu & Banzhaf, 2010).

Intrusion detection systems are generally categorised into signature based or anomaly based. Signature based IDSs use a database of rules or so called *signatures* to classify network connections, whereas anomaly based IDSs create a normal user profile and identify anything that does not match this profile as an attack. In the former, known intrusions can be detected efficiently with a low false alarm rate. Thus, in most of the commercial systems, this approach has been widely used.

One of the major challenges in the signature based intrusion detection is the detection of new intrusions. To address this issue, the database of rules should be regularly updated with a manual or automatic process. A manual process is usually done by a network administrator by finding new signatures and adding them to the database of rules, while an automatic process can be done with the help of

supervised learning algorithms. Another way to address this problem is to switch to an anomaly detection system, which has the capability of detecting new types of attacks. However, the major problem in anomaly detection systems is discovering the boundaries between normal and anomalous behaviour. In addition to that, as the normal traffic pattern is also changing over time, an anomaly detection system needs to adapt itself to this changing behaviour.

Nowadays, due to the exponential growth of technologies and the increased number of available hacking tools, in both approaches, adaptability should be considered as a key requirement. To be able to react to network intrusions as fast as possible, an automatic or at least semi-automatic detection phase is required. This will decrease the amount of damage to legitimate users because an early detection system supplies more time for a proper reaction.

1.1.1 Essential Requirements of an IDS

The main features that an IDS should have are effectiveness, adaptability and extensibility (Lee, 1999). Effectiveness refers to the ability of an IDS in detection of slight variations of known attacks and extensibility means that an IDS can be customised easily for different environments.

A majority of proposed IDSs in the past are based on signature detection (Gómez et al., 2013) because of their effectiveness in detection of known attacks. Two examples of widely used open-source signature-based tools are Snort (Roesch, 1999) and ClamAV (Kojm, 2004). Both have their own database of signatures (more than 4000 rules in Snort database and over 800,000 in ClamAV repository). However, these systems raise two important issues for further research: 1) a human is responsible for generation of signatures and updating the database and as a result the cost of developing and maintaining this database is an important issue. They also suffer from the time lag between facing new attacks and manually updating the signatures. and 2) they do not work against new and unknown attacks.

To automate the process of signature generation, the use of computational algorithms is a promising method with adaptation, fault tolerance, high computational speed and error resilience in the face of noisy information characteristics (Wu & Banzhaf, 2010). Additionally, to update the database of signatures for a dynamic environment, an incremental learning approach can be used to repeatedly learn new attacks.

To summarise, the laborious manual process of signature creation for IDSs, non-adaptability of them to a dynamically changing environment and considering extensibility as an important feature for the design of IDSs introduce a new direction of research and the motivation for this thesis.

1.2 Research Motivation and Objectives

As the number of malicious computer users and their intrusive behaviour increase, the task of designing comprehensive network protection systems become more and more difficult. This thesis focuses on the majority class of IDSs, which is based on signature detection and proposes a method for automatically generating signatures for network intrusion detection. The main disadvantage of signature-based detection systems is the impossibility of detecting new intrusions because they only look for patterns that match the signatures (or rules) stored in their databases. These systems need frequent updates to keep the database of signatures current. Creation, test and distribution of these signatures are often carried out by human experts. This involves examining and analysing the malicious traffic for extracting information needed for signature-based detectors. After creation of signatures, they should be tested against some captured network data and if they perform well, they will be added to the repository of signatures. To facilitate the signature creation process, a computational intelligence method can be used in a supervised learning mode in building an automatic rule-based IDS. More specifically in this thesis, a Genetic-based Machine Learning (GBML) technique is proposed and its application to IDS

problems is evaluated. The proposed approach uses a genetic algorithm (GA) to generate an optimal set of rules for intrusion detection.

Although the application of nature inspired rule based systems, such as genetic algorithms, for classification tasks is a promising line of work, there are some challenges that need to be addressed, such as scalability (how do they deal with datasets with a high number of attributes?), flexibility (can they deal with both balanced and imbalanced (i.e. a problem where the total number of a class of data is far less than the total number of another class of data) datasets?), understandability of the model (how complex is the final ruleset?), etc. These challenges are discussed below.

One of the challenges in evolutionary rule learning techniques is dealing with datasets with a large number of attributes. Standard feature selection strategies are often used to find relevant attributes and reduce the search space. However, these methods are only able to initially filter the dataset for the use of a classification method and thus the classifier is forced to use the same subset of features for every rule (signature) it generates. Therefore, the same subset of attributes are seen for the whole solution (i.e. ruleset). For example, if the feature selection method chose five of the 10 received features as the relevant attributes for the classifier, then the final ruleset generated by the classifier includes rules that are all the same size (each rule comprises five features). Recently, some other approaches have been proposed to integrate feature selection and learning in a concurrent manner ([Bacardit & Krasnogor, 2009a](#)). These fine grained feature selection approaches are more flexible and produce rules with different sets of attributes. This also reduces complexity of the final ruleset generated for the classification task by giving the flexibility to learning system to only use and keep relevant attributes during the evolutionary course. Therefore, two advantages of an integrated feature selection and learning approach are 1) a more efficient learning process and 2) a more compact and hence understandable final solution.

In many real world situations (e.g., intrusion detection, risk management and medical applications), cases where one class (usually the positive class) represents only

a very small fraction of the entire dataset can often be seen. For example, in network intrusion detection, the number of intrusive instances is typically a very small fraction of the total network traffic records. Similarly, for classifying the cancerous pixels from normal ones in mammogram images, the cancerous instances represent only a very small fraction of the entire image (Chawla et al., 2003). Dealing with these cases is considered one of the emergent challenges in data mining (Yang & Wu, 2006). Since the minority class is considered the target class in the learning and prediction process, the cost associated with misclassification of even one example of that class should be higher than that of the other class. Therefore, a rule learning system's design should be flexible enough to adapt to different types of problems and datasets.

Another important challenge for nature inspired rule based systems is the complexity of the final ruleset. In most domains, experts are interested in human-understandable models. A common metric used in the literature for measuring complexity is the number of rules. To avoid generating too many rules and to minimize overlapping regions covered by rules, cooperation between rules should be considered when evolving rules.

Therefore, this thesis aims at designing a genetic-based rule learning classifier, which can automatically generate a concise set of easy to understand rules for intrusion detection. This system decreases human effort in creating and maintaining rules for rule-based systems. Another goal of this thesis is to make the detection system adaptable to the changes in the real environment using an incremental learning approach. The contributions of the proposed approach to both the GBML and network intrusion detection fields are explained in the following section.

1.3 Thesis Contributions

The main contributions of this thesis are:

- Automatic rule creation for intrusion detection:

The proposed nature inspired approach in this thesis is able to automatically generate signatures (rules) to detect anomalous traffic. This is accomplished by implementing a genetic algorithm with new features, which are explained later, to autonomously derive a set of rules from network data. This facilitates and speeds up the process of rule extraction from network audit data, which has been traditionally carried out by human experts for signature based IDSs.

- Use of statistical features for detector rules:

The types of rules generated in this thesis are statistical based as it is believed that the generated packets by today's attack tools with improved packet crafting characteristics will most likely distort statistical measurements of the composition of network traffic. For example, flooding attacks produce huge amount of normal packets to disrupt normal victim's services. To be able to detect this kind of network intrusion, statistical methods can be used. Aiming for generation of statistical based rules, the input features to the proposed intrusion detection system are real-valued attributes. Accordingly, the system is named ESR-NID (Evolving Statistical Rulesets for Network Intrusion Detection). Examples of these statistical measures are number of data bytes from source to destination, number of data bytes from destination to source, Entropy of source IP address, Entropy of source port number, Entropy of destination port number, Entropy of packet type and Entropy of packet size. The rules generated by ESR-NID are not dependent on the categorical features extracted from packets and can be used for detection of a wide range of intrusions. Most of the existing signature based IDSs lack the ability to detect slight variations of known attacks because they look for exact matches against network packets. Example of these features are source IP address, source port number, protocol and destination port number.

- Development and analysis of a flexible genetic-based rule learning technique:
One of the main contributions of the thesis is the development of an effective genetic algorithm with new features to generate accurate and compact rulesets

for network intrusion detection. The proposed model, ESR-NID, has two main features:

1. An advanced two-stage evaluation approach:

ESR-NID uses two functions to find the best cooperating rules in each generation of an evolutionary run. These are a well-defined fitness function, which considers the cooperation of rules to eliminate redundancy and minimize the overlap between the rules and a performance function, which decides on the best ruleset in a cooperative manner and provides flexibility in dealing with different problems, including imbalanced problems. ESR-NID is also flexible enough to accommodate alternative fitness and performance functions based on the designer's needs and preferences. To show this, a new problem is defined for exploring the use of ESR-NID for detecting normal instances instead of intrusions. The aim is to design a detection system that matches maximum number of normal instances. This can be achieved by changing the performance function in ESR-NID model. Accordingly, a set of experiments is conducted to show the flexibility aspect of ESR-NID.

2. An adaptive elitism mechanism:

To avoid the laborious task of finding the best elite value through a trial-and-error method and also losing good cooperative rules over the generations, an adaptive approach is proposed for ESR-NID to adjust the number of elites copied into each new generation. This enhances the learning process by keeping the best performed rules from one generation to the next.

Analysis of the proposed method: The performance of ESR-NID is evaluated against three sources of data: synthetic datasets, NSL-KDD dataset (a new improved version of the KDD99) and a combined DARPA/CAIDA dataset. Preliminary evaluations of the method are carried out against the first set of data as the ground truth is known for synthetic problems. Four different scenarios are designed for generation of synthetic datasets. The other two sets of data are used for evaluating the ESR-NID within the context of a

real-world problem (network intrusion detection). The outcomes of ESR-NID are compared against five machine learning methods from two categories of GA-based and non-GA-based algorithms: J48, kNN and JRip (non-GA-based) and GASSIST-ADI and MPLCS (GA-based).

- Incremental learning:

Finally, a methodology is developed to make ESR-NID adaptable to environment changes. Using this approach, ESR-NID is able to frequently update its database of rules to detect new attacks. The adaptability of ESR-NID is evaluated against synthetic datasets and NSL-KDD dataset.

1.4 Thesis Structure

The remainder of this thesis is structured as follows: In chapter 2, the context of this thesis is set by providing a brief introduction to the intrusion detection problem and the existing detection systems. After describing the important factors that should be considered in the design and implementation of intrusion detection systems, an overview of capabilities of machine learning techniques and their application to rule learning is provided. Additionally, a review of some of the recent and most related work to the proposed approach is provided.

In chapter 3, the design and implementation of the proposed approach, ESR-NID, for generating optimised rulesets for network intrusion detection is presented. The contributions of ESR-NID to the genetic based machine learning techniques are also explained in more detail.

Chapter 4 introduces the synthetic datasets generated for evaluation of ESR-NID. Through the first set of experiments, an adaptive elitism mechanism is proposed for enhancing the proposed algorithm. Additionally, an algorithm tuning process is conducted to find the best settings for ESR-NID for all the subsequent experiments.

In chapter 5, ESR-NID is more extensively evaluated for network intrusion detection. This is achieved through a series of experiments against NSL-KDD datasets.

Additionally, by combining the new attack traces in CAIDA DDoS 2007 data with the attack-free DARPA data, another dataset is created for testing the performance of ESR-NID. The flexibility in the use of ESR-NID on a different problem, where the aim is to detect normal instances instead of intrusions, is tested in this chapter as well.

In chapter 6, an incremental learning approach is proposed to make ESR-NID adaptable to a dynamic environment. To evaluate the proposed model, an evaluation strategy is developed to carry out a series of experiments. Synthetic datasets and the NSL-KDD dataset are again used as the sources of data in this chapter.

Finally, chapter 7 summarizes the contributions of this thesis, points out the limitations of the work and discusses the future directions for continuing further research in this area.

Chapter 2

Background and Related Work

2.1 Introduction

This chapter first provides a brief introduction to the network intrusion problem and then an overview of intrusion detection systems (IDSs) and their selection criteria. Following the introductory concepts, a brief background on machine learning techniques with emphasis on rule learning approaches will be covered. The rule learning techniques are broadly categorised into evolutionary and non-evolutionary algorithms. Under each category, a set of selected machine learning techniques are reviewed. Finally, a survey of past research on the use of nature-inspired techniques for rule based intrusion detection systems, that is related to the work presented in this dissertation is provided.

2.2 Overview of Network Intrusions

Nowadays, many variations of network attacks are seen and they are increasingly becoming more varied and sophisticated. To improve computer and network security, knowing the existing attack types and categorising them based on their patterns is very useful. A standard taxonomy of attacks will decrease confusion between

organisations and researchers who are dealing with computer and network attacks on a regular basis to provide effective methods to combat them.

A comprehensive taxonomy of network and computer attacks that takes into account all parts of the attack (from the vulnerability to the target to the attack itself) can be found in [Hansman & Hunt \(2005\)](#). This taxonomy is designed using the concept of dimensions. There are four dimensions introduced for attack classification. The first dimension is based on the *attack vector* (i.e. the method by which an attack reaches its target). For example, the Melissa virus uses email as its main method of propagation. Therefore, it will be categorised in the category of mass-mailing *worms*. As defining the attack vector for some types of attacks is not easy, the category that best matches the following definitions are chosen in the taxonomy. These are virus (propagates through some form of infected files), Trojans (a benign looking program that serves some malicious purposes), buffer overflows (crashes a process by overflowing its buffer), denial of service attacks (makes services unavailable to legitimate users), network attacks (by manipulating network protocols), physical attacks (by damaging physical components of a network), password attacks (aims at gaining a password) and information gathering attacks (aims at gaining important information). The second dimension covers the target(s) of attacks. There are two main categories in this dimension: hardware and software. Each category can be broken down into multiple sub-classes. Hardware targets include computer components (such as CPUs and hard-disks), network equipment (such as routers and switches) and peripheral devices (such as monitors). On the other hand, software targets are classified into operating systems and applications. The list of potential targets increases each year. The third dimension in this taxonomy covers the vulnerabilities that the attack uses. For this dimension, the CVE (Common Vulnerabilities and Exposure) standard ([CVE, 2015](#)) for vulnerabilities is used. For categorising an attack in this level, one should recognise the vulnerability or vulnerabilities that an attack exploits and assign CVE entries to it. Finally, the fourth dimension takes into account the possibility for an attack to have a payload or effects beyond itself. This dimension consists of five categories: first dimension attack payload, corruption of information, disclosure of information, theft of service and subversion.

There are some other methods in the literature, which are aimed at classifying security threats using various factors. For example, some are only focused on vulnerabilities ([Abbott et al., 1976](#); [Lough, 2001](#)) and some others categorise the attacks based on attacker motivation and objectives, for example, [Howard & Longstaff \(1998\)](#)'s taxonomy.

This research, however, is only concerned with distinguishing intrusive behaviour from legitimate behaviour. Therefore, any set of actions that attempt to compromise the integrity, confidentiality or availability of network resources is considered an attack or network intrusion (this definition of intrusion has been adopted from [Heady et al. \(1990\)](#)). Intrusions can be broadly categorised into two classes: anomalies and misuse intrusions. Anomalies are deviations from normal behaviour while misuses are known patterns of attacks. As misuses are known patterns of attacks, detecting novel attacks is a challenging task for detection systems designed for this category of intrusions. On the other hand, because the behaviour of normal patterns varies in different environments and as a result the system profiles should be updated from time to time, detection of anomaly intrusions is computationally expensive ([Mukkamala et al., 2002](#)). Some examples of intrusions are unauthorized modifications of system files to facilitate illegal access to the system, unauthorized modifications of router tables to deny use of the network and unauthorized use of computing resources ([Bishop, 2003](#)).

As the number of devices connected to the Internet grows and the resources facilitated by bandwidth-providers increase, the adversary-class has less concerns with the network bandwidth constraint and easily produces highly sophisticated network-based attacks ([Baig et al., 2011](#)). Therefore, there is still a need for highly efficient and intelligent defence mechanisms to stand against network threats. In the next section, an overview of intrusion detection systems as the key element of defence mechanisms against network attacks will be presented.

2.3 Overview of Intrusion Detection Systems

A complete security solution against network attacks has three main security components: attack prevention, attack detection, and attack reaction ([Thomas, 2001](#); [Yang et al., 2004](#)). These components are presented in Figure 2.1.

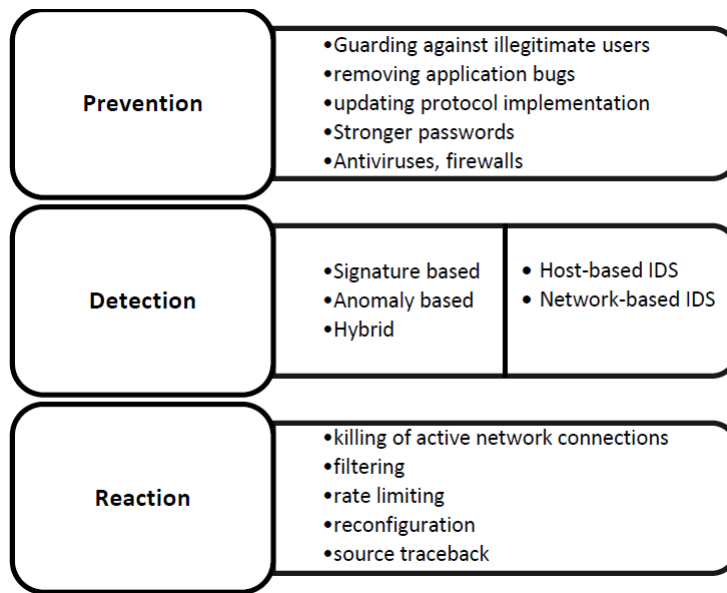


FIGURE 2.1: Components of a comprehensive defence mechanism.

Prevention - The prevention phase aims to increase the overall security of the system by installation of required security devices, removing application bugs, updating protocol implementation, and improving the security of all computers linked to the Internet. In this stage, the network administrator tries to secure the system from illegitimate users by implementing preventive methods. Although it is not possible to prevent all attacks, the goal is to raise the bar for attackers to launch DoS attacks ([Glenn, 2003](#)).

Detection - A good defence system should have a proper attack detection phase before any reaction. The goal of every attack detection mechanism is to detect intrusions before any serious damage. The term intrusion means any unauthorized attempt to access, falsify, change or destroy information in order to make a system unreliable ([Spafford & Zamboni, 2000](#)). A good system can detect attacks in a short

period of time with a low proportion of false positives. Due to the growing number of intrusions, researchers are increasingly trying to develop Intrusion Detection Systems (IDSs). An IDS consists of three major components: sensor, analyser, and user interface (Allen et al., 2000). The key feature of such a system is its ability to provide a view of malicious activities and give out alerts, which inform the network administrators and facilitate the process of reaction.

Based on the analysis methods used, there are two broad groups of IDSs: signature based detection and anomaly based detection. Mechanisms that combine anomaly based detection and signature based detection are known as hybrid detection systems, which update an attack signature database using data about discovered attacks through an anomaly detection phase. Anomaly based detection systems explore intrusions based on deviations from normal behaviours and thus they can detect some new or modified attacks. On the other hand, signature based detection systems can identify an attack if the monitored traffic matches the anomaly patterns (signatures) in their databases. Signature based mechanisms usually work faster and produce less false positives, but they need prior knowledge of intrusions and consequently, in the case of novel attacks, a network system is left vulnerable until the signature database is updated. Table 2.1 shows pros and cons of these detection methodologies (Liao et al., 2013). A hybrid system that can use the advantages of both signature based and anomaly based IDSs provides a more comprehensive protection against a wide range of attacks.

TABLE 2.1: Pros and cons of signature and anomaly based IDSs.

	Signature-based	Anomaly-based
Pros	<ul style="list-style-type: none"> - Simple and effective in detection of known attacks - Detail contextual analysis 	<ul style="list-style-type: none"> - Effective in detection of new vulnerabilities - Facilitates detection of privilege abuse
Cons	<ul style="list-style-type: none"> - Ineffective in detection of unknown attacks and variants of known attacks - Hard to keep signatures up-to-date - Time consuming and expensive to extract and maintain the knowledge 	<ul style="list-style-type: none"> - Weak accuracy of normal profiles as observed events are constantly changing - Not effective during recreation of behaviour profiles

There are different methods and techniques involved in each category including threshold detection, statistical measurement, rule-based methods, and evolutionary computation methods (Haag et al., 2007). For example in a threshold detection approach, the user specifies threshold values for network traffic and any deviation from the values is considered an attack and the system produces an alarm. This approach is a very common technique used in sensor networks. The detection system recognises an event of interest, when the sensory readings exceed a predefined threshold value (Baquer & Khan, 2007). Similarly, a statistical technique determines a network's normal traffic distributions and if these distributions change significantly, the system triggers an alarm. On the other hand, there are sets of predefined rules provided for rule-based systems. A rule-based classifier fires once a match is found between an input record and a rule in the system. Finally, the application of evolutionary computation techniques becomes a key technology in many research efforts in this area, but improvements are likely possible. These techniques with learning and adaptive capabilities have often been used together with rule-based methods to acquire knowledge of normal and abnormal behaviour (Wu & Banzhaf, 2010; Baig et al., 2011). Artificial neural networks (ANNs), fuzzy logic (FL), genetic algorithms (GAs), genetic programming (GP), swarm intelligence (SI) and artificial immune systems (AISs) are examples of computational intelligence approaches, that have been used (Wu & Banzhaf, 2010) for solving intrusion detection problems.

Considering the source of data being used for intrusion detection, IDSs are classified into two categories; Host-based IDS (HIDS) and Network-based IDS (NIDS) (Northcutt & Novak, 2002). A HIDS operates on a single host and monitors events occurring within an individual computer system. So, HIDS provide protection for critical computers that may house sensitive information. On the other hand, NIDSs are not restricted to packets going to a specific host since all the machines in the network are protected using this NIDS. A NIDS monitors traffic in a network segment and analyses the traffic in order to identify suspicious activities. In order to combine the advantages of both host-based and network-based IDSs and reduce human interaction, correlators are used to prioritize alerts from several different detection

systems. Correlators are able to weigh alerts, cluster related alerts, and assign priorities regarding the level of harm to a critical resource. This technique can help analysts by reducing the quantity of information they should analyse to recognise attacks ([Haines et al., 2003](#)).

Reaction - Finally, as the Internet is a resource-sharing architecture, a reaction mechanism should be employed when an attack is underway. The first advantage of having a good reaction technique is to save bandwidth that will be wasted by attack traffic and the second advantage is to separate the packets belonging to attack flow and normal flow of traffic. However, it needs to make sure that in the filtration phase, only attack traffic is filtered, and it has no impact on legitimate traffic ([Peng et al., 2007](#)). It is possible to combine any detection mechanism with one or more reaction techniques for achieving better results ([Mölsä, 2005](#)). There are several reaction mechanisms proposed in the literature such as killing of active network connections, filtering, defining new policies and reconfiguring, and source traceback mechanism. Responses to attacks could either be in an active or passive mode. In an active scheme, the reaction mechanism reacts against attack traffic in real-time while in a passive mode, the attack traffic log will be analysed passively to find the sources of attacks, for example, see [Chen et al. \(2004\)](#).

As the focus of this thesis is on the intrusion detection problem, in the next section, some of the important features that an intrusion detection system should possess will be described.

2.3.1 Selection Criteria for Intrusion Detection Systems

Although there are many different IDSs available to secure networks, attackers on the other hand are evolving and coming up with new attack strategies against network infrastructure. There are several factors that should be considered when choosing a cost-effective defence. Several factors affecting the selection process are presented in Table 2.2 ([Mölsä, 2005](#); [Chebroly et al., 2005](#)).

TABLE 2.2: Factors affecting the selection process of defence mechanisms

Factor	Explanation
Effectiveness	How capable is an IDS in detection of attacks? Is there a possibility of false positives?
Adaptability	How capable is a defence mechanism in learning changes in the environment over time?
Misusability	Can an attacker use a defence mechanism for achieving a DoS condition?
Collateral damage	Does a defence mechanism cause any negative side effects? Does it harm the legitimate traffic?
Proactivity	Can a defence mechanism prevent attacks?
Completeness	What kind of other defence mechanisms are required?
Reaction time	How fast does a defence mechanism react to attacks?
Ease of implementation	Is it feasible to implement a defence mechanism?
Ease of use	Is the interface easy to use? Is it configurable for different environments?
Installation place	What is the optimal place to implement a defence mechanism?

Effectiveness and adaptability are the two key characteristics of intrusion detection systems.

Effectiveness - An effective IDS should be general enough to detect different types of attacks and at the same time it should not misclassify legitimate activity as an attack (Chebrolu et al., 2005). The only way to evaluate the effectiveness of an intrusion detection system is to measure its ability to accurately detect intrusions (detection rate) without raising too many false alarms (false positives/negatives). These two metrics are the most common ones used in IDS evaluation studies (Eskin et al., 2002; Lee & Shields, 2002; Lee & Stolfo, 1998). The detection rate is generally defined as the percentage of attacks detected by the IDSs and the false positive rate is the percentage of normal traffic that has been mistakenly detected as malicious

traffic by IDSs. There is always a tradeoff between the detection rate and false alarm rate. The analysis of their trade-off can be represented as a receiver operating characteristic (ROC) curve. Some ROC curves of real IDSs can be seen in several research efforts ([Durst et al., 1999](#); [Lippmann et al., 2000](#)). A ROC curve clearly presents the relationship between the number of false positives and true positives. An effective IDS can detect a high number of intrusions with an acceptable level of false alarms.

Adaptability - refers to an IDS' ability in learning changes in the environment over time and adjusting to them ([Bai & Kobayashi, 2003](#)). For both signature and anomaly based detection systems, this is a major challenge and a highly desired characteristic. In signature based systems, the database of signatures should be regularly updated either manually, which is time-consuming and laborious, or automatically to be able to detect unknown intrusions. Although anomaly based detection systems are able to detect new types of intrusions, they also need to adapt to constantly changing normal behaviour ([Wu & Banzhaf, 2010](#)).

One solution to the problem of facing new attacks is considering the use of biologically inspired approaches such as artificial immune systems (AISs). The principles of immunology have been adapted by the evolutionary computation community to develop AISs in the hope that these systems will adjust over time ([Forrest & Hofmeyr, 1999](#); [Dasgupta & González, 2002](#)). Another research direction is the use of incremental learning techniques to provide the ability of repeatedly updating the system when new data becomes available ([Nasr et al., 2014](#)). The later is a broader solution as it will adapt the system to both changes in attack and normal behaviour.

In this dissertation, the main advantage of signature based IDSs, which is an effective detector of known attacks, will be used to provide high detection accuracy with few false positives. However, to overcome the problem of developing and maintaining the rules (signatures) for the signature based system, this thesis investigates the existing learning solutions and proposes an evolutionary computation based rule learning technique to automatically create IDS rulesets (signatures).

The following section demonstrates the promising capabilities of machine learning techniques in designing effective and adaptable intrusion detection systems. However, more emphasis will be put on rule learning techniques as the proposed approach belongs to this category of machine learning techniques. Rule learning is a kind of machine learning technique, that uses rules for knowledge representation. The rule learning classification approaches can be broadly divided into two categories: evolutionary rule learners (GA-based) and non-evolutionary (non-GA-based) rule learners. One of the advantages of rule learning techniques is that they are very competitive in terms of interpretability since rules can often be understood easily by human experts ([Fernández et al., 2010](#)).

2.4 Machine Learning

Machine learning is a field of artificial intelligence (AI), that studies learning techniques that improve their behaviour based on past observations ([Michalski et al., 2013](#)). The goal of these techniques is to adapt to their environment and learn from their past experience, which has attracted researchers from different fields including computer science, engineering, mathematics, and cognitive science ([Alpaydin, 2004](#)). In general, a learning technique transforms information obtained from the environment into some new form for future use.

The three most common types of machine learning algorithms are supervised learning, unsupervised learning and reinforcement learning. Every record in any dataset prepared for machine learning algorithms is usually represented using a set of features. The features might be continuous, categorical or binary. If the data is labelled, then the learning technique is referred to as supervised learning. In these techniques, a mapping will be found between the set of input values and their corresponding desired outputs. Support vector machines, neural networks, decision trees and k-nearest neighbour are examples of supervised learning techniques ([Kotsiantis, 2007](#)). The proposed algorithm in this thesis works in a supervised environment and thus the learning algorithm is provided with the class of the input example. Another

kind of machine learning is unsupervised learning, where records in datasets are unlabelled. Finding regularities in the input data is the aim of unsupervised learning techniques, which will find unknown, but useful classes of items. This is done by finding structure in a collection of unlabelled data points. One common method in this category is clustering, which finds clusters of input data ([Alpaydin, 2004](#)). Finally, reinforcement learning will learn behaviour through trial-and-error interactions with a dynamic environment. In this approach, the learner is not told which actions to take. It will discover the best ones by trying each action in turn and being positively reinforced for good actions. ([Kaelbling et al., 1996](#); [Kotsiantis, 2007](#)).

Machine learning techniques can be applied to intrusion detection problems in different ways. For example, automatically generating signatures or rules for signature based IDSs, extracting interesting features to improve the performance of IDS or learning the normal behaviour of a protected system for an anomaly based IDS ([Shafi, 2008](#)).

In the following sections, a set of selected machine learning techniques from the categories of non-evolutionary and instance-based approaches are explained. These are C4.5 (a decision tree learner), RIPPER (a propositional rule learner) and kNN (instance-based learning scheme). Additionally, a number of evolutionary algorithms are described to provide a sufficient background on the proposed technique in this dissertation.

2.4.1 Non-evolutionary Rule Learning Algorithms

Rule-based methods generally work as follows. A set of rules is manually defined or automatically learned by the system. Each rule uses a set of features and the input data points are also represented by those features. A rule-based classifier fires once a match is found between an input record and a rule in the system. A rule consists of a pattern and an action. When a pattern matches a sequence of values, the corresponding action will be fired ([Aggarwal & Zhai, 2012](#)). The goal of rule-based systems is to construct the smallest ruleset that is consistent with the

training data. If the number of rules learned by the system is large, the learning algorithm is only replicating the training data and will not generalise well. This is also called an overfitting problem which is a key problem for all learning algorithms (Fürnkranz et al., 2012). Prepruning of rules during the learning and post-pruning after an overfitting ruleset has been learned are two standard techniques used to avoid this problem.

One example of a rule learning techniques is a separate-and-conquer or covering algorithm, which is presented in Algorithm 1. Basically, it searches for a rule that explains a part of training dataset, separates these instances, and repeatedly conquers the remaining instances by learning more rules until no instance remains (Fürnkranz, 1999). The final ruleset generated by this algorithm can be ordered in terms of rule accuracy so the most accurate rules are considered first when the system classifies a new instance.

Algorithm 1 A general pseudo-code for rule learners (Kotsiantis, 2007)

```

Initialize ruleset to a default (usually empty) or a rule assigning all objects to the
most common class
Initialize instances to either all available instances in the training data or all
instances not correctly handled by rule set
repeat
    Find best, the best rule with respect to instances
    if such a rule can be found then
        Add best to ruleset
        Set instances to all instances not handled correctly by ruleset
    end if
until no rule best can be found for instance, because no instances remain

```

Two dominant schemes for rule learning are RIPPER (for Repeated Incremental Pruning to Produce Error Reduction) (Cohen, 1995) and C4.5 (a decision tree learner explained in Section 2.4.1.2) (Quinlan, 1993) algorithms. Both perform an optimisation process on the set of initial rules, which is a complex and heuristic process according to Frank & Witten (1998). In an intrusion detection context, a rule-based algorithm can be used to learn rules for the most interesting class, which is usually the attack class.

2.4.1.1 Repeated Incremental Pruning to Produce Error Reduction (RIPPER)

RIPPER implements a propositional rule learner, which was proposed by [Cohen \(1995\)](#) as an optimised version of Reduced Error Pruning (REP). RIPPER defines a rule-based model and seeks to improve it iteratively by using different heuristic techniques. The aim is to increase the accuracy of the ruleset by replacing or revising individual rules. Finally, the constructed ruleset is used to classify new instances.

Experiments in the literature showed that the classification accuracy of RIPPER is comparable to that produced by C4.5. However, the time complexity of RIPPER is less than C4.5 and, thus, makes RIPPER more attractive for large datasets. Since RIPPER is able to induce rules from examples that are encoded using boolean, numeric, or nominal attributes, it has been used in a wide range of practical applications ([Yang et al., 2001](#); [Gaonjur et al., 2008](#)).

The experiments using RIPPER algorithm in this thesis are conducted using the WEKA (The Waikato Environment for Knowledge Analysis) software system. WEKA is a tool for data analysis and includes implementations of data pre-processing, classification, regression, clustering, association rules, and visualization by different algorithms ([Witten & Frank, 2005](#); [Zhao & Zhang, 2008](#)). The implemented approaches in WEKA include instance-based learning algorithms, statistical learning and rule-based methods. The JRip algorithm in WEKA implements Repeated Incremental Pruning to Produce Error Reduction.

2.4.1.2 Decision Trees

Among the existing classification techniques, decision trees are simple yet successful for explaining the relationship between some measurements about the input instance and its target class ([Rokach, 2008](#)). In classification tasks, a decision tree is more appropriately referred to as a classification tree, which classifies an input example (such as flow of traffic) to a predefined set of classes (such as normal/attack) based on its attributes values (such as total packet count or destination port entropy).

The decision tree consists of a *root* node, *internal* nodes with outgoing edges and *leaves* (also known as decision nodes). Based on a certain discrete function of the input attribute values, each internal node divides the instance space into two or more sub-spaces. Finally, each leaf is assigned to one class in a classification task.

Decision trees can also be turned into a ruleset by generating one rule for each path from the root to each leaf. Decision makers prefer decision trees that are easy to understand and not complex because their complexity has a crucial effect on their accuracy (Breiman et al., 1984). For measuring the tree complexity, the total number of nodes, total number of leaves, tree depth and number of attributes are usually used.

C4.5 (Quinlan, 1993) and its predecessor ID3 (Quinlan, 1986) are two well known examples of algorithms that summarise training data in the form of a decision tree. These algorithms employ a greedy approach, that use an information theoretic measure as their guide. C4.5 has the advantage of pruning the decision trees, which simplifies the trees and reduces the probability of over fitting the training data. It also accepts both continuous and categorical attributes (Anyanwu & Shiva, 2009). This enhanced algorithm has been used as a benchmark when measuring the performance of other machine learning algorithms (Hall, 1999). J48, which is a slightly modified version of C4.5 provided with WEKA, is used in this thesis.

For binary problems, a rule-based technique which learns a set of rules for only the positive class is more comprehensible than decision trees. On the other hand, for multi-class problems, rule-based learners must be run separately for each class. Finally, all the rulesets generated are combined and as a result, these sets might be inconsistent or incomplete. This is usually solved by ordering the rules and a rule is only fired when none of the preceding rules are applicable. The divide-and-conquer technique used by decision trees is usually more efficient than the approach used by rule-based systems in small datasets (Kotsiantis, 2007). When the classification problem is more complex and challenging, such as the intrusion detection task, use of these techniques requires care to produce a comprehensive system with the least complexity.

2.4.2 k-Nearest Neighbour (kNN)

The *nearest neighbour* algorithm is from the category of instance-based or lazy-learning algorithms. These algorithms delay the induction or generalization process until classification is performed. Comparing to eager-learning algorithms such as decision trees, instance-based methods need less computational resources during the training. However, for classification process, more computation time is needed (Kotsiantis et al., 2006). The kNN classifies an unseen instance by finding the k closest previously observed examples, and assigning a label to the test instance by identifying the most frequent class in this neighbourhood. The kNN has been listed among the most influential data mining algorithms in the research community according to Wu et al. (2008). One of the key issues affecting the performance of kNN is the choice of k . Being sensitive to noise points is the result of a small k and including too many instances from other classes during classification is the result of a too large k . Another issue is the choice of the distance measure used to compare instances. Various measures can be used to compute the distance between two points such as Cosine measure, Euclidean measure, Chi-square and hyperrectangle distance function. The most commonly used one is the Euclidean distance function (Wilson & Martinez, 2000), which is defined as:

$$E(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2} \quad (2.1)$$

where \vec{x} and \vec{y} are the two input vectors, m is the number of input attributes, and x_i and y_i are the input values of attribute i . In some algorithms, weighting schemes are used to provide more accurate results by altering the distance measurements and voting influence of each instance. This will give more weight to highly reliable training objects (Domeniconi et al., 2002). A weighted distance computation can be defined as:

$$D(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^m w_i (x_i - y_i)^2} \quad (2.2)$$

where w_i is the weight of attribute i .

In this thesis, the IBk algorithm in WEKA toolkit is used, which is a k-nearest-neighbor classifier that uses an Euclidean distance measure. If more than one neighbour is selected, the predictions of the neighbours can be weighted according to their distance to the test instance. Two different methods are implemented in WEKA for converting the distance into a weight: inverse-distance weighting (weight neighbours by the inverse of their distance) and similarity weighting (weight neighbours by 1-their distance).

2.4.3 Evolutionary Rule Learning Algorithms

The focus of this section is on the specific machine learning paradigm used in this thesis. According to [Michalewicz \(1996\)](#), Genetic-based Machine Learning (GBML) approaches use evolutionary computation techniques (EC) as their learning technique to search in the space of possible solutions ([Holland et al., 2000](#)).

Evolutionary Computation (EC) is a research area inspired from the process of natural evolution. Natural evolution can be presented as a population of individuals in a given environment, that compete with each other for survival and reproduction. The success of an individual is determined by fitness rules in the environment ([Eiben & Smith, 2003](#)). In the area of computer science, evolutionary computation can be used for problem solving. Given a set of candidate solutions for a problem as the individuals, an evolutionary computation algorithm finds the fittest individuals or the best solutions to the problem as follows:

The evolutionary process starts with generating a population of individuals (each referred to as *chromosome*). In the standard genetic algorithm each member of the

Algorithm 2 Evolutionary Computation

```

Initialize population
while termination condition not satisfied do
    Evaluate fitness value of each individual
    Select intermediate population
    Apply genetic operators (crossover and mutation) to the individuals
    Create a new population
end while
return the best individuals in current population

```

initial population is a binary of string of length l , which corresponds to the problem encoding. Each bit in the chromosome is called a *gene*.

The individuals generated for the first population are the candidate solutions for the target problem. Next, in an iterative manner, the population of individuals goes through evaluation and breeding stages with the aim of coming closer to solving the problem in hand.

In the evaluation stage, each individual is evaluated and a set of fit individuals (intermediate population) is selected. Selection of individuals is based on the fitness value assigned by the evaluator. These individuals in the intermediate population are used as the breeding ground for the next population.

In the breeding stage, by applying the genetic operators, the EC algorithm tries to provide better solutions in the new generated population. Crossover is a mechanism for reproducing in evolutionary computation algorithms. This can be done for example by selecting two individuals from the intermediate population and choosing a random point to split the parents organisms' strings (with a probability of p_c). To produce two new children, the substrings from each parent are swapped. The parameter p_c , or the crossover probability is the probability that crossover occurs at a particular mating. Next, to introduce diversity into the population, mutation is applied to each individual with a probability of p_m . This parameters determines how likely it is that each part of an individual will be changed to a different value. In addition to the use of crossover and mutation operators to modify a population of potential solutions, the application of reproductive techniques can also be used. There are two reproductive techniques introduced in the literature: generational

and steady-state (Syswerda, 1991). The generational genetic algorithm replaces the entire population with a new population, while the steady-state genetic algorithm replaces only a few members during a generation (Vavak & Fogarty, 1996). In different comparative studies, these approaches were compared against several problems (Vavak & Fogarty, 1996; Dahal & McDonald, 1998; Rogers & Prügel-Bennett, 1999; Goldberg & Deb, 1991). However, definitive statements about the performance of these two reproductive schemes are difficult to make.

Defining the right chromosome representation for each problem is a key issue in genetic algorithm work (Michalewicz, 1996). The encoding mechanism for representing the problem's solutions depends on the nature of the problem. For example, for solving the travelling salesman problem, the variables can be binary quantities representing the inclusion or exclusion of an edge in the Hamiltonian circuit. Integer and real-valued representations are two other types of encoding schemes.

Evolutionary rule-based systems are a kind of GA-based machine learning, that uses sets of rules for knowledge representation (such as IF-THEN prediction rules) and genetic algorithms as the learning mechanism (Freitas, 2002) - the type of GBML technique used in this thesis. When an evolutionary algorithm is used to learn rules, there are two learning styles: Michigan and Pittsburgh approaches. Depending on which method is used in the evolutionary algorithm design, the individual representation for the classification problem can either be a ruleset or just a single rule. The Michigan style evolutionary learning system operates at the level of the individual rule. On the other hand, an individual in the Pittsburgh approach represents a complete solution to the classification problem. The main advantage of the former is that the evaluation of an individual takes into account rule interactions. As a result, no conflict between individuals arises, the problem that can often be seen in the Michigan style evolutionary learning systems. On the other hand, maintenance and evaluation of complete rulesets in the Pittsburgh approach is much more computationally expensive than the Michigan style, which has a simpler chromosome representation (Casillas et al., 2007; Freitas, 2008).

The advantages of GBML approaches are production of interpretable models, no assumption of prior relationships among features and the possibility of obtaining precise and compact rulesets (García et al., 2009). Examples of GBMLs are XCS (Wilson, 1995), UCS (Bernadó-Mansilla & Garrell-Guiu, 2003), GASSIST (Bacardit & Garrell, 2004) and MPLCS (Bacardit & Krasnogor, 2009b), which are explained in the following subsections and a summary of their components is also provided in Table 2.3.

2.4.3.1 XCS

A genetic algorithm is the main component of a learning classifier system (LCS i.e. further explained in Section 2.5.2) such as XCS (Wilson, 1995). The condition-action rules in LCSs are referred to as classifiers. The original XCS was mainly applied to binary problems and a ternary representation in the form of $\{0, 1, \#\}$, where $\#$ means it can match either of the input values, was utilised in its implementation (Holland et al., 2000). An interval based representation was later introduced for XCS by Wilson (2000, 2001), where the condition part is replaced by a conjunction of interval predicates. The intervals are in the form of (l_i, u_i) , where l_i is the lower bound and u_i is the upper bound of an interval. The genetic algorithm in XCS tries to find new classifiers, which contribute to the existing knowledge and eliminate classifiers that do not contribute much. There are three different sets considered in the XCS algorithm: [P]: population, [M]: match set and [A]: action set. When an input instance is presented to XCS, it will be processed in one of two modes: explore and exploit. During the explore phase, a match set will be created of all the matching classifiers in the population set. If all of the actions are not presented in the match set, a *covering* operator will be activated to create new classifiers with the actions that are not in [M]. An action set will be then created by randomly choosing an action and selecting the classifiers advocating that action. Next, the selected action is sent to the environment and depending on the input example label, a reward will be returned to the system. XCS uses this reward to update the parameters of the classifiers in [A]. Fitness is also updated based on the classifier's reward prediction. A GA is applied to the action set if the average time since the last application of

GA to the classifiers in $[A]$ exceeds a threshold. During the GA process, two parents are selected with probability proportional to their fitness. Then, after reproducing, crossing over and mutating the parents, two offspring will be generated and inserted to the population. If the number of classifiers in the population exceeds a user-defined threshold, a classifier will be removed stochastically to keep the population size constant. On the other hand, during the exploit phase, a test input example is presented to the system and the algorithm has to predict the corresponding action. A prediction for each matching action is calculated using the fitness weighted average of the predictions of all the classifiers in $[M]$ that advocated the given action. Finally, the system will return the action with the highest prediction (Shafi et al., 2009).

The most important feature of XCS is its fitness calculation, which is based on the accuracy of the classifier's payoff prediction. This is in contrast to the use of only prediction in previous strength-based LCSs, which was originally introduced by Holland (1975). In traditional LCSs, a *strength* value is assigned to each rule. This value shows the reward that the system can expect if that rule is fired. In both selection of individuals in the evolutionary process and controlling the participating rules for decision making (or prediction), this strength value will be used. This might eliminate some low-rewarded classifiers from the population which are still well suited for the environmental niche (Urbanowicz & Moore, 2009). Another feature of XCS is that the GA is applied over niches (i.e. the match set) instead of the whole population. This will increase the proportion of accurate classifiers in the population. The general and simple design of XCS was a great start for its following implementations. As an example, UCS (Bernadó-Mansilla & Garrell-Guiu, 2003) is a well-known variant of XCS that has been designed specifically for supervised learning tasks. Both XCS and UCS are Michigan style LCSs. The Michigan-style is characterized by a population of rules and the GA operates at the individual rules level. The final optimized solution is then represented by the entire population of rules.

2.4.3.2 GASSIST

Genetic Algorithms based claSSifier sySTem (GASSIST) ([Bacardit, 2004](#)) is a Pittsburgh GBML algorithm that evolves individuals representing complete problem solutions. An individual consists of an ordered, variable-length ruleset and the GA operates at the level of a ruleset. GASSIST uses the minimum description length principle for its fitness function design. This fitness takes into account both the accuracy and the complexity of ruleset. An adaptive discretization intervals rule representation ([Bacardit & Garrell, 2003](#)) for real-valued attributes is used in GASSIST. This representation can be converted into an interval-rule later by taking the lower and upper cut-points of each attribute. To keep the ruleset small, GASSIST additionally introduces a new deletion operator after a predefined number of iterations to remove the rules of an individual which do not match any training input instance.

2.4.3.3 MPLCS

Memetic Pittsburgh Learning Classifier System (MPLCS) ([Bacardit & Krasnogor, 2009b](#)) incorporates two kinds of local search operators into GASSIST: (1) rule-wise mechanism and (2) ruleset-wise mechanism.

Rule-wise operators are rule cleaning (RC), rule splitting (RS) and rule generalizing (RG) operators. The first two edit the rules to eliminate some of their misclassifications, which make the rules more specific. The RG operator edits the rules to make them more general by adding literals to the rules in order to classify more instances correctly.

The ruleset-wise local search mechanism has three main stages: evaluation of the candidate rules, selection of the rules that will form the offspring ruleset and generation of the final individual. This approach recombines rules from many different parents to generate a single ruleset with maximum possible accuracy and compactness.

In (Bacardit & Krasnogor, 2009b), both classes of operators were tested separately and together against four different datasets. The aim was to find the MPLCS variants that produce optimal solutions and converge to them using the least amount of training samples. Different configurations of MPLCS were also evaluated in terms of their ability to cope with noise and required amount of computational effort. The results showed that a proper equilibrium of specificity and generality pressures is the key for successful learning. The best configuration of MPLCS was the one that used the three rule-wise operators simultaneously across the whole population.

TABLE 2.3: A summary of the reviewed GA-based rule learning techniques.

Method	Learning	Evaluation Metric	Evaluation Style	Chromosome Representation	Type of GA
XCS	Reinforcement	accuracy	Michigan-style	interval representation	steady-state
UCS	Supervised	accuracy	Michigan-style	interval representation	steady-state
GASSIST	Supervised	minimum description length principle (accuracy and complexity)	Pittsburgh-style	adaptive discretization intervals	generational
MPLCS	Supervised	minimum description length principle (accuracy and complexity)	Pittsburgh-style	adaptive discretization intervals	generational

2.5 Rule-based nature-inspired approaches for IDSs

Network Intrusion Detection Systems (NIDS) can be categorised into behaviour-based (anomaly-based) and rule-based (signature-based) systems. Behaviour-based systems create a model of normal behaviour and when input values fall outside the norm, they raise alarms. On the other hand, rule-based systems use rulesets to detect known network threats. An important issue for rule-based systems is the cost of developing and maintaining the rulesets. In traditional rule-based systems, a human expert is responsible for creating, testing and distributing the rules through a manual process of trial and error ([Vollmer et al., 2011](#)). To address this issue, the use of nature-inspired approaches can be investigated in rule-based IDSs. In this section, some of the work related to the proposed approach is presented. These are from the category of rule-based systems, which use nature-inspired learning techniques. Their learning approaches are grouped into Artificial Immune System (AIS) (i.e. applies to the category of anomaly detection) and GA-based (applicable to both anomaly and signature based systems).

2.5.1 Artificial Immune System

The biological immune system can be broadly divided into two categories: Innate (non-specific) Immunity and Adaptive (specific) Immunity, which are linked together and influence each other. This system is able to categorize all cells within our bodies as self-cells or non-self-cells. The interesting mechanism of adaptive immune system in responding to previously unknown foreign cells and building a long-term response to them in the body, has attracted the attention of researchers in computer science recently. This nature inspired computational intelligence technique is known as an Artificial Immune System (AIS) ([Aickelin et al., 2014](#); [Kubi, 2002](#)).

A standardized framework is defined by [De Castro & Timmis \(2002\)](#), which has three basic elements: representation of components, affinity measures, and immune algorithms as presented in [Figure 2.2](#). At the first layer, the chromosome data structure representation will be defined. Bit string, real-valued vector and length

are examples of a chromosome representation. In the next layer, one or more affinity measures are defined to measure the interactions of the system's elements. For example, for the bit string representation, Hamming distance can be used and for real-valued vectors, Euclidean distance is a good measurement. Finally, an immune algorithm with a proper choice of mutation, selection, and evaluation methods will be selected to complete the framework.

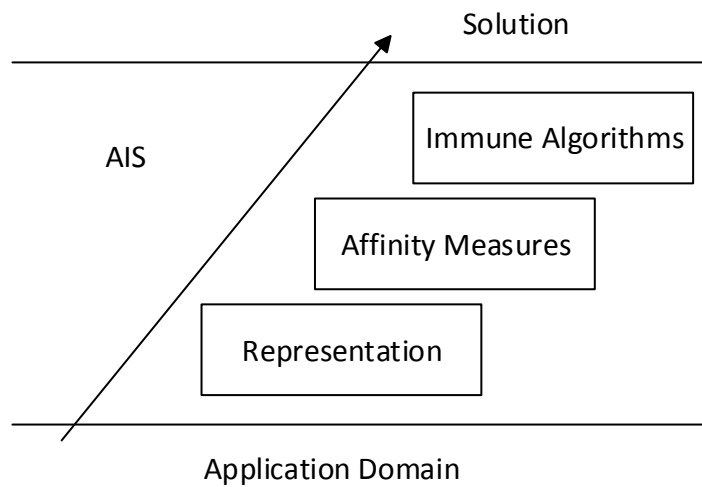


FIGURE 2.2: Layered AIS Framework (De Castro & Timmis, 2002).

The effectiveness and efficiency of an AIS strongly depends on the search algorithm used within it. The purpose of searching is divided into two phases: initialization and scanning phase. In the former, a set of optimal detectors will be chosen to detect anomalies and in the latter, based on a user-defined matching threshold, the selected detectors search for the anomalies (Haag et al., 2007).

This technique has recently been used for network protection by a number of researchers (Hou & Dozier, 2005; Forrest & Hofmeyr, 1999). The AIS based intrusion detection systems are in the category of anomaly detection. They generate non-self patterns for anomalous records based on received normal data. In all of the previously developed AISs for network protection, the success of the natural system has been emulated to overcome the common problem of IDSs in detecting novel attacks. They evolve a population of coded detectors (e.g., binary strings, network packets,

etc.) to match malicious instances, so the system is able to classify the network traffic as either “self” or “non-self” (Forrest & Hofmeyr, 1999). The evolutionary process in AISs developed for network protection is presented in Listing 2.1. Initially, a set of immature detectors is randomly generated. These detectors then go through two phases of training. First, for a pre-defined amount of time, they are exposed to the normal instances and if any detector matches a self packet, it will be removed from the population. This process is also referred to as the negative selection mechanism (Kim & Bentley, 1999). After the negative selection phase, the surviving mature detectors are able to detect non-self packets but they still have to go through another phase of learning. During this phase, detectors that fail to match a certain number of non-self instances will die. Finally, a set of detectors that remain in the population after two phases of training are used as memory detectors in order to detect threats against the system (Britt et al., 2007; Hou & Dozier, 2005).

```

Create a population of detectors randomly (Pop)
//negative selection phase
for some predefined amount of time
    Compare each detector to self instances
    if (detector matches the self)
        Remove the detector
    else
        Keep the detector as a Mature Detector
//second phase of learning
for some predefined amount of time
    Expose Mature Detectors to non-self
    if (Mature Detector matches a certain number of non-self instances)
        Promote to Memory Detector
    else
        Remove the detector
//Detection Phase
while (condition is true)
    Use each element of Pop to perform detection

```

LISTING 2.1: Training Detectors in an AIS.

The negative selection phase, which is the core of these systems, enables them to perform a form of passively proactive protection and this gives them the advantage

of detecting previously unknown attacks. In this approach, it is assumed that the normal behaviour of a system shows stable patterns over time.

The most commonly used coding scheme for AIS-based IDSs is binary strings (Wu & Banzhaf, 2010), where a matching rule is a r -contiguous bit string accordingly. Since real-valued representation is closer to the problem space (Gonzalez et al., 2002), Hou (2006) proposed a constraint-based AIS with a novel detector representation. A data sample is a vector of integers, (x_0, x_1, \dots, x_n) , where n is the number of fields and x_i is the value of the corresponding field. The range of each field can be chosen based on an analysis on the actual data set. Accordingly, detectors are coded as interval constraints which are in the form of vector of intervals, $(lb_0..ub_0, lb_1..ub_1, \dots, lb_n..ub_n)$. Each interval covers the possible values of the corresponding field in the pattern representation. For the matching rule, an any- r -interval matching rule is used to find a match between a data sample and a constraint-based detector. Based on this matching rule, if any r fields representing the data sample fall within the corresponding r intervals of a detector then that detector matches the pattern. By changing the value of r to a higher value, we can have a more specific detector but as a result, a more specific detector will match fewer patterns. Based on this novel idea, in Hou & Dozier (2005), the representation of packets is in the form of $(ip_address, port, src)$, where $ip_address$ is the remote host address, $port$ represents the port number on the receiving host, and src is 0 for incoming and 1 for outgoing packets. Accordingly, a detector is represented as $(lb_0..ub_0, lb_1..ub_1, lb_2..ub_2, lb_3..ub_3, lb_{port}..ub_{port}, src)$, where the first 4 intervals are used for the possible IP addresses, the fifth interval is for the port number, and src value decides whether the detector should be used on incoming or outgoing traffic. For the experiments in Hou & Dozier (2005), a $r=3$ threshold value has been used. For performance measurements, two popular metrics in anomaly detection system evaluation have been used: false positive (self instances are erroneously detected as non-self) and false negative rates (nonsself instances are not detected). One of the major disadvantage of AISs is that there is no way to know which nonsself instances were not detected successfully. This is also referred to as a *hole (vulnerability)* in the detection system (Hofmeyr & Forrest, 2000).

Another challenge in AIS implementations is the length of detectors to be used in the system (Britt et al., 2007). Short strings are more general, which leads to matching of more entities (and more false positives), while long strings are very specific but we need more of them to effectively map the appropriate search space. Therefore, defining a proper length for the detectors is very important because the number of detectors in the final set has a direct impact on the time needed for the recognition of self and non-self patterns and the memory requirements. Additionally, the size of the self space and the complexity of the matching function will affect the training time. The size of the self space is important because during the negative selection phase, every detector has to be checked against the self instances. More attention needs to be paid on the selection of matching function, since it affects both the learning phase and the recognition phase. Correlation Coefficient, Hamming Distance, r-contiguous-bits matching function, and constraint-based matching are the examples of matching functions used in the literature. A constraint-based approach has the advantages of being more understandable by humans, and having some fuzziness comparing to the rigid bitwise approach.

Additionally, according to Wu & Banzhaf (2010), evaluation of effectiveness of AIS-based IDSs in real-environments and exploring their adaptation to changes in self data (normal patterns) are open areas for further research. The incremental learning approach used in this thesis can be a solution for the second challenge (i.e. adaptation to environment changes).

2.5.2 GA-based Techniques

Genetic algorithms have been used as a powerful searching technique to find the optimal set of rules for anomaly detection systems. In these types of classification problems, GAs perform a global search using a population of individuals. During the evolutionary process, a fitness function encourages rules, which accurately classify the training instances. The final set of rules produced by GAs will be used for predicting unseen instances. In the area of network security, this method has also provided good results in the identification of network intrusion attempts.

In [Vollmer et al. \(2011\)](#), a GA is used to produce a set of near optimal rules for the Snort rule engine. Snort is an open source IDS developed by [Roesch \(1999\)](#). Snort has a simple rule language that matches against network packets for generating alerts or logging messages. In this work, they assumed that the anomalous traffic has already been identified in advance. Thus, their proposed system will be working in an offline mode to process pre-captured data files for rule generation. The focus of this study is on ICMP traffic and thus, the GA population representation includes the following fields: source IP address, destination IP address, ICMP id, type, code, sequence number and packet size. These fields and their acceptable values are extracted directly from the Snort rule syntax definition. Each individual in the population of solutions in their proposed system is presented using a variable length vector: $\vec{v}_i = (x_0, \dots, x_n)$ of mixed data type values x . For designing the fitness function, which judges the desirability of rules, three criteria are considered: complete rule match, grammar check and partial rule match. The first criterion is tested by running Snort with a candidate rule on a test packet and evaluating the results (a large value for a successful alert). Secondly, the Perl based *dumbpig* grammar checker is used to check the candidate rule format (a greater value between 0.0 to 1.0 will be assigned if the tool finds fewer issues). Finally, a match function is used to compute the number of fields in a rule that matched an evaluation packet. The final output of the GA is a set of rules, which will be sorted according to their fitness values and any duplicates will be removed. Then, the resulting top three rules are proposed as possible rule definitions to be distributed. The ICMP test data is generated using existing packet creation tools such as *Nemesis*, *packETH* and *ISIC*. Testing showed that the generated rules were specific to the packets and produced a low false positive rate. To provide a broader coverage of all possible network anomalies, further investigation on other protocols including TCP and UDP is needed.

As another solution to the problem of manual processing of rule creation in IDSs, a Pareto-based multi-objective evolutionary algorithm (MOEA) is used in [Gómez et al. \(2013\)](#) to optimize the automatic rule generation of a signature-based IDS (i.e. Snort). They proposed a new MOEA within the detection engine of Snort,

which will optimize the rule generation while minimizing two different objectives: the non-detected hostile traffic (i.e. the false negatives) and the non-hostile traffic erroneously detected as hostile (i.e. the false positives). Pareto-optimization (Goldberg, 1989) is used in this algorithm to address the problem involved in algorithms with one single aggregate objective function, which only provide a single compromise solution. Pareto-optimization establishes a relationship between solutions according to Pareto-dominance relations. A solution ($s1$) is dominant over another ($s2$) if $s1$ is better than $s2$ in at least one objective, and not worse in the other objectives. The aim of a Pareto-based multi-objective approach is to find the Pareto-optimal set. In the proposed algorithm, each individual contains a certain number of Snort signatures (rules). The focus of this study is on the payload part of rules, which examines data contained in the network packet. MOEA-Snort has two optimization modes: a single aggregated objective function (a function that combines both above-mentioned objectives) and Pareto-optimization. The second enables the algorithm to obtain a set of non-dominated solutions. The performance of MOEA-Snort is tested on an attack-free DARPA 1999 dataset combined with 173 attacks (on web server and ftp). The results showed that the rules obtained using the single aggregate objective function are very affected by the weights used in the objective function. The Pareto-optimization mode, on the other hand, has the advantage of producing a wide set of solutions and so it is very useful for the decision maker to choose not from one, but several solutions.

As another example of the use of genetic algorithms for intrusion detection problem, Baig et al. (2011) proposed a GA-based method which falls in the category of rule-based intrusion detection systems. Each individual (chromosome) in their system is a rule, which represents the status of a network connection. The dataset used in this study is the KDD99 dataset, which was first distributed as part of a competition (1999 KDD Cup competition) sponsored by the International Conference on Knowledge Discovery in Databases (Tavallaei et al., 2009). To find the relevant features and eliminate redundancies, some initial preprocessing is done on the original KDD99 dataset. Then, for each type of attack, a set of effective chromosomes using a combination of features are constructed (i.e. the initial chromosome generation

phase). A binary format chromosome is selected due to ease of use for the crossover and mutation phases. For example, a chromosome like $(tcp, ftp, REJ, 0, 0, 0.91, xx)$ is constructed for a Satan attack which includes the following features: *protocol_type*, *service*, *flag*, *src_bytes*, *dst_bytes*, *dst_host_error_rate*, and *dst_host_srv_rerror_rate*. During the evolution process, the system tries to form some new chromosomes with enhanced detection rate for the attacks. A bit-flip mutation and a 1-point crossover were used to generate the next generation chromosomes. For testing the system, the new chromosomes go through a pattern matching exercise and get rewards based on the number of correctly detected connections. Finally, analysis of the ROC curve showed that the false alarm rate decreases with increasing attack detection rate. One challenging task in rule-based systems, such as signature based intrusion detection systems, is reducing the signature database size. In this system, as the final set of evolved chromosomes are responsible for intrusion detection, the number of rules generated by the system is equal to the population size. For KDD99 dataset, 31 chromosomes provided an acceptable detection rate, but more investigation is needed on other datasets. In the design and implementation of rule-based intrusion detection systems, the cooperation of rules should be taken into account to decrease the size of the final rulset. In Baig et al. (2011)'s proposed approach, however, each rule is evaluated in isolation from others in the population. Therefore, the final ruleset might include some redundant rules or conflict between the rules.

In Gong et al. (2005), a similar GA-based approach is proposed for network intrusion detection. Seven network features, which have high likelihood of being involved in intrusions were extracted from the data to form a fixed length chromosome. Two modules were implemented for the training stage in an offline environment and the testing (detection stage) in the real-time environment. In the first stage, a set of optimal rules were generated. Each rule was in the form of an if-then clause, where the first six features compose the condition part and the last feature (attack name) is the outcome. Figure 2.3, shows an encoded form of a chromosome including different types of genes. A support-confidence framework (Lu & Traore, 2004) is used to evaluate each rule in the system. Once the training stage on the benchmark DARPA datasets (MIT Lincoln Laboratory, 2002) for ruleset generation

is finished, the generated rules are tested to classify incoming network connections from a different subset of 1998 DARPA data. The proposed system is implemented on top of a Java-based evolutionary computation research system called ECJ (Luke et al., 2006). The fitness function used in this system consumes a lot of memory because it needs the whole training data to be loaded before any computation. Another limitation of the proposed approach is that the generated rules are biased to the training subset. The author proposed two techniques which may solve this problem: 1) selecting appropriate number of generations; or 2) selecting appropriate number of best-fit rules.

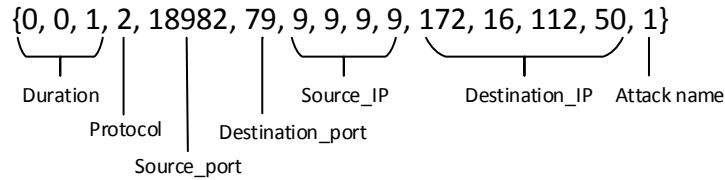


FIGURE 2.3: Encoded Form of a Rule (Gong et al., 2005).

Another example of the use of GAs for IDSs can be found in Li (2004). Simple rules in the form of *if {condition} then {act}* are evolved for network traffic using GAs. Nine attributes of network connections are included in the condition part: source IP address, destination IP address, source port number, destination port number, duration, state, protocol, number of bytes sent by originator and number of bytes sent by responder. Altogether there are 57 genes in each chromosome. In their system, the fitness of each chromosome is computed separately, which means a rule will receive a bonus if it finds an anomalous behaviour or a penalty if it matches a normal connection. Thus, there is a chance of redundant rules in the final optimised ruleset, which cover some overlapping regions of search space. The evolution process in their system starts with a population of randomly generated rules and continues with applying crossover and mutation operators toward finding an optimised ruleset. Finally, these rules will be added into the rule database used by the IDS. For testing the system, the DARPA dataset in tcpdump binary format was

used. Additionally, the authors added two statistical attributes of network traffic to overcome the problem involved in other similar approaches, which only used the categorical features of raw data. The disadvantage of using the categorical features is that these attributes do not present normal or abnormal behaviours of networks in all cases since they are extracted from packet headers. For example, in port scanning attempts, the attacker only uses some scanning tools to find the available ports in a server and as a result the generated packets are all normal and can not be easily diagnosed by signature-based systems or systems that are constructed based on only categorical features.

Another similar rule-based method used for IDSs is a Learning Classifier System (LCS), which was first introduced by [Holland et al. \(2000\)](#) as a genetic-based machine learning technique. The GA module is responsible for searching a large space by first randomly generating a population of individuals and then selecting the best and producing new individuals by mutation and crossover operators, iteratively. For evaluating the quality of individuals (or evaluating the fitness), LCSs utilise reinforcement learning or supervised learning techniques. As an example, in [Shafi & Abbass \(2009\)](#), a supervised signature learning classifier was used to evolve a set of classifiers, where each classifier is in the form of a simple rule. Two main metrics were used to evaluate the performance of rules: Accuracy and Fitness. Accuracy is the ratio between the large number of times a classifier was successful in detecting an instance and the number of times it has been fired. Fitness is then calculated as a function of accuracy. In this system, during the training phase, a set of matching classifiers for an input instance is built from the population of individuals. Then based on their updated performance parameters, the system finds the classifiers, which predict the instance class (label) correctly. Finally, a GA is applied to the selected classifiers to form the next generation population. One problem of LCSs is the number of rules generated by the system, which makes the resulting ruleset less transparent, and is an issue for time critical applications. To solve this problem, the authors proposed two generalisation operators to modify rule boundaries, which reduces the overlapping regions covered by the individuals.

The appropriateness of evolutionary rule learning techniques for detection of Session Initiation Protocol (SIP) based flooding attacks and malware detection has been evaluated in (Akbar & Farooq, 2009) and (Shafiq et al., 2009) comparative studies, correspondingly. In these studies, the performance of evolutionary rule learning algorithms such as XCS, UCS and GASSIST is compared to some non-evolutionary classifiers such as RIPPER and C4.5 in terms of classification accuracy, number of rules and processing overhead. The results from (Akbar & Farooq, 2009), showed that evolutionary classifiers can even detect low intensity SIP floods in realtime. However, in Shafiq et al. (2009), although the evolutionary algorithms provided acceptable performance results, the non-evolutionary techniques outperformed evolutionary ones especially in terms of comprehensibility of rules (due to large rulesets). Therefore, an important direction for the future of these algorithms is improving the comprehensibility of the final rulesets. This will also reduce the complexity of the rule learning process. Another issue found with LCSs such as XCS and UCS for intrusion detection is that LCSs have difficulty in achieving high accuracy on rare classes (Orriols-Puig & Bernadó-Mansilla, 2006). They also need a large population of individuals to cover high dimensional search spaces. This might lead to complex models for IDSs and generating the model can be computationally expensive.

2.5.2.1 Adapting to Environment Changes

Learning and adapting to the changing environment and detecting both known and unknown intrusions is a key issue in IDSs (Dasgupta, 1999; Shafi et al., 2006). A typical approach to learn new information is to discard current classifier and create a new classifier using all of the data that have been collected thus far. This approach is used in common neural networks such as multilayer perceptron and radial basis function network (Polikar et al., 2001). A similar incremental approach is used in Pietraszek (2004), in which subsequent new training examples are added to the training set and the classifier is retrained on the entire training set. Since it is not feasible to retrain the system on every single new training sample, the training examples are handled in batches. The size of these batches can be fixed or decided by the administrators or it can be dependant on the current performance

of the classifier. The former approach is used in Pietraszek (2004). If the current classifier accuracy drops below a pre-defined threshold, they rebuild the classifier using the entire training set. This approach, however, might not be feasible for some applications and is very resource consuming as it should store the original training data and add the new data to it. The size of this storage will grow dramatically over time and the training time in the system will also increase.

To address this issue, an incremental learning approach can be deployed to repeatedly train the network when new data becomes available and not forget the previous knowledge gained. In Nasr et al. (2014), a learnable IDS is proposed using decision trees and incremental learning, that promotes adaptability to a dynamic changing environment. Their system consists of two phases: offline training and incremental online testing. During the offline phase, pairwise datasets are generated for 1-vs-1 model classification. Therefore, for the five attack types in the KDD99 and NSL-KDD dataset, they produced 10 datasets containing 2 different classes in each dataset. Using the existing training datasets, the algorithm produces 10 unique classifiers. Next, in the online phase, the incoming records will be classified using the previously generated model from offline phase. On each single record, a Bagging approach votes all five classes. Then, the Bagging component flags the record by the class that received the maximum vote. Next, the corresponding classifiers in the database will be updated based on the information obtained from record features and predicted results. To test the system, 20% of NSL-KDD training dataset was used during the offline training phase and 20% of NSL-KDD testing dataset was fed into the online phase as a stream. The results showed that their system is capable of learning new rules from the new input data.

2.6 Summary of Challenges in GA-based IDSs

In network intrusion problem, because the pattern of normal and intrusive behaviour is very similar, the normal and attack classes have some overlap as seen in the most well-known dataset in this area, KDD99. This makes the problem harder for

generalisation and considered a challenging problem for learning algorithms ([Shafi, 2008](#)). In the reviewed approaches, GAs were used as a tool for automatically generating knowledge for rule-based IDSs. Each rule in these systems is considered a classifier and the goal is to find and add a set of most fit classifiers to the rule database to provide an acceptable detection rate. The results showed that GAs are a promising learning technique for IDSs and fulfill the effectiveness criteria. Each of the existing approaches offers its own strength and weaknesses. The challenges found in this GA-based IDSs which require further research are:

- Inflexibility of the system (e.g., fixed length chromosomes and use of packet header features)
- Use of out-dated data for testing (e.g., DARPA and KDD99 datasets)
- No consideration for cooperation among rules (complex models)
- Not adjustable to the changes in a dynamic environment
- Failure to eliminate irrelevant features

The use of fixed length chromosomes in system design forces the rule learner algorithm to only generate same size rules as the output. Simpler models can be produced if variable-length chromosomes were used in the design of algorithm. Another solution to that is defining a new representation of chromosomes with one bit associated to each feature of the domain. This method has been introduced in [Bacardit et al. \(2009\)](#) to improve the scalability of rule-based evolutionary learning. This new representation not only provided competent learning performance but it also reduced the system run-time. This is because by the use of the proposed individual representation, the evolutionary algorithm is attempting to provide a high accuracy while reducing the subset of selected features. As a result, a feature selection is performed at the rule-wise level and the final ruleset includes one or more rules that each can have different set of relevant attributes.

Another area worthy of exploration is the complexity of models generated by these approaches. One of the weaknesses found in the reviewed GA-based rule learning

techniques used for IDSs is that they did not take into account the cooperation of rules in the course of evolutionary run. By not considering this, the learning system may end up generating a set of rules, where each of them is a good classifier by itself but one specific rule might cover a region of search space that has been covered by another rule. So, numerous rules for each class will be produced. Thus, more attention need to be paid to the degree of cooperation of the rules when they are evaluated by the GA, which results in a more concise set of rules. This is one of the requirements in the design and implementation of IDSs. The rules (or signatures) generated by rule learning techniques should be understandable by network administrators and analysts. Thus, further analysis and verification can be carried out by experts ([Helmer et al., 2002](#); [Pietraszek, 2004](#)).

By addressing the above-mentioned issues (more specifically inflexibility of the system, no consideration for cooperation among rules, not adjustable to the changes in a dynamic environment), this thesis contributes to both intrusion detection and GA-based rule learning techniques.

2.7 Summary

This chapter set the context of this thesis by providing a brief introduction to intrusion detection problem and the existing defensive solutions. After stating the important factors that should be considered in the design and implementation of intrusion detection systems, an overview of the capabilities of machine learning techniques and their applications for rule learning is provided. A literature review of existing techniques relating to the proposed approach in this thesis is then provided. In particular, nature-inspired rule learning techniques for IDSs are reviewed. Next, by summarizing the existing challenges in GA-based IDSs, the areas for further research in this thesis are shaped.

In the following chapter, the design and implementation of the proposed GA-based approach for generating optimised rulesets for network intrusion detection will be presented.

Chapter 3

ESR-NID: A Framework for Evolving Statistical Rulesets for Network Intrusion Detection

3.1 Introduction

This chapter presents the design and implementation of a GA-based classification approach, called ESR-NID, that generates optimised rulesets for network intrusion detection through a learning stage. ESR-NID aims to address some of the issues found in the existing GA-based IDSs. These are inflexibility of the system (e.g., fixed length chromosomes and use of packet header features), no consideration for cooperation among rules (complex models) and not adjustable to the changes in a dynamic environment.

The generated rules as the output of ESR-NID are used for classifying unseen data points. This is a general view of the proposed system, which classifies it as a machine learning algorithm. Specifically, as depicted in Figure 3.1, among the machine learning algorithms, ESR-NID belongs to the supervised machine learning category. A supervised machine learning system consists of three parts: a learning module,

a model and a classification module. The learning module is responsible for constructing a model based on a labelled training dataset. The model in the proposed system contains a set of rules for future prediction. These rules, when applied to the records in the test dataset, will predict labels of the test set using the classification module.

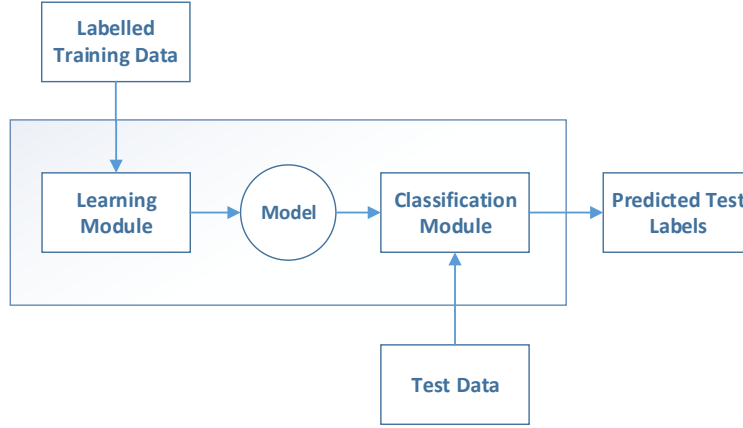


FIGURE 3.1: General view of supervised machine learning algorithms.

Due to the amount of data to be analysed and the similarity of malicious and normal traffic, intrusion detection is considered a complex real world problem. The aim is to develop a generic approach, which is not a protocol-specific mechanism and thus can be used against different existing attack tools and future malicious attempts and can be also deployed at any point of network infrastructure.

The proposed framework (ESR-NID) is depicted in Figure 3.2. The two important characteristics of ESR-NID are the new representation of individuals and an advanced two-stage evaluation approach to find the best set of cooperative rules. ESR-NID is implemented on top of a third party Java-based package called ECJ (Luke et al., 2006), an evolutionary computation research system, developed at George Mason University. There are three main stages in ESR-NID: pre-processing, evolutionary process and post-processing. In the pre-processing stage, the focus is on the preparation of data for the system, which is done through normalization. In the next stage, the evolutionary process starts, with two components to read the

preprocessed data and to initialize the population of individuals. Then, for a number of generations (i.e. until the termination condition), the evaluator, selection and breeder modules will operate to select a set of fit individuals from the population and to apply genetic operators (mutation and crossover) to each individual, respectively. Next, after completion of the evolutionary run, through a post-processing step, the final ruleset for the classification task will be produced. In the following subsections, the framework components are explained in more detail. Accordingly, in Figure 3.2, the related subsection number is provided. Thus, the explanation for each component can be found in the corresponding subsection.

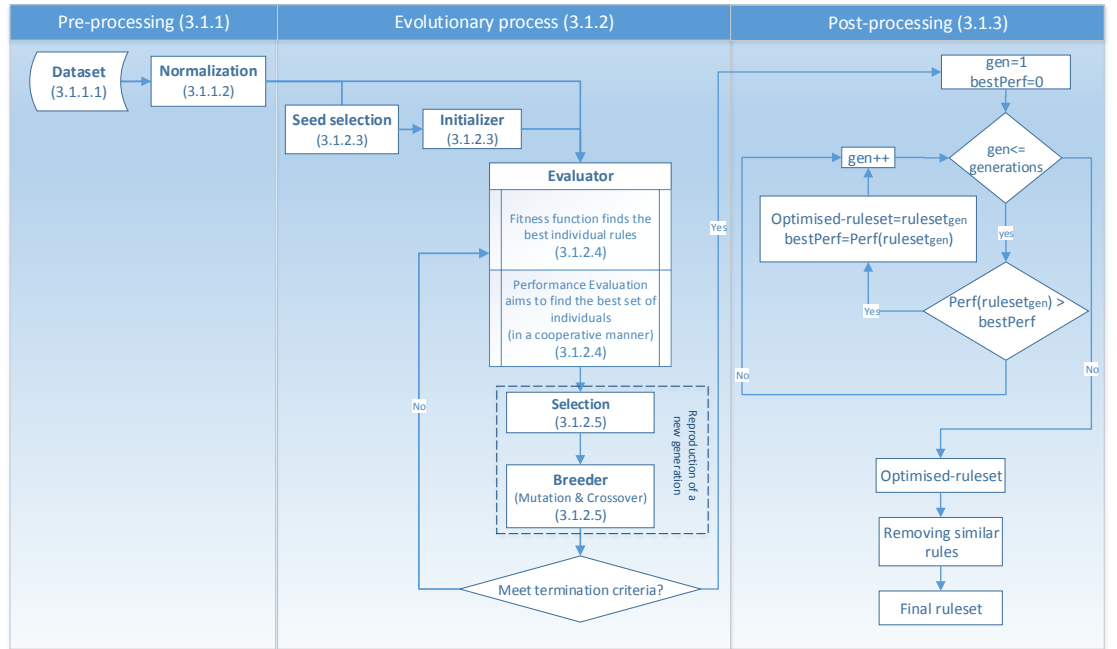


FIGURE 3.2: The architecture of the ESR-NID system.

In the following sections, the two stages of ESR-NID and the components involved in each stage are introduced.

3.1.1 Pre-processing Stage

In the pre-processing stage, the input dataset will be processed through normalization. Next, the characteristics of input data to ESR-NID and the data normalization module will be explained.

3.1.1.1 Input data

ESR-NID accepts input with continuous features. This data is in the form of vectors of floats, (x_1, \dots, x_n) , where n is the number of features in the dataset and x_i is the value of the corresponding feature. Additionally, each record in the dataset should have a class label, “normal” or “anomaly”.

3.1.1.2 Data Normalization

Since the aim is to design a generic system, that can be applied to a dataset with arbitrary distribution, in ESR-NID, at the beginning of the classification process, a data normalization component is used. A data normalization is usually needed if some distance measures (e.g., Euclidean distance) are used in the approach. By normalizing the data, the values of all features are in the same range. In this work, the feature scaling function in equation (3.1) is used, so that the values of features fall in the range of $[0, 1]$.

$$x' = \frac{x - \min_x}{\max_x - \min_x} \quad (3.1)$$

where x is the original value, x' is the normalized value and \min_x and \max_x are the minimum and maximum possible values for variable x .

3.1.2 Evolutionary Process

The evolutionary process starts with generating a population of individuals. This is done through the seed selection and initializer modules. The individuals generated for the first population are candidate solutions for the target problem. Next, in an iterative manner, the population of individuals goes through evaluation, selection and breeding stages with the aim of coming closer to solving the problem in hand. This has been implemented using evaluator, selection and breeder components in ESR-NID (details can be found in section 3.1.2.4 and section 3.1.2.5). This iterative process will be repeated until the termination criteria is met. An upper limit on the

number of generations is the termination method used in ESR-NID, which stops the evolution once a user-specified number of generations is reached.

In the evaluation stage, each individual is evaluated and a set of fit individuals (intermediate population) is then selected. These individuals in the intermediate population are used as the breeding ground for the next population. The evaluator component uses two functions to find the best cooperating rules in each generation: fitness function and performance function. The choice of these two functions is based on the designer's needs and preferences and this makes ESR-NID a flexible approach.

In the breeding stage, by applying the genetic operators, the EC algorithm tries to provide better solutions in the new generated population.

When GAs are used for rule discovery, depending on how the rules are encoded in the population of individuals, there are two approaches: Michigan and Pittsburgh. Next, these two approaches will be explained by focusing on the selected one in ESR-NID.

3.1.2.1 Michigan Versus Pittsburgh Approach

In GAs for rule discovery, rules can be encoded in the population of individuals (chromosomes) using two approaches: Pittsburgh and Michigan. In the Michigan approach, an individual represents a single prediction rule, while in the Pittsburgh approach, each individual represents a set of prediction rules. Depending on which kind of rules the system is going to discover, one of these two approaches will be chosen. In the Pittsburgh approach, each chromosome represents an independent solution (i.e. a ruleset). The quality of a candidate solution in the Pittsburgh approach is evaluated by taking into account the interactions among all the rules included in the individual (Bojarczuk et al., 2004; Freitas, 2003). This leads to use of more computational resources and some modifications might be applied on the standard genetic operators to make them applicable on relatively complex individuals in this approach. Examples of using the Pittsburgh approach in classification problems can

be found in [De Jong et al. \(1994\)](#), [Bacardit \(2004\)](#) and [Aguilar-Ruiz et al. \(2003\)](#). On the other hand, if we aim to find a small set of high quality prediction rules (e.g., detecting rare events), the Michigan approach might be more suitable. The advantage of this approach is that the individuals are simpler and shorter and also the search space is smaller and thus, less time is needed to compute the individuals' fitness values. However, if the system requires to evaluate the quality of a ruleset (i.e. taking rule interactions into account), an extra level of evaluation is needed to co-evolve a set of individuals which cooperatively provide a solution to the problem ([Freitas, 2003](#)). Examples of GAs in classification which used the Michigan approach are [Orriols-Puig et al. \(2009\)](#) and [Urbanowicz & Moore \(2010\)](#). The former is a rule-based evolutionary learning system, which uses a steady-state EA at the rule level to evolve a set of linguistic fuzzy rules to collaborate to cover all the input instances. At the end of the evolutionary run, a ruleset reduction method is designed to obtain a minimum set of rules. However, in the latter, the evolved solution is represented by the entire rule population. This may result in a large set of overlapping rules if the population size was initially set to a large number.

In [Abadeh et al. \(2011\)](#), both Pittsburgh and Michigan approaches were used for a fuzzy classifier for intrusion detection in computer networks. In their research, a Pittsburgh approach was more effective in evolving fuzzy rules. Moreover, [Shafi & Abbass \(2009\)](#) proposed a supervised learning classifier system (UCS) to learn signatures for network intrusion detection based on the Michigan style.

ESR-NID is based on a Michigan style, where each individual representing a rule is evaluated in cooperation with other individuals and the aim is to obtain the best set of rules via a two-stage evaluation process in each generation.

3.1.2.2 Individual Representation

To briefly review the characteristics of input data, the preprocessed data in ESR-NID is in the form of vectors of floats, (x_1, \dots, x_n) , where n is the number of features in the dataset and x_i is the value of the corresponding feature. Since the original dataset goes through a normalization stage, the range of values for each feature is

Field0			Field1			...	Fieldn		
LB0	UB0	on/off	LB1	UB1	on/off		LBn	UBn	on/off

FIGURE 3.3: The structure of a chromosome in the proposed system. LB_i , UB_i are real values between 0 and 1 and represent lower bound and upper bound of feature i and on/off component is a binary value used to activate or deactivate feature i in the chromosome. Thus, the system can produce variable length rules.

between 0 and 1. Accordingly, the basic structure of a detector rule in the system is defined as follows :

Rule: if *cond*, then *anomaly*

where $cond = x_{i_1} \in [LowerBound_{i_1}, UpperBound_{i_1}]$ and ... and

$x_{i_k} \in [LowerBound_{i_k}, UpperBound_{i_k}]$;

where $i_k \in [1, n]$ and $LowerBound_i$ and $UpperBound_i \in \mathbb{R}$ (i.e. a real number).

The chromosome structure representing a rule in ESR-NID is shown in Figure 3.3. It contains one field of three genes for each feature. Depending on the number of features in the input dataset, the system can be customized by increasing or decreasing the length of the chromosome. Each field has 3 components: LB_i , UB_i are real values between 0 and 1, and a binary on/off value signifies whether the feature is active in the rule. For reporting the optimised ruleset, the LB_i is mapped back to a value in the original range of the feature using the equation

$$LowerBound_i = min + LB_i \times (max - min) \quad (3.2)$$

where min and max are the minimum and maximum values for that feature and $LowerBound_i$ is the original value of lower bound of feature i . Similarly, the value of the upper bound in the original range is also calculated as

$$UpperBound_i = LowerBound_i + UB_i \times (max - LowerBound_i) \quad (3.3)$$

In the proposed individual representation, the binary *regulatory* value for each feature in the chromosome enhances the evolutionary algorithm (EA) by allowing it to dynamically find the required features to solve the problem. Standard feature selection methods usually filter the input data by keeping the same subset of features for the whole solution (i.e. a ruleset), whereas the proposed individual representation in ESR-NID has a built-in fine grained feature selection process, which provides varied length rules with different sets of relevant features. Using the regulatory genes, ESR-NID is performing a feature selection at a rule-wise level. Thus, the key features may be different for each learned rule. For example, an optimised population of 3 rules is presented below (e.g. the regulatory genes for x_1 and x_3 are on and for x_2 and x_4 are off in Rule₁):

Rule₁: if $x_1 \in [LowerBound_1, UpperBound_1]$ and $x_3 \in [LowerBound_3, UpperBound_3]$
then *anomaly*

Rule₂: if $x_2 \in [LowerBound_2, UpperBound_2]$ then *anomaly*

Rule₃: if $x_2 \in [LowerBound_2, UpperBound_2]$ and $x_3 \in [LowerBound_3, UpperBound_3]$
and $x_4 \in [LowerBound_4, UpperBound_4]$ then *anomaly*

In Rule₁, features 1 and 3 are active, whereas in Rule₂, only feature 2 and in Rule₃, features 2, 3 and 4 are active. This new form of representation reduces the time needed to evaluate each rule and helps the algorithm to find the set of features with most predictive ability.

3.1.2.3 Seed Selection and Initializer

Most GA-based systems use random selection of seeds for initialization of the population. Experiments in the past showed that non-random initialization, or *inoculation*, is able to improve the performance of evolutionary algorithm in terms of average solution quality and runtime (Surry & Radcliffe, 1996). This method incorporates domain knowledge by using one or more high quality solutions. ESR-NID has two methods of initialization: random (default option in ESR-NID) and a combination of random and non-random. The method for initializing a population is determined

by an input parameter to the algorithm. In a random initialization, the initial population of individuals are randomly generated (and the seed selection module is not used), whereas in the latter, one or more individuals are read from a file produced by the seed selection module. For this, a user-defined interval is used by the seed selection module to create rules for a random subset of anomalous records from the training set. This is depicted in Algorithm 3, Seed selection.

Algorithm 3 Seed selection

```

Select a random subset of anomalous records from training set
for each record rec in selected subset do
  for each feature f of rec do
     $LowerBound(f) \leftarrow f - interval$ 
     $UpperBound(f) \leftarrow f + interval$ 
     $cond(f) \leftarrow f \in [LowerBound(f), UpperBound(f)]$ 
  end for
  return rule(rec)
end for

```

The output of Algorithm 3 is: rule(*rec*) = if *condition*, then *anomaly*, where *condition* = $f_1 \in [LowerBound(f_1), UpperBound(f_1)]$ and $f_2 \in [LowerBound(f_2), UpperBound(f_2)]$ and ... and $f_n \in [LowerBound(f_n), UpperBound(f_n)]$. The output of the seed selection module will be written into a file to be later used by the initializer module in ESR-NID. These individuals must be written in a computer-readable fashion into the file.

In ESR-NID, the size of the population and the number of non-random seeds for initialization are user defined input parameters to the system. If the number of non-random seeds is less than the size of the population, the initializer will randomly generate the rest of the individuals in the population.

As explained before, in ESR-NID, an optimal solution for the classification problem is encoded by a set of rules using a Michigan approach. For this, the evaluator component is implemented using a fitness function (designed specifically for the proposed system) and a performance function, which are described next.

3.1.2.4 Fitness Function and Performance Function

In ESR-NID, the evaluator component aims to derive a set of classification rules from the provided dataset, which can cooperatively provide a good coverage of search space. For this, a two-stage evaluation method is proposed. In the first stage, based on a fitness value, the best individual rules are found in the population and in the second stage, a performance function decides on the best ruleset in a cooperative manner.

To design an appropriate fitness function, the following factors are considered:

- The number of anomalies correctly detected.
- The number of errors (i.e. the number of normal instances incorrectly detected as anomalies)
- The overlapping with other rules; a higher score will be given to the rules that detect anomalies which were not identified by many other rules.
- A weighting, on the scores and errors, to control the balance between them and to address the influence of imbalanced datasets.

To explain the first two factors, suppose that Figure 3.4 shows a simple binary classification problem with one detection rule constructed to classify five abnormal instances. This rule also misclassifies two normal instances. Since the aim of an IDS is to detect the abnormal instances, the fitness function should reward a detection rule for its correctly detected abnormal instances. At the same time, a good system should produce a low false positive rate. Therefore, the fitness function should penalise a detection rule for incorrectly detecting normal instances as abnormal records. The false negative rate is not taken into account here in designing the fitness function, because the main concern is measuring the performance of a single rule in detection of abnormal instances. As the final output of ESR-NID is an optimised ruleset, those abnormal instances not detected correctly by this rule will be classified by some other rules in the optimised ruleset.

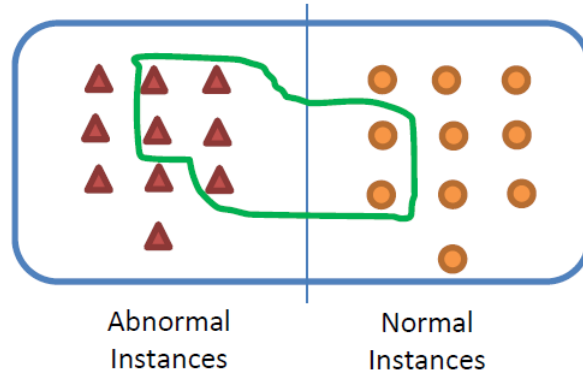


FIGURE 3.4: An example of a binary classification problem. One detection rule is covering a set of input instances.

Considering the first two factors listed above, a naïve fitness function is formed as below:

$$fitnessfunction(1) = no.ofInstancesCorrectlyDetected - no.ofErrors \quad (3.4)$$

It rewards correct classifications made by an individual rule and penalises incorrect ones, but does not take into account class imbalance, or overall coverage in combination with other rules.

One of the challenges that needed to be overcome in this research was to design a fitness function that can measure the degree of cooperation between the rules. This can be addressed by giving a higher score to those rules that detect anomalies which were not identified by many other rules. Thus, a compact set of unique rules can be found as opposed to collecting too many rules, that cover overlapping regions of search space. Moreover, to counter imbalance in data, particularly in network intrusion detection datasets, where the malicious traffic instances are far fewer than the instances of normal traffic, some adjustments are needed to weigh errors in the fitness function. This is important for intrusion detection because without addressing this issue, the system will not be properly trained using the limited number of anomalous instances.

To cover all above mentioned factors for designing the fitness function, the following functions are proposed:

$$\begin{aligned}
 fitnessfunction(2) = & (2p - 1) * no.ofInstancesCorrectlyDetectedby1rule \\
 & + (2p - 2) * no.ofInstancesCorrectlyDetectedby2rules \\
 & + \dots + (2p - p) * no.ofInstancesCorrectlyDetectedbyAllRules \\
 & - (p) * \left(\frac{no.ofabnormalInstances}{no.ofnormalInstances} \right) * no.ofErrors
 \end{aligned} \tag{3.5}$$

$$\begin{aligned}
 fitnessfunction(3) = & (p) * no.ofInstancesCorrectlyDetectedby1rule \\
 & + (p - 1) * no.ofInstancesCorrectlyDetectedby2rules \\
 & + \dots + (1) * no.ofInstancesCorrectlyDetectedbyAllRules \\
 & - (p) * \left(\frac{no.ofabnormalInstances}{no.ofnormalInstances} \right) * no.ofErrors
 \end{aligned} \tag{3.6}$$

where p is the number of individuals (rules) in the population. Also knowing that a fitness function is measuring the fitness of one individual (rule) in the population:

- *no.ofInstancesCorrectlyDetectedby1rule* is the number of abnormal instances being correctly detected by only the rule under evaluation. This rewards the rule for detecting unique instances.
- *no.ofInstancesCorrectlyDetectedby2rule* is the number of abnormal instances being correctly detected by the rule under evaluation and another rule in the population.
- ...
- *no.ofInstancesCorrectlyDetectedbyAllrule* is the number of abnormal instances being correctly detected by all the rules in the population.
- and *no.ofErrors* is the number of normal instances incorrectly detected as anomalies.

To better understand the proposed fitness functions, suppose that the proposed system tries to generate an optimised set of three rules for detection of abnormal

instances in an example problem presented in Figure 3.5. Below, the calculations and outputs of the three proposed fitness functions for the green rule are provided.

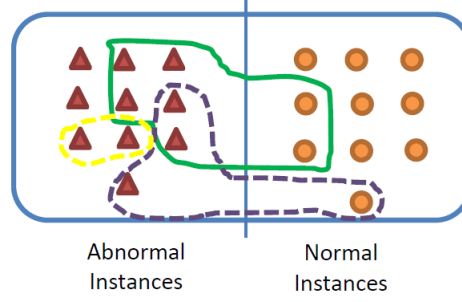


FIGURE 3.5: A ruleset with three rules (corresponding to regions marked in green, yellow and purple respectively) for detection of abnormal instances.

The value of parameters in the proposed fitness functions (i.e. equations 3.5 and 3.6) for the green rule in Figure 3.5 is as follows:

- $p = 3$
- $no.ofabnormalInstances = 10$
- $no.ofnormalInstances = 10$
- $no.ofInstancesCorrectlyDetectedby1rule = 3$
- $no.ofInstancesCorrectlyDetectedby2rule = 2$
- $no.ofInstancesCorrectlyDetectedbyAllrule = 0$
- $no.ofErrors = 2$

Accordingly, the output of the three fitness functions is calculated as follows:

- $fitness\ function(1) = 5 - 2 = 3$
- $fitness\ function(2) = (2*3-1)* 3$
 $+ (2*3-2)* 2$
 $+ (2*3-3)* 0$
 $- 3*\frac{10}{10}* 2 = 17$

$$\begin{aligned}
\bullet \text{ fitness function}(3) &= 3 * 3 \\
&+ 2 * 2 \\
&+ 1 * 0 \\
&- 3 * \frac{10}{10} * 2 = 7
\end{aligned}$$

As can be seen, fitness function (2) has higher weightings for scoring the rules than fitness function (3) and both functions use the same weight for penalising the rules for false positives. As a result, fitness function (2) puts more emphasis on detection of anomalies than on reducing false alarms. Based on the pre-set objectives in different environments, one can decide on the sensitivity of the system on detecting abnormal instances or preventing the false positives. To take care of the class imbalance problem in the data, a ratio of the *no.of abnormalInstances* to the *no. of normalInstances* is applied to the error in both functions. Therefore, when the number of normal instances is greater than the number of abnormal instances in a dataset, if a detection rule misclassifies a normal instance, it receives less penalties in comparison to a dataset with twice the abnormal records and a smaller number of normal records. This is shown in Figure 3.6.

To provide a better understanding of how different fitness functions select a good solution for a classification problem, these function are explored in different scenarios which are simple and easy to understand and at the same time, representative of a variety of situations.

Evaluating Fitness Functions: In this section, the proposed fitness functions ((2) and (3)) will be investigated on balanced and imbalanced data with different complexity of rulesets (that can be measured by the amount of coverage and false positive ratio). For this purpose, five different scenarios are designed (i.e. presented in Table 3.1) as follows:

1. A balanced data to test a set of rules with good coverage and low false positives
2. A balanced data to test a set of rules with poor coverage and very low false positives

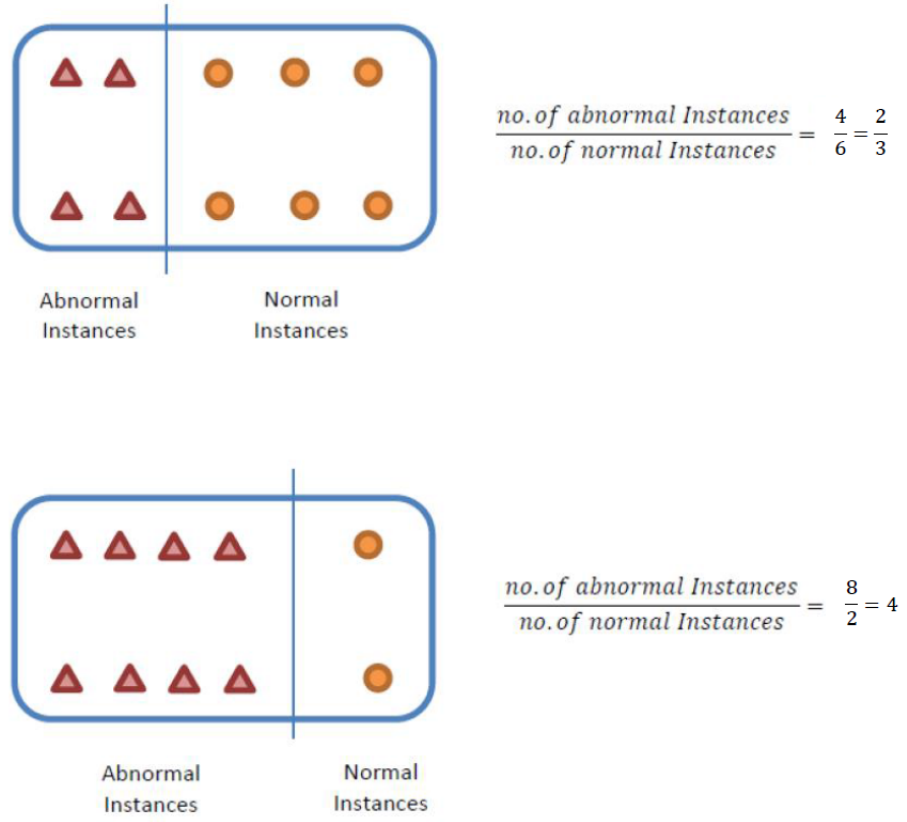


FIGURE 3.6: Weighting on the false positives in the proposed fitness functions (fitness function (2) and (3)) to take care of the problem of the class imbalance in a dataset.

3. A balanced data to test a set of rules with good coverage but high false positives
4. A balanced data to test a set of rules with full coverage but high false positives
5. An imbalanced data to test a set of rules with full coverage but high false positives

In these scenarios, if a ruleset classifies approximately 80 percent of the abnormal instances, then it provides a “good coverage”. Similarly, if all the abnormal instances are classified correctly by a ruleset, then a “full coverage” is obtained.

The results are then compared to the support-confidence function (i.e. equation (3.7)), as the support-confidence framework (Lu & Traore, 2004; Gong et al., 2005) is the most commonly used fitness function in the literature.

Supposing a rule is in the form of *if A then B*, the support-confidence framework is defined as follows:

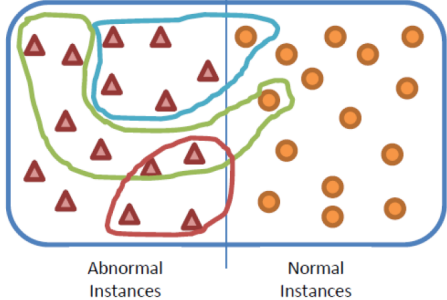
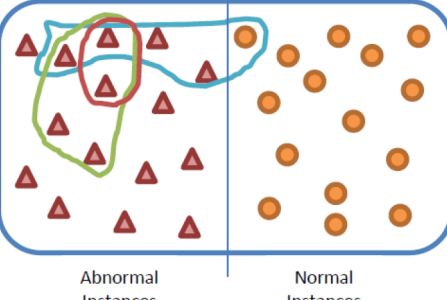
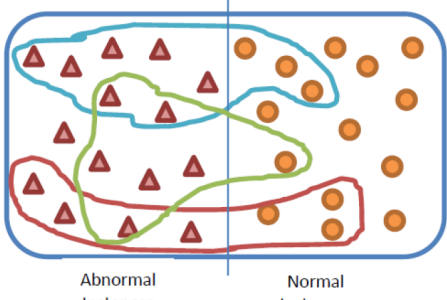
$$\begin{aligned}
 support &= |A \& B| / N \\
 confidence &= |A \& B| / |A| \\
 fitness &= w1 * support + w2 * confidence
 \end{aligned} \tag{3.7}$$

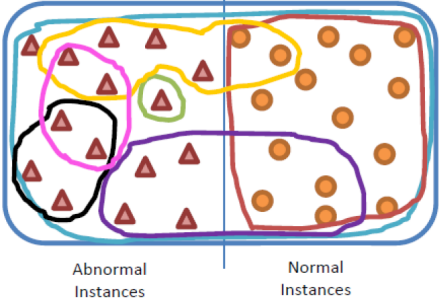
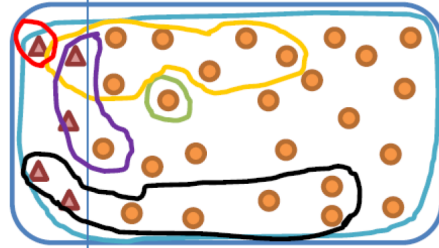
where *support* is the ratio of the number of records covered by the ruleset to the total number of records in the dataset and *confidence* factor is calculated by dividing the number of records covered by the ruleset by the number of records that only match the condition *A*. To control the balance between the two metrics, *w1* and *w2* are used in the *fitness* equation. Two sets of values are used for *w1* and *w2* in the literature (Pal & Hasan, 2012; Pawar & Bichkar, 2014; Jadhav & Gaikwad, 2014; Gong et al., 2005; Zuo et al., 2002), when the support-confidence framework is utilised for designing an IDS: $w1=0.2$, $w2=0.8$ and $w1=0.5$, $w2=0.8$ respectively. Therefore, these two sets of weights are used and evaluated against the fitness functions proposed in this research.

Since the support-confidence function rewards the rules with best coverage and least errors, it will be useful in problems where the goal is to find a set of high quality rules and the interaction among the rules is not important. However, ESR-NID searches for a set of rules, which cooperatively work well together. Thus, in fitness function (2) and (3), those rules that detect anomalies which were not detected by other rules, will be rewarded. To also control the balance of data, some weightings on scores and errors are considered in the functions. Below, using the scenarios in Table 3.1, the proposed fitness functions were evaluated against the support-confidence framework.

A fundamental goal of IDSs is to increase detection rate and decrease false alarm rate. Accordingly, ESR-NID should generate a set of rules with good coverage of abnormal records (high detection rate) and the least coverage of normal instances (low false positive rate).

TABLE 3.1: A comparison of fitness functions for different scenarios.

Scenario description	Rule	Support-confidence framework		fitness function (2)	fitness function (3)
		w1=0.2 w2=0.8	w1=0.5 w2=0.8		
Scenario 1: <i>Good coverage/not many errors</i> 	Green	0.72	0.78	25	13
	Blue	0.70	0.75	22	12
	Red	0.82	0.86	18	10
Scenario 2: <i>Not good coverage/hardly any errors</i> 	Green	0.83	0.88	21	11
	Blue	0.66	0.70	14	6
	Red	0.81	0.83	7	3
Scenario 3: <i>Good coverage/lots of errors</i> 	Green	0.83	0.88	21	11
	Blue	0.66	0.70	14	6
	Red	0.81	0.83	7	3

<p>Scenario 4: <i>Full coverage/lots of errors</i></p>  <p>Abnormal Instances Normal Instances</p>	Black	0.82	0.86	46	22
	Pink	0.82	0.86	44	20
	Yellow	0.90	0.65	44	14
	Green	0.80	0.81	12	6
	Purple	0.42	0.46	20	-4
	Blue	0.50	0.65	72	-18
	Red	0	0	-105	-105
<p>Scenario 5: <i>Imbalanced data</i></p>  <p>Abnormal Instances Normal Instances</p>	Purple	0.54	0.56	17.8	7.8
	Red	0.80	0.81	10	5
	Black	0.24	0.26	14	4
	Green	0	0	-1.2	-1.2
	Yellow	0.12	0.13	1.8	-3.2
	Blue	0.16	0.21	19	-6

In the first scenario, a balanced dataset is provided for a sample binary classification problem. It is assumed that the system produced a ruleset of three rules with a good coverage of abnormal instances and 2 misclassified normal records. The aim in this scenario is to see how different fitness functions ranked the rules considering the their cooperation in the classification task. A rule that detects abnormal instances, which were not identified by others should be ranked higher. In this scenario, the green rule has the best coverage (6 abnormal instances, which includes 4 unique points that were not detected by any other rules) with only one false positive among other rules. As an example, details of the calculations of fitness for the green rule in the first scenario are explained below:

- support = $|A \& B|/N$
 confidence = $|A \& B|/|A|$
 Support-confidence framework = $w1 * support + w2 * confidence$
 for $w1 = 0.2$, $w2 = 0.8$,
 Support-confidence framework = $0.2 * \frac{6}{30} + 0.8 * \frac{6}{7} = 0.72$
 for $w1 = 0.5$, $w2 = 0.8$,
 Support-confidence framework = $0.5 * \frac{6}{30} + 0.8 * \frac{6}{7} = 0.78$

- $p = 3$
 $no.ofabnormalInstances = 15$
 $no.ofnormalInstances = 15$
 $no.ofInstancesCorrectlyDetectedby1rule = 4$
 $no.ofInstancesCorrectlyDetectedby2rule = 2$
 $no.ofInstancesCorrectlyDetectedbyAllrule = 0$
 $no.ofErrors = 1$
 fitness function(2) = $(2*3-1)* 4$
 $+ (2*3-2)* 2$
 $+ (2*3-3)* 0$
 $- 3* \frac{15}{15} * 1 = 25$

- fitness function(3) = $3*4$
 $+ 2*2$
 $+ 1*0$
 $- 3* \frac{15}{15} * 1 = 13$

This has been found by the proposed fitness functions, whereas the support-confidence fitness function ranked the green rule as the second best rule. This is because the support fitness function aims to find the rule with maximum detection rate and minimum false positive rate. The cooperation between rules is not considered in this function and thus the blue rule which classifies more unique abnormal instances and only produces one error is ranked after the red and green rules. This becomes more problematic in the second scenario by rewarding a redundant rule which classifies the instances being already detected by some other rules.

The second scenario is presenting a situation with a redundant rule. In this scenario, the green and blue rules are providing the same coverage as offered by three rules together. Thus, the red rule would be redundant. Since the cooperation of rules is one of the main factors considered in designing ESR-NID fitness function, in this scenario, different functions are tested to see how they rank a redundant rule. The aim is to decrease the size of the optimised ruleset generated by ESR-NID by eliminating the redundant rules. This can be done with a fitness function that gives a lower rank to those rules. According to the proposed fitness functions, the green and blue rules are ranked first and the red rule is the last. On the other hand, in the support-confidence framework, the red rule is ranked before the blue rule because the cooperation of rules is not considered in this framework.

In the third scenario, there is a ruleset of three rules, which misclassifies about half of the normal instances. As mentioned before, a good classification system should prevent a high false positive rate. Based on ESR-NID's objectives, in this scenario, the green rule with an acceptable coverage of abnormal instances and only one false positive should be the best among others. Secondly, between the blue and red rules, the system should pick the blue one because of the higher number of unique abnormal records detected. As shown in the results, this ranking of rules is obtained

through the use of all four fitness functions (i.e. support-confidence framework with two different set of weights, fitness function (2) and fitness function (3)).

The fourth scenario is prepared to test some special cases like the blue rule that will just randomly guess the class of instances and yet appears to provide a full coverage of all abnormal records. A well-designed fitness function should not reward a rule like this. In this scenario, fitness function (3) still provides reasonable results by considering the two important factors of detection and false alarm rate and the cooperation of rules by prioritising the following rules: black, pink, yellow, green and purple. The blue and red rules should be the worst because the blue rule indiscriminately classifies all the instances as abnormal and the red one produces a high false positive rate. Although the latter has been taken into account in all four functions, the former is an issue in fitness function (2). Between the two variations of support-confidence functions, this is more problematic in the second one which similarly ranked the blue and yellow rules as the fourth best. This is in contrast to the first support-confidence function that ranked the yellow rule as the best among all the rules. All in all, in prioritising the best rules and penalising the worst, first support-confidence function and fitness function (3) provided more acceptable results than the other two functions.

Finally, all the above mentioned aspects should be considered in situations with imbalanced data. Thus, in the fifth scenario, these functions are examined on an imbalanced dataset. As it can be seen from the results, fitness function (3) accurately chooses the purple, red and black rules as the best and the imperfect blue rule is the lowest-ranked rule. However, this is an issue in fitness function (2) which ranked the blue rule as the best among the six rules. This is less problematic in the two variations of support-confidence function that ranked the blue rule as the fourth best rule. Similar results were produced when an experiment was conducted on an imbalanced dataset with more abnormal instances (25 instances) than normal records (5 instances) and the same issue of prioritising the imperfect blue rule has been raised in the case of using fitness function (3) and support-confidence functions.

Ind0	f0
Ind1	f1
Ind2	f2
Ind3	f3
Ind4	f4
Ind5	f5
Ind6	f6
Ind7	f7
Ind8	f8
Ind9	f9
Ind10	f10
Ind19	f19

Gen0

FIGURE 3.7: A sample of sorted population based on the fitness value in generation0.

While the fitness value indicates the quality of a rule, it is not sufficient for making a decision on which set of rules are able to cooperatively cover the search space. Therefore, a performance function is adopted to evaluate the performance of a group of rules for detection, which will be introduced next. Through this two-stage evaluation process, ESR-NID is able to find a set of classification rules for the specific dataset.

Performance Evaluation: The performance function helps the system find the optimal set of cooperating rules through the process explained below:

The generational EA in ESR-NID sorts the population of individuals by fitness in each generation (see Figure 3.7) and then using the Algorithm 4, the best value of n and the corresponding performance value will be found.

For evaluating a classifier system, a confusion matrix, which includes true positive, true negative, false positive, and false negative is usually calculated. The aim is often to maximize the true positive and true negative rates and minimize the false positive and false negative rates. There are different metrics in the literature for evaluation of classifiers including *accuracy*, *precision*, *recall*, *F-value* and *g-performance* (Weng

Algorithm 4 Finding best value of n

```

 $perfMax = 0$ 
for  $n$  from 1 to  $popSize$  do
  if  $performance(first\ n\ rules) > perfMax$  then
     $perfMax \leftarrow performance(first\ n\ rules)$ 
     $best-n \leftarrow n$ 
  end if
   $n \leftarrow n + 1$ 
end for
return  $best-n$ 
return  $perfMax$ 

```

& Poon, 2008; Sokolova et al., 2006; Seliya et al., 2009; Kotsiantis et al., 2006).

Any of these functions can be used as a performance function in the ESR-NID framework depending on the nature of the problem and the given data. For example, in network intrusion detection, the number of intrusive instances is typically a very small fraction of the total network traffic records. Similarly, in medical databases, for classifying the cancerous pixels from normal ones in mammogram images, the cancerous instances represent only a very small fraction of the entire image (Chawla et al., 2003). A fairly high detection rate of the minority class is required in these applications because, for example, the cost of misclassifying a patient with cancer as non-cancerous can be very high or missing an attack (i.e. a false negative) can cause serious damage to an organization. Therefore, ESR-NID should aim to evolve a set of rules with high true positive and true negative rates. In these cases, if ESR-NID, for example, uses accuracy as a performance function, a ruleset which labels everything with the majority class can achieve high accuracy and this will not improve the search process in ESR-NID. Therefore, for domains with imbalanced data, classification accuracy is not an appropriate performance measurement.

By providing this flexibility in the choice of performance function, ESR-NID can be configured specifically to meet the domain-specific requirements for different problems. This option can not be found in other machine learning techniques and thus makes ESR-NID a more flexible model.

In this research, to address the problem of balance in datasets, the *g-performance* is used as a performance function. The *g-performance* is calculated by the geometric

mean of the accuracies on positive and negative examples ($g = \sqrt{a^+ * a^-}$) to avoid the poor behaviour of some learners under the circumstances of having imbalanced datasets. This metric was used in [Kubat & Matwin \(1997\)](#); [García & Herrera \(2009\)](#) and [Barandela et al. \(2003\)](#) to measure the performance of classification over imbalanced datasets. Evolving the system using this measure allows the classifier to maximize the true positive and true negative rates at the same time.

$$performance\ function = \sqrt{TP_{rate} * TN_{rate}} \quad (3.8)$$

where TP_{rate} is the percentage of true positive cases correctly classified as positive, and TN_{rate} is the percentage of true negative cases correctly classified as negative. The range of values for *g-performance* is between 0 and 1, where in the case of an ideal classifier it has a value of 1.

As explained before, depending on the application domain and the system designer's needs and preferences, the performance function in ESR-NID can be changed to evolve a more specific set of rules. For instance, another performance function utilised in this research is:

$$performance\ function = \sqrt[3]{TP_{rate} * TP_{rate} * TN_{rate}} \quad (3.9)$$

which puts more emphasis on detection of anomalous records than normal instances. Using this function, one can provide higher detection rate of one class if not missing records of this class is really critical in an application.

The two-stage evaluation component of ESR-NID makes it a flexible and customisable model by providing the ability to change the fitness function and performance function based on the designer's needs and preferences. Therefore, variants of classifiers can be generated by ESR-NID for different problems. This will be shown in the experiments conducted in [Section 4.10](#).

3.1.2.5 Reproduction of A New Generation

In this section, the reproduction of a new population of individuals is explained. This process is implemented using two modules: selection and breeder. The former is responsible for choosing the individuals for reproduction based on their fitness values and the latter uses two operators (crossover and mutation) to create offspring by mixing and altering the genes of parents.

As an operational characteristic of GAs, elitism ensures that the best chromosome(s) are passed, unchanged, to the next generation and the EA does not waste time re-discovering previously discarded good candidates. This will often improve the EA performance (Reed et al., 2001; Dumitrescu et al., 2000).

Initially, ESR-NID was designed with a fixed elitism mechanism. In the fixed elitism scheme, the number of elites can be set as an input parameter (*breed.elite.0 = 10*), that is the number of candidates in a generation that should be sent to the next generation, rather than created via evolution. Alternatively the number of elites can be defined as a proportion of the population: *breed.elite-fraction.0 = 0.25*. In both, in each generation, the EA sends a fixed number of individuals to the next generation. Through this process, some potential individuals may be lost. To overcome this issue, further investigation is needed on the behaviour of the EA in each generation. Later, in Section 4.4.3, the proposed adaptive approach to address this issue will be explained. This approach aims to adaptively adjust the number of elites copied to each new generation.

The new population will consist of these individuals (*elite*) from the previous generation and *popSize-elite* newly bred individuals. The newly bred individuals are created through a chain of selection and breeding operators (Figure 3.8). Among all the available selection methods such as Random Selection, Fitness-Proportionate Selection, Stochastic Universal Sampling and Tournament Selection, the Tournament Selection method is used in ESR-NID (Goldberg & Deb, 1991). This method first chooses T individuals randomly, where T is the *tournament size*. Then, among those T individuals, it will return the fittest one. The most common setting for T

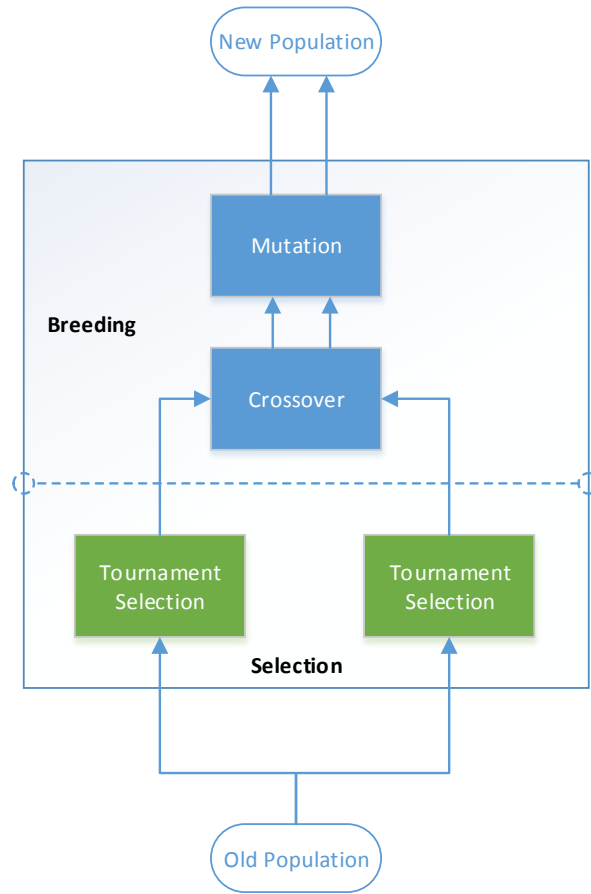


FIGURE 3.8: The breeding procedure (Luke et al., 2006).

as well as the chosen value for ESR-NID, is two. In Figure 3.8, the leaves of the tree (green boxes) are selection methods, responsible for choosing individuals from the old population. These individuals are then handed to the breeding operators (crossover and mutation). Finally the breeder module will send the new individuals to the new population.

In ESR-NID, a two-point crossover, with the crossover points restricted to the boundaries between fields, is used in the breeder component. As can be seen from Figure 3.9, two crossover points are selected randomly. To create the next generation, the field(s) from the beginning of the chromosome to the first crossover point is copied from parent A. Then the field(s) from the first to the second crossover point is copied from parent B and finally the rest of field(s) are copied from the first parent. All

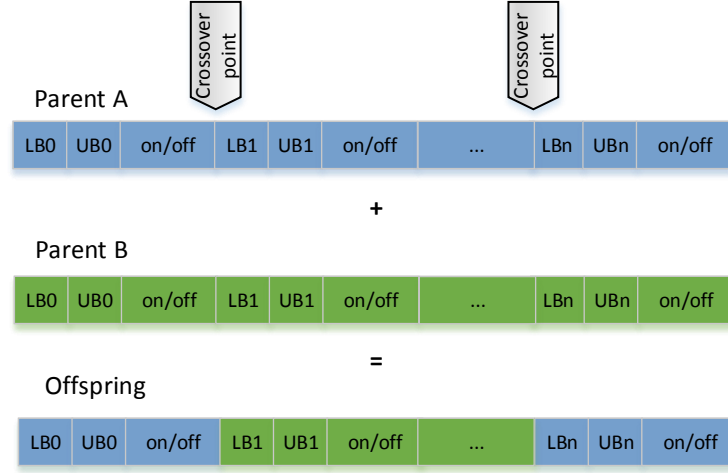


FIGURE 3.9: Two point crossover

the individuals used by the breeder module undergo a two-point crossover in the ESR-NID framework.

Additionally, depending on which gene is to be mutated, different mutation operators are also used. The upper and lower bound values are mutated by Uniform mutation. However, each gene is only mutated with a certain probability, which is defined as an input parameter to the algorithm (i.e. mutation probability). Uniform mutation sets the gene to a random value between its minimum and maximum legal values. On the other hand, the regulatory value, which has a binary value, is mutated by bit-flipping. The bit-flipping mutation simply flips a randomly selected bit.

3.1.3 Post-processing

As explained before, in each generation of the evolutionary process, the best performing ruleset can be found using a two-stage evaluation method. When the evolutionary process ends, the optimal performance value achieved along the generations will be found and the corresponding ruleset will be reported as the output of the second stage in ESR-NID. However, to produce a more concise ruleset for the use

of IDS, similar rules are removed from the final optimised ruleset through a post-processing step. Similarity between rules is determined by comparing the values of their active features, using a user-defined cut-off for the number of digits of precision in floating-point numbers as features in our approach are continuous real values. An example of this process is shown in Figure 3.10. A cut-off point of 2 digits is defined to remove similar rules from the generated optimal ruleset by ESR-NID. As can be seen from this figure, first, rule0 in the optimised ruleset is moved to the final ruleset. Then, rule1 in the optimised ruleset will be checked against the current rule(s) in the final ruleset (i.e. only rule0) and because the ranges of active features up to two decimal points in rule1 are similar to rule0 in the final ruleset, rule1 will be removed. This process continues until all the rules in the optimised ruleset are checked and consequently either moved to the final ruleset or removed.

3.2 Summary

In this chapter, a GA-based framework, called ESR-NID, was introduced for generating optimised rulesets for network intrusion detection. The input data to ESR-NID first goes through a pre-processing stage which prepares the data for the evolutionary stage. At the starting point of the evolutionary process, the seed selection and initializer modules are responsible for creating some initial individuals for the EA. If the number of selected individuals by seed selection is less than the predefined population size, the initializer is set to randomly generate the rest of individuals in the population. Next, for a fixed number of generations (i.e. the termination condition of EA), the evaluator, selection and breeder modules will iteratively attempt to find the best solutions to the problem. One of the contributions in this work is the new form of individual representation using the regulatory genes to represent statistical detector rules for IDSs. The binary regulatory value for each feature in the chromosome enhances the evolutionary algorithm by allowing it to find the required features to solve the problem. ESR-NID also contributes to the genetic based machine learning techniques by proposing a two-stage evaluation process to derive a set of classification rules from the provided data. In the first stage, the

Optimised ruleset

0	0.3245	0.6523	on	0	0.2588	off	0.3358	0.9856	on
1	0.3247	0.6533	on	0.25	0.2645	off	0.3350	0.9886	on
2	0.5780	0.7580	on	0.2306	0.2596	on	0.8563	0.9523	on
3	0.5963	0.6523	on	0.336	0.4488	off	0.8566	0.9556	off
4	0.3456	0.6231	off	0	0.2896	on	0.6725	0.8560	off
5	0.3245	0.6523	off	0	0.2855	on	0.3358	0.9856	off

Final ruleset

Cut-off = 2 digits

0	0.3245	0.6523	on	0	0.2588	off	0.3358	0.9856	on
1									
2									
...									

...

Final ruleset

0	0.3245	0.6523	on	0	0.2588	off	0.3358	0.9856	on
1	0.5780	0.7580	on	0.2306	0.2596	on	0.8563	0.9523	on
2	0.5963	0.6523	on	0.336	0.4488	off	0.8566	0.9556	off
3	0.3456	0.6231	off	0	0.2896	on	0.6725	0.8560	off

FIGURE 3.10: Post-processing of final optimised ruleset.

best individual rules in the population are found based on a fitness value. Then, in the second stage, the best ruleset will be found using a performance function. For this, ESR-NID uses a Michigan style rule discovery method, where each individual is evaluated in cooperation with other individuals and the best set of rules are found. Toward implementation of this two-stage evaluation process, three fitness functions were proposed. These functions were designed to address the challenges involved in this research including the class imbalance in data and a tool to measure the degree of cooperation between the rules.

As discussed, the two distinct features of ESR-NID were an advanced two-stage evaluation approach and an adaptive elitism mechanism. These make ESR-NID a flexible model that can automatically derive a set of classification rules from the provided dataset.

In the next chapter, a better understanding of the behaviour of the proposed system will be obtained and its rule generation ability will be tested on some synthetic data.

Chapter 4

Performance Evaluation of ESR-NID on Synthetic Problems

4.1 Introduction

For evaluation, training and testing of detection and classification systems, it is essential to have suitable data. When synthetic data is generated to evaluate an approach, the properties of data can be controlled to meet various conditions and as the ground truth is known, the performance of the proposed approach can be validated rigorously.

In this chapter, first, the choice of dataset is introduced. There are four different scenarios designed to generate data for system evaluation. After introducing the evaluation strategy used in this chapter, ESR-NID will be evaluated against the first dataset generated for a medium size problem. Through this preliminary investigation of the performance of ESR-NID, an adaptive elitism mechanism is proposed for enhancing the proposed algorithm.

Next, an algorithm tuning process is conducted in order to find the best fitness function and GA parameter settings for ESR-NID. The chosen fitness function and the best values found for GA parameters are then used for all subsequent experiments.

In the experiments in this chapter, five machine learning methods from the literature were also used for comparing the performance of ESR-NID. These are C4.5 (a decision tree learner), IB1 (an instance based learner) and RIPPER (a rule induction method) from the category of non-GA-based algorithms and Genetic Algorithm based Classifier System with Adaptive Discretization Intervals (GASSIST-ADI) (Bacardit & Garrell, 2003) and Memetic Pittsburgh Learning Classifier System (MPLCS) (Bacardit & Krasnogor, 2009b) from the category of GA-based rule learning techniques. For the first three classifiers, the Weka version of the C4.5 algorithm, known as J48, an implementation of RIPPER called JRip, and kNN, a linear nearest neighbours search algorithm were used. The other two classifiers, GASSIST-ADI and MPLCS, are provided by the KEEL software (Alcala-Fdez et al., 2009). For each of these five algorithms, parameter tuning is also carried out to find the respective best performing model.

4.2 Choice of Dataset

The synthetic data used in all the experiments in this chapter is generated using a Python script, which provides the ability to easily generate data for more complex classification problems by only modifying the input parameters such as the number of data points, the dimensionality, and the number of clusters needed for evaluating the proposed system. Moreover, as the final optimised rulesets generated by ESR-NID are small for these simple to complex scenarios, they can be evaluated against the ground truth.

The four scenarios designed for evaluating ESR-NID in this chapter are listed below:

- A 3-dimensional medium size problem with one normal cluster and two anomalous clusters.
- A 3-dimensional problem with two normal clusters and five anomalous clusters.
- A problem with 6 input features (6-dimension).

- A problem with 12 input features (12-dimension).

More details of these scenarios are explained in the following sections.

4.2.1 A 3-Dimensional Medium Size Problem

As depicted in Figure 4.1, a three dimensional problem is designed in this section to evaluate ESR-NID. Using three features, the generated data can be easily visualised for the preliminary experiments. This is not possible for higher dimension data used in subsequent experiments.

In this problem, 500 data points for normal records (lighter green area) and two clusters of 250 points for anomalous records (darker red area) with some overlap with the green region, are generated to test the accuracy of ESR-NID in detecting the anomalous records. The overlap between the normal and anomalous clusters is provided to simulate the similarities between normal and abnormal behaviours in real-world problems. The rules used to generate the data for the problem in Figure 4.1 are as follows:

Green: *if $f1 \in [0, 0.3]$ and $f2 \in [0.3, 0.6]$ and $f3 \in [0, 0.2]$ then normal*

Red: *if $f1 \in [0, 0.3]$ and $f2 \in [0, 0.2]$ and $f3 \in [0.4, 0.8]$ then anomaly*
if $f1 \in [0, 0.1]$ and $f2 \in [0.5, 0.9]$ and $f3 \in [0, 0.3]$ then anomaly

As there are only two clusters of anomalous records in this problem, ESR-NID does not require a large set of rules for generating an acceptable classification rate and thus the generated rules by the system and their feature boundaries can be easily compared to the above listed rules for the Red regions.

4.2.2 A 3-Dimensional Problem with More Clusters of Hits

To test the ability of ESR-NID to generate an adequate number of rules when there are more clusters of hits (anomalous records), a problem with two clusters of normal

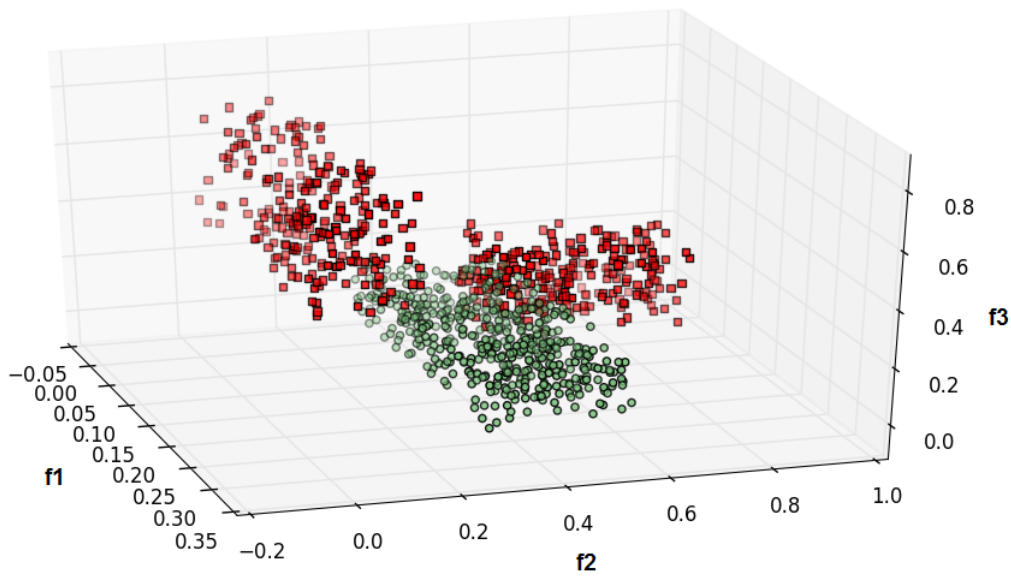


FIGURE 4.1: A medium-size problem. Red (darker) data points represent “anomalous” records, and green (lighter) points are “normal”.

records (lighter green area) and five clusters of anomalous records (darker red area), is constructed as shown in Figure 4.2.

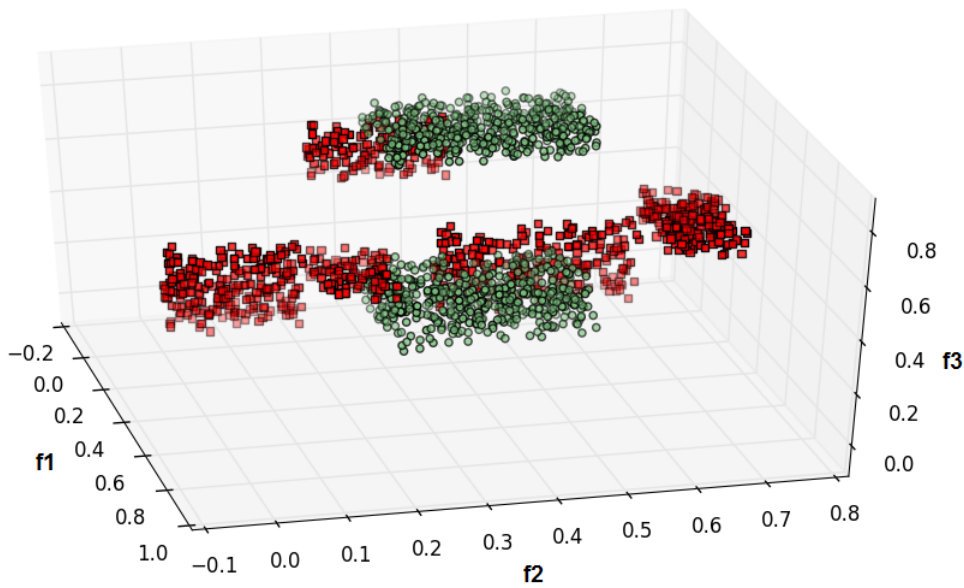


FIGURE 4.2: A problem to test the performance of ESR-NID in detection of more clusters of hits. This problem includes two “normal” clusters and five “anomalous” clusters.

To generate this dataset, the following rules were the input to the Python script used for data generation in this chapter:

Green: *if $f1 \in [0, 0.3]$ and $f2 \in [0.3, 0.6]$ and $f3 \in [0.7, 0.8]$ then normal*

if $f1 \in [0, 0.3]$ and $f2 \in [0.3, 0.6]$ and $f3 \in [0, 0.2]$ then normal

Red: *if $f1 \in [0, 0.1]$ and $f2 \in [0.4, 0.7]$ and $f3 \in [0, 0.3]$ then anomaly*

if $f1 \in [0, 0.1]$ and $f2 \in [0, 0.2]$ and $f3 \in [0, 0.3]$ then anomaly

if $f1 \in [0.5, 0.8]$ and $f2 \in [0.6, 0.7]$ and $f3 \in [0.6, 0.7]$ then anomaly

if $f1 \in [0.1, 0.2]$ and $f2 \in [0.2, 0.4]$ and $f3 \in [0.6, 0.8]$ then anomaly

if $f1 \in [0.7, 0.9]$ and $f2 \in [0.1, 0.2]$ and $f3 \in [0.6, 0.7]$ then anomaly

4.2.3 Problems With More Input Features

In 2- or 3-dimensional spaces, it is usually relatively easy to understand the properties of data and thus they are easier problems for the classifiers. To get a deeper understanding of the behaviour of ESR-NID on high-dimensional real-valued data, two sets of synthetic data were generated for problems with 6 and 12 input features. The aim is to evaluate the performance of ESR-NID to generate classification rules for more complex problems. High dimensional data points are typically derived from complex real-world objects such as images ([Hinneburg et al., 2000](#)). In the computer security research community, researchers tend to use two well-known datasets for the evaluation of their techniques for intrusion detection problem: the DARPA-Lincoln dataset and the KDD99 dataset ([Shafi, 2008](#)). Investigation on the number of attributes being used in these research activities showed that a high dimensional dataset in this area consists of 41 features (i.e. the number of features for each vector in KDD99 dataset). Since several features have higher possibilities to be involved in network intrusions, some select a smaller set of features for intrusion detection. For example, in [Depren et al. \(2005\)](#), six basic features of KDD99 dataset were used and [Gong et al. \(2005\)](#) utilised seven features of the DARPA dataset in their GA-based classification approaches.

The following two datasets are generated for the evaluation of ESR-NID against problems with more input features to show that the proposed system can also be used on higher dimensional datasets.

First, the following rules were used to generate a dataset with 6 input features:

Green: *if $f1 \in [0, 0.3]$ and $f2 \in [0.3, 0.6]$ and $f3 \in [0, 0.2]$ and $f4 \in [0, 0.3]$ and $f5 \in [0.3, 0.6]$ and $f6 \in [0, 0.2]$ then normal*

Red: *if $f1 \in [0, 0.3]$ and $f2 \in [0, 0.2]$ and $f3 \in [0.4, 0.8]$ and $f4 \in [0, 0.3]$ and $f5 \in [0, 0.2]$ and $f6 \in [0.4, 0.8]$ then anomaly*

if $f1 \in [0, 0.1]$ and $f2 \in [0.5, 0.9]$ and $f3 \in [0, 0.3]$ and $f4 \in [0, 0.1]$ and $f5 \in [0.5, 0.9]$ and $f6 \in [0, 0.3]$ then anomaly

Then, a more complex problem with 12 features was designed using the following rules. In this problem, 2 clusters of normal records and 3 clusters of anomalous records were considered.

Green: *if $f1 \in [0, 0.1]$ and $f2 \in [0.1, 0.2]$ and $f3 \in [0.2, 0.3]$ and $f4 \in [0.3, 0.4]$ and $f5 \in [0.5, 0.6]$ and $f6 \in [0.1, 0.2]$ if $f7 \in [0, 0.5]$ and $f8 \in [0, 0.3]$ and $f9 \in [0, 0.5]$ and $f10 \in [0, 0.3]$ and $f11 \in [0.9, 1]$ and $f12 \in [0.6, 0.9]$ then normal*

if $f1 \in [0.2, 0.3]$ and $f2 \in [0.2, 0.3]$ and $f3 \in [0.2, 0.4]$ and $f4 \in [0.3, 0.8]$ and $f5 \in [0.4, 0.7]$ and $f6 \in [0.6, 0.7]$ if $f7 \in [0.7, 0.9]$ and $f8 \in [0.8, 0.9]$ and $f9 \in [0.9, 1]$ and $f10 \in [0.1, 0.2]$ and $f11 \in [0.1, 0.6]$ and $f12 \in [0, 0.2]$ then normal

Red: *if $f1 \in [0, 0.3]$ and $f2 \in [0.3, 0.5]$ and $f3 \in [0.1, 0.3]$ and $f4 \in [0.3, 0.4]$ and $f5 \in [0.6, 0.9]$ and $f6 \in [0.7, 0.9]$ if $f7 \in [0.1, 0.3]$ and $f8 \in [0.1, 0.3]$ and $f9 \in [0.1, 0.3]$ and $f10 \in [0.1, 0.3]$ and $f11 \in [0.1, 0.7]$ and $f12 \in [0.1, 0.3]$ then anomaly*

if $f1 \in [0, 0.5]$ and $f2 \in [0, 0.2]$ and $f3 \in [0.1, 0.5]$ and $f4 \in [0.2, 0.6]$ and $f5 \in [0.5, 0.7]$ and $f6 \in [0.5, 0.6]$ if $f7 \in [0.7, 0.9]$ and $f8 \in [0.8, 0.9]$ and $f9 \in [0.9, 1]$ and $f10 \in [0, 0.2]$ and $f11 \in [0.1, 0.2]$ and $f12 \in [0.1, 0.2]$ then anomaly

if $f1 \in [0, 0.1]$ and $f2 \in [0.1, 0.2]$ and $f3 \in [0.2, 0.3]$ and $f4 \in [0.3, 0.4]$ and $f5 \in [0.5, 0.6]$ and $f6 \in [0.1, 0.2]$ if $f7 \in [0, 0.5]$ and $f8 \in [0, 0.3]$ and $f9 \in [0, 0.5]$ and $f10 \in [0, 0.3]$ and $f11 \in [0.8, 1]$ and $f12 \in [0.6, 0.9]$ then anomaly

4.3 Evaluation Strategy Against Synthetic Datasets

In the experiments in this chapter, to provide an accurate estimate of classification rate, a standard k -fold cross-validation technique is used. In k -fold cross-validation, the dataset is randomly split into k mutually exclusive subsets of approximately equal size. A predictive model is thus trained and tested k times; each time, it is trained on $k-1$ subsets and tested on the remaining one subset. Finally, the k results from the folds can be averaged to produce a single estimation. For this research, the number of folds (k) is set to 10 (i.e. a common value used in the literature) and the folds are stratified so that they contain approximately the same proportions of two types of class labels as the original dataset. This is called a stratified cross-validation. Additionally, the evaluation method used 3 different seeds to provide a total of 30 runs using the 10 folds.

For the experiments conducted in this chapter, the seed selection module in the ESR-NID framework was not used and the initial population of individuals was randomly generated by setting seed selection to “random”.

4.4 Preliminary Experiments for Performance Evaluation

In this section, the performance of ESR-NID is tested against the first problem depicted in Figure 4.1 (i.e. a 3-dimensional medium size problem). These preliminary experiments were carried out to identify and rectify any performance issue. As will be seen, it was found that a better elitism mechanism was needed.

4.4.1 Parameter Settings

For the experiments in this section, ESR-NID is configured using the following settings:

- Fitness Function = fitness function (3) (Equation (3.6))
- Performance function = $\sqrt{TP_{rate} * TN_{rate}}$
- Mutation probability = 0.1
- Population size = 10
- Generations = 300
- Elite = 25%
- Seed selection: random
- Number of runs = 30

Some preliminary experiments were conducted to find a suitable value for the number of generations and the results did not show any improvements in the system performance when the number of generations is increased.

4.4.2 System Performance

As explained before, a two-stage evaluation process is deployed in ESR-NID, where the fitness value is used for ranking the individuals in each generation and a performance function is responsible for finding the best set of individuals who are working well cooperatively to cover the search space. At the end of the evolutionary run, the ruleset that produced the best performance value will be reported as the optimised ruleset.

Figure 4.3, shows the performance of ESR-NID during the learning stage for only one fold of the dataset generated for the problem shown in Figure 4.1. A performance value of nearly 58% is seen for the initial population of rules (i.e. generation 1). As shown by the graph, there is a lot of fluctuation in the performance of ESR-NID in the first 100 generations. This issue will be explained next.

As initial ESR-NID uses a fixed elitism approach, in each generation, a proportion of the population (*elite* = 25%) will be sent to next generation. Consequently, this

may lead to loss of some of the individuals from the best performing set of rules between generations (found using the performance function value). To provide an illustrative example of this issue, a fixed elitism mechanism is applied to a population of individuals in Figure 4.4. In this example, a population of 20 individuals is sorted in descending order of fitness. To produce the next generation of individuals, a fixed elitism approach used by the algorithm loads the first five individuals from initial generation into next generation. However, the best performing ruleset found using the performance function in this example is a set consisting of the first 10 rules in the population. Therefore, in using a fixed elitism approach, half of the individuals in the best performing ruleset are lost. This is the reason that performance of the proposed evolutionary algorithm fluctuates with generations in Figure 4.3. To solve this issue, a more accurate approach is needed, which in each generation decides how many individuals are needed to be sent to next generation. For this, the performance value of the best ruleset in each generation would be a good indication of which individuals are good to keep and send to next generation.

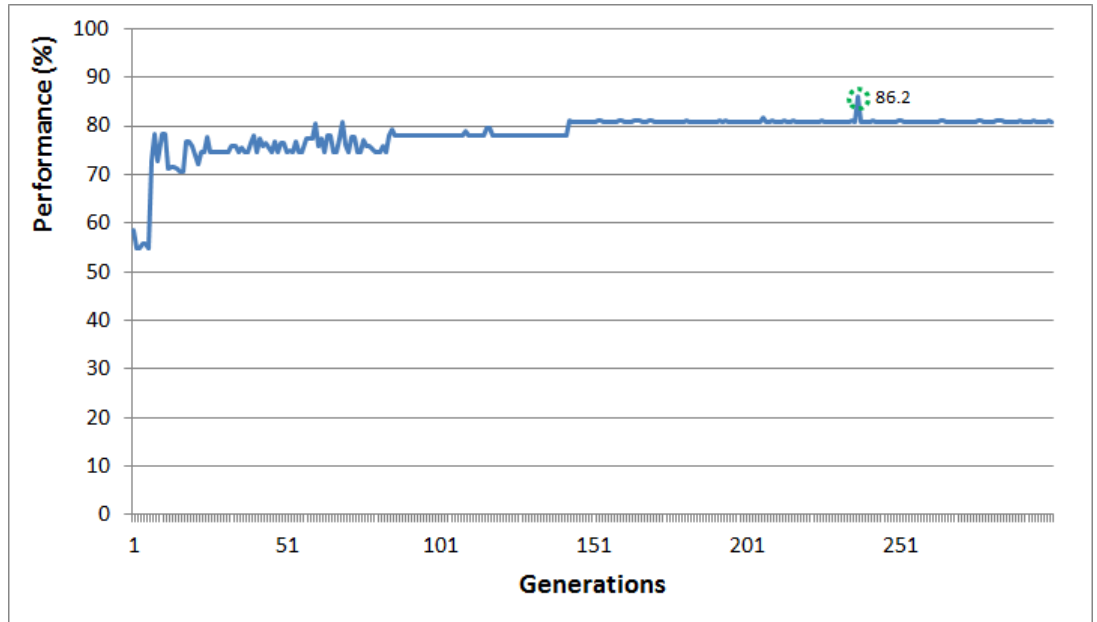


FIGURE 4.3: Performance of ESR-NID on the problem shown in Figure 4.1 vs generations for only one fold.

Next, the proposed adaptive elitism approach will be introduced to address the issue found in the above experiment. By adaptively adjusting the number of elites copied

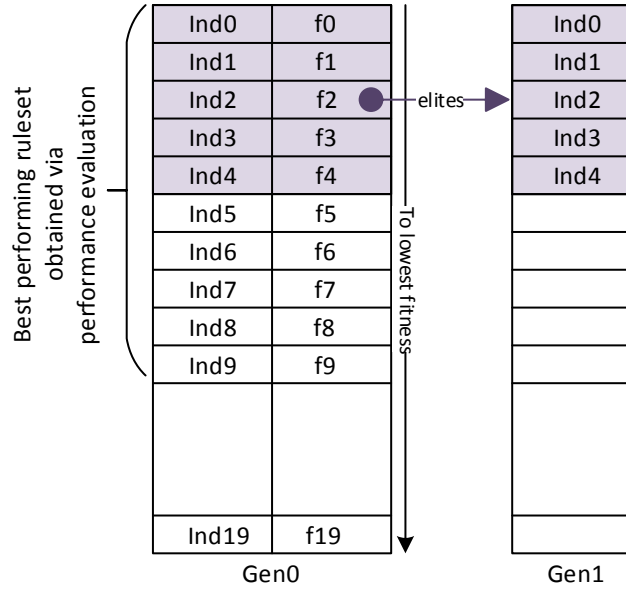


FIGURE 4.4: Fixed elitism approach (population-size=20 and elite=25%).

into each new generation, this ensures that cooperating rules are kept together and not lost from one generation to the next.

4.4.3 Adaptive Elitism Mechanism

In this section, the proposed adaptive approach to address the problem of losing good rules from a previous generation is described. The aim is to adaptively adjust the number of elites copied into each new generation. This also means that there is no need to select some arbitrary number of elites apriori. Thus, in each generation, the top n rules are designated elite, where n is chosen to maximize the performance measure as shown in Algorithm 5. A sorted population (*sortedPop*) of individuals based on the fitness value will be given to this algorithm, which in a loop with a maximum of *popSize* iterations checks the performance of first n rules in the sorted population. n is initialized to one and will be incremented by one in each iteration. If the performance of first n rules is greater than the *perfMax* (i.e. initialized to zero), the performance of first n rules will be assigned to the *perfMax* and the *elite*

value will be equal to n . Finally, the *elite* value will be returned as the output of the algorithm.

Algorithm 5 Finding elite value

```

sortedPop = sorted population pop based on fitness values
perfMax = 0
for  $n$  from 1 to popSize do
    if performance(first  $n$  rules in sortedPop) > perfMax then
        perfMax  $\leftarrow$  performance(first  $n$  rules in sortedPop)
        elite  $\leftarrow n$ 
    end if
     $n \leftarrow n + 1$ 
end for
return elite

```

After finding the elite value (the value of n that gives the best performance), the breeder module will use this number to populate the next generation. The new population will consist of n best individuals from previous generation and *popSize*- n newly bred individuals. The newly bred individuals are created through a chain of selection and breeding operators.

To compare ESR-NID using the new adaptive elitism mechanism with the fixed approach used in the previous section, the algorithm is configured as before (refer to the parameter settings in Section 4.4.1) and is applied to the same problem (i.e., shown in Figure 4.1). Figure 4.5 shows the performance of ESR-NID during the learning phase for only a single fold. To provide a fair comparison with the previous experiment, the same fold is selected for presenting the results. As can be observed, the performance value is increased to more than 94% in the new enhanced system and less fluctuation across generations is seen in the performance figure.

As explained before, in ESR-NID framework, the final ruleset to be used in a classification task is the best performing ruleset found after the post-processing stage. The maximum performance value, which is the ultimate goal, indicates which ruleset in which generation is the best. Here, a final ruleset of four rules with 94.5% performance at generation 96 is the final output of ESR-NID.

As can be seen in Figure 4.5, the performance value decreases in generation 147. In ESR-NID, the breeder module uses the *elite* value found by Algorithm 5 to copy

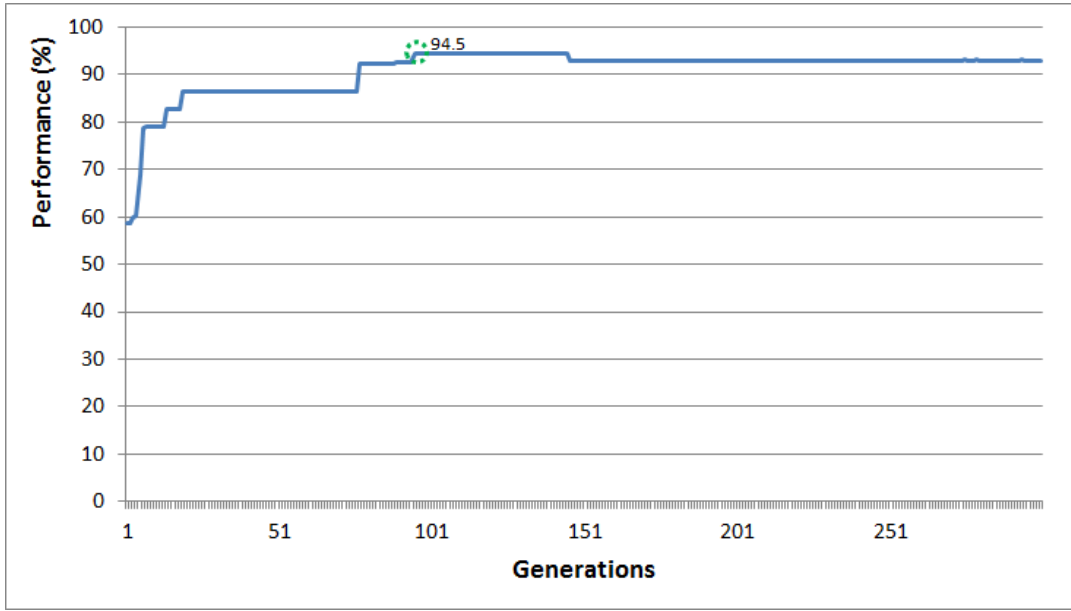


FIGURE 4.5: Performance of the enhanced ESR-NID on the problem shown in Figure 4.1 vs generations for only one fold.

the *elite*-best individuals from the previous generation. Because it is not possible to check the performance of all combinations of individuals in a population, ESR-NID can not always assure to send the optimal combination to the next generation. To provide a clearer understanding of this issue, the process of finding the elites for the next generation in ESR-NID is depicted in the top row of Figure 4.6 in contrast to a resource-consuming process in the bottom row.

In this figure, the elite value found by Algorithm 5, is five, which means the next generation (i.e. $generation_1$) will receive the 5-best individuals (i.e. ranked based on fitness value) from the current generation (i.e. $generation_0$). Among these five individuals, $individual_1$ is not contributing much to the optimised ruleset. By looking at the ranges of features, it can be concluded that rule1 is a more general version of rule0. As the proposed fitness function is designed to reward unique rules, rule0 (i.e. a more specific rule) received a higher fitness value compared to rule1, which is more general. But rule1 (with a good fitness value) is not necessarily assisting the other rules to provide the best performance value. This is because of some false positives (and as a result an inferior true negative rate) produced by rule1. If there were no limitations on computational resources, one could try all the combinations of individuals to find the best performing ruleset, which consists of $individual_0$,

*individual*₂, *individual*₃, and *individual*₄. However ESR-NID will not consider this combination, so some decreases of performance value may be seen in some generations. Although performance can drop in some generations, ESR-NID chooses the ruleset from the generation with the best performance.

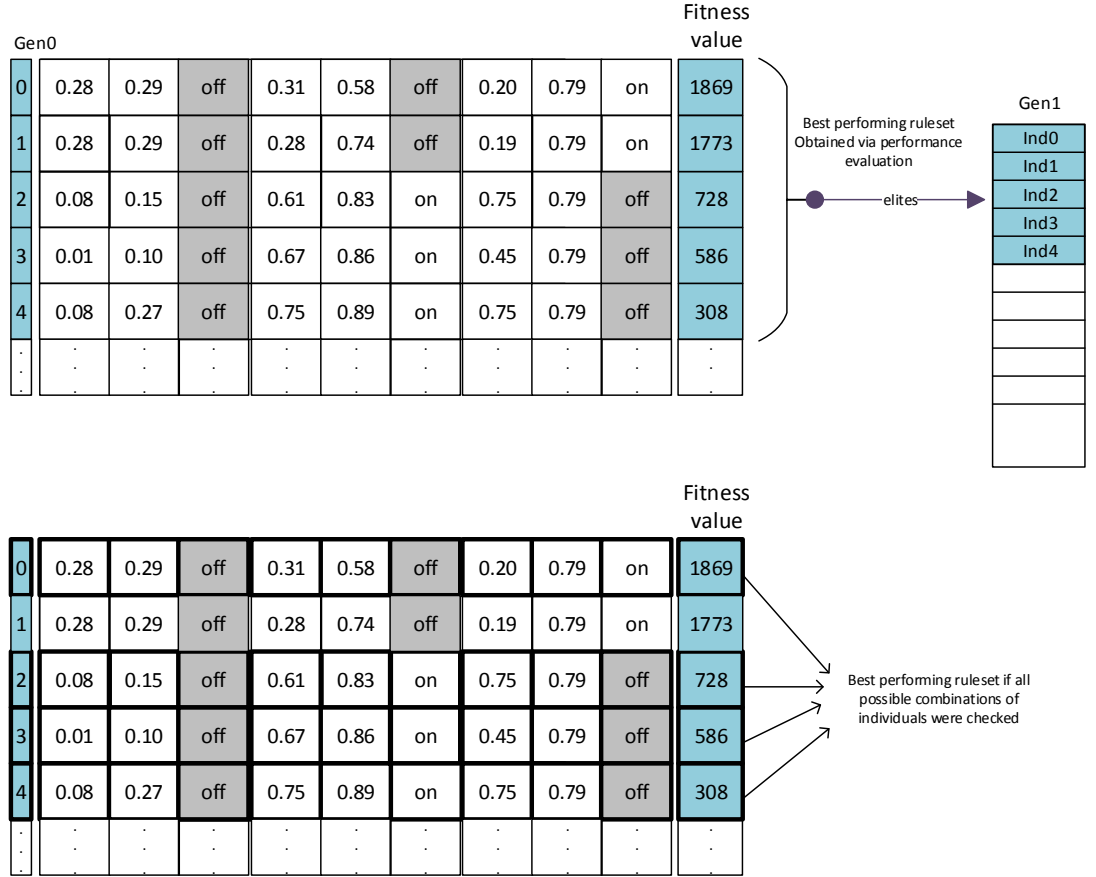


FIGURE 4.6: Adaptive elitism approach. Individuals are sorted in descending order of fitness values. $\text{Performance}(\text{Ind0}, \text{Ind2}, \text{Ind3}, \text{Ind4}) > \text{Performance}(\text{Ind0}, \text{Ind1}, \text{Ind2}, \text{Ind3}, \text{Ind4})$.

Now, to provide the final evaluation results for 30 runs, Figure 4.7 compares the performance of the enhanced evolutionary algorithm to the original algorithm, which uses the fixed elitism approach. A box plot, that shows minimum, lower quartile, median, upper quartile, and maximum performance values over the 30 runs is utilised for describing the results. As can be seen in Figure 4.7, a better performance value (ranged between 86% and 96%) is obtained when the adaptive mechanism is used.

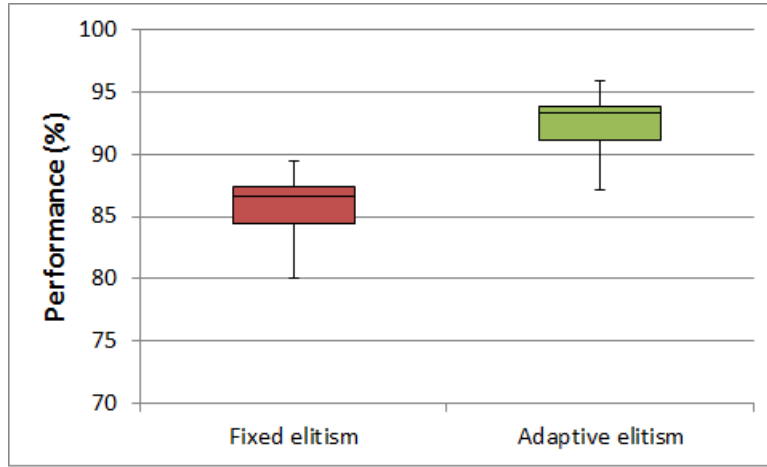


FIGURE 4.7: Comparing the performance of ESR-NID using fixed elitism and adaptive elitism mechanisms (the results are for a total of 30 runs).

Through the previous preliminary experiments, the performance of ESR-NID was evaluated for a medium size problem and after fixing one issue (elitism), it showed promising results for the classification task. Therefore, for the rest of experiments, the adaptive elitism approach will be used. However, more investigation is needed to find the best parameter settings for the algorithm. In the following section, through the process of algorithm tuning, the impact of proposed fitness functions on the performance of ESR-NID will also be evaluated.

4.5 Algorithm Tuning

In this section, the three fitness functions introduced before (equations (3.4), (3.5), (3.6)) and different combinations of mutation probabilities and population sizes are investigated for ESR-NID configuration. This step is required to identify a suitable configuration of ESR-NID for future experiments in this research. The synthetic data generated for the medium size problem in Figure 4.1 is again used in this section. The experiments are completed based on Algorithm 6, which facilitates the selection of best fitness function and set of GA parameters for use in later experiments. To keep the total number of evaluations the same (15000 evaluations) in all the experiments, when the population size increases, the number of generations will be decreased.

Algorithm 6 Algorithm tuning process

```

for each fitness function  $ff$  in (equations (3.4), (3.5), (3.6)) do
  for each mutation probability  $mu$  in (0.1, 0.3, 0.5) do
    for each population size  $popSize$  in (50, 100, 200) do
      run EA
    end for
  end for
end for
return best combination of  $ff$ ,  $mu$ ,  $popSize$ 

```

The results obtained from different combinations of mutation probabilities and population sizes for three different fitness functions used in ESR-NID are outlined in Figures 4.8-4.10. The performance figure is the *g-performance* function introduced in Equation 3.8. A box plot is used to show minimum, lower quartile, median, upper quartile, and maximum performance values over the runs. For ESR-NID, 3 runs for each of the 10 folds resulted in 30 runs for each combination of mutation rate and population size. In addition to performance figures, the average number of rules needed in ESR-NID is also presented in these figures. In each figure, the best configuration of ESR-NID is selected and marked based on the five values used for presenting the results. When the performance range is wide for a specific combination of fitness function, mutation probability and population size, that combination is considered less reliable as it does not provide an acceptable performance value for all the runs.

Based on the results presented in Figure 4.8, by using fitness function (1) in ESR-NID, a quite wide range of performance values, with minima between about 74% and 89%, can be seen. The average number of rules also varies between 7 and 12 rules. The results are dependent on the choice of mutation rate and population size. Comparing all the nine configurations, the fourth one is chosen as the best when ESR-NID uses fitness function (1).

The variability of performance values and the average number of rules become less when using fitness function (2). As can be seen in Figure 4.9, the minimum performance value is between about 87% and 91%, depending on mutation rate and population size. The average number of rules in this figure is between 2 and 3 rules.

In this figure, the sixth combination of mutation probability and population size is considered the best when ESR-NID uses the second fitness function.

Finally, the variability of performance value using fitness function (3) is less than that for the other two fitness functions, with minima consistently around 90%, regardless of mutation rate and population size. The average number of rules here is between 3 and 4 rules. Among all the combinations of mutation probability and population size, the first configuration ($\mu=0.1$, $popSize=50$ and generations=300) is considered the most reliable one for ESR-NID using fitness function (3).

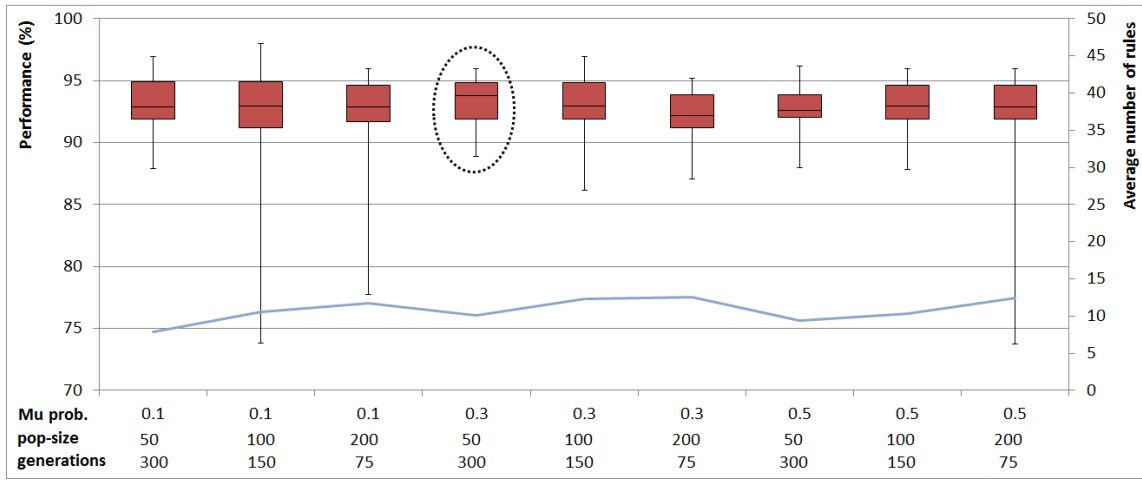


FIGURE 4.8: Performance of ESR-NID using fitness function (1). Left y-axis shows the performance value and right y-axis presents the number of rules used for classification. On x-axis, different combinations of mutation probabilities and population sizes for ESR-NID are depicted. As explained before, when the population size increases, the number of generations will be decreased to keep the total number of evaluations the same in all the experiments.

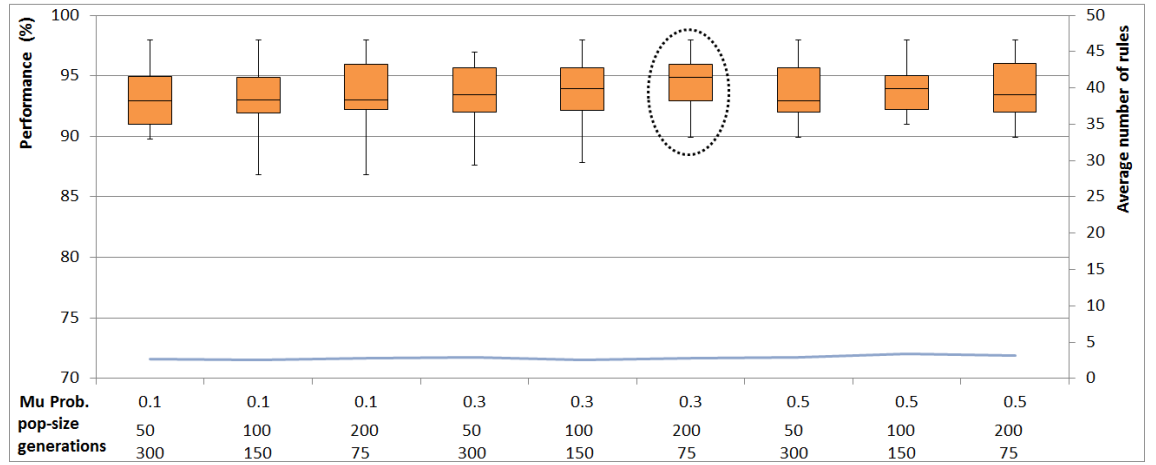


FIGURE 4.9: Performance of ESR-NID using fitness function (2) and different combinations of mutation probabilities and population sizes.

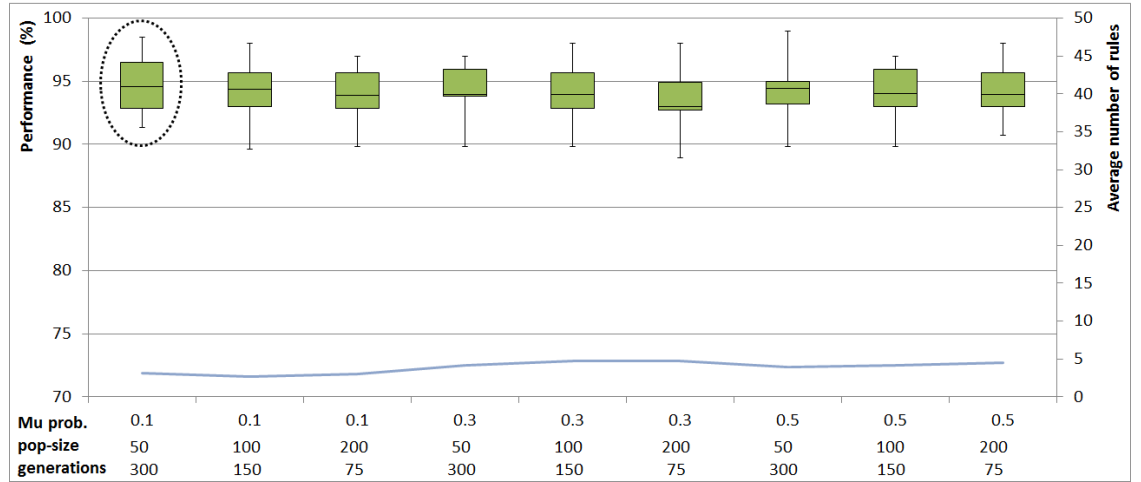


FIGURE 4.10: Performance of ESR-NID using fitness function (3) and different combinations of mutation probabilities and population sizes.

Now, to decide on the best fitness function and GA parameters, Figure 4.11 compares the best results found in each experiment on the three fitness functions. As can be seen, the least performance range is for the fitness function (1) with bigger evolved rulesets than the other two functions. These issues are solved by enhancing the fitness function. Less variation of performance values and smaller rulesets are the results achieved by fitness functions (2) and (3). However, fitness functions (3) provided better performance result than fitness function (2) and thus it can be concluded that the fitness function (3) is the most reliable among the three tested functions. Therefore, for the subsequent experiments, ESR-NID will be configured

with fitness function (3), mutation probability of 0.1 and population size of 50 and the evolutionary process will be running for 300 generations.

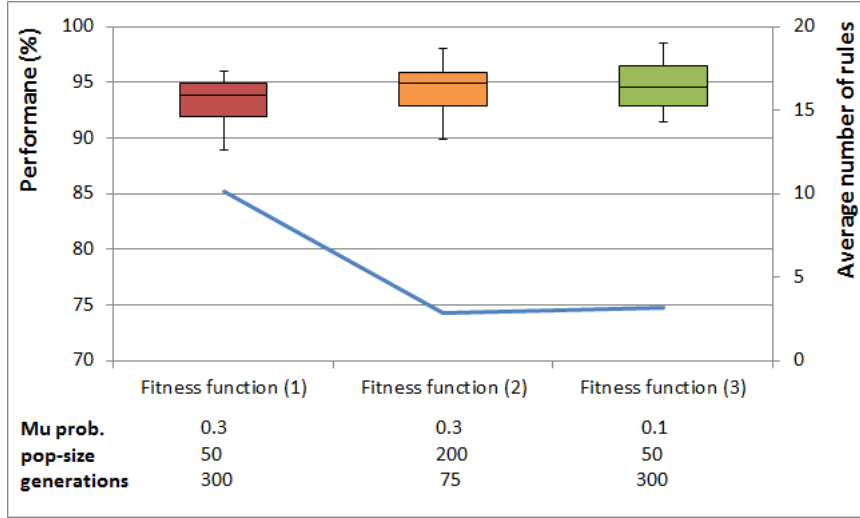


FIGURE 4.11: Performance of ESR-NID using different fitness functions.

Next, the process of parameter tuning will be done for the other three comparison methods and then the results from the best configuration of ESR-NID found in this section will be compared to the best outputs of other machine learning approaches.

4.6 Parameter Tuning for Other Classifiers

In order to properly evaluate the performance of ESR-NID and allow fair comparison between different approaches, a parameter tuning process is also carried out for J48, kNN, JRip, GASSIST-ADI and MPLCS. Since there are many possible combinations of parameter values to test, manually setting and searching for optimal values would be very time consuming. For the first three algorithms, Weka's inbuilt cross-validated parameter selection facility (CVParameterSelection) was used. CVParameterSelection uses internal cross-validation to find the best values for one or more parameters within a specified range (Sönströd et al., 2009). For the other two GA-based techniques, the parameter tuning was limited to crossover probability and population size. The parameter tuning process here is done against the medium size problem shown in Figure 4.1.

For J48, five parameters listed in Table 4.1, which might have influence on resulting decision tree (Koblar, 2012), were optimised. Table 4.1 also provides the tested values in this work, Weka’s default values and the best values obtained for designing a decision tree model for the introduced synthetic classification task.

TABLE 4.1: Parameters of the J48 machine learning algorithm used in optimization.

Parameters	Tested values	Default values	Best value
Minimum number of instances in a leaf (M)	[1..4] step size:1	2	1
Use of pruned trees (U)	yes/no	no	no
Confidence factor used in postpruning (C)	[0.02..0.5] step size:0.02	0.25	0.02
Subtree raising operation in postpruning (S)	yes/no	yes	no
Use of binary splits (B)	yes/no	no	no

Similarly, for kNN, the number of neighbours, k, and the weighting technique are the two evaluated parameters for kNN configurations (as presented in Table 4.2). Two weighting methods included in Weka implementation are inverse-distance weighting (weight neighbours by the inverse of their distance) and similarity weighting (weight neighbours by 1-their distance). However, the default technique used in Weka is the equal weighting (weight neighbours by their distance) (Lavesson & Davidsson, 2006).

TABLE 4.2: Parameters of the kNN machine learning algorithm used in optimization.

Parameters	Tested values	Default values	Best value
Neighbors (k)	[1..80] step size:1	1	56
Weighting	equal, inverse-distance, similarity	equal	equal

Moreover, as presented in Table 4.3, five parameters of JRip algorithm according to MeeraGandhi et al. (2010), were optimised.

Finally, the best values found through optimization for the two parameters of GASSIST-ADI and MPLCS are presented in Table 4.4.

TABLE 4.3: Parameters of the JRip machine learning algorithm used in optimization.

Parameters	Tested values	Default values	Best value
Number of folds for reduced error pruning (F)	[1..10] step size:1	3	3
Minimal weight of instances within a split (N)	[1..10] step size:1	2	3
Number of runs of optimisations (O)	[1..10] step size:1	2	4
The seed of randomization (S)	[1..10] step size:1	1	5
Not use pruning (P)	yes/no	no	no

TABLE 4.4: Parameters of GASSIST-ADI and MPLCS used in optimization.

Parameters	Tested values	Default values	Best value
Crossover probability	0.2, 0.4, 0.6, 0.8	0.6	0.6
Population size	100, 300, 500, 1000	300	300

4.7 Performance Comparison

In this section, the performance of ESR-NID is compared to five classifiers using their best configurations found in section 4.5 and 4.6, respectively. For J48, kNN and JRip, which are deterministic, 10 runs for 10 folds are conducted and for GASSIST-ADI and MPLCS, 3 runs for each of the 10 folds resulted in 30 runs. Similarly for the proposed algorithm, for each combination of mutation rate and population size, 30 runs were carried out. Thus, the maximum and minimum values for GASSIST-ADI, MPLCS and the proposed EA can be expected to show slightly wider spread than for J48, kNN and JRip. As can be seen in Figure 4.12 and Table 4.5, ESR-NID, kNN, JRip and GASSIST-ADI provided similar performance results whereas MPLCS performed slightly better and J48 slightly worse than the other methods. The specification of each model provided in Table 4.5 shows that the complexity of ESR-NID model is also comparable to other rule-based models (J48, JRip, GASSIST-ADI and MPLCS).

Additionally, an example of a final rule set generated by ESR-NID compared to J48, JRip, GASSIST-ADI and MPLCS outputs, is presented in Table 4.6. The outputs are interpreted to match the format that ESR-NID uses for the representation of rules. As can be seen, all three input features were needed for classification of instances in these approaches. ESR-NID generated a concise ruleset similar to JRip

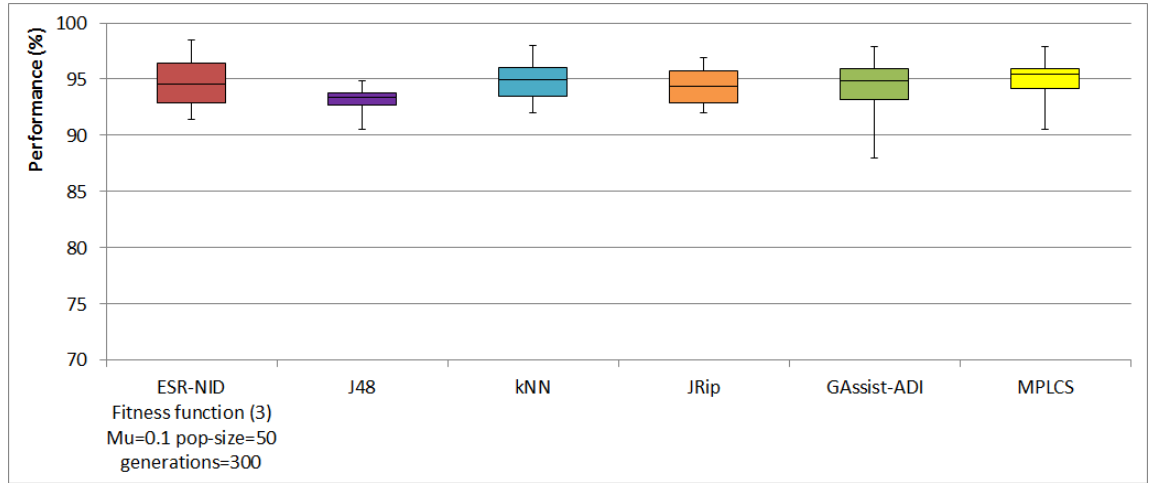


FIGURE 4.12: Performance of ESR-NID using the best configuration found compared to J48, kNN, JRip, GASSIST-ADI and MPLCS approaches.

TABLE 4.5: Details of the results presented in Figure 4.12.

Classifier	Specifications	Performance				
		min	q1	median	q3	max
ESR-NID	number of rules: 3	91.3	92.8	94.5	96.4	98.4
J48	size of tree: 7 number of leaves: 4	90.5	92.7	93.3	93.8	94.8
kNN	k = 56	91.9	93.4	94.9	96	98
JRip	number of rules: 3	91.9	92.8	94.3	95.7	96.9
GASSIST-ADI	number of rules: 4	88	93.1	94.8	95.9	97.9
MPLCS	number of rules: 4	90.5	94.1	95.4	95.9	97.9

output, without having redundancy, as opposed to J48 and GASSIST-ADI outputs. J48 algorithm checks the F3 condition repeatedly for both leaves and GASSIST-ADI has three similar rules with approximately the same ranges for F1 and F2 conditions.

Based on the preliminary experimental findings and the satisfactory outcome of performance comparison of ESR-NID against other machine learning methods, for the subsequent experiments, ESR-NID will be configured as follows:

- Fitness Function = fitness function (3) (Equation (3.6))
- Performance Function = $\sqrt{TP_{rate} * TN_{rate}}$
- Mutation probability = 0.1
- Population size = 50

TABLE 4.6: Comparing the final ruleset generated by ESR-NID, J48, JRip, GASSIST-ADI and MPLCS.

Approach	Output
The rules used to generate the anomalous records for the problem in Figure 4.1	rule1: <i>if</i> $F1 \in [0, 0.3]$ <i>and</i> $F2 \in [0, 0.2]$ <i>and</i> $F3 \in [0.4, 0.8]$ <i>then anomaly</i> rule2: <i>if</i> $F1 \in [0, 0.1]$ <i>and</i> $F2 \in [0.5, 0.9]$ <i>and</i> $F3 \in [0, 0.3]$ <i>then anomaly</i>
ESR-NID	rule1: <i>if</i> $F3 \in [0.20, 0.82]$ <i>then anomaly</i> rule2: <i>if</i> $F1 \in [0, 0.10]$ <i>and</i> $F2 \in [0.57, 0.95]$ <i>then anomaly</i>
J48 (with interpretation)	$F3 \leq 0.19$ $F1 \leq 0.09$ $F2 \leq 0.50$: Normal $F2 > 0.50$: Anomaly $F1 > 0.09$: Normal $F3 > 0.19$: Anomaly rule1: <i>if</i> $F3 \in [0.19, 1]$ <i>then anomaly</i> rule2: <i>if</i> $F1 \in [0, 0.09]$ <i>and</i> $F2 \in [0.50, 1]$ <i>and</i> $F3 \in [0, 0.19]$ <i>then anomaly</i>
JRip (with interpretation)	$(F3 \geq 0.20)$: Anomaly $(F2 \geq 0.59)$ <i>and</i> $(F1 \leq 0.09)$: Anomaly Others: Normal rule1: <i>if</i> $F3 \in [0.20, 1]$ <i>then anomaly</i> rule2: <i>if</i> $F1 \in [0, 0.09]$ <i>and</i> $F2 \in [0.59, 1]$ <i>then anomaly</i>
GASSIST-ADI (with interpretation)	$F3$ is $[> 0.19]$ Anomaly $F1$ is $[< 0.01]$ $F2$ is $[> 0.53]$ Anomaly $F1$ is $[< 0.06]$ $F2$ is $[> 0.55]$ Anomaly $F1$ is $[< 0.11]$ $F2$ is $[> 0.59]$ Anomaly Default rule – $>$ Normal rule1: <i>if</i> $F3 \in [0.19, 1]$ <i>then anomaly</i> rule2: <i>if</i> $F1 \in [0, 0.01]$ <i>and</i> $F2 \in [0.53, 1]$ <i>then anomaly</i> rule3: <i>if</i> $F1 \in [0, 0.06]$ <i>and</i> $F2 \in [0.55, 1]$ <i>then anomaly</i> rule4: <i>if</i> $F1 \in [0, 0.11]$ <i>and</i> $F2 \in [0.59, 1]$ <i>then anomaly</i>
MPLCS (with interpretation)	$F2$ is $[> 0.78]$ Anomaly $F3$ is $[> 0.19]$ Anomaly $F1$ is $[< 0.10]$ $F2$ is $[> 0.58]$ Anomaly Default rule – $>$ Normal rule1: <i>if</i> $F2 \in [0.78, 1]$ <i>then anomaly</i> rule2: <i>if</i> $F3 \in [0.19, 1]$ <i>then anomaly</i> rule3: <i>if</i> $F1 \in [0, 0.10]$ <i>and</i> $F2 \in [0.58, 1]$ <i>then anomaly</i>

- Generations = 300

and the other comparison methods will also be designed with the parameters best values found in Table 4.1, 4.2 and 4.3.

In the next two sections, to evaluate the performance of ESR-NID on more complex problems, further experiments are conducted against the other synthetic datasets introduced in Section 4.2.2 and 4.2.3: a 3-dimensional problem with more clusters of hits and two problems with 6 and 12 input features.

4.8 Performance Evaluation of ESR-NID against a 3-Dimensional Problem with More Clusters of Hits

Similar to the previous experiments, ESR-NID and the other comparison methods are tested against the problem shown in Figure 4.2 with two clusters of normal records and five clusters of anomalous instances. The results are presented in Figure 4.13. Additionally, Table 4.7 provides more details of the five values used for presenting the box plot. The results show that a final rule set of 6 rules (on average) provided by ESR-NID is comparable in performance and reliability to the other methods. The median performance for ESR-NID is slightly better than GASSIST-ADI, the same as J48 and slightly worse than kNN, JRip and MPLCS. The minimum performance provided by ESR-NID, J48 and JRip methods is more than 90% but for kNN, GASSIST-ADI and MPLCS, this is between about 87% and 90%.

To present the simplicity and readability of ESR-NID output, Table 4.8 compares an example of a final ruleset generated by ESR-NID to J48, JRip, GASSIST-ADI and MPLCS outputs. ESR-NID only uses two input features (F2 and F3) for classification of anomalous records, while the other methods needed three of the input features to produce a similar performance value.

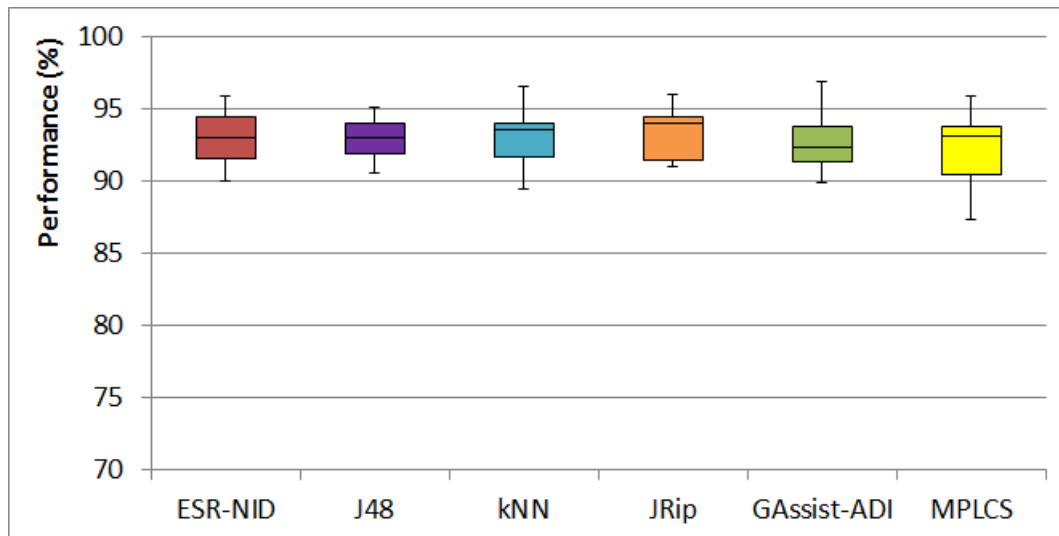


FIGURE 4.13: Performance of different techniques on the problem shown in Figure 4.2. Six rules are produced by ESR-NID, J48 and JRip for classification of anomalous records.

TABLE 4.7: The performance of different techniques on the problem shown in Figure 4.2.

Classifier	Specifications	Performance				
		min	q1	median	q3	max
ESR-NID	average number of rules: 6	90	91.5	92.9	94.3	95.7
J48	size of tree: 11 number of leaves: 6	90.5	91.9	92.9	93.8	95
kNN	-	89.4	91.6	93.3	93.8	96.3
JRip	number of rules: 6	91	91.4	93.8	94.2	95.8
GASSIST-ADI	number of rules: 4	89.9	91.3	92.3	93.7	96.9
MPLCS	number of rules: 5	87.3	90.4	93.1	93.7	95.9

Additionally, in Figure 4.14, the six rules generated by ESR-NID are plotted on the seven clusters of data shown in Figure 4.2 (i.e. five anomalous and two normal clusters). This shows the coverage of anomalous instances by the generated ruleset, that results in a performance value of about 93%.

TABLE 4.8: Comparing the final ruleset generated by ESR-NID, J48, JRip, GASSIST-ADI and MPLCS.

Approach	Output
ESR-NID	rule1: <i>if</i> $F3 \in [0.20, 0.31]$ <i>then anomaly</i> rule2: <i>if</i> $F2 \in [0.03, 0.40]$ <i>and</i> $F3 \in [0.42, 0.69]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.03, 0.27]$ <i>then anomaly</i> rule4: <i>if</i> $F2 \in [0.05, 0.29]$ <i>then anomaly</i> rule5: <i>if</i> $F2 \in [0.60, 0.79]$ <i>then anomaly</i> rule6: <i>if</i> $F2 \in [0.01, 0.09]$ <i>then anomaly</i>
J48	$F2 \leq 0.30$: Anomaly $F2 > 0.30$ $F3 \leq 0.69$ $F3 \leq 0.19$ $F1 \leq 0.09$ $F2 \leq 0.40$: Normal $F2 > 0.40$: Anomaly $F1 > 0.09$: Normal $F3 > 0.19$: Anomaly $F3 > 0.69$: Normal
JRip	rule1: ($F2 \leq 0.30$): Anomaly rule2: ($F2 \geq 0.60$): Anomaly rule3: ($F1 \leq 0.09$) and ($F3 \leq 0.29$) and ($F3 \geq 0.19$): Anomaly rule4: ($F1 \leq 0.16$) and ($F3 \leq 0.69$) and ($F3 \geq 0.61$): Anomaly rule5: ($F2 \leq 0.41$) and ($F3 \geq 0.60$) and ($F3 \leq 0.69$): Anomaly rule6: Others: Normal

GASSIST-ADI	<p>rule1: F3 is [0.20, 0.69] Anomaly</p> <p>rule2: F1 is [< 0.19] F2 is [< 0.30] Anomaly</p> <p>rule3: F1 is [< 0.10] F2 is [> 0.49] F3 is [< 0.63] Anomaly</p> <p>rule4: F1 is [< 0.10] F2 is [> 0.40] F3 is [0.05, 0.64] Anomaly</p> <p>rule5: Default rule – $>$ Normal</p>
MPLCS	<p>rule1: F1 is [< 0.10] F2 is [< 0.20][> 0.49] F3 is [0.03, 0.15] Anomaly</p> <p>rule2: F1 is [< 0.10] F2 is [< 0.30][0.40, 0.49] F3 is [0.10, 0.30] Anomaly</p> <p>rule3: F2 is [< 0.30][> 0.59] Anomaly</p> <p>rule4: F3 is [0.20, 0.69] Anomaly</p> <p>rule5: Default rule – $>$ Normal</p>

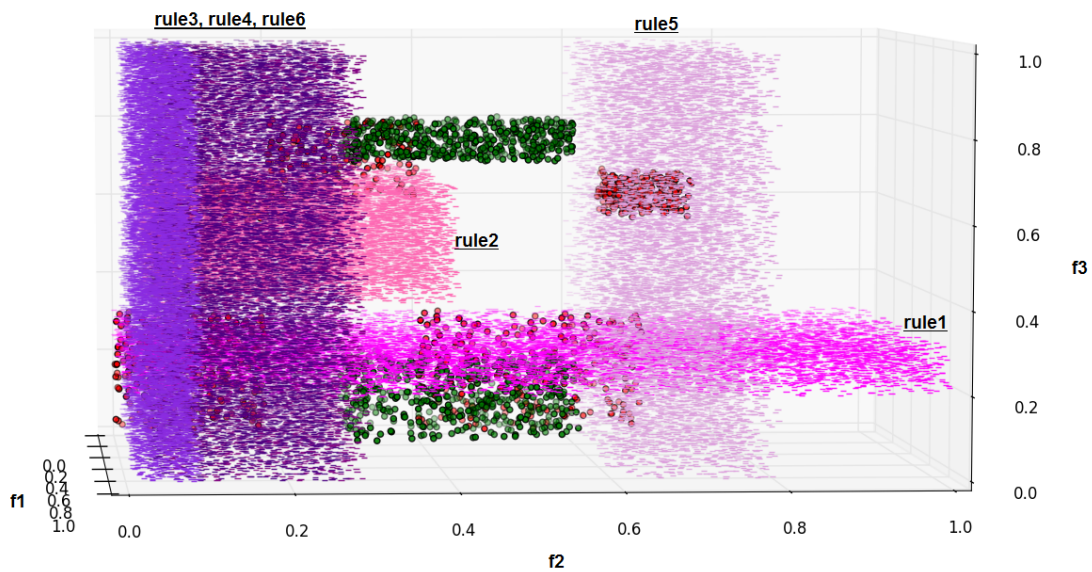


FIGURE 4.14: The final ruleset evolved by ESR-NID for classifying the anomalous records in the problem shown in Figure 4.2. f1, f2 and f3 are the three features in the dataset generated by the rules in Section 4.2.1.

Comparing the output of ESR-NID, J48 and JRip, some similarities can be found between the rulesets. For example the highlighted conditions in the JRip rules are relatively equivalent to ESR-NID rules as shown below:

- $(F2 \leq 0.30): \text{Anomaly}$ \longrightarrow rule3, rule4 and rule6 in ESR-NID
- $(F2 \geq 0.60): \text{Anomaly}$ \longrightarrow rule5 in ESR-NID
- $(F1 \leq 0.09)$ and $(F3 \leq 0.29)$ and $(F3 \geq 0.19): \text{Anomaly}$ \longrightarrow rule1 in ESR-NID
- $(F1 \leq 0.16)$ and $(F3 \leq 0.69)$ and $(F3 \geq 0.61): \text{Anomaly}$
- $(F2 \leq 0.41)$ and $(F3 \geq 0.60)$ and $(F3 \leq 0.69): \text{Anomaly}$ \longrightarrow rule2 in ESR-NID
- Others: Normal

For example, the coverage area provided by the first rule in JRip, $(F2 \leq 0.30): \text{Anomaly}$, is approximately the same as rule3: *if* $F2 \in [0.03, 0.27]$ *then anomaly*, rule4: *if* $F2 \in [0.05, 0.29]$ *then anomaly* and rule6: *if* $F2 \in [0.01, 0.09]$ *then anomaly* in the ESR-NID final ruleset.

Similarly, in Figure 4.15, the highlighted parts of the tree from J48 provide approximately the same coverage as the one produced by ESR-NID rules.

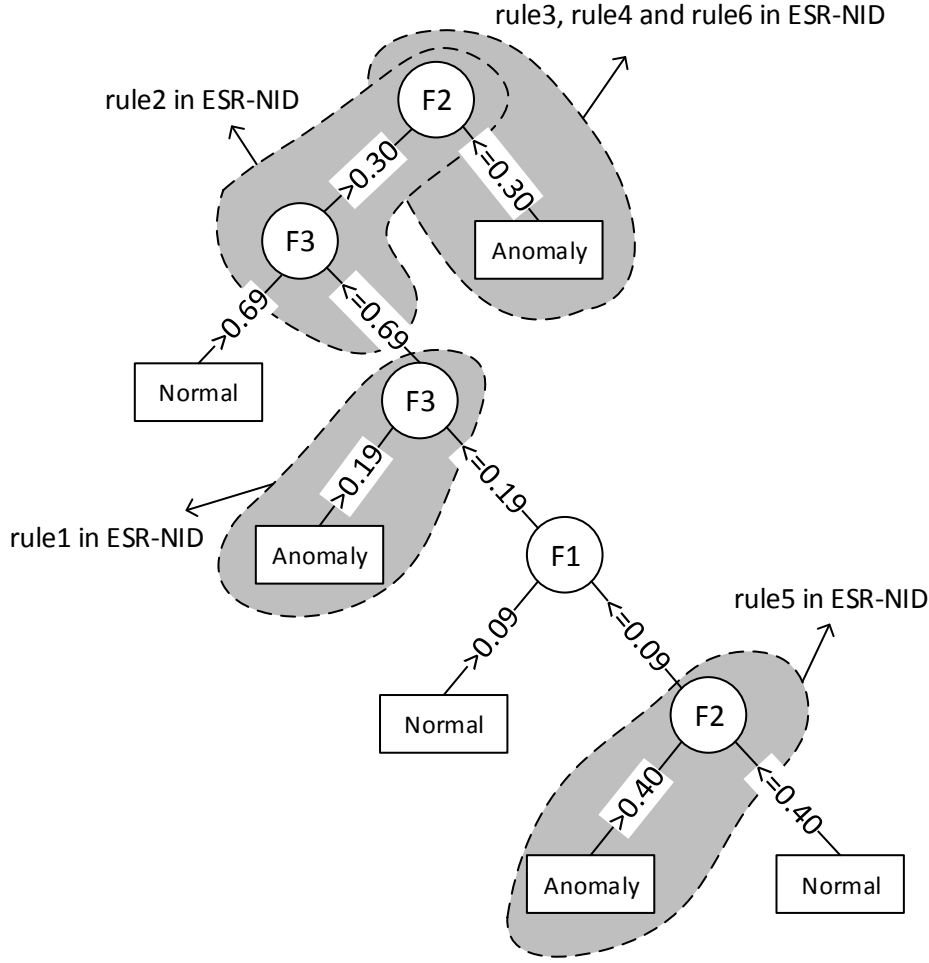


FIGURE 4.15: Comparing the output of J48 to the ruleset generated by ESR-NID.

Finally, some similarities can be seen between ESR-NID output and GASSIST-ADI and MPLCS final rulesets as shown below:

(GASSIST-ADI)

- F3 is $[0.20, 0.69]$ | Anomaly \rightarrow rule1 and rule2(cond2) in ESR-NID
- F1 is $[< 0.19]$ | F2 is $[< 0.30]$ | Anomaly \rightarrow rule3, rule4 and rule6 in ESR-NID
- F1 is $[< 0.10]$ | F2 is $[> 0.49]$ | F3 is $[< 0.63]$ | Anomaly \rightarrow rule5 and rule1 and rule2(cond2) in ESR-NID
- F1 is $[< 0.10]$ | F2 is $[> 0.40]$ | F3 is $[0.05, 0.64]$ | Anomaly \rightarrow rule5 and rule1 and rule2(cond2) in ESR-NID
- Default rule \rightarrow Normal

(MPLCS)

- F1 is [< 0.10] | F2 is [< 0.20][> 0.49] | F3 is [$0.03, 0.15$] | Anomaly \rightarrow rule5, rule6 and rule1 in ESR-NID
- F1 is [< 0.10] | F2 is [< 0.30][$0.40, 0.49$] | F3 is [$0.10, 0.30$] | Anomaly \rightarrow rule3, rule4, rule6 and rule1 in ESR-NID
- F2 is [< 0.30][> 0.59] | Anomaly \rightarrow rule3, rule4, rule5 and rule6 in ESR-NID
- F3 is [$0.20, 0.69$] | Anomaly \rightarrow rule1 and rule2(cond2) in ESR-NID
- Default rule – $>$ Normal

As an example, the first rule in GASSIST-ADI ruleset, F3 is [$0.20, 0.69$] | Anomaly, provides approximately the same coverage as the combination of rule1: *if* $F3 \in [0.20, 0.31]$ and cond2 in rule2: $F3 \in [0.42, 0.69]$ provides in ESR-NID model.

4.9 Performance Evaluation of ESR-NID against a Problem with More Input Features

In this section, ESR-NID is evaluated on problems with more features. For this, two sets of synthetic data were generated for problems with 6 and 12 input features. Similar to the previous section, the results from J48, kNN, JRip, GASSIST-ADI and MPLCS were compared to the result obtained from ESR-NID.

Figure 4.16 and Table 4.9 illustrate the performance of these techniques on a problem with 6 features. Less variation of performance values can be seen from ESR-NID, J48, kNN, GASSIST-ADI and MPLCS classifiers compared to JRip. Comparing the size of generated ruleset for classification, ESR-NID, JRip, GASSIST-ADI and MPLCS are the best with 5 rules. Therefore, considering the performance value and simplicity of the output, ESR-NID and GASSIST-ADI can be ranked as the best ones between the five rule-based classifiers.

When the data is more complex with 12 input features, as can be seen from Figure 4.17 and Table 4.10, J48, JRip, GASSIST-ADI and MPLCS performed slightly better than ESR-NID and kNN performed slightly worse. All the rule-based classifiers produced the same size output of 5 rules. This concludes that ESR-NID can be used in higher dimensions by producing an optimised ruleset of reasonable size with an acceptable performance value.

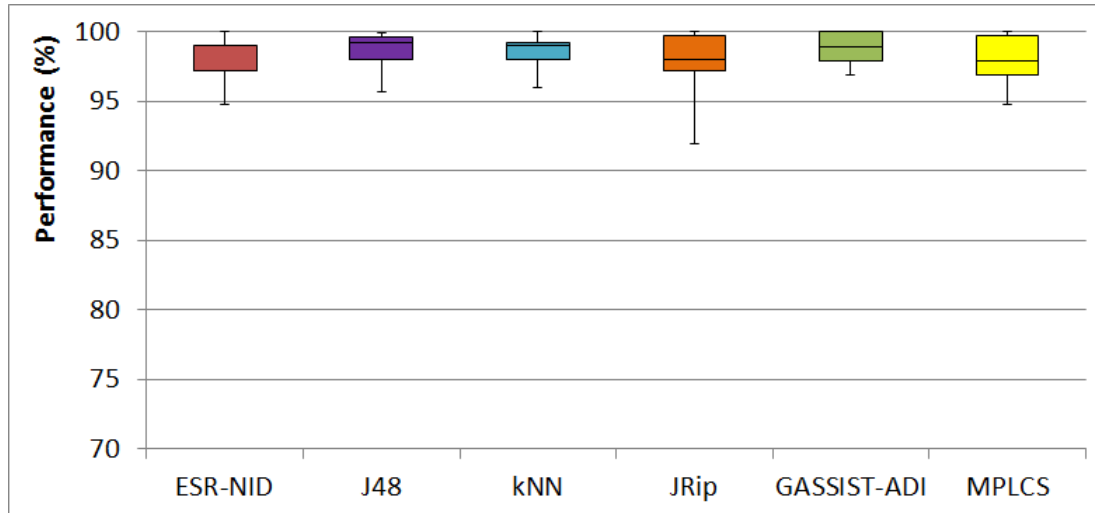


FIGURE 4.16: Performance of different techniques on a problem with 6 features.

TABLE 4.9: Performance of different techniques on a problem with 6 features.

Classifier	Specifications	Performance				
		min	q1	median	q3	max
ESR-NID	average number of rules: 5	94.7	97.2	98.9	98.9	99.9
J48	size of tree: 13 number of leaves: 7	95.7	97.9	99.1	99.5	99.8
kNN	-	96	98	99	99.2	100
JRip	number of rules: 5	92	97.2	97.9	99.7	99.9
GASSIST-ADI	number of rules: 5	96.9	97.9	98.9	100	100
MPLCS	number of rules: 5	94.8	96.9	97.9	99.7	100

4.10 Using a Different Performance Function

Throughout this chapter ESR-NID was configured using g-performance, calculated as $\sqrt{TP_{rate} * TN_{rate}}$. To present one of the advantages of ESR-NID as opposed to

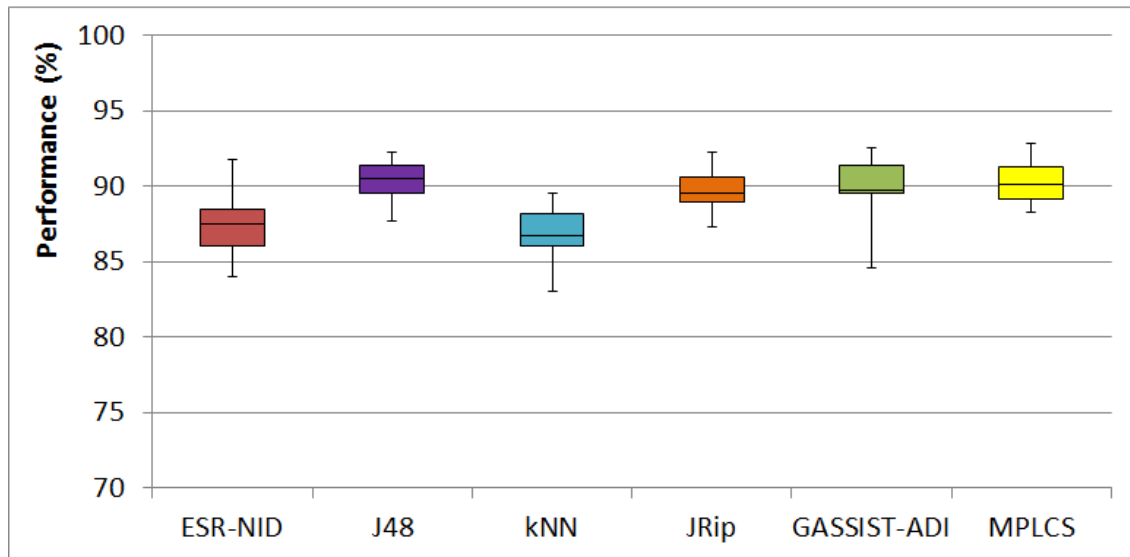


FIGURE 4.17: Performance of different techniques on a problem with 12 features.

TABLE 4.10: Performance of different techniques on a problem with 12 features.

Classifier	Specifications	Performance				
		min	q1	median	q3	max
ESR-NID	average number of rules: 5	84	86	88	89	92
J48	size of tree: 9 number of leaves: 5	87.7	89.5	90.5	91.4	92.3
kNN	-	83.1	86	86.7	88	89.3
JRip	number of rules: 5	87.3	89	89.5	90.6	92.3
GASSIST-ADI	number of rules: 5	84.4	89.3	89.5	91.2	92.3
MPLCS	number of rules: 5	88.3	89.1	90.1	91.3	92.8

other tested techniques, in this section, another performance metric is used. By providing the ability to change the performance function, ESR-NID generates variants of classifiers for different problems. Here, classification accuracy is used as the performance function in ESR-NID.

The classification accuracy can be reported using the standard confusion matrix approach. This metric is the most commonly used measure for evaluating the performance of classifiers (Alpaydin, 2004; Witten & Frank, 2005). A confusion matrix for a two-class problem is illustrated in Table 4.11.

Based on the confusion matrix in Table 4.11, accuracy is defined as follows:

TABLE 4.11: Confusion matrix for a two-class problem.

		Predicted	
		Normal	Attack
Actual	Normal	a	b
	Attack	c	d

$$\text{accuracy} = \frac{a + d}{a + b + c + d} \quad (4.1)$$

The customised ESR-NID using accuracy as the performance function is evaluated against the latest synthetic problem with 12 input features. To provide a better comparison, first, Table 4.12 and Figure 4.18 show the g-performance results achieved using the g-performance function and then Table 4.13 and Figure 4.19 present the classification accuracy results. As can be seen from the results, the customised ESR-NID is performing approximately the same as the default configuration of ESR-NID using g-performance in terms of both g-performance and classification accuracy. The customised ESR-NID, however, in this problem produced more rules compared to ESR-NID and other rule-based methods. Therefore, for this specific problem (i.e. a balanced problem with 12 input features), either of these functions would provide an acceptable performance result. This may be because, on a balanced dataset g-performance and accuracy would be expected to rank classifiers similarly. Therefore, in the next subsection, the case of imbalanced datasets is considered.

TABLE 4.12: Performance of different techniques on a problem with 12 features.

Classifier	Specifications	g-performance				
		min	q1	median	q3	max
ESR-NID	average number of rules: 5	84	86	88	89	92
Customised ESR-NID	average number of rules: 7	84.1	85.9	88.3	90	92
J48	size of tree: 9 number of leaves: 5	87.7	89.5	90.5	91.4	92.3
kNN	-	83.1	86	86.7	88	89.3
JRip	number of rules: 5	87.3	89	89.5	90.6	92.3
GASSIST-ADI	number of rules: 5	84.4	89.3	89.5	91.2	92.3
MPLCS	number of rules: 5	88.3	89.1	90.1	91.3	92.8

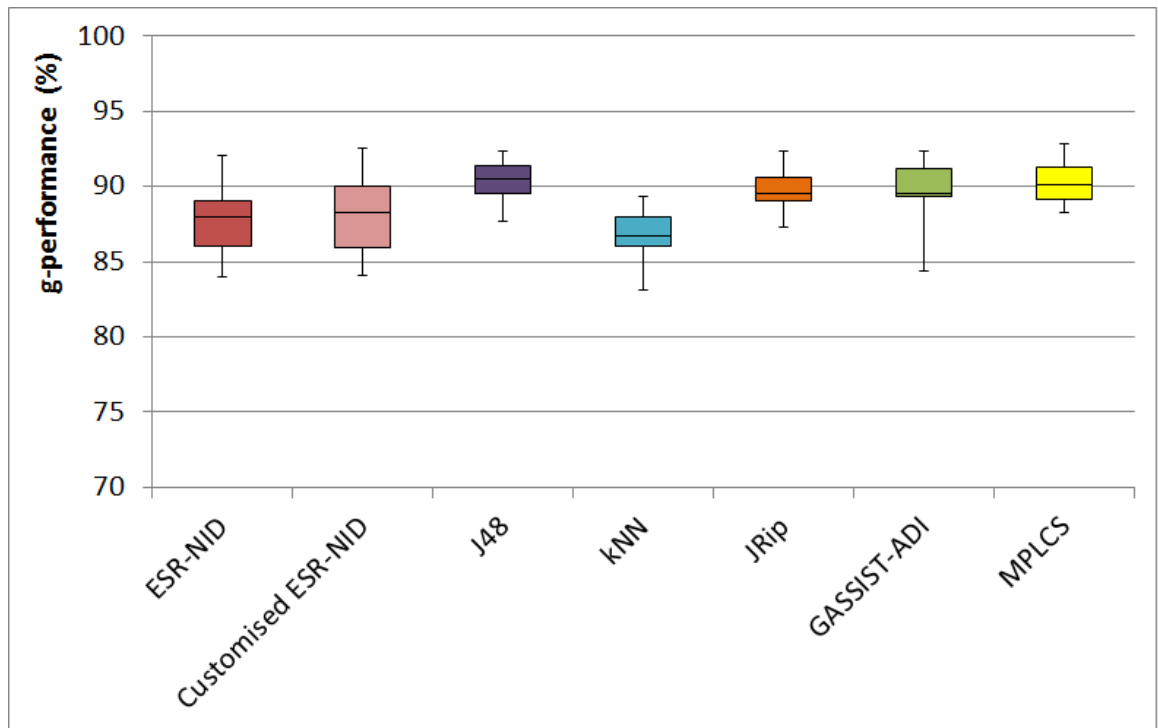


FIGURE 4.18: Performance of different techniques on a problem with 12 features.

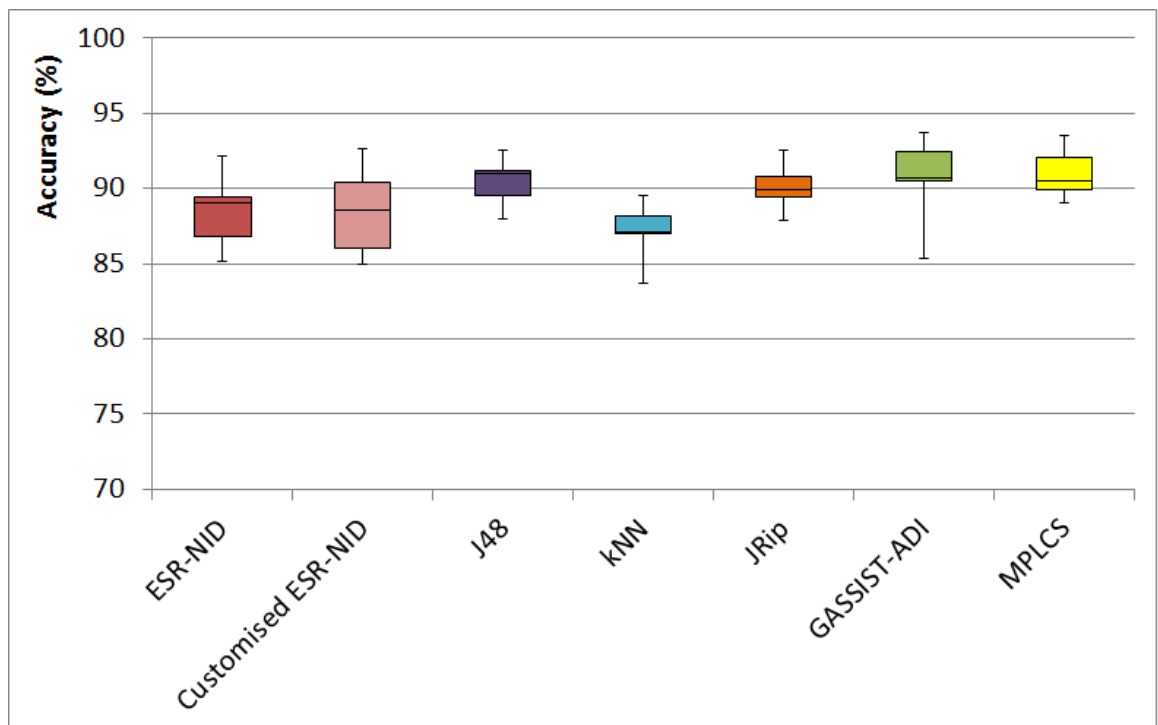


FIGURE 4.19: Accuracy of different techniques on a problem with 12 features.

TABLE 4.13: Accuracy of different techniques on a problem with 12 features.

Classifier	Specifications	Performance (accuracy)				
		min	q1	median	q3	max
ESR-NID	average number of rules:5	85.1	86.8	89	89.4	92.1
Customised ESR-NID	average number of rules: 7	85	86	88.5	90.4	92.1
J48	size of tree: 9 number of leaves: 5	88	89.5	91	91.2	92.5
kNN	-	83.7	87	87.1	88.2	89.5
JRip	number of rules: 5	87.9	89.4	89.9	90.8	92.5
GASSIST-ADI	number of rules: 5	84.8	90	90.2	91.9	93.2
MPLCS	number of rules: 5	89	89.9	90.5	92	93.5

4.10.1 Imbalanced Dataset

In this section, to present the influence of use of a different performance function on the output of ESR-NID, an imbalanced dataset with 12 input features is generated using the same rules utilised to generate the complex problem with 12 features in Section 4.2.3. In this imbalanced problem, 1000 data points for normal records and 20 points for anomalous instances are generated. The classification task is only carried out using the default version of ESR-NID (using g-performance function) and the customised ESR-NID (using accuracy). Tables 4.14 and 4.15 show minimum, lower quartile, median, upper quartile, and maximum performance values over the 30 runs.

To provide a more detailed comparison of the two configurations of ESR-NID, in addition to the average values of g-performance and accuracy for 30 runs, the average of true positive and true negative rates are also presented in Tables 4.16 and 4.17. In this experiment, the default configuration of ESR-NID outperformed the customised ESR-NID in terms of true positive rate and as a result, it produced better g-performance as can be seen from the results. This is because accuracy does not differentiate between the number of correct classification of different classes and its erroneous conclusions on imbalanced datasets derive a set of rules with lower true positive rate. For example, if the ratio of imbalance in a dataset is 1:100 (i.e. 1 positive instance versus 99 negatives), if a classifier produces 99% of accuracy, it is not actually accurate if it does not correctly classify the only positive instance

in the dataset. The use of geometric mean of the true rates instead of utilising accuracy has been studied in the past to address the problem of balance in datasets (Fernández et al., 2010).

TABLE 4.14: Performance of two configurations of ESR-NID on an imbalanced problem with 12 features.

Classifier	g-performance				
	min	q1	median	q3	max
ESR-NID	86.6	100	100	100	100
Customised ESR-NID	86.6	86.6	100	100	100

TABLE 4.15: Accuracy of two configurations of ESR-NID on an imbalanced problem with 12 features.

Classifier	Performance (accuracy)				
	min	q1	median	q3	max
ESR-NID	99.5	100	100	100	100
Customised ESR-NID	99.5	99.5	100	100	100

TABLE 4.16: Average performance of ESR-NID against an imbalanced problem with 12 input features.

g-performance	accuracy	True positive rate	True negative rate
97.3	99.9	95	100

TABLE 4.17: Average performance of customised ESR-NID against an imbalanced problem with 12 input features.

g-performance	accuracy	True positive rate	True negative rate
93.7	99.7	88.3	100

Since the default configuration of ESR-NID, which is configured using g-performance function, produced acceptable classification results in both cases of balanced and imbalanced problems, this configuration will be used for the rest of experiments in this thesis except in Section 5.4, where further experiments explore the idea of using different performance functions in another context (i.e. detecting normal instances).

4.11 Summary

This chapter evaluated ESR-NID using some synthetic datasets, that can be easily constructed and manipulated to create classification problems with desired features. After preliminary evaluations of the method, an adaptive elitism mechanism was introduced, which adaptively adjusts the number of elites copied into each new generation. This ensures that cooperating rules are kept together and not lost from one generation to the next. This also means that there is no need to select some arbitrary number of elites a priori. Next, some experiments were conducted to evaluate three different fitness functions: a naïve one and two advanced ones, in which the rules can evolve cooperatively to provide a final optimised ruleset that covers the area of search precisely. The fitness functions have also been designed with the aim of reducing the number of rules needed for classification problems. Through these experiments, the impact of GA parameters on the performance of the enhanced ESR-NID were also investigated. The results obtained from ESR-NID along with the five comparison methods, J48, kNN, JRip, GASSIST-ADI and MPLCS showed that the fitness function (3), introduced in Equation (3.6), is the most reliable function among the three tested fitness functions. These results also suggested the best set of GA parameters to fix the system for further experiments. Next, to test the accuracy of ESR-NID further, some experiments on a set of more complex synthetic datasets were conducted. The results showed that ESR-NID worked effectively compared to other comparison machine learning techniques by generating small and easily understood rulesets. The size of generated rulesets by ESR-NID was always equal or less than the other comparing rule-based methods. This difference becomes more pronounced for real complex problems as shall be seen later.

In the next chapter, ESR-NID will be tested within the context of a real-world problem. This will show the flexibility of ESR-NID to be applied on different problems with continuous-valued features.

Chapter 5

Performance Evaluation of ESR-NID on Network Intrusion Detection Problems

5.1 Introduction

In the previous chapter, a number of synthetic datasets were used to evaluate ESR-NID to see if the proposed approach can handle:

- The similarity of normal and anomalous records in the dataset
- The increasing number of clusters of anomalous records
- The increasing complexity in feature space

To demonstrate the flexibility of ESR-NID when applied to a real world problem, this chapter describes a set of experiments conducted to test the system as a network intrusion detection system.

Similar to the previous chapter, the performance of ESR-NID is compared to five machine learning methods from two categories of GA-based and non-GA-based algorithms: J48, kNN and JRip (non-GA-based) and GASSIST-ADI and MPLCS

(GA-based). As before, the goal is to evaluate the accuracy, reliability and the understandability of the rules of the resulting rule-based classifier.

For this, standardized sets of data that evaluate the effectiveness and efficiency of IDSs are needed. Section 5.2, introduces the existing choices with the focus on the NSL-KDD dataset ([Tavallaee et al., 2009](#)) and a combined DARPA/CAIDA dataset, which are used for the experiments in this chapter.

5.2 Choice of Dataset

Based on the reviewed research work, there are two publicly available datasets that have been widely used in either signature-based or anomaly-based detection system evaluations: the DARPA-Lincoln datasets and the KDD99 dataset.

The first Intrusion Detection Evaluation (IDEVAL) program was run in 1998 by the MIT Lincoln Lab under the sponsorship of DARPA and US Air Force Research Labs (AFRL). The 1998 off-line evaluation included 10 systems, 38 attack types, weeks of background traffic, which resulted in a database of attacks and background traffic (seven weeks of training data and two weeks of test data captured by a program named tcpdump ([Jacobson et al., 1989](#))). This evaluation was limited to only one intrusion detection system developed under DARPA's sponsorship, only attacks against UNIX hosts and background traffic similar to that of one Air Force base. Moreover, in 1999, further off-line and real-time evaluations were conducted by AFRL based on recommendations from 1998 to enhance the analysis and cover more attack types. Major changes in the 1999 evaluation are ([Haines et al., 2001](#); [Lippmann et al., 2000](#)):

- Addition of a Windows NT workstation as a victim
- Addition of an inside tcpdump sniffer machine
- Collection of both Windows NT audit events and inside tcpdump sniffing data

- Focusing on determining the ability of systems to detect unseen attacks and analysing why systems miss new attacks

The attacks found in these datasets fall into four major categories: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L) and probing. Additionally, some scenario specific datasets, which include the network traffic collected from both the DMZ and the inside part of the evaluation network, were also distributed in 2000 ([MIT Lincoln Laboratory, 2000](#)).

A processed version of the 1998 DARPA dataset, the KDD99 dataset, was also distributed as part of a competition (1999 KDD Cup competition) sponsored by the International Conference on Knowledge Discovery in Databases. The task for this contest was to learn a predictive model (i.e. a classifier) that is able to distinguish between legitimate and illegitimate connections in a computer network. The KDD99 dataset consists of a large number of network connections, both normal and attacks. This data is partitioned into about five million records of training data and approximately 0.3 million records of test data. A connection corresponds to a time-stamped session of data transfer between two computers. Each connection in KDD99 has 41 features (32 continuous and 9 discrete) and is labeled as normal or a specific attack type as presented in Table 5.1. These features can be broken down into the following four categories ([Stolfo et al., 2000](#); [Stewart, 2009](#); [Shafi, 2008](#)):

- Basic features (presented in Table 5.2): the features that are common to all network connections such as duration and protocol_type. These features could help in detection of attacks targeting protocol and service vulnerabilities.
- Traffic features based on a time window (presented in Table 5.3): the features that are measured using a two-second time window. The time window is used to examine the connections in the past two seconds which have the same destination host or the same service as that of the current connection.
- Host based traffic features (presented in Table 5.4): similar to the previous category, host based traffic features capture the number of connections to the same host, port or service by a destination host in the past 100 connections.

- Connection-based content features based on domain knowledge (presented in Table 5.5): this category of features may or may not be useful in detecting malicious activities and is based on domain knowledge. This category is usually used in detecting R2L and U2R attacks by monitoring statistics disclosed in the audit logs or in the payload section of the packets.

TABLE 5.1: Attack types in KDD99 dataset and their categorisation.

Category	Attack type
Probe	ipsweep, mscan, nmap, portsweep, satan, saint
DoS	apache2, back, land, mailbomb, neptune, pod, processtable, smurf, teardrop, udpstorm
U2R	buffer_overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm
R2L	ftp_write, guess_passwd, httptunnel, imap, multihop, named, phf, sendmail, snmpgetattack, snmpguess, spy, warezclient, warezmaster, worm, xlock, xsnoop

TABLE 5.2: Basic features in KDD99 dataset.

Feature Name	Description	Type
duration	length of the connection in seconds	continuous
protocol_type	type of protocol, e.g., tcp, udp, etc.	nominal
service	network service on the destination, e.g., http, telnet, etc.	nominal
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	nominal
land	1 if connection is from/to the same host/-port; 0 otherwise	binary
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

TABLE 5.3: Traffic features using two-second time windows in KDD99 dataset.

Feature Name	Description	Type
count	number of connections to the same host as the current connection in the past two seconds	continuous
error_rate	% of same host connections that have “SYN” errors	continuous
error_rate	% of same host connections that have “REJ” errors	continuous
same_srv_rate	% of same host connections to the same service	continuous
dif_srv_rate	% of same host connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
srv_error_rate	% of same service connections that have “SYN” errors	continuous
srv_error_rate	% of same service connections that have “REJ” errors	continuous
srv_diff_host_rate	% of same service connections to different hosts	continuous

TABLE 5.4: Traffic features using windows of 100 connections in KDD99 dataset.

Feature Name	Description	Type
dst_host_count	number of connections to the same host in the past 100 connections	continuous
dst_host_error_rate	% of connections that have “SYN” errors	continuous
dst_host_error_rate	% of connections that have “REJ” errors	continuous
dst_host_same_srv_rate	% of connections to the same service	continuous
dst_host_dif_srv_rate	% of same host connections to different services	continuous
dst_host_srv_count	number of connections to the same service in the past 100 connections	continuous
dst_host_srv_error_rate	% of same service connections that have “SYN” errors	continuous
dst_host_srv_error_rate	% of same service connections that have “REJ” errors	continuous
dst_host_srv_diff_host_rate	% of same service connections to different hosts	continuous
dst_host_same_src_port_rate	% of connections from the same source port	continuous

TABLE 5.5: Connection-based content features based on domain knowledge in KDD99 dataset.

Feature Name	Description	Type
hot	hot indicators e.g., access to system directories, creation, and execution of programs, etc.	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	binary
num_compromised	number of compromised states on the destination host (e.g., file/path “not found” errors, and “Jump to” instructions, etc.)	continuous
root_shell	1 if root shell is obtained; 0 otherwise	binary
su_attempted	1 if “su root” command attempted; 0 otherwise	binary
num_root	number of “root” accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_host_login	1 if the login belongs to the “host” list; 0 otherwise	binary
is_guest_login	1 if the login is a “guest” login; 0 otherwise	binary

5.2.1 NSL-KDD dataset

Although the KDD99 dataset has been widely used for evaluation of IDSs, it has two major issues which affect the performance of evaluated systems. First, KDD99 contains a large number (over 75%) of repeated records. Secondly, the dataset is very asymmetric, containing disproportionately more records that are “easy” to classify. This large number of redundant records and low average level of difficulty results in high detection rates (about 98%) even for very simple machine learning methods. These two problems were addressed with a new version of KDD99 proposed by [Tavallae et al. \(2009\)](#), NSL-KDD. First, NSL-KDD does not include redundant (repeated) records. Second, records from KDD99 are resampled based on an estimate of how difficult each record is to classify. To decide on the difficulty of the records, 7 learners, each trained 3 times over three subsets of the KDD99 training set, were employed to label the KDD99 training and test sets. Using these learners, 21 predicted labels were provided for each record. A *#successfulPrediction*

value, initialised to zero, is also assigned to each record of the dataset. Since the KDD99 dataset provides the correct label for each record, if a specific learner correctly predicted the label of a given record, its *#successfulPrediction* value will be incremented. Therefore, using this process, the number of learners correctly label that given record was calculated. Finally, the *#successfulPrediction* values for the KDD99 dataset records were grouped into 5 difficulty groups: 0-5, 6-10, 11-15, 16-20 and 21 (i.e. the highest value for *#successfulPrediction* and coveys the fact that all learners correctly label a given record). The results showed that 97.97% and 86.64% of the records in the KDD99 training and test sets were correctly labelled by all 21 classifiers. To create a more challenging subset of the KDD99 dataset, the number of selected records from each difficulty group is inversely proportional to the percentage of records in the original dataset (so records that are difficult to classify are selected more often). These changes allow for greater discrimination between performance levels of different algorithms. Additionally, the number of records in the training and test datasets is not too large, which enables researchers to run experiments on the complete set and avoids the need to randomly select a small portion of the much larger KDD99 for training and testing, as has been done in the past. Table 5.6 and 5.7 illustrate the statistics of the reduction of repeated records in the KDD99 train and test sets, respectively.

TABLE 5.6: Statistics of redundant records in the KDD99 training set (Tavallae et al., 2009).

	Original Records	Distinct Records	Reduction Rate
Attacks	3,925,650	262,178	93.32%
Normal	972,781	812,814	16.44%
Total	4,898,431	1,074,992	78.05%

TABLE 5.7: Statistics of redundant records in the KDD99 test set (Tavallae et al., 2009).

	Original Records	Distinct Records	Reduction Rate
Attacks	250,436	29,378	88.26%
Normal	60,591	47,911	20.92%
Total	311,027	77,289	75.15%

Due to the easy access to these datasets, they are still being used by many researchers for IDS evaluation. Therefore, for the first set of experiments in this chapter, NSL-KDD will form the basis of the training and testing data.

5.2.2 Combined DARPA/CAIDA dataset

As KDD99 and NSL-KDD datasets are now more than a decade old and are not considered standard and appropriate by some members of the community due to their inclusion of old attacks (Sommer & Paxson, 2010), an alternative method that has been used by other researchers in this domain (Bhatia et al., 2011; Xiang et al., 2011; Bhuyan et al., 2014), can be considered for IDS evaluation. By combining the new attack traces found in the low-rate DDoS attack scenario from CAIDA DDoS 2007 (CAIDA, 2007) with the MIT Lincoln Laboratory Scenario (attack-free) inside tcpdump dataset as the normal network traffic, two newer sets of training and testing datasets are produced to test the proposed algorithm further. The normal traffic scenario is the data from Thursday in the third training week, which does not contain any attacks. A selected sampling period of this traffic is shown in Figure 5.1.

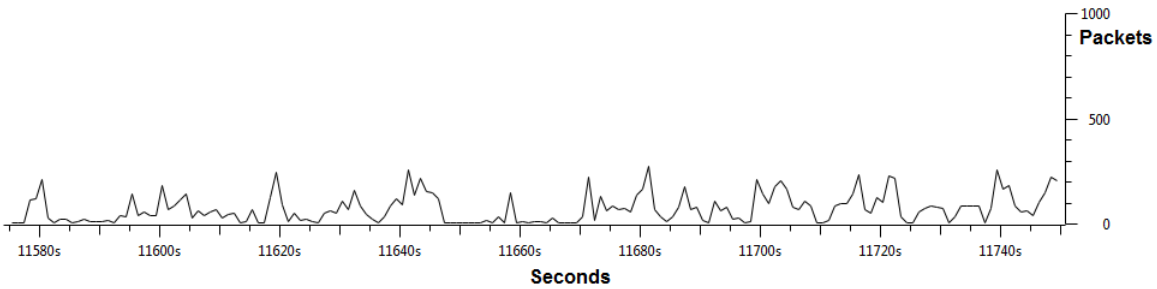


FIGURE 5.1: Normal traffic scenario from MIT Lincoln Laboratory.

Additionally, the CAIDA dataset contains 5 minutes (i.e., 300 s) of anonymized traffic of a DDoS attack on August 4, 2007. Non-attack traffic has been removed as much as possible from this data. According to Moore et al. (2006), if there are more than 10000 packets per second over the network, a high-rate attack is achieved and with about 1000 packets per second, only 60% of full attack is achievable and it is called a low-rate attack. As a result, CAIDA DDoS attack scenario is categorised

under low-rate attacks category as can be seen in Figure 5.2. More details of traffic feature of the CAIDA low-rate DDoS attack scenario can be found in Table 5.8.

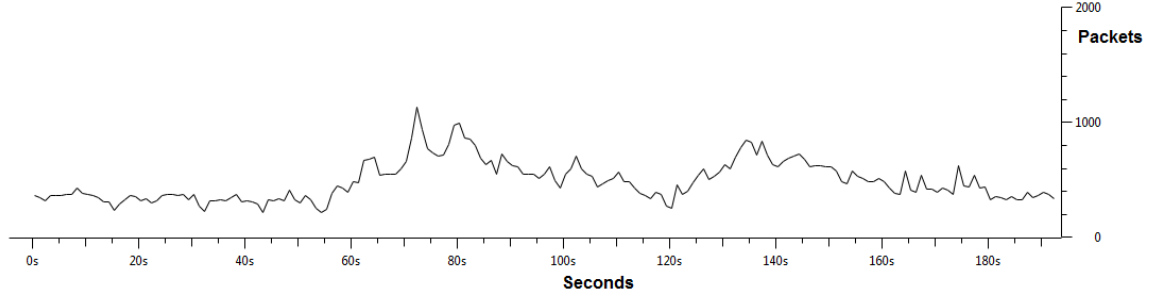


FIGURE 5.2: Low-rate DDoS attack scenario from CAIDA.

TABLE 5.8: Traffic features and details of CAIDA DDoS attack scenario (Moore et al., 2006; Xiang et al., 2011).

Maximum capture length for interface 0: 65000
First timestamp: 1186260576.487629000
Last timestamp: 1186260876.482457000
Unknown encapsulation: 0
IPv4 bytes: 37068253
IPv4 pkts: 166448
IPv4 traffic: 8079
Unique IPv4 addresses: 136
Unique IPv4 source addresses: 132
Unique IPv4 destination addresses: 136
Unique IPv4 TCP source ports: 4270
Unique IPv4 TCP destination ports: 3348
Unique IPv4 UDP source ports: 1
Unique IPv4 UDP destination ports: 1
Unique IPv4 ICMP type/codes: 2

5.2.3 Features

Identifying suitable features that provide worthwhile information for intrusion detection is a challenging task. A combination of features extracted from raw packets (packet level data) or a set of attributes related to a particular network flow collected over a short period of time (flow level data), have been previously used in the literature as the input to detection systems to classify incoming packets or flow of traffic (Wang & Stolfo, 2004; Bhuyan et al., 2015). These features are usually

categorised into basic, time-based, connection-based and content-based features (examples can be found in Tables 5.2, 5.3, 5.4 and 5.5). Additionally, some statistical approaches such as entropy-based and volume-based methods have been proposed in the past for identifying malicious activities in network traffic. Entropy-based detectors are of particular interest in this study. They are simple statistical measures, which discriminate DDoS traffic from legitimate (Xiang et al., 2011; Feinstein et al., 2003; Zhang et al., 2010; Zi et al., 2010; Sqalli et al., 2011). Simple calculation, high sensitivity, low false positive rate are the advantages of entropy-based approaches (Zhang et al., 2010). As an example, Figure 5.3 illustrates the effect of an attack on the entropy of IP source addresses in a time window of 10000 packets.

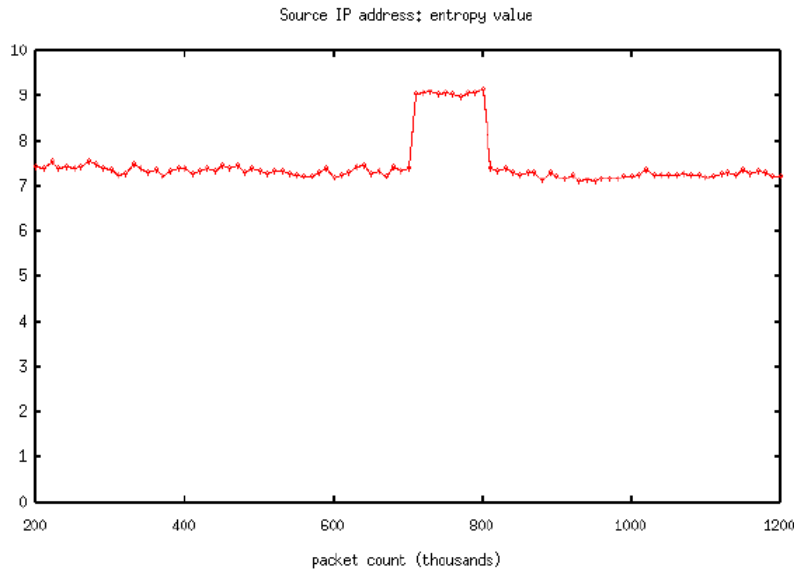


FIGURE 5.3: Entropy for a brief distributed denial of service attack (Feinstein et al., 2003)

NSL-KDD is a processed dataset based on the data captured in DARPA'98 IDS evaluation program and it consists of single connection vectors each of which contains 41 features and a class label (i.e. normal or attack). Thus, for evaluation of IDSs, any subset of these features can be utilised. For the experiments in this chapter, 32 continuous features of NSL-KDD (i.e. listed in Table 5.9) are used as ESR-NID is designed to classify instances based on the provided continuous input features. Additionally, all continuous values are scaled linearly between 0 and 1.

TABLE 5.9: KDD99 continuous features.

ID	Feature Name
1	duration
4	src.bytes
5	dst.bytes
8	wrong_fragment
9	urgent
10	hot
11	num_failed_logins
13	num_compromised
16	num_root
17	num_file_creations
18	num_shells
19	num_access_files
20	num_outbound_cmds
23	count
24	error_rate
25	error_rate
26	same_srv_rate
27	diff_srv_rate
28	srv_count
29	srv_error_rate
30	srv_rerror_rate
31	srv_diff_host_rate
32	dst_host_count
33	dst_host_srv_count
34	dst_host_same_srv_rate
35	dst_host_diff_srv_rate
36	dst_host_same_src_port_rate
37	dst_host_diff_host_rate
38	dst_host_error_rate
39	dst_host_srv_error_rate
40	dst_host_rerror_rate
41	dst_host_srv_rerror_rate

However, for the combined DARPA/CAIDA dataset, the captured raw network traffic will be preprocessed to extract suitable features. For this, using an entropy-based approach, entropy of selected packet attributes are calculated for a time-window of 10 seconds. These are *Entropy of source IP address*, *Entropy of source port number*, *Entropy of destination port number*, *Entropy of packet type* and *Entropy of packet size*. Entropy is a measure of the uncertainty associated with a random variable. The entropy H (that can be computed on a sample of consecutive packets) is defined as follows (Shannon, 1948):

$$H = - \sum_{i=1}^n p_i \log_2 p_i \quad (5.1)$$

Here, an information source has n independent symbols and each of them with probability of choice p_i .

After calculating the entropy values, the features were scaled linearly between 0 and 1 to be used by ESR-NID.

5.3 Experiments for Network Intrusion Detection

In this section, for evaluating the proposed algorithm for intrusion detection, ESR-NID is tested on the NSL-KDD dataset and the combined DARPA/CAIDA dataset using the tuned parameters found in Chapter 4 and then the results are analysed by comparing against five well-known machine learning techniques (J48, kNN, JRip, GASSIST-ADI and MPLCS). For these experiments, two separate sets of data are used for training and testing phases. Since the proposed algorithm is stochastic, each experiment is repeated 30 times with a different seed and the average results are reported. Similarly, for GASSIST-ADI and MPLCS, average values for 30 runs are presented. J48, kNN and JRip are deterministic, so only the value from a single run is reported for each test.

5.3.1 Parameter Settings

As mentioned before, no specific parameter tuning was carried out for ESR-NID when applied to these datasets. Instead the following settings that were found to be most reliable in the experiments with synthetic datasets were used.

- Fitness function = $(p) * no.ofInstancesCorrectlyDetectedby1rule$
 $+ (p - 1) * no.ofInstancesCorrectlyDetectedby2rules$
 $+ ... + (1) * no.ofInstancesCorrectlyDetectedbyAllRules$
 $- (p) * (\frac{no.ofabnormalInstances}{no.ofnormalInstances}) * no.ofErrors$
 , where p is the number of individuals (rules) in the population and $no.ofErrors$ is the number of normal instances incorrectly detected as anomalies.
- Mutation probability = 0.1
- Generations = 300
- Population size = 50
- Performance Function = g-performance
- Number of runs = 30

For the experiments in this chapter, the seed selection module was utilised to insert a set of rules as a proportion (20%) of the initial population. This enables the algorithm to evolve faster than with random initialisation.

For the other algorithms used for comparison, the best values found through parameter optimisation in section 4.6 are used in these experiments.

5.3.2 Experiments with NSL-KDD Dataset

As part of the proposed framework, a feature selection module can be optionally applied to input features to find the most relevant ones. To identify the important input features in the NSL-KDD dataset for building an efficient and effective

IDS, [Mukherjee & Sharma \(2012\)](#) investigated three feature selection methods implemented in WEKA 3.6: Correlation-based Feature Selection (CFS), Information Gain (IG) and Gain Ratio (GR). The result showed that the feature subset identified by CFS gave better classification accuracy than IG and GR. While a CFS method measures the individual predictive ability of each attribute along with the degree of redundancy between them, the Consistency Subset Evaluator (CSE) measures the inconsistency of a feature set given different class labels. With the former approach, the BestFirst search method is used, which performs greedy hill climbing with backtracking, whereas with CSE approach, the GreedyStepwise searches greedily through the space of attribute subsets. The CSE and CFS approaches were used in [Khor et al. \(2010\)](#) for finding relevant features for classifying network intrusions. The classification accuracy obtained using a Naïve Bayes Classifier (NBC) based on feature sets generated by these approaches were compared and NBC utilizing CSE showed better performance results. Both [Mukherjee & Sharma \(2012\)](#) and [Khor et al. \(2010\)](#) conducted their experiments on the KDD dataset.

In this section, three sets of experiments are carried out based on the three sets of features selected from NSL-KDD dataset: 8 continuous features (chosen by CFS method), 15 continuous features (chosen by CSE method) and all 32 continuous features (although some continues features in this dataset represent finite sets of values, the ranges of values are large enough to be treated as continues features). Table 5.10 and 5.11, show the 8 and 15 continuous attributes selected by CFS and CSE approaches, respectively.

TABLE 5.10: Eight features extracted using CFS and BestFirst.

Feature Name
src_bytes
dst_bytes
num_root
same_srv_rate
diff_srv_rate
srv_error_rate
dst_host_srv_diff_host_rate
dst_host_srv_error_rate

TABLE 5.11: Fifteen features extracted using CSE and GreedyStepwise.

Feature Name
duration
src_bytes
dst_bytes
count
srv_diff_host_rate
dst_host_count
dst_host_srv_count
dst_host_same_srv_rate
dst_host_diff_srv_rate
dst_host_same_src_port_rate
dst_host_diff_host_rate
dst_host_serror_rate
dst_host_srv_serror_rate
dst_host_rerror_rate
dst_host_srv_rerror_rate

In these experiments, the proposed system is trained using the pre-processed NSL-KDD training dataset and then the learned model (i.e., the optimised ruleset) is used for the classification of the test data, as this is the common practice in studies using NSL-KDD in the literature.

Table 5.12 presents the results obtained using ESR-NID, J48, JRip, kNN, GASSIST-ADI and MPLCS when applied to the NSL-KDD dataset with 8 features. For each classifier, in addition to the evaluation metrics (i.e. accuracy and number of rules for rule-based methods), g-performance value, true negative rate (TNrate) and true positive rate (TPrate) are also provided. As can be seen from the results, ESR-NID has similar performance (76.5%) to the other algorithms except that kNN does slightly better with 77% accuracy.

TABLE 5.12: Comparing the performance of ESR-NID on the NSL-KDD dataset with 8 features with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 245 number of leaves: 123	76.3	96.6	60.3	76
kNN	1 nearest neighbour	77.6	97	62.2	77
JRip	number of rules: 39	76.4	96.5	60	76.1
GASSIST-ADI	number of rules: 5	76.1	97	59.8	76
MPLCS	number of rules: 5	76	97	59.6	76
ESR-NID	number of rules: 9	76.5	96.5	60.8	76.2

Additionally, Table 5.13, 5.14 and 5.15 illustrate examples of the final rulesets generated by ESR-NID, GASSIST-ADI and MPLCS for classification of the NSL-KDD dataset with 8 features. In contrast to these, the outputs of the JRip and J48 cannot be presented due to their complexity (39 rules for JRip and 123 rules for J48).

TABLE 5.13: A ruleset generated by ESR-NID for the NSL-KDD dataset with 8 features.

rule1: <i>if</i> <i>diff_srv_rate</i> $\in [0.04\ 0.29]$ <i>then anomaly</i>
rule2: <i>if</i> <i>dst_bytes</i> $\in [0\ 1261360321]$ and <i>num_root</i> $\in [0\ 149]$ and <i>srv_error_rate</i> $\in [0\ 0.53]$ and <i>dst_host_srv_error_rate</i> $\in [0.34\ 1]$ <i>then anomaly</i>
rule3: <i>if</i> <i>srv_error_rate</i> $\in [0.98\ 1]$ and <i>dst_host_srv_diff_host_rate</i> $\in [0\ 0.02]$ and <i>dst_host_srv_error_rate</i> $\in [0\ 0.10]$ <i>then anomaly</i>
rule4: <i>if</i> <i>dst_bytes</i> $\in [0\ 26198748]$ and <i>num_root</i> $\in [0\ 149]$ <i>srv_error_rate</i> $\in [0.47\ 1]$ and <i>same_srv_rate</i> $\in [0\ 0.93]$ and <i>dst_host_srv_error_rate</i> $\in [0\ 0.75]$ <i>then anomaly</i>
rule5: <i>if</i> <i>dst_host_srv_diff_host_rate</i> $\in [0.23\ 0.93]$ and <i>dst_host_srv_error_rate</i> $\in [0\ 0.97]$ <i>then anomaly</i>
rule6: <i>if</i> <i>dst_bytes</i> $\in [0\ 26198748]$ and <i>srv_error_rate</i> $\in [0.61\ 0.92]$ and <i>dst_host_srv_diff_host_rate</i> $\in [0\ 0.02]$ <i>then anomaly</i>

TABLE 5.14: A ruleset generated by GASSIST-ADI for the NSL-KDD dataset with 8 features.

rule1: Att <i>dst_host_srv_error_rate</i> is $[>0.57]$ anomaly
rule2: Att <i>dst_bytes</i> is $[<436645800]$ Att <i>dst_host_srv_diff_host_rate</i> is $[>0.25]$ anomaly
rule3: Att <i>srv_error_rate</i> is $[>0.11]$ Att <i>dst_host_srv_diff_host_rate</i> is $[<0.09]$ anomaly
rule4: Default rule – > normal

TABLE 5.15: A ruleset generated by MPLCS for the NSL-KDD dataset with 8 features.

rule1: Att <i>num_root</i> is $[<2800.5]$ Att <i>dst_host_srv_diff_host_rate</i> is $[>0.25]$ anomaly
rule2: Att <i>num_root</i> is $[<5974.4]$ Att <i>same_srv_rate</i> is $[<0.96]$ Att <i>diff_srv_rate</i> is $[<0.12]$ Att <i>dst_host_srv_diff_host_rate</i> is $[<0.5]$ Att <i>dst_host_srv_error_rate</i> is $[<0.95]$ anomaly
rule3: Att <i>srv_error_rate</i> is $[>0.16]$ Att <i>dst_host_srv_diff_host_rate</i> is $[<0.04]$ anomaly
rule4: Att <i>srv_error_rate</i> is $[<0.5]$ Att <i>dst_host_srv_error_rate</i> is $[>0.8]$ anomaly
rule5: Default rule – > normal

The second sets of experiments are against the NSL-KDD dataset with 15 features chosen by CSE method. Table 5.16 illustrates the results obtained from ESR-NID, J48, kNN and JRip methods against the NSL-KDD dataset with 15 features. This time, the proposed algorithm performed slightly better than the kNN, GASSIST-ADI and MPLCS and slightly worse than the J48 and JRip. The complexity of the ESR-NID model will be medium and it sits between GASSIST-ADI and MPLCS with low complexity and J48 and JRip with high complexity.

TABLE 5.16: Comparing the performance of ESR-NID on the NSL-KDD dataset with 15 features with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 371 number of leaves: 186	78.4	97.2	63.3	77.9
kNN	1 nearest neighbour	75	93.4	60.3	74.6
JRip	number of rules: 46	77.8	96.1	63.1	77
GASSIST-ADI	number of rules: 6	75.3	92.1	61.6	74.7
MPLCS	number of rules: 6	73.3	91.4	58.8	72.8
ESR-NID	number of rules: 18	76.4	92.7	63.3	75.9

Finally, the ESR-NID is evaluated against the NSL-KDD dataset with all 32 continuous features and the results are presented in Table 5.17. In this set of experiments, ESR-NID performed slightly better than kNN, JRip, GASSIST-ADI and MPLCS and slightly worse than J48. Similar to the previous experiments (i.e. with 15 features) the complexity of the ESR-NID model is in between the GASSIST-ADI, MPLCS and J48, KNN.

TABLE 5.17: Comparing the performance of ESR-NID on the NSL-KDD dataset with all 32 continuous features with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 323 number of leaves: 162	82.8	95	72.3	82
kNN	1 nearest neighbour	75.2	93	60.9	75.1
JRip	number of rules:38	76.3	95	61.3	76.8
GASSIST-ADI	number of rules: 6	74.9	93.4	60.2	74.5
MPLCS	number of rules: 5	75.5	90.8	62.9	74.9
ESR-NID	number of rules: 19	78.1	87	70	78

In conclusion, the results indicate that ESR-NID has similar performance to the other algorithms in all cases, except that J48 does slightly better when there are many more features, and kNN, JRip, GASSIST-ADI and MPLCS do slightly worse. In all cases, ESR-NID discovers rulesets up to an order of magnitude smaller than those found by J48, and having only 23-50% as many rules as JRip. When the input data is more complex and includes all the attributes, although GASSIST-ADI and MPLCS generated less rules than ESR-NID, this decreased the effectiveness of

the system in detection of anomalous records (and as a result the accuracy of the system) as can be seen from the true positive rates in Table 5.17.

5.3.3 Experiments with DARPA/CAIDA Dataset

This section presents the results obtained from the experiments on the combined DARPA/CAIDA dataset. As can be seen from Table 5.18, in this classification problem, ESR-NID, kNN, GASSIST-ADI and MPLCS produced better results than J48 and JRip classifiers. The complexity of the ESR-NID model is also low and comparable to other classifiers as can be seen in Table 5.19. ESR-NID and GASSIST-ADI are similarly using two features from the collected statistical attributes (i.e. *Entropy of source IP address*, *Entropy of source port number*) while J48, JRip and MPLCS are only utilising *Entropy of source IP address*. In the GASSIST-ADI ruleset, one redundant rule can be seen because the instances that can be detected by *Entropy of source IP address is [> 0.37] | anomaly* rule could also be classified using *Entropy of source IP address is [> 0.43] | anomaly* rule. As a result, ESR-NID ruleset is considered a more compact model. The J48 approach partitioned only one attribute (i.e. *Entropy of source IP address*) range into two intervals using a single cut point. This resulted into a poor classification accuracy of 84%. This is in contrast to JRip and MPLCS approaches, which provided several intervals of the same feature using a set of cut points and thus generated better classification accuracy.

As the complexity of these rulesets is low, the attributes ranges can be easily compared among different approaches. For example, both ESR-NID and GASSIST-ADI generated approximately the same range for the *Entropy of source port number* (i.e. [0.09, 0.32] in ESR-NID ruleset and [0.09, 0.29] in GASSIST-ADI ruleset).

TABLE 5.18: Comparing the performance of ESR-NID on the combined DARPA/-CAIDA dataset with J48, kNN, JRip, GASSIST-ADI and MPLCS performances.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 3 number of leaves: 2	81.9	86.7	77.4	84
kNN	1 nearest neighbour	98.3	100	96.7	98
JRip	number of rules:3	96.7	100	93.5	97.9
GASSIST-ADI	number of rules: 4	99.2	100	98.5	98.9
MPLCS	number of rules: 4	98.3	100	96.7	98.9
ESR-NID	number of rules: 2	97.8	99.7	96	98.4

TABLE 5.19: Comparing the final ruleset generated by ESR-NID, J48, JRip, GASSIST-ADI and MPLCS.

Approach	Output
ESR-NID	rule1: <i>if Entropy of source IP address</i> $\in [0.38, 0.50]$ <i>then anomaly</i> rule2: <i>if Entropy of source port number</i> $\in [0.09, 0.32]$ <i>then anomaly</i>
J48	<i>Entropy of source IP address</i> > 0.37 : anomaly <i>Entropy of source IP address</i> ≤ 0.37 : normal
JRip	(<i>Entropy of source IP address</i> ≥ 0.39): anomaly (<i>Entropy of source IP address</i> ≤ 0.27) and (<i>Entropy of source IP address</i> ≥ 0.17): anomaly Others: normal
GASSIST-ADI	<i>Entropy of source port number</i> is $[0.09, 0.29]$ anomaly <i>Entropy of source IP address</i> is $[> 0.43]$ anomaly <i>Entropy of source IP address</i> is $[> 0.37]$ anomaly Default rule – $>$ normal
MPLCS	<i>Entropy of source IP address</i> is $[> 0.44]$ anomaly <i>Entropy of source IP address</i> is $[0.10, 0.32]$ anomaly <i>Entropy of source IP address</i> is $[> 0.39]$ anomaly Default rule – $>$ normal

5.4 ESR-NID for Detecting Normal Instances

In this section, a set of experiments will be conducted to explore the use of ESR-NID for detecting normal instances. When there is not enough knowledge about attacks in the captured data, it would be more useful to utilise ESR-NIS as an anomaly

detector to generate rules for normal background traffic. Any deviation from the normal model will be detected as an attack and as a result ESR-NID will fire an alarm showing suspicious behaviour in the network.

In this experiment, the NSL-KDD training and testing datasets with all continuous features are used and the results are compared against the other machine learning methods. Here, the positive examples are the normal records. Table 5.20 summarizes the results. The accuracy of the J48 model is the best but this model is very complex with 162 leaves. After J48, kNN and JRip provide slightly better results than ESR-NID, GASSIST-ADI and MPLCS with approximately similar classification accuracy values. If the aim in designing a classifier for detection of normal instances is to make it more accurate in matching normal cases (positives), then TPrate can also be used as a metric for evaluating the approaches. Here, J48 and kNN produced higher TPrates than ESR-NID while kNN, GASSIST-ADI and MPLCS provided less values. Among the tested rule-based methods, the complexity of ESR-NID model is medium as it is between GASSIST-ADI and MPLCS with low complexity and JRip and J48 with high complexity.

Comparing Table 5.17 and 5.20 results, when the classifiers were designed for detection of anomalies, less rules were generated by JRip, GASSIST-ADI, MPLCS and ESR-NID approaches. However, this is not the case for J48 since in both experiments for detection of anomalies and normal instances, the complexity of the generated models is the same (size of tree: 323, number of leaves: 162). Moreover, the JRip, GASSIST-ADI, MPLCS and ESR-NID classifiers produced higher accuracy results for detecting intrusions compared to the results achieved for detecting normal instances.

Since the aim of the experiments in this section is to develop a model that matches maximum number of normal instances, a system designer might need to put more emphasis on detection of these input records. One of the aspects of ESR-NID is its flexibility on the choice of fitness and performance functions. In this section, ESR-NID will be evaluated with a different performance function with the aim of providing better true positive rate. A better true positive rate increases the sensitivity of the

TABLE 5.20: Performance of ESR-NID for detecting normal instances.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 323 number of leaves: 162	83.5	72	97	83
kNN	1 nearest neighbour	75.6	60.9	93.9	75.1
JRip	number of rules:41	74.9	57.7	97.4	74.7
GASSIST-ADI	number of rules:7	72.9	56.9	93.4	72.6
MPLCS	number of rules:7	73.7	58.3	93.3	73.4
ESR-NID	number of rules:22	72.3	55.9	95.1	72.3

detection system to match most of the input normal records. For this purpose, the following performance function is used:

$$performance\ function = \sqrt[3]{TP_{rate} * TP_{rate} * TN_{rate}} \quad (5.2)$$

As can be seen from Table 5.21, the true positive rate increases when a different performance function with more emphasis on this metric (TP_{rate}) is used in the design of IDS. Although the customised model produced more rules compared to the previous model, it provided a more accurate classifier with 75.4% accuracy. It also outperformed kNN, JRip, GASSIST-ADI and MPLCS in terms of accuracy.

TABLE 5.21: Performance of customised ESR-NID as an anomaly detection system.

Classifier	Specifications	g-performance	TNrate	TPrate	Accuracy
J48	Size of tree: 323 number of leaves: 162	83.5	72	97	83
kNN	1 nearest neighbour	75.6	60.9	93.9	75.1
JRip	number of rules:41	74.9	57.7	97.4	74.7
GASSIST-ADI	number of rules:7	72.9	56.9	93.4	72.6
MPLCS	number of rules:7	73.7	58.3	93.3	73.4
ESR-NID (Performance function 3.8)	number of rules:22	72.3	55.9	95.1	72.3
Customised ESR-NID (Performance function 5.2)	number of rules:25	76.3	58.8	99.2	75.4

5.5 Summary

This chapter evaluated ESR-NID on network intrusion detection problems. For this, the existing publicly available datasets and the selected ones for this study were introduced. NSL-KDD and a combined DARPA/CAIDA datasets are the selected datasets for the experiments in this chapter. Since ESR-NID is designed to classify examples based on the provided continuous input features, some statistical features of network traffic were used for the classification task. Processed NSL-KDD dataset has 32 continuous and 9 discrete features. Therefore, 32 continuous attributes were selected. Additionally, a feature selection module was applied to the continuous features of NSL-KDD dataset to reduce the size of search space by selecting the most relevant attributes. Two different feature selection methods were used in this chapter: Correlation-based Feature Selection (CFS) and Consistency Subset Evaluator (CSE). These two methods selected 8 and 15 attributes of the NSL-KDD dataset with 32 continuous features. In addition to evaluation of ESR-NID on the NSL-KDD dataset with 8 and 15 features, it was also evaluated on the data with all the 32 features. For the combined DARPA/CAIDA dataset, the raw network traffic should be pre-processed to extract suitable features for intrusion detection. In this study, entropy of selected packet attributes are calculated over a time-window of 10 seconds. These are Entropy of source IP address, Entropy of source port number, Entropy of destination port number, Entropy of packet type and Entropy of packet size. For both datasets, in the pre-processing stage of ESR-NID framework, the input continuous values were scaled linearly between 0 and 1. For these experiments, ESR-NID was configured using the tuned parameters found in the previous chapter. The performance of ESR-NID was compared against five well-known machine learning techniques. These techniques are from two categories of GA-based and non-GA-based algorithms: J48, kNN and JRip (non-GA-based) and GASSIST-ADI and MPLCS (GA-based). The results showed that the performance of ESR-NID is comparable to the other tested methods and produces compact, easily understood rulesets.

Additionally, ESR-NID was evaluated for generating an effective model for detecting

normal instances. This can be a useful method when there is not enough information about attacks in the captured data. One of the advantages of ESR-NID is that depending on the problem domain and the characteristics of the dataset, it is customisable by changing the fitness and performance function to provide customised optimal results. This option can not be found in other techniques and thus makes ESR-NID a more flexible model. This has been shown in an experiment, where the aim is to increase the true positive rate. Thus, a new performance function was introduced, which puts more emphasis on detection of hits (i.e. normal records in this experiment). The use of ESR-NID as an anomaly detector is an area that requires further investigation in the future.

Chapter 6

Adaptation in a Dynamic Environment

6.1 Introduction

This chapter presents how ESR-NID can be utilised for incremental learning in a dynamic environment. Using the proposed approach, ESR-NID will be able to frequently update its database of rules to detect new attacks.

The real environment in which an IDS is deployed is continuously changing because of network topology and technology changes ([Kuwatly et al., 2004](#)) and certain worms or attacks gain and lose popularity or new attacks come into existence ([Pietraszek, 2004](#)). High false positive and false negative rates produced by deployed IDSs can be the result of such changes in the environment. This has been considered as one of the challenging problems in the community and thus requires attention from intrusion detection analysts ([Pietraszek, 2004](#)). An IDS should be able to adapt to the changing environment with the least amount of manual intervention. Various approaches have been suggested in the literature for learning new information when new training data becomes available. With the increasing and diverse types of novel network attacks, researchers attempt to use incremental learning in IDSs to enable

them to adapt themselves to new attacks without forgetting previously learned information. This also increases IDS's performance, efficiency and sustainability (Nasr et al., 2014). For example, one approach is to discard the existing classifier and retrain using the entire accumulated training data. However, it would be very expensive to rebuild the classifier after each new input instance becomes available. To address this problem, a batch incremental learning approach can be used to handle training examples in batches. The size of these batches can be decided based on a constant value or depending on the current performance of the classifier (Pietraszek, 2004; Polikar et al., 2001). This approach, however, suffers from the drawback that the size of the training dataset grows infinitely over time and as a result the training time increases.

To address this issue, in this chapter, an incremental learning technique is incorporated into ESR-NID that meets the following criteria:

- preserves previously acquired knowledge (i.e. rules)
- does not require access to the original data used to train the current classifier
- learns additional information from new data and updates the database of signatures with the new information

The proposed framework for ESR-NID to make it adaptable to environment changes is explained in the next section.

6.2 Incremental Learning for ESR-NID

An effective IDS should be able to incrementally learn and adapt to changes in the environment, the behaviour of users and the pattern of attacks. In this section, an incremental learning approach is proposed for enhancing ESR-NID. Figure 6.1 gives an overview of the proposed incremental learning model. It consists of two phases: startup/classification phase and update phase. The startup phase (shown

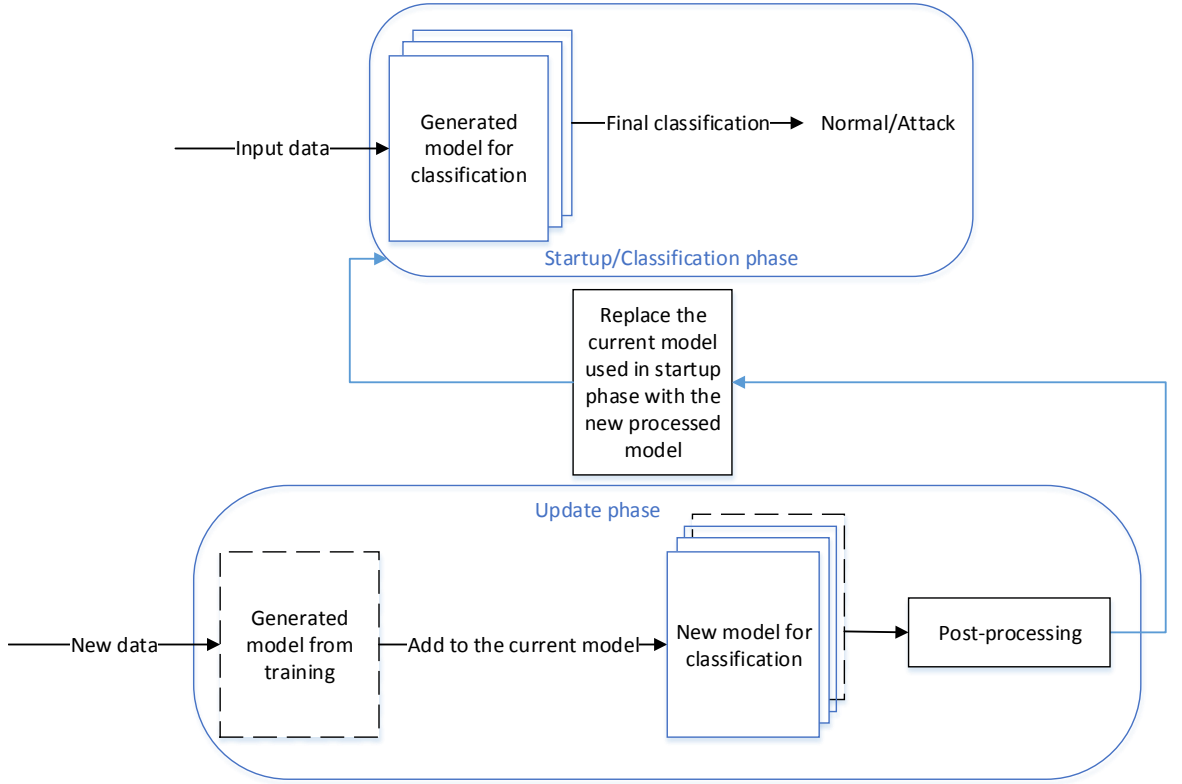


FIGURE 6.1: Incremental learning used in ESR-NID.

in the upper box in Figure 6.1) is the stage that an IDS is designed to perform classification in its intended environment for the first time. In the startup phase, an initial set of training examples is collected and given to ESR-NID to generate a classification model for the given environment.

Once the startup phase is accomplished, the existing model will operate continuously in the classification mode. However, further learning might be needed from time to time, as defined by network administrators.

In the update phase, ESR-NID updates its database of rules (signatures) using the new batch of data accumulated over the pre-defined time window. ESR-NID uses this new training data to generate some new rules. These rules are added to the current database of signatures to constitute a new model for classification. The new model then go through a post-processing stage (similar to the one in Section 3.1.3) to produce a more concise ruleset for the use of IDS. In the post-processing stage similar rules are removed. Similarity between rules is determined by comparing the

values of their active features, using a user-defined cut-off for the number of digits of precision in floating-point numbers as features are continuous real values. Finally, the new processed model will be used for classification tasks in the classification phase.

In this framework, instead of preserving all the training data, only the previous rule-set is maintained, which is much smaller than the whole training dataset. Another advantage of this framework is that less time and computational resources will be needed to train the system on only a batch of new data, compared to learning by retraining the system using all of the data that has been accumulated thus far.

To evaluate the proposed incremental learning approach for ESR-NID, a set of experiments is carried out using the evaluation strategy explained in the next section. First, ESR-NID will be trained using the existing training data in the startup phase to generate a model for classification of normal and attack records. Then, for the update phase, two different learning techniques are used, which lead to further comparisons. These are traditional learning and incremental learning (i.e. the method used for ESR-NID). For this phase, it is assumed that a new data is captured over a pre-defined time interval and the system is ready to be updated.

6.3 Evaluation Strategy

In the experiments in this chapter, two sets of data are always provided: old data and new data. The old data is used in the startup phase for generating a model for IDS, while it is assumed that the new data is the new incoming flow of network traffic that has been collected over a pre-defined time interval. As the new data becomes available, the existing IDS needs to be updated. For the updating phase, as can be seen in Figure 6.2, the proposed incremental learning approach for ESR-NID is compared with a traditional learning scheme, which requires the entire old and new data to adapt to a changing environment.

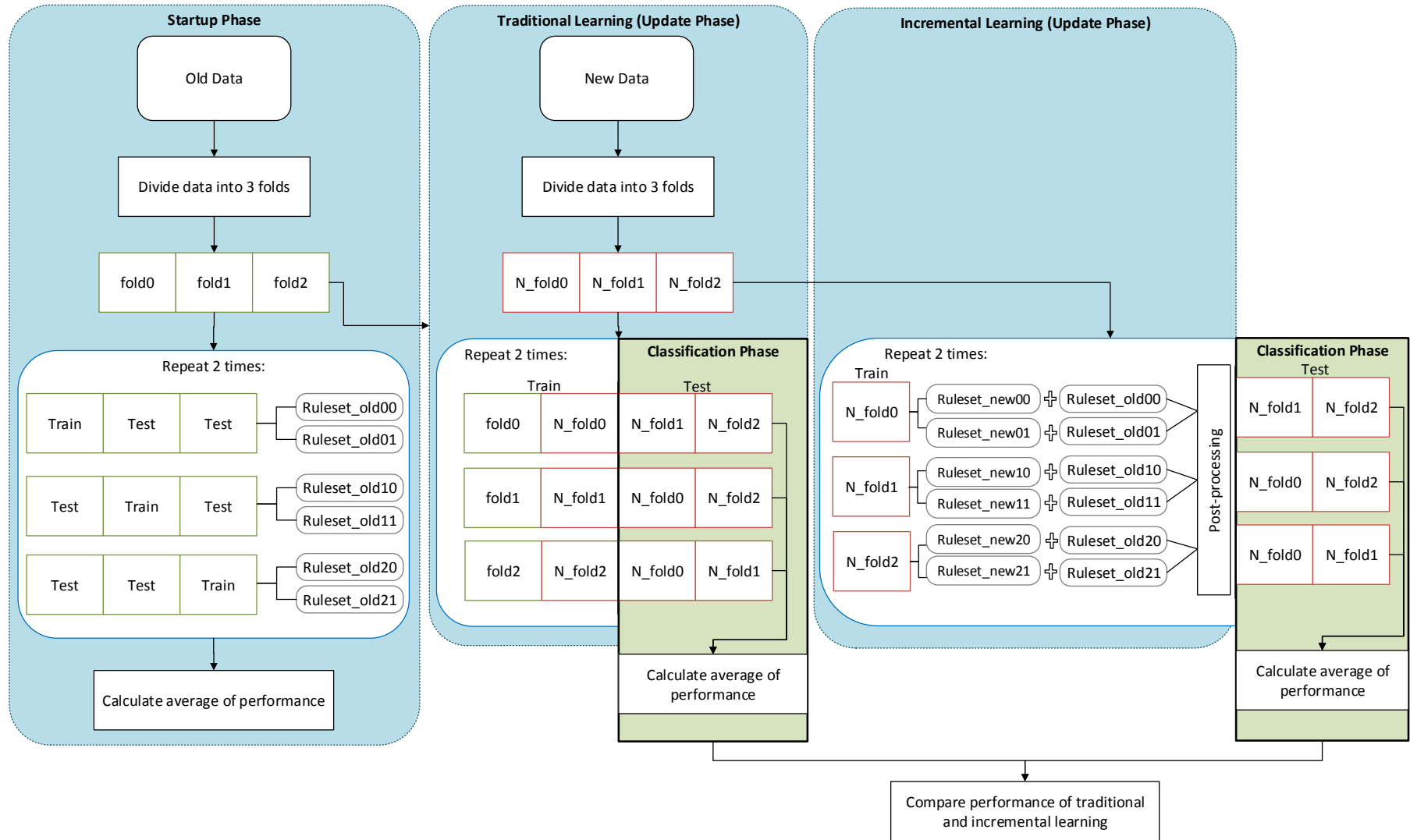


FIGURE 6.2: Evaluation strategy used in the experiments in this chapter.

To provide an estimate of classification rate, the old and new datasets are divided into three folds to be used in a reversed 3-fold cross-validation approach, where 2 folds are used to test the fitness of model. However, if there were no time constraints, more folds could be used to improve performance estimation. The folds are stratified so that they contain approximately the same proportions of two types of classes as the original dataset. Additionally, the evaluation method in this chapter used two different seeds to provide a total of six runs using the three folds of data. Therefore, the evaluation does not depend on a single outcome. Using the evaluation strategy presented in Figure 6.2, three different sets of experiments will be conducted for each scenario in this chapter to compare the performance of traditional and incremental learning schemes. The first set is designed to generate a model for intrusion detection based on the existing data (referred to as old data). This set is similar to the experiments conducted in Chapters 4 and 5 for generating classifiers for different classification tasks. The other two sets of experiments are related to the update phase. Two different methods of learning are used in these experiments: traditional learning and incremental learning. In the traditional learning, the system needs both old and new datasets because in each run, ESR-NID is trained on one fold of old data (e.g., fold0) combined with one fold of new data (e.g., N_fold0) and tested on the remaining folds of new data (e.g., N_fold1 and N_fold2). Therefore, in the traditional learning experiments, storing the old data is essential for future retraining. On the other hand, for incremental learning, the system only requires the new data and previously generated rulesets (e.g., Ruleset_old00) in the startup phase. In these experiments, ESR-NID is trained twice on the three folds of new data using two different seeds. Then, the produced rulesets (e.g., Ruleset_new00) are added to the rulesets generated for the old data (e.g., Ruleset_old00) and after post-processing of rulesets, they are tested against the folds of new data (e.g., N_fold1 and N_fold2) that were kept apart. As the proposed incremental learning approach only stores the previous acquired rules, it requires less storage compared to the traditional learning scheme.

In the evaluation strategy presented in Figure 6.2, an average of performance is

calculated for each set of experiments and thus comparison of traditional and incremental learning approaches is facilitated.

In the following sections, 3 scenarios are defined for experiments on synthetic datasets. The problem with 6 input features explained in Section 4.2.3 is used as the old data for these scenarios and additionally 3 new datasets are also generated. Moreover, the NSL-KDD dataset with 8 features (listed in Table 5.10) is used for the experiments on the NSL-KDD dataset. This dataset has two separate training and testing data. The NSL-KDD training data represents the old data and the NSL-KDD testing data is used as the new incoming data for the experiments in this chapter.

6.3.1 Experiments with Synthetic Datasets

In this section, the problem with 6 input features defined in Chapter 4 for evaluation of ESR-NID against higher dimensional datasets is used as the old dataset. The following rules were used to generate this dataset:

Normal: *if* $f1 \in [0, 0.3]$ *and* $f2 \in [0.3, 0.6]$ *and* $f3 \in [0, 0.2]$ *and* $f4 \in [0, 0.3]$ *and* $f5 \in [0.3, 0.6]$ *and* $f6 \in [0, 0.2]$ *then normal*

Attack1: *if* $f1 \in [0, 0.3]$ *and* $f2 \in [0, 0.2]$ *and* $f3 \in [0.4, 0.8]$ *and* $f4 \in [0, 0.3]$ *and* $f5 \in [0, 0.2]$ *and* $f6 \in [0.4, 0.8]$ *then anomaly*

Attack2: *if* $f1 \in [0, 0.1]$ *and* $f2 \in [0.5, 0.9]$ *and* $f3 \in [0, 0.3]$ *and* $f4 \in [0, 0.1]$ *and* $f5 \in [0.5, 0.9]$ *and* $f6 \in [0, 0.3]$ *then anomaly*

Three different scenarios were then designed to generate new datasets for evaluation of the proposed learning approach for ESR-NID. These are:

- Scenario 1: a new version of an old attack becomes available (old attacks appear in the new data).
- Scenario 2: a completely new attack becomes available (old attacks appear).

- Scenario 3: a completely new attack becomes available (old attacks do not appear).

In the first two scenarios, the new data contains the same normal records (that existed in the old dataset) and both old and new attacks. However, in the third one, it is assumed that the old attacks are not popular any more and thus in the new data, there is no record of previous attacks. The reason to evaluate the system on synthetic data is that the properties of data can be controlled to meet various conditions and validation of final rulesets against the rules used to generate the data can be easily carried out.

After presenting the parameters defined for the experiments in the next section, the following three sections will explain the experiments in more details and discuss the classification results for each synthetic problem.

6.3.1.1 Parameter Settings

For the experiments in this section, ESR-NID is configured using the following settings:

- Fitness Function = fitness function (3) (Equation (3.6))
- Performance function = g-performance ($\sqrt{TP_{rate} * TN_{rate}}$)
- Mutation probability = 0.1
- Population size = 50
- Generations = 300
- Number of runs = 6

6.3.1.2 Scenario 1

This scenario simulates the case of facing a new variation of an existing network attack in the new flow of network traffic. For generating the data for the first scenario, the following rules were used:

Normal: *if* $f1 \in [0, 0.3]$ *and* $f2 \in [0.3, 0.6]$ *and* $f3 \in [0, 0.2]$ *and* $f4 \in [0, 0.3]$ *and* $f5 \in [0.3, 0.6]$ *and* $f6 \in [0, 0.2]$ *then normal*

Attack1: *if* $f1 \in [0, 0.3]$ *and* $f2 \in [0, 0.2]$ *and* $f3 \in [0.4, 0.8]$ *and* $f4 \in [0, 0.3]$ *and* $f5 \in [0, 0.2]$ *and* $f6 \in [0.4, 0.8]$ *then anomaly*

Attack2: *if* $f1 \in [0, 0.1]$ *and* $f2 \in [0.5, 0.9]$ *and* $f3 \in [0, 0.3]$ *and* $f4 \in [0, 0.1]$ *and* $f5 \in [0.5, 0.9]$ *and* $f6 \in [0, 0.3]$ *then anomaly*

Attack3 (a new variation of an old attack (i.e. attack1)): *if* $f1 \in [0.1, 0.4]$ *and* $f2 \in [0.1, 0.3]$ *and* $f3 \in [0.5, 0.9]$ *and* $f4 \in [0.1, 0.4]$ *and* $f5 \in [0.1, 0.3]$ *and* $f6 \in [0.5, 0.9]$ *then anomaly*

Using the evaluation strategy presented in Figure 6.2, three sets of experiments were conducted and the average results were calculated. Table 6.1 shows the average results for different phases. Additionally, the g-performance results (only for traditional and incremental learning phases) are presented using a box plot in Figure 6.3 (incremental learning has a tighter interquartile range than traditional learning). To compare the final rulesets generated through traditional and incremental learning, Table 6.2 provides examples of the old ruleset generated during startup phase and rulesets obtained from traditional learning and incremental learning when new training data becomes available. The attacks that can be detected using each rule are also listed in the third column of Table 6.2. For example, in startup phase, rule1 matches attack number 1 and 2 in the old dataset. In the ruleset produced from the incremental learning approach, rule3, which was existed in the database of signatures generated during the startup phase, is a general version of rule6. This rule can be eliminated from the final ruleset by improving the post-processing stage. As a result, the number of rules can be reduced and less complex model would be produced for the classification task. However, for the synthetic problems, which are only

designed for testing and evaluating the proposed system, this extra post-processing step is not carried out and as will be seen later, this issue (i.e. having redundant rules) rarely happens in more complicated high dimensional real problems (See the experiments with NSL-KDD dataset in Section 6.3.2).

The results in Table 6.1 show that the performance of proposed incremental learning for ESR-NID (97%) is slightly worse than traditional learning (97.35%). The incremental learning approach also produced more rules compared to the traditional learning approach because it keeps the previous rules in its database. However, the incremental learning scheme benefits from less required storage because it only needs to maintain the database of rules which is much smaller than the whole training data needed for each re-training phase in the traditional approach. Comparing the true positive rates, the incremental learning produced a slightly better result (98.1% compared to 97.1%).

TABLE 6.1: Average results for startup and update phases in the first scenario.

Phase	Number of rules	g-performance	TNrate	TPrate	Accuracy
Startup	4	96.7	97.08	96.34	96.67
Traditional Learning (update)	5	97.35	97.52	97.19	97.37
Incremental Learning (update)	7	97	95.8	98.1	96.9

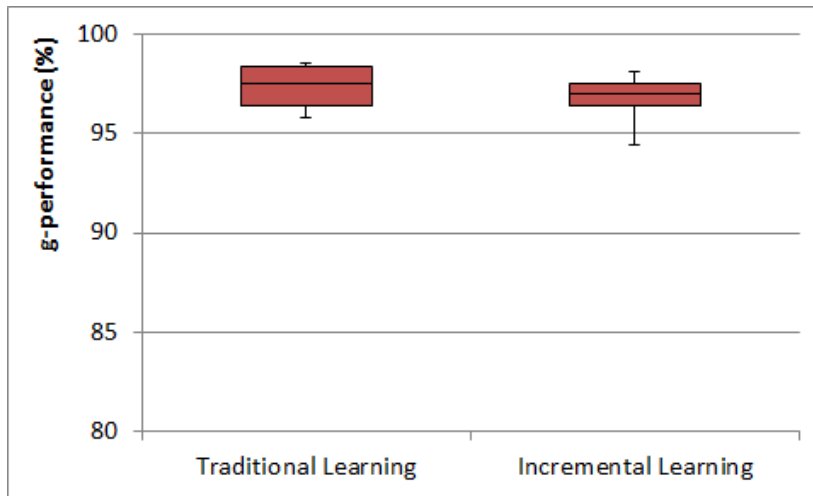


FIGURE 6.3: Performance of different learning approaches in the first scenario.

TABLE 6.2: Comparing the final rulesets generated by ESR-NID during startup and update phases in the first scenario.

Phase	Ruleset	Attack Number
Startup	rule1: <i>if</i> $F3 \in [0.19, 0.78]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.59, 0.89]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.60, 0.89]$ <i>then anomaly</i>	1, 2 1, 2 2 2
Traditional Learning (Update)	rule1: <i>if</i> $F3 \in [0.19, 0.89]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.60, 0.88]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.59, 0.88]$ <i>then anomaly</i>	1, 2, 3 1, 2, 3 2 2
Incremental Learning (Update)	rule1: <i>if</i> $F3 \in [0.19, 0.78]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.59, 0.89]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.60, 0.89]$ <i>then anomaly</i> rule5: <i>if</i> $F3 \in [0.22, 0.85]$ <i>then anomaly</i> rule6: <i>if</i> $F2 \in [0.61, 0.89]$ <i>then anomaly</i>	1, 2, 3 1, 2, 3 2 2 1, 2, 3 2

6.3.1.3 Scenario 2

The second scenario is for the case of seeing a completely new attack in the new collected data. Here, it is assumed that old attacks are still popular and can be seen in the new data. So, the following rules were used for generating the data:

Normal: *if* $f1 \in [0, 0.3]$ *and* $f2 \in [0.3, 0.6]$ *and* $f3 \in [0, 0.2]$ *and* $f4 \in [0, 0.3]$ *and* $f5 \in [0.3, 0.6]$ *and* $f6 \in [0, 0.2]$ *then normal*

Attack1: *if $f1 \in [0, 0.3]$ and $f2 \in [0, 0.2]$ and $f3 \in [0.4, 0.8]$ and $f4 \in [0, 0.3]$ and $f5 \in [0, 0.2]$ and $f6 \in [0.4, 0.8]$ then anomaly*

Attack2: *if $f1 \in [0, 0.1]$ and $f2 \in [0.5, 0.9]$ and $f3 \in [0, 0.3]$ and $f4 \in [0, 0.1]$ and $f5 \in [0.5, 0.9]$ and $f6 \in [0, 0.3]$ then anomaly*

Attack3 (completely new attack): *if $f1 \in [0.5, 0.8]$ and $f2 \in [0.2, 0.4]$ and $f3 \in [0.8, 1]$ and $f4 \in [0.5, 0.7]$ and $f5 \in [0.9, 1]$ and $f6 \in [0.9, 1]$ then anomaly*

A set of experiments similar to those in the previous section was carried out to compare the performance of ESR-NID using traditional and incremental learning. Table 6.3 presents the average results. The g-performance rates for the two different learning approaches are also demonstrated in Figure 6.4. To provide a more detailed picture of the final rulesets generated during different phases, Table 6.4 presents the rulesets and the attacks that each rule is able to detect. As old attacks appear in the new data, when ESR-NID used traditional learning approach for this problem, it generated a set of more general rules that are useful for detecting both the old and new attacks. For example, rule2 in the traditional learning phase is a general version of rule2 in the startup phase. The wider range produced by traditional learning for F6 attribute helps the system classify the new attack (Attack3) as well as the old attacks. Using the incremental learning scheme, ESR-NID utilised the previous ruleset generated during the startup phase for detecting old attacks and generated a set of rules, which can be useful for classification of both attacks as the new data contains all types of attacks.

In this scenario, similar to the previous scenario, the traditional learning produced slightly better g-performance, accuracy and true negative rates than the incremental learning. However, the true positive rate for the incremental learning is slightly better than the traditional learning. Despite the larger ruleset generated by the incremental learning approach, this method is more desirable and less expensive because it does not require to maintain the whole training data over time.

TABLE 6.3: Average results for startup and update phases in the second scenario.

Phase	Number of rules	g-performance	TNrate	TPrate	Accuracy
Startup	4	96.7	97.08	96.34	96.67
Traditional Learning (update)	6	98.78	99.79	97.79	98.59
Incremental Learning (update)	8	97.1	95.67	98.63	97.4

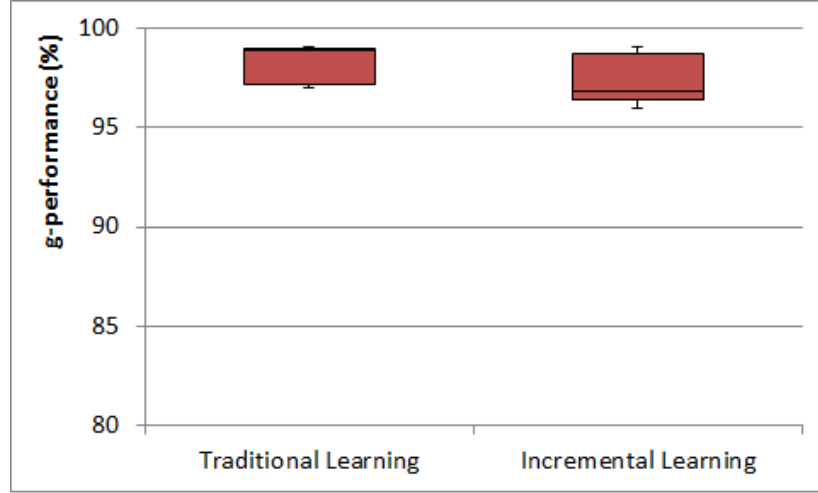


FIGURE 6.4: Performance of different learning approaches in the second scenario.

6.3.1.4 Scenario 3

This section evaluates the use of incremental learning for ESR-NID in a scenario that a completely new attack becomes available assuming that previous attack types lose popularity and they will not exist in the new data. For this scenario, the following rules were used for generating the new training data:

Normal: *if* $f1 \in [0, 0.3]$ *and* $f2 \in [0.3, 0.6]$ *and* $f3 \in [0, 0.2]$ *and* $f4 \in [0, 0.3]$ *and* $f5 \in [0.3, 0.6]$ *and* $f6 \in [0, 0.2]$ *then normal*

Attack3 (completely new attack): *if* $f1 \in [0.5, 0.8]$ *and* $f2 \in [0.2, 0.4]$ *and* $f3 \in [0.8, 1]$ *and* $f4 \in [0.5, 0.7]$ *and* $f5 \in [0.9, 1]$ *and* $f6 \in [0.9, 1]$ *then anomaly*

The average results are presented in Table 6.5. A box plot is also used to compare the g-performance of traditional and incremental learning approaches in Figure 6.5.

TABLE 6.4: Comparing the final rulesets generated by ESR-NID during startup and update phases in the second scenario.

Phase	Ruleset	Attack Number
Startup	rule1: <i>if</i> $F3 \in [0.19, 0.78]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.59, 0.89]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.60, 0.89]$ <i>then anomaly</i>	1, 2 1, 2 2 2
Traditional Learning (Update)	rule1: <i>if</i> $F3 \in [0.20, 0.98]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.99]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.60, 0.88]$ <i>and</i> $F5 \in [0.34, 0.91]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.64, 0.99]$ <i>then anomaly</i> rule5: <i>if</i> $F5 \in [0.50, 0.98]$ <i>then anomaly</i> rule6: <i>if</i> $F5 \in [0.64, 0.85]$ <i>then anomaly</i>	1, 2, 3 1, 2, 3 2, 3 2, 3 2, 3 2
Incremental Learning (Update)	rule1: <i>if</i> $F3 \in [0.19, 0.78]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.59, 0.89]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.60, 0.89]$ <i>then anomaly</i> rule5: <i>if</i> $F3 \in [0.20, 0.99]$ <i>then anomaly</i> rule6: <i>if</i> $F5 \in [0.60, 0.99]$ <i>then anomaly</i> rule7: <i>if</i> $F6 \in [0.20, 0.99]$ <i>then anomaly</i>	1, 2 1, 2 2 2 1, 2, 3 2, 3 1, 2, 3

Additionally, the final rulesets generated from different phases of evaluation process are demonstrated in Table 6.6. In this scenario, although the average results for the traditional learning are slightly better than the incremental learning, the same number of rules is generated for both approaches. As the old attacks do not appear in the new batch of data, the incremental learning approach generated a set of new rules (rule5, rule6 and rule7), which are only useful for classifying the new attack (Attack3). These rules will be added to the previous rules in the database of signature for future classification. On the other hand, when ESR-NID uses the traditional learning approach, a set of rules will be generated for the entire data (both old and new data). Examples of these rules are rule1 and rule2 from the traditional learning phase, which are useful for all three types of attacks.

TABLE 6.5: Average results for startup and update phases in the third scenario.

Phase	Number of rules	g-performance	TNrate	TPrate	Accuracy
Startup	4	96.7	97.08	96.34	96.67
Traditional Learning (update)	7	99.3	98.8	99.8	99
Incremental Learning (update)	7	98.5	97.3	99.5	98.4

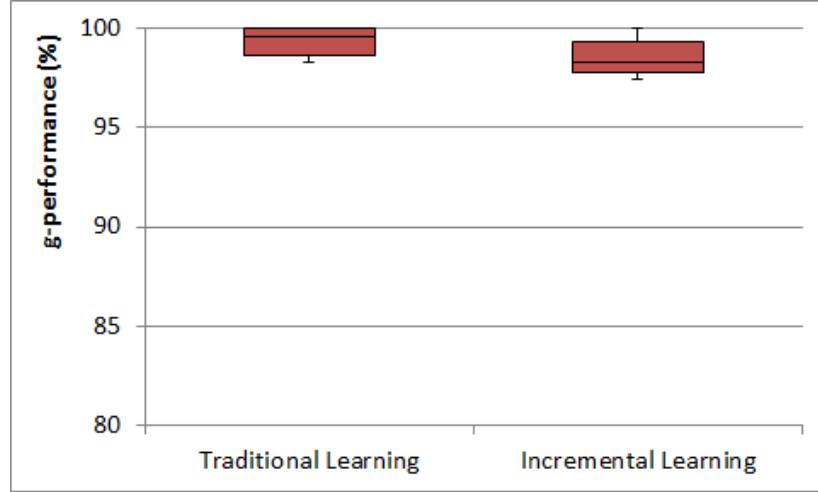


FIGURE 6.5: Performance of different learning approaches in the third scenario.

6.3.2 Experiments with NSL-KDD Dataset

To evaluate the incremental learning for ESR-NID, a set of experiments is also conducted using the evaluation strategy presented in Figure 6.2 against the NSL-KDD dataset with 8 features. The NSL-KDD training dataset includes 21 different attacks out of the 37 present in the test dataset. Therefore, the NSL-KDD test dataset contains both old and new attacks and it will be used as the new incoming data for the experiments in this section. The same settings that were found to be most reliable in the experiments with synthetic datasets were also used for the experiments against NSL-KDD dataset.

The average results for different phases of startup, traditional and incremental learning are presented in Table 6.7. In this real intrusion detection problem, the incremental learning produced better g-performance, accuracy and true positive rates using

TABLE 6.6: Comparing the final rulesets generated by ESR-NID during startup and update phases in the third scenario.

Phase	Ruleset	Attack Number
Startup	rule1: <i>if</i> $F3 \in [0.19, 0.78]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.59, 0.89]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.60, 0.89]$ <i>then anomaly</i>	1, 2 (old) 1, 2 (old) 2 (old) 2 (old)
Traditional Learning (Update)	rule1: <i>if</i> $F3 \in [0.20, 0.99]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.24, 0.98]$ <i>then anomaly</i> rule3: <i>if</i> $F5 \in [0.61, 0.99]$ <i>then anomaly</i> rule4: <i>if</i> $F6 \in [0.21, 0.85]$ <i>then anomaly</i> rule5: <i>if</i> $F6 \in [0.20, 0.75]$ <i>then anomaly</i> rule6: <i>if</i> $F2 \in [0.62, 0.88]$ <i>and</i> $F5 \in [0.08, 0.99]$ rule7: <i>if</i> $F2 \in [0.61, 0.82]$ <i>and</i> $F5 \in [0.08, 0.99]$	1, 2 (old), 3 (new) 1, 2 (old), 3 (new) 2 (old), 3 (new) 1, 2 (old) 1, 2 (old) 2 (old) 2 (old)
Incremental Learning (Update)	rule1: <i>if</i> $F3 \in [0.19, 0.78]$ <i>then anomaly</i> rule2: <i>if</i> $F6 \in [0.20, 0.79]$ <i>then anomaly</i> rule3: <i>if</i> $F2 \in [0.59, 0.89]$ <i>then anomaly</i> rule4: <i>if</i> $F5 \in [0.60, 0.89]$ <i>then anomaly</i> rule5: <i>if</i> $F4 \in [0.52, 0.69]$ <i>then anomaly</i> rule6: <i>if</i> $F2 \in [0.23, 0.48]$ <i>and</i> $F6 \in [0.89, 0.97]$ <i>then anomaly</i> rule7: <i>if</i> $F3 \in [0.81, 0.96]$ <i>and</i> $F5 \in [0.84, 0.93]$ <i>then anomaly</i>	1, 2 (old) 1, 2 (old) 2 (old) 2 (old) 3 (new) 3 (new) 3 (new)

more rules compared to the traditional learning. Comparison of these two learning approaches is also presented in Figure 6.6 using g-performance box plots. Moreover, the final rulesets generated from startup and two update phases are demonstrated in Table 6.7. In these rulesets, there is a less issue of seeing extra rules (i.e. a rule that is more general than another) compared to the previous synthetic scenarios. In the presented rulesets in Table 6.8, there is only one example of this case in the traditional learning phase ruleset. In this ruleset, rule3 is able to classify the examples that are classified by rule2. Therefore, by removing rule2 from the final ruleset, the classification rate will not be affected. This can be achieved through an improved post-processing stage, which leads to a less complex model for the classification task. Comparing the two models generated from traditional learning and incremental learning, some similar rules can be found. For example, rule1 from the

incremental learning is quite similar to rule6 from the traditional learning. Another example is the combination of rule4 and rule8 from the incremental learning, which approximately provides the same coverage as rule9 from the traditional learning model.

TABLE 6.7: Average results for startup and update phases against the NSL-KDD dataset.

Phase	Number of rules	g-performance	TNrate	TPrate	Accuracy
Startup	13	91.59	94.33	88.93	91.82
Traditional Learning (update)	14	80.6	95.92	67.73	79.88
Incremental Learning (update)	20	82.04	73.86	91.53	83.83

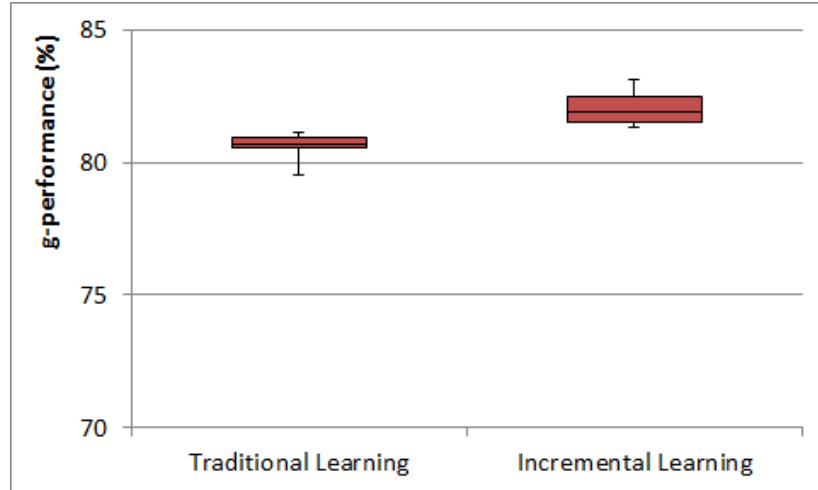


FIGURE 6.6: Performance of different learning approaches against the NSL-KDD dataset.

6.4 Summary

In this chapter, due to the need for an IDS to adapt to changes in the real environment, an efficient model was proposed to incrementally update the database of signatures (rules) in ESR-NID. The proposed framework consists of two phases: startup/classification phase and update phase. During the startup phase, an initial set of collected training samples are fed into the ESR-NID to generate an initial model for the given environment. This initial IDS is able to classify normal and

TABLE 6.8: Comparing the final rulesets generated by ESR-NID during startup and update phases for the NSL-KDD dataset.

Phase	Ruleset
Startup	<p>rule1: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $dst_host_srv_error_rate \in [0.39, 1]$ then anomaly</p> <p>rule2: if $dst_bytes \in [0, 130993]$ and $diff_srv_rate \in [0.05, 0.46]$ then anomaly</p> <p>rule3: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.16, 1]$ then anomaly</p> <p>rule4: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.8, 1]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule5: if $dst_bytes \in [0, 139307]$ $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.12, 1]$ and $diff_srv_rate \in [0.05, 0.92]$ and $dst_host_srv_error_rate \in [0, 0.85]$ then anomaly</p> <p>rule6: if $src_bytes \in [0, 1174619101]$ and $dst_bytes \in [0, 130993]$ and $srv_error_rate \in [0, 0.01]$ and $dst_host_srv_diff_host_rate \in [0.2, 0.86]$ and $dst_host_srv_error_rate \in [1, 0.01]$ then anomaly</p> <p>rule7: if $same_srv_rate \in [0.72, 0.97]$ then anomaly</p> <p>rule8: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.19, 0.36]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule9: if $same_srv_rate \in [0.54, 0.63]$ then anomaly</p> <p>rule10: if $diff_srv_rate \in [0.91, 0.97]$ and $dst_host_srv_diff_host_rate \in [0, 0.22]$ then anomaly</p> <p>rule11: if $diff_srv_rate \in [0.79, 0.82]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule12: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $srv_error_rate \in [0, 0.04]$ and $same_srv_rate \in [0.05, 0.23]$ and $diff_srv_rate \in [0.52, 0.55]$ and $dst_host_srv_diff_host_rate \in [0, 0.54]$ then anomaly</p>

Traditional Learning (Update)	<p>rule1: if $dst_bytes \in [0, 336232536]$ and if $num_root \in [0, 0.74]$ and $diff_srv_rate \in [0.03, 0.47]$ then anomaly</p> <p>rule2: if $num_root \in [0, 0.74]$ and $diff_srv_rate \in [0.03, 0.58]$ then anomaly</p> <p>rule3: if $num_root \in [0, 0.74]$ and $diff_srv_rate \in [0.03, 0.99]$ then anomaly</p> <p>rule4: if $num_root \in [0, 0.74]$ and $same_srv_rate \in [0.01, 0.49]$ then anomaly</p> <p>rule5: if $num_root \in [0, 0.74]$ and $same_srv_rate \in [0.01, 0.92]$ and $dst_host_srv_diff_host_rate \in [0, 0.16]$ then anomaly</p> <p>rule6: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $dst_host_srv_error_rate \in [0.17, 1]$ then anomaly</p> <p>rule7: if $dst_bytes \in [0, 130993]$ and $dst_host_srv_error_rate \in [0.85, 1]$ then anomaly</p> <p>rule8: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $same_srv_rate \in [0.03, 0.94]$ and $dst_host_srv_diff_host_rate \in [0, 0.16]$ then anomaly</p> <p>rule9: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.09, 1]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule10: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $dst_host_srv_diff_host_rate \in [0.22, 0.61]$ then anomaly</p> <p>rule11: if $dst_bytes \in [0, 1107936940]$ and $srv_error_rate \in [0.55, 0.90]$ and $dst_host_srv_diff_host_rate \in [0, 0.03]$ then anomaly</p> <p>rule12: if $dst_bytes \in [0, 1107936940]$ and $srv_error_rate \in [0.33, 0.44]$ then anomaly</p> <p>rule13: if $same_srv_rate \in [0.76, 0.98]$ then anomaly</p> <p>rule14: if $dst_host_srv_error_rate \in [0.69, 0.77]$ then anomaly</p> <p>rule15: if $dst_bytes \in [4259411, 1013436963]$ and $num_root \in [0, 0.74]$ then anomaly</p> <p>rule16: if $src_bytes \in [24210394, 1221033923]$ and $num_root \in [0, 0.74]$ then anomaly</p> <p>rule17: if $dst_bytes \in [0, 879630256]$ and $num_root \in [0, 0.74]$ and $dst_host_srv_diff_host_rate \in [0.90, 0.96]$ then anomaly</p>
-------------------------------	--

Incremental Learning (Update)	<p>rule1: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $dst_host_srv_error_rate \in [0.39, 1]$ then anomaly</p> <p>rule2: if $dst_bytes \in [0, 130993]$ and $diff_srv_rate \in [0.05, 0.46]$ then anomaly</p> <p>rule3: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.16, 1]$ then anomaly</p> <p>rule4: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.8, 1]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule5: if $dst_bytes \in [0, 139307]$ $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.12, 1]$ and $diff_srv_rate \in [0.05, 0.92]$ and $dst_host_srv_error_rate \in [0, 0.85]$ then anomaly</p> <p>rule6: if $src_bytes \in [0, 1174619101]$ and $dst_bytes \in [0, 130993]$ and $srv_error_rate \in [0, 0.01]$ and $dst_host_srv_diff_host_rate \in [0.2, 0.86]$ and $dst_host_srv_error_rate \in [1, 0.01]$ then anomaly</p> <p>rule7: if $same_srv_rate \in [0.72, 0.97]$ then anomaly</p> <p>rule8: if $num_root \in [0, 0.74]$ and $srv_error_rate \in [0.19, 0.36]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule9: if $same_srv_rate \in [0.54, 0.63]$ then anomaly</p> <p>rule10: if $diff_srv_rate \in [0.91, 0.97]$ and $dst_host_srv_diff_host_rate \in [0, 0.22]$ then anomaly</p> <p>rule11: if $diff_srv_rate \in [0.79, 0.82]$ and $dst_host_srv_diff_host_rate \in [0, 0.01]$ then anomaly</p> <p>rule12: if $dst_bytes \in [0, 130993]$ and $num_root \in [0, 0.74]$ and $srv_error_rate \in [0, 0.04]$ and $same_srv_rate \in [0.05, 0.23]$ and $diff_srv_rate \in [0.52, 0.55]$ and $dst_host_srv_diff_host_rate \in [0, 0.54]$ then anomaly</p> <p>rule13: if $dst_bytes \in [0, 178]$ then anomaly</p> <p>rule14: if $dst_bytes \in [0, 1013]$ and $same_srv_rate \in [0, 0.78]$ then anomaly</p> <p>rule15: if $dst_bytes \in [0, 1013]$ and $dst_host_srv_error_rate \in [0.05, 0.99]$ then anomaly</p> <p>rule16: if $dst_bytes \in [0, 29381]$ and $dst_host_srv_error_rate \in [0.05, 0.82]$ then anomaly</p> <p>rule17: if $srv_error_rate \in [0.07, 0.99]$ then anomaly</p> <p>rule18: if $dst_bytes \in [8140, 8318]$ then anomaly</p> <p>rule19: if $num_root \in [0.5, 583]$ then anomaly</p> <p>rule20: if $dst_bytes \in [283505, 284305]$ then anomaly</p> <p>rule21: if $dst_bytes \in [386889, 513338]$ then anomaly</p>
-------------------------------	--

attack records. Therefore, after the startup phase, the produced model will be deployed in its intended environment to operate in the classification mode. However, from time to time, the network administrator might need to update the existing model because of changes in the environment such as modification in the network topology, technology changes and arrival of new types of attacks. For learning new information, in the update phase, ESR-NID is exposed to the new collected data and a new set of rules will be generated and then added to the existing database of rules. The updated database, however, goes through a post-processing stage, which results in removing the similar rules from the database. Finally, the existing model for classification in the startup/classification phase will be replaced by the new processed database.

Using the proposed incremental learning approach, ESR-NID does not need to store the entire training data from the startup day until now. This is the issue of traditional learning, where the system requires huge resources to store the whole training data accumulated thus far and retrain the classifier on the entire data every time an update is needed. For evaluating the proposed model, a series of experiments were carried out when ESR-NID was using the traditional learning and the incremental learning schemes. These experiments included 3 scenarios of synthetic problems and a real intrusion detection problem using NSL-KDD dataset. The results from the experiments against synthetic problems showed that the performance of proposed incremental learning for ESR-NID is slightly worse than the traditional learning and the incremental learning also produced more rules compared to the traditional learning. However, the incremental learning approach benefits from less required storage because it only keeps the generated rules in its database. This is in contrast to the infinitely growing size of repository of raw training data required for traditional learning. Additionally, the results from the experiments against the real intrusion detection problem using NSL-KDD dataset showed that incremental learning is better than traditional learning in terms of g-performance, accuracy, true positive rates and number of rules. This concludes that incremental learning is more effective and efficient for intrusion detection problems with large amount of raw data.

As a future work, some other post-processing methods can be implemented to reduce the number of rules. For example, one rule might be more general than another, which can be eliminated from the final ruleset without affecting the classification rate.

Chapter 7

Conclusions

The detection of network intrusions is a challenging task due to the changes in the real environment over time and availability of advanced tools to attackers. In order to deal with this challenge, effective intrusion detection systems that can adapt to the changes in the environment and variations of attacks are needed.

In this thesis, the selection criteria for intrusion detection systems were reviewed. Examples of these factors are effectiveness, adaptability, ease of implementation and installation place. Amongst all, effectiveness, adaptability and flexibility are the focus of this study. To overcome these challenges in the design and implementation of IDSs, a nature-inspired machine learning approach is proposed in this dissertation. The proposed technique, ESR-NID, is used to acquire knowledge of normal and abnormal behaviour in the form of rules.

ESR-NID uses a genetic algorithm as a base learner to extract signatures for intrusion detection. The database of signatures should be updated over time to adapt to environment changes. For this, ESR-NID utilises an incremental learning approach to incrementally learn the changes in the behaviour of users and the pattern of attacks.

In a series of experiments against different sources of data, the performance of ESR-NID is evaluated and compared with five well-known machine learning techniques.

These techniques are from two categories of GA-based and non-GA-based algorithms. The results show that the performance of ESR-NID is comparable to the other tested methods and produces compact, easily understood rulesets for classification problems.

In the next section, a summary of contributions of the proposed approach to both network intrusion detection and genetic-based machine learning fields are explained.

7.1 Summary of Contributions

The main contributions of this research are outlined as follows:

- **Automatic rule creation for intrusion detection:**

After reviewing the literature, it was found that construction and maintenance of rules for signature based intrusion detection systems is a challenging task. To reduce the cost and time associated with extraction of signatures (rules) from network data, a genetic based rule learning technique was developed in this thesis with some new features, which contributes to the field of GBML.

- **Use of statistical features for detector rules:**

The attack detection process in this thesis is a statistical based approach that provides the ability to detect a wider range of network intrusions. The use of statistical measures overcomes the challenge involved in some of the existing signature based IDSs that are dependent on network packet features such as source IP address and protocol. These systems are unable to detect flooding attacks, which send huge amount of normal packets toward victim to disrupt its services to legitimate users. As the input features to ESR-NID are statistical measures of network traffic, it used a real-valued chromosome representation. Accordingly, only the continuous features of NSL-KDD dataset were used in the intrusion detection process. When a raw network traffic was used for evaluation of the system (i.e. the combined DARPA/CAIDA dataset), an extra pre-processing stage was used to extract suitable statistical features.

- **Development and analysis of a flexible genetic-based rule learning technique:**

An effective GA-based approach was proposed and developed in this thesis with two distinct features. These are an advanced two-stage evaluation approach and an adaptive elitism mechanism. The former makes ESR-NID a flexible model by giving system designers the choice of selecting appropriate fitness and performance functions for different application domains. The two-stage evaluation component in ESR-NID aimed to derive a set of classification rules from the provided dataset, which can cooperatively provide a good coverage of search space. For this, a fitness function was designed for the evaluation of each rule in the system, which takes the cooperation of rules into consideration. Additionally, a performance function was used in this component, which operates on a higher level to evolve the rules cooperatively. This resulted into compact, easily understood rulesets for classification tasks. To show that how variant classifiers can be produced for different problems, first in Chapter 4, ESR-NID was configured using a different performance function and tested against a synthetic problem. Then in Chapter 5, the use of ESR-NID for detecting normal instances was explored and showed how the flexibility aspect of the proposed model helps the designers to put more emphasis on detection of instances from the desired class.

The latter feature, adaptive elitism mechanism, was proposed to address the problem of losing good rules over the generations. This problem was found through the preliminary experiments conducted for evaluation of ESR-NID using a fixed elitism mechanism. The proposed elitism mechanism for ESR-NID adaptively determines the amount of elitism in the selection process. This ensures that cooperating rules are kept together and not lost from one generation to the next. This also means that there is no need to find the best elite value through a trial-and-error method.

- **Incremental learning:**

Finally, to make ESR-NID adaptable to the environment changes, an incremental learning method was introduced. Using this approach, ESR-NID does

not need to store the original data used for training the classifier. Instead it preserves the previously acquired knowledge (i.e. ruleset) and learns additional information from new data and updates the database of signatures with the new information. This addresses the storing of the entire training data and the increased training time challenges in the traditional learning approaches. These techniques discard the existing classifier and retrain the system using the entire accumulated training data. The proposed incremental learning for ESR-NID consisted of startup/classification and update phases. During the startup phase, a classification model is designed using the initial set of training examples. This phase will then be called classification phase throughout the system life cycle. From time to time, ESR-NID updates the database of rules during the update phase. This is achieved by retraining the system on the training data and adding the generated rules to the current database. Before the new model can be used for classification, the combined rules go through a post-processing stage to produce a more concise ruleset for the use of IDS. This is because in most domains, experts are interested in simple human-understandable models.

7.2 Limitations

There are a number of limitations in the work presented in this thesis, which are listed below.

- The system is aimed at learning statistical signatures of network traffic to detect a wider range of attacks. Therefore, only real-valued attributes can be used as the input to ESR-NID.
- The system evaluation on intrusion detection was only limited to the publicly available datasets that have been used for IDSs evaluations in the literature. If there were no constraints on the availability of large scale traffic collection

infrastructure and privacy concerns, additional real network traffic could be collected for testing the system.

- The final evaluations of the incremental learning approach for ESR-NID is limited to only six runs per experiment (2 different initial seeds and 3 folds). Therefore, the performance measure reported is the average across six runs. If there were no time constraints, more folds and seed values could be used to improve performance estimation.

7.3 Future Work

Several directions can be taken to extend this work. The potential areas for future research are summarized below.

- Improving ESR-NID: with some evolved rulesets, the fitness and performance measurements that drive the algorithm are insensitive to small modifications to boundary values (because there are no training examples nearby). The next step is to investigate efficient local generalisation and specialisation operators that could exploit this fact. For example, they could be used in a memetic version of the algorithm, combining evolution with local modification.
- Classification of attack types: in this thesis, ESR-NID is used as a binary classifier to detect network intrusions. However, identifying the type of attacks is another challenge and hence ESR-NID can be extended to discover rulesets for multi-class classification problems.
- Improving the incremental learning component: in this thesis, an incremental learning approach is utilised for ESR-NID to re-train the system once a new batch of data becomes available. Using this, ESR-NID generates ruleset for the new data, and then this ruleset will be added to the existing database of rules. Next, the similar rules will be removed through a post-processing stage and finally the post-processed ruleset will be used for future classification

tasks. One step for improving this approach is to extend the post-processing stage by keeping the general rules and eliminating the more specific ones in the ruleset without affecting the classification rate. This decreases the size of ruleset and minimize the complexity of the classification model. The similarity between rules can be more accurately determined using the Jaccard index (or Jaccard similarity coefficient) ([Real & Vargas, 1996](#)), which is a statistic used for comparing the similarity and diversity of sample sets. Jaccard index as a similarity metric describes the degree of overlap between the rules and is defined as the ratio of the intersection to the union of the pairwise compared variables ([Pang-Ning et al., 2005](#)).

- Applying ESR-NID to other areas: the focus of this research was on intrusion detection problem, however, this can be extended for other classification problems. It would be nice to evaluate the developed system with datasets from other areas.

Bibliography

- Abadeh, M. S., Mohamadi, H., & Habibi, J. (2011). Design and analysis of genetic fuzzy systems for intrusion detection in computer networks. *Expert Systems with Applications*, 38(6), 7067–7075.
- Abbott, R. P., Chin, J. S., Donnelley, J. E., Konigsford, W. L., Tokubo, S., & Webb, D. A. (1976). Security analysis and enhancements of computer operating systems. Technical report, Defense Technical Information Center (DTIC) Document. Retrieved from <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA436876>.
- Aggarwal, C. C. & Zhai, C. (2012). *Mining text data*. Springer.
- Aguilar-Ruiz, J. S., Riquelme, J. C., & Toro, M. (2003). Evolutionary learning of hierarchical decision rules. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(2), 324–331.
- Aickelin, U., Dasgupta, D., & Gu, F. (2014). Artificial immune systems. In *Search Methodologies* (pp. 187–211). Springer.
- Akbar, M. A. & Farooq, M. (2009). Application of evolutionary algorithms in detection of SIP based flooding attacks. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, (pp. 1419–1426). ACM.
- Alcala-Fdez, J., Sanchez, L., Garcia, S., del Jesus, M. J., Ventura, S., Garrell, J., Otero, J., Romero, C., Bacardit, J., Rivas, V. M., et al. (2009). KEEL: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3), 307–318.

- Allen, J., Christie, A., Fithen, W., McHugh, J., & Pickel, J. (2000). State of the practice of intrusion detection technologies. Technical report, Defense Technical Information Center (DTIC) Document. Retrieved from <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA375846>.
- Alpaydin, E. (2004). *Introduction to machine learning*. MIT press.
- Anyanwu, M. N. & Shiva, S. G. (2009). Comparative analysis of serial decision tree classification algorithms. *International Journal of Computer Science and Security*, 3(3), 230–240.
- Bacardit, J. (2004). *Pittsburgh genetics-based machine learning in the data mining era: representations, generalization, and run-time*. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain. Retrieved from <http://sci2s.ugr.es/keel/pdf/keel/tesis/bacardit.pdf>.
- Bacardit, J., Burke, E. K., & Krasnogor, N. (2009). Improving the scalability of rule-based evolutionary learning. *Memetic Computing*, 1(1), 55–67.
- Bacardit, J. & Garrell, J. M. (2003). Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system. In *Genetic and Evolutionary Computation—GECCO 2003*, (pp. 1818–1831). Springer.
- Bacardit, J. & Garrell, J. M. (2004). Analysis and improvements of the adaptive discretization intervals knowledge representation. In *Genetic and Evolutionary Computation—GECCO 2004*, (pp. 726–738). Springer.
- Bacardit, J. & Krasnogor, N. (2009a). A mixed discrete-continuous attribute list representation for large scale classification domains. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, (pp. 1155–1162). ACM.
- Bacardit, J. & Krasnogor, N. (2009b). Performance and efficiency of memetic pittsburgh learning classifier systems. *Evolutionary computation*, 17(3), 307–342.

- Bai, Y. & Kobayashi, H. (2003). Intrusion detection systems: technology and development. In *Advanced Information Networking and Applications, 2003. AINA 2003. 17th International Conference on*, (pp. 710–715). IEEE.
- Baig, Z. A., Khan, S., Ahmed, S., & Sqalli, M. H. (2011). A selective parameter-based evolutionary technique for network intrusion detection. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, (pp. 65–71). IEEE.
- Baqer, M. & Khan, A. (2007). Energy-efficient pattern recognition approach for wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, (pp. 509–514). IEEE.
- Barandela, R., Sánchez, J. S., Garcia, V., & Rangel, E. (2003). Strategies for learning in class imbalance problems. *Pattern Recognition*, 36(3), 849–851.
- Bernadó-Mansilla, E. & Garrell-Guiu, J. M. (2003). Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary computation*, 11(3), 209–238.
- Bhatia, S., Mohay, G., Tickle, A., & Ahmed, E. (2011). Parametric differences between a real-world distributed denial-of-service attack and a flash event. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, (pp. 210–217). IEEE.
- Bhuyan, M. H., Bhattacharyya, D., & Kalita, J. (2014). Information metrics for low-rate DDoS attack detection: A comparative evaluation. In *Contemporary Computing (IC3), 2014 Seventh International Conference on*, (pp. 80–84). IEEE.
- Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2015). Towards Generating Real-life Datasets for Network Intrusion Detection. *International Journal of Network Security*, 17(6), 675–693.
- Bishop, M. (2003). *Computer security: art and science*. Addison-Wesley, Boston, Mass.

- Bojarczuk, C. C., Lopes, H. S., Freitas, A. A., & Michalkiewicz, E. L. (2004). A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. *Artificial Intelligence in Medicine*, 30(1), 27–48.
- Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.
- Britt, W., Gopalaswamy, S., Hamilton, J. A., Dozier, G. V., & Chang, K. H. (2007). Computer defense using artificial intelligence. In *Proceedings of the 2007 spring simulation multiconference-Volume 3*, (pp. 378–386). Society for Computer Simulation International.
- CAIDA (2007). CAIDA dataset. Accessed July 2015. Retrieved from http://www.caida.org/data/passive/ddos-20070804_dataset.xml.
- Casillas, J., Carse, B., & Bull, L. (2007). Fuzzy-XCS: A Michigan genetic fuzzy system. *Fuzzy Systems, IEEE Transactions on*, 15(4), 536–550.
- Chawla, N. V., Lazarevic, A., Hall, L. O., & Bowyer, K. W. (2003). SMOTEBoost: Improving prediction of the minority class in boosting. In *Knowledge Discovery in Databases: PKDD 2003* (pp. 107–119). Springer.
- Chebrolu, S., Abraham, A., & Thomas, J. P. (2005). Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, 24(4), 295–307.
- Chen, L.-C., Longstaff, T. A., & Carley, K. M. (2004). Characterization of defense mechanisms against distributed denial of service attacks. *Computers & Security*, 23(8), 665–678.
- Cohen, W. W. (1995). Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, (pp. 115–123).
- Cole, E. (2011). *Network security bible*, volume 768. John Wiley & Sons.
- CVE (2015). Common Vulnerabilities and Exposures. The MITRE Corporation. Available at <http://cve.mitre.org/>.

- Dahal, K. P. & McDonald, J. (1998). Generational and Steady-State Genetic Algorithms for Generator Maintenance Scheduling Problems. In *Artificial Neural Nets and Genetic Algorithms*, (pp. 259–263). Springer.
- Dasgupta, D. (1999). Immunity-based intrusion detection system: a general framework. In *Proc. of the 22nd NISSC*, volume 1, (pp. 147–160).
- Dasgupta, D. & González, F. (2002). An immunity-based technique to characterize intrusions in computer networks. *Evolutionary Computation, IEEE Transactions on*, 6(3), 281–291.
- De Castro, L. N. & Timmis, J. (2002). *Artificial immune systems: a new computational intelligence approach*. Springer.
- De Jong, K. A., Spears, W. M., & Gordon, D. F. (1994). *Using genetic algorithms for concept learning*. Springer.
- Depren, O., Topallar, M., Anarim, E., & Ciliz, M. K. (2005). An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert systems with Applications*, 29(4), 713–722.
- Domeniconi, C., Peng, J., & Gunopulos, D. (2002). Locally adaptive metric nearest-neighbor classification. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(9), 1281–1285.
- Dumitrescu, D., Lazzerini, B., Jain, L. C., & Dumitrescu, A. (2000). *Evolutionary computation*. CRC press.
- Durst, R., Champion, T., Witten, B., Miller, E., & Spagnuolo, L. (1999). Testing and Evaluating. *Communications of the ACM*, 42(7), 53.
- Eiben, A. E. & Smith, J. E. (2003). *Introduction to evolutionary computing*. springer.
- Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security* (pp. 77–101). Springer.

- Evans, D. (2011). The internet of things. *How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG)*.
- Feinstein, L., Schnackenberg, D., Balupari, R., & Kindred, D. (2003). Statistical approaches to DDoS attack detection and response. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, (pp. 303–314). IEEE.
- Fernández, A., García, S., Luengo, J., Bernadó-Mansilla, E., & Herrera, F. (2010). Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study. *Evolutionary Computation, IEEE Transactions on*, 14(6), 913–941.
- Forrest, S. & Hofmeyr, S. (1999). Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan-Kaufmann, San Francisco, CA*, (pp. 1289–1296).
- Frank, E. & Witten, I. H. (1998). Generating Accurate Rule Sets Without Global Optimization. In *Proc. Of The 15th INT. Conference On Machine Learning*.
- Freitas, A. A. (2002). *Data mining and knowledge discovery with evolutionary algorithms*. Springer.
- Freitas, A. A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing* (pp. 819–845). Springer.
- Freitas, A. A. (2008). A review of evolutionary algorithms for data mining. In *Soft Computing for Knowledge Discovery and Data Mining* (pp. 79–111). Springer.
- Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1), 3–54.
- Fürnkranz, J., Gamberger, D., et al. (2012). *Foundations of rule learning*. Springer Science & Business Media.

- Gaonjur, P., Tarapore, N., Pukale, S., & Dhore, M. (2008). Using Neuro-Fuzzy Techniques to reduce false alerts in IDS. In *Networks, 2008. ICON 2008. 16th IEEE International Conference on*, (pp. 1–6). IEEE.
- García, S., Fernández, A., Luengo, J., & Herrera, F. (2009). A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing*, 13(10), 959–977.
- García, S. & Herrera, F. (2009). Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary Computation*, 17(3), 275–306.
- Glenn, M. (2003). A summary of DoS/DDoS prevention, monitoring and mitigation techniques in a service provider environment. *SANS Institute*, Aug, 21, 34.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Goldberg, D. E. & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1, 69–93.
- Gómez, J., Gil, C., Baños, R., Márquez, A. L., Montoya, F. G., & Montoya, M. (2013). A Pareto-based multi-objective evolutionary algorithm for automatic rule generation in network intrusion detection systems. *Soft Computing*, 17(2), 255–263.
- Gong, R. H., Zulkernine, M., & Abolmaesumi, P. (2005). A software implementation of a genetic algorithm based approach to network intrusion detection. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on*, (pp. 246–253). IEEE.
- Gonzalez, F., Dasgupta, D., & Kozma, R. (2002). Combining negative selection and classification techniques for anomaly detection. In *Evolutionary Computation*,

2002. *CEC'02. Proceedings of the 2002 Congress on*, volume 1, (pp. 705–710). IEEE.
- Haag, C. R., Lamont, G. B., Williams, P. D., & Peterson, G. L. (2007). *An artificial immune system-inspired multiobjective evolutionary algorithm with application to the detection of distributed computer network intrusions*. Springer.
- Haines, J., Ryder, D. K., Tinnel, L., & Taylor, S. (2003). Validation of sensor alert correlators. *IEEE Security & Privacy*, 1(1), 46–56.
- Haines, J. W., Lippmann, R. P., Fried, D. J., Zissman, M., & Tran, E. (2001). 1999 DARPA intrusion detection evaluation: Design and procedures. Technical report, DTIC Document.
- Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato.
- Hamdi, M. & Boudriga, N. (2005). Computer and network security risk management: Theory, challenges, and countermeasures. *International journal of communication systems*, 18(8), 763–793.
- Hansman, S. & Hunt, R. (2005). A taxonomy of network and computer attacks. *Computers & Security*, 24(1), 31–43.
- Heady, R., Luger, G. F., Maccabe, A., & Servilla, M. (1990). *The architecture of a network level intrusion detection system*. Department of Computer Science, College of Engineering, University of New Mexico.
- Helmer, G., Wong, J. S., Honavar, V., & Miller, L. (2002). Automated discovery of concise predictive rules for intrusion detection. *Journal of Systems and Software*, 60(3), 165–175.
- Hinneburg, A., Aggarwal, C. C., & Keim, D. A. (2000). What is the nearest neighbor in high dimensional spaces?
- Hofmeyr, S. A. & Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary computation*, 8(4), 443–473.

- Holland, J. H. (1975). *Adaptation in natural and artificial system: an introduction with application to biology, control and artificial intelligence*. Ann Arbor, University of Michigan Press.
- Holland, J. H., Booker, L. B., Colombetti, M., Dorigo, M., Goldberg, D. E., Forrest, S., Riolo, R. L., Smith, R. E., Lanzi, P. L., Stolzmann, W., et al. (2000). What is a learning classifier system? In *Learning Classifier Systems* (pp. 3–32). Springer.
- Hou, H. (2006). *Genertia: a system for vulnerability analysis, design and redesign of immunity-based anomaly detection system*. PhD thesis, Computer Science and Software Engineering, Auburn University.
- Hou, H. & Dozier, G. (2005). Immunity-based intrusion detection system design, vulnerability analysis, and GENERTIA's genetic arms race. In *Proceedings of the 2005 ACM symposium on Applied computing*, (pp. 952–956). ACM.
- Howard, J. D. & Longstaff, T. A. (1998). A common language for computer security incidents. *Sandia National Laboratories*.
- Jacobson, V., Leres, C., & McCanne, S. (1989). The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*.
- Jadhav, M. L. & Gaikwad, C. (2014). Intrusion Detection System Using GA. *IOSR Journal of Computer Engineering (IOSR-JCE)*, 16(2), 82–84.
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 237–285.
- Khor, K.-C., Ting, C.-Y., & Amnuaisuk, S.-P. (2010). Forming an optimal feature set for classifying network intrusions involving multiple feature selection methods. In *Information Retrieval & Knowledge Management, (CAMP), 2010 International Conference on*, (pp. 179–183). IEEE.
- Kim, J. & Bentley, P. (1999). Negative selection and niching by an artificial immune system for network intrusion detection. In *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference*, (pp. 149–158).

- Koblar, V. (2012). OPTIMIZING PARAMETERS OF MACHINE LEARNING ALGORITHMS. Technical report.
- Kojm, T. (2004). ClamAV. URL <http://www.clamav.net>.
- Kotsiantis, S. (2007). Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, 31, 249–268.
- Kotsiantis, S., Kanellopoulos, D., Pintelas, P., et al. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, 30(1), 25–36.
- Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: a review of classification and combining techniques. *Artificial Intelligence Review*, 26(3), 159–190.
- Kubat, M. & Matwin, S. (1997). Addressing the Curse of Imbalanced Training Sets: One-Sided Selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*.
- Kubi, J. (2002). Immunology, Fifth Edition by Richard A. Goldsby, Thomas J. Kindt, Barbara A. Osborne. New York.
- Kuwatly, I., Sraj, M., Al Masri, Z., & Artail, H. (2004). A dynamic honeypot design for intrusion detection. In *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on*, (pp. 95–104). IEEE.
- Lavesson, N. & Davidsson, P. (2006). Quantifying the impact of learning algorithm parameter tuning. In *Proceedings of the 21st national conference on Artificial intelligence-Volume 1*, (pp. 395–400). AAAI Press.
- Lee, S. & Shields, C. (2002). Challenges to automated attack traceback. *IT professional*, 4(3), 12–18.
- Lee, W. (1999). *A data mining framework for constructing features and models for intrusion detection systems*. PhD thesis, Columbia University. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.714&rep=rep1&type=pdf>.

- Lee, W. & Stolfo, S. J. (1998). Data mining approaches for intrusion detection. In *Usenix Security*.
- Li, W. (2004). Using genetic algorithm for network intrusion detection. *Proceedings of the United States Department of Energy Cyber Security Group*, 1–8.
- Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1), 16–24.
- Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation. *Computer networks*, 34(4), 579–595.
- Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E., Wyschogrod, D., Cunningham, R. K., et al. (2000). Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*, volume 2, (pp. 12–26). IEEE.
- Lough, D. L. (2001). *A Taxonomy of Computer Attacks with Applications to Wireless Networks*. PhD thesis, VirginiaTech. <https://vtechworks.lib.vt.edu/handle/10919/27242>.
- Lu, W. & Traore, I. (2004). Detecting new forms of network intrusion using genetic programming. *Computational Intelligence*, 20(3), 475–494.
- Luke, S., Panait, L., Balan, G., Paus, S., Skolicki, Z., Bassett, J., Hubley, R., & Chircop, A. (2006). Ecj: A java-based evolutionary computation research system. *Downloadable versions and documentation can be found at the following url: <http://cs.gmu.edu/ecjlab/projects/ecj>*.
- MeeraGandhi, G., Appavoo, K., & Srivasta, S. (2010). Effective network intrusion detection using classifiers decision trees and decision rules. *Int. J. Advanced network and application, Vol2*.

- Michalewicz, Z. (1996). *Genetic algorithms+ data structures= evolution programs*. springer.
- Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.
- MIT Lincoln Laboratory (2000). DARPA intrusion detection scenario specific datasets. Retrieved from <http://www.ll.mit.edu/ideval/data/2000data.html>.
- MIT Lincoln Laboratory (2002). *DARPA Intrusion Detection Evaluation Data Sets*. <http://www.ll.mit.edu/ideval/data/>.
- Mölsä, J. (2005). Mitigating denial of service attacks: a tutorial. *Journal of computer security*, 13(6), 807–837.
- Moore, D., Shannon, C., Brown, D. J., Voelker, G. M., & Savage, S. (2006). Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2), 115–139.
- Mukherjee, S. & Sharma, N. (2012). Intrusion detection using naive Bayes classifier with feature reduction. *Procedia Technology*, 4, 119–128.
- Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 2, (pp. 1702–1707). IEEE.
- Nasr, A. A., Ezz, M. M., & Abdulmageed, M. Z. (2014). Use of Decision Trees and Attributional Rules in Incremental Learning of an Intrusion Detection Model. *International Journal of Computer Networks & Communications Security*, 2(7).
- Northcutt, S. & Novak, J. (2002). *Network intrusion detection*. Sams Publishing.
- Orriols-Puig, A. & Bernadó-Mansilla, E. (2006). Bounding XCS's parameters for unbalanced datasets. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, (pp. 1561–1568). ACM.

- Orriols-Puig, A., Casillas, J., & Bernadó-Mansilla, E. (2009). Fuzzy-UCS: a Michigan-style learning fuzzy-classifier system for supervised learning. *Evolutionary Computation, IEEE Transactions on*, 13(2), 260–283.
- Pal, B. & Hasan, M. A. M. (2012). Neural network & genetic algorithm based approach to network intrusion detection & comparative analysis of performance. In *Computer and Information Technology (ICCIT), 2012 15th International Conference on*, (pp. 150–154). IEEE.
- Pang-Ning, T., Steinbach, M., Kumar, V., et al. (2005). *Introduction to data mining* (1st ed.). Boston : Pearson Addison Wesley.
- Pawar, S. & Bichkar, R. (2014). Selecting GA Parameters for Intrusion Detection. *International Journal of Applied Information Systems (IJ AIS)*, 6(7), 15–20.
- Peng, T., Leckie, C., & Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys (CSUR)*, 39(1), 3.
- Pietraszek, T. (2004). Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection*, (pp. 102–124). Springer.
- Polikar, R., Upda, L., Upda, S. S., & Honavar, V. (2001). Learn++: An incremental learning algorithm for supervised neural networks. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 31(4), 497–508.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1), 81–106.
- Quinlan, J. R. (1993). *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann.
- Real, R. & Vargas, J. M. (1996). The probabilistic basis of Jaccard’s index of similarity. *Systematic biology*, 380–385.

- Reed, P. M., Minsker, B. S., & Goldberg, D. E. (2001). The practitioner's role in competent search and optimization using genetic algorithms. *Bridges*, 10(40569), 97.
- Roesch, M. (1999). Snort: Lightweight Intrusion Detection for Networks. In *LISA*, volume 99, (pp. 229–238).
- Rogers, A. & Prügel-Bennett, A. (1999). Genetic drift in genetic algorithm selection schemes. *Evolutionary Computation, IEEE Transactions on*, 3(4), 298–303.
- Rokach, L. (2008). *Data mining with decision trees: theory and applications*, volume 69. World scientific.
- Santos, O. (2007). *End-to-end Network Security: Defense-in-depth*. Pearson Education.
- Seliya, N., Khoshgoftaar, T. M., & Van Hulse, J. (2009). A study on the relationships of classifier performance metrics. In *Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on*, (pp. 59–66). IEEE.
- Shafi, K. (2008). *An online and adaptive signature-based approach for intrusion detection using learning classifier systems*. PhD thesis, Australian Defence Force Academy.
- Shafi, K., Abbass, H., & Zhu, W. (2006). An adaptive rule-based intrusion detection architecture. In *Proceedings of the 2006 RNSA security technology conference. Canberra, Australia*, (pp. 307–319).
- Shafi, K. & Abbass, H. A. (2009). An adaptive genetic-based signature learning system for intrusion detection. *Expert Systems with Applications*, 36(10), 12036–12043.
- Shafi, K., Kovacs, T., Abbass, H. A., & Zhu, W. (2009). Intrusion detection with evolutionary learning classifier systems. *Natural Computing*, 8(1), 3–27.
- Shafiq, M. Z., Tabish, S. M., & Farooq, M. (2009). On the appropriateness of evolutionary rule learning algorithms for malware detection. In *Proceedings of the*

- 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, (pp. 2609–2616). ACM.
- Shannon, C. E. (1948). A note on the concept of entropy. *Bell System Tech. J.*, 27, 379–423.
- Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In *AI 2006: Advances in Artificial Intelligence* (pp. 1015–1021). Springer.
- Sommer, R. & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, (pp. 305–316). IEEE.
- Sönströd, C., Johansson, U., & Löfström, T. (2009). Evaluating Algorithms for Concept Description.
- Spafford, E. H. & Zamboni, D. (2000). Intrusion detection using autonomous agents. *Computer networks*, 34(4), 547–570.
- Sqalli, M. H., Firdous, S. N., Baig, Z., & Azzedin, F. (2011). An Entropy and Volume-Based Approach for Identifying Malicious Activities in Honeynet Traffic. In *Cyberworlds (CW), 2011 International Conference on*, (pp. 23–30). IEEE.
- Stewart, I. (2009). A Modified Genetic Algorithm and Switch-Based Neural Network Model Applied to Misuse-Based Intrusion Detection.
- Stolfo, J., Fan, W., Lee, W., Prodromidis, A., & Chan, P. K. (2000). Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection. *Results from the JAM Project by Salvatore*.
- Surry, P. D. & Radcliffe, N. J. (1996). Inoculation to initialise evolutionary search. In *Evolutionary Computing* (pp. 269–285). Springer.
- Syswerda, G. (1991). A study of reproduction in generational and steady state genetic algorithms. *Foundations of genetic algorithms*, 2, 94–101.

- Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A.-A. (2009). 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*.
- Thomas, R. (2001). Managing the Threat of Denial-of-Service Attacks. *CERT Coordination Center*, 10.
- Urbanowicz, R. J. & Moore, J. H. (2009). Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009, 1.
- Urbanowicz, R. J. & Moore, J. H. (2010). The application of michigan-style learning classifiersystems to address genetic heterogeneity and epistasisin association studies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, (pp. 195–202). ACM.
- Vavak, F. & Fogarty, T. C. (1996). Comparison of steady state and generational genetic algorithms for use in nonstationary environments. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, (pp. 192–195). IEEE.
- Vollmer, T., Alves-Foss, J., & Manic, M. (2011). Autonomous rule creation for intrusion detection. In *Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium on*, (pp. 1–8). IEEE.
- Wang, K. & Stolfo, S. J. (2004). Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection*, (pp. 203–222). Springer.
- Weng, C. G. & Poon, J. (2008). A new evaluation measure for imbalanced datasets. In *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*, (pp. 27–32). Australian Computer Society, Inc.
- Wilson, D. R. & Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3), 257–286.
- Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary computation*, 3(2), 149–175.

- Wilson, S. W. (2000). Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems* (pp. 209–219). Springer.
- Wilson, S. W. (2001). Mining oblique data with XCS. In *Advances in Learning Classifier Systems* (pp. 158–174). Springer.
- Witten, I. H. & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.
- Wu, Q., Shiva, S., Roy, S., Ellis, C., & Datla, V. (2010). On modeling and simulation of game theory-based defense mechanisms against dos and ddos attacks. In *Proceedings of the 2010 spring simulation multiconference*, (pp. 159). Society for Computer Simulation International.
- Wu, S. X. & Banzhaf, W. (2010). The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1), 1–35.
- Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Philip, S. Y., et al. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1), 1–37.
- Xiang, Y., Li, K., & Zhou, W. (2011). Low-rate DDoS attacks detection and trace-back by using new information metrics. *Information Forensics and Security, IEEE Transactions on*, 6(2), 426–437.
- Yang, H., Luo, H., Ye, F., Lu, S., & Zhang, L. (2004). Security in mobile ad hoc networks: challenges and solutions. *Wireless Communications, IEEE*, 11(1), 38–47.
- Yang, J., Tiyyagura, A., Chen, F., & Honavar, V. (2001). Feature subset selection for rule induction using RIPPER. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, (pp. 1800).
- Yang, Q. & Wu, X. (2006). 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(04), 597–604.

- Zhang, J., Qin, Z., Ou, L., Jiang, P., Liu, J., & Liu, A. (2010). An advanced entropy-based DDOS detection scheme. In *Information Networking and Automation (ICINA), 2010 International Conference on*, volume 2, (pp. V2–67). IEEE.
- Zhao, Y. & Zhang, Y. (2008). Comparison of decision tree methods for finding active objects. *Advances in Space Research*, 41(12), 1955–1959.
- Zi, L., Yearwood, J., & Wu, X.-W. (2010). Adaptive clustering with feature ranking for ddos attacks detection. In *Network and System Security (NSS), 2010 4th International Conference on*, (pp. 281–286). IEEE.
- Zuo, J., Tang, C., & Zhang, T. (2002). Mining predicate association rule by gene expression programming. In *Advances in web-age information management* (pp. 92–103). Springer.