Setareh Roshan

An Online Anomaly-Detection Neural Networks-based Clustering for Adaptive Intrusion Detection Systems

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 11.12.2015

Thesis supervisor:

Prof. N. Asokan

Thesis advisor:

D.Sc. (Tech.) Yoan Miche

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Setareh Roshan

Title: An Online Anomaly-Detection Neural Networks-based Clustering for Adaptive Intrusion Detection Systems

Date: 11.12.2015 Language: English Number of pages: 8+67

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor: Prof. N. Asokan

Advisor: D.Sc. (Tech.) Yoan Miche

In the evolving nature of today's world of network security, threats have become more and more sophisticated. Although different security solutions such as firewalls and antivirus software have been deployed to protect systems, external attackers are still capable of intruding into computer networks. This is where intrusion detection systems come into play as an additional security layer.

Despite the large volume of research conducted in the field of intrusion detection, finding a perfect solution of intrusion detection systems for critical applications is still a major challenge. This is mainly due to the continuous emergence of security threats which can bypass the outdated intrusion detection systems.

The main objective of this thesis is to propose an adaptive design of intrusion detection systems which offers the capability of detecting known and novel attacks and being updated according to new trends of data patterns provided by security experts in a cost-effective manner. The proposed intrusion detection system uses an anomaly-based technique and is constructed on the basis of Extreme Learning Machine method which is a variant of neural networks. In this work, two novel approaches are also proposed to enhance the speed of partial updates for the learning model according to new information fed into the system. The performance of the proposed intrusion detection system is evaluated as a network-based solution using NSL-KDD data set. The evaluation results indicate that the system provides an average detection rate of 81 % while having a false positive rate of 3 % in detecting known and novel attacks. In addition, the obtained results show that the system is capable of adapting to the new input information and data injected into the system by a human security expert.

Keywords: Intrusion Detection System, Anomaly Detection, Clustering, ELM, Neural Networks

Acknowledgement

Firstly, I wish to thank my employer 'Nokia Networks' for providing the opportunity to work on this interesting thesis topic.

I would like to express my sincere gratitude to my supervisor Professor N. Asokan for supervising my thesis and providing me with his valuable advice and encouraging comments throughout this work.

I am very grateful to my instructor Yoan Miche for his patience, never-ending support and guidance during this thesis work. Every day of working with him has been a great learning opportunity for me. It is certainly an honor to have him as an instructor.

I want to thank my colleagues for making the office a very pleasant place to work in. My special thanks go to my boss Christine Bejerasco for reviewing this work and for her encouragement and support during the time I wrote this document.

I would like to thank all my friends who have been cheering me up with their love and enthusiasm during the difficulties while doing this thesis. I have been extremely fortunate to have such nice buddies around.

Finally, this thesis would not have been the same without the unconditional love and support from my family which have been my inspiration to strive for the very best things in life.

Setareh Roshan,

Espoo, December 2015.

Contents

List of Tables

List of Figures

Abbreviations and Acronyms

BP	Backpropagation
CLUS-ELM	Clustering Extreme Learning Machine
CM	Clustering Manager
DARPA	Defense Advanced Research Projects Agency
DoS	Denial of Service
EE	Evaluation Engine
ELM	Extreme Learning Machine
FD	Fault Detector
IDS	Intrusion Detection System
IP	Internet Protocol
LAN	Local Area Network
LVM	Levenberg-Marquardt
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
OS-ELM	Online Sequential Extreme Learning Machine
RAN	Resource Allocation Network
R2L	Remote to Local
SGBP	Stochastic Gradient Descent Backpropagation
SLFN	Single hidden Layer Feedforward Neural Network
TCP	Transmission Control Protocol
UM	Update Manager
U2R	User to Root

Chapter 1

Introduction

1.1 Background

In today's world, internet and computer networks are widely used in different businesses. Business requirements have made corporations deploy their own information systems on internet. These information systems may utilize different technologies such as distributed data storage systems, encryption and authentication techniques, remote access and web services [51]. While the usage of these technologies provides lots of opportunities for enterprises to achieve their commercial goals, it also increases the chance of intruders to launch attacks against the information systems. With unknown vulnerabilities in software design and network protocols, cyber-crimes have grown significantly during the past decade.

As information is a crucial asset of every organization [35], any security breach in information systems which can put confidentiality, integrity and availability of the information at risk may cause severe loss for the organization. According to [4], the annual cost from cyber-crimes to the global economy is estimated to be more than $400 billion in 2014. This statistic stresses the importance of developing proper solutions to ensure the safety of information systems.

In order to protect information systems against intruders, different preventive solutions have been deployed by organizations. These solutions include user authentication, access control mechanisms and firewalls [29]. Although the mentioned solutions secure information systems against unauthorized access by outsiders [27], they can't provide the necessary protection against internal attackers. As a result, intruders can launch attacks from a compromised user account without being identified.

This is where intrusion detection systems come into play as the second line of defense. Intrusion detection is the task of monitoring a system or network in order to detect malicious activities which try to compromise the confidentiality, integrity and availability of the information on that system or network. An intrusion detection system (IDS) which is deployed on a system or network generates alarms upon detection of malicious activities. These systems are capable of detecting intrusive

activities launched by both insiders and outsiders in a system or network. Intrusion detection systems can be of two types of anomaly-based and signature-based. Anomaly-based IDSs try to find the abnormal patterns of activities in a system by comparing the activities with the normal profile of the system. On the other hand, signature-based IDSs compare the patterns of the activities in the monitored system to the predefined patterns of existing intrusions and trigger an alarm if a match is found.

Despite the fact that intrusion detection has been a vast area of research for many years [62, 56, 14, 38], there is no perfect example of intrusion detection systems which can be used in critical applications. This is mainly due to the evolving nature of today's world of cyber-security where novel attacks are emerging continuously while security solutions remain the same. In order to address this issue, an effective design of intrusion detection system must have the capability of detecting new and unknown attacks as well as being updated according to new trends of data patterns provided by security experts in a cost-effective manner.

1.2 Scope

The goal of this thesis is to design a new intrusion detection system which addresses the problem of adaptability of existing IDSs. The proposed solution is capable of detecting novel attacks while providing acceptable rates of detection and false alarms. In addition, the proposed methodology uses a fast procedure for updating the IDS according to the new patterns of data coming from both existing and new attacks. The proposed IDS has the capability of receiving the input from a human security expert and being updated according to that input with a low computational cost. The update cases may consist in modifying the existing data, adding new unlabeled or labeled data, and adding new classes of data into the system. Moreover, the proposed IDS is capable of detecting human errors in categorizing the data and providing the appropriate correction proposals.

1.3 The Structure of Thesis

The rest of this thesis is organized as follows.

Chapter 2 provides the necessary background in the field of intrusion detection.

Chapter 3 describes the problem tackled in this thesis as well the criteria used to evaluate the proposed solution. In addition, the test data set which is used for evaluation of the proposed intrusion detection system is described.

Chapter 4 presents the mathematical methods and algorithms which are applied in the proposed solution.

Chapter 5 introduces the proposed intrusion detection system. Different components of the system and the novel approaches of this work are discussed in detail in this chapter.

Chapter 6 provides an evaluation of the proposed approaches according to the defined criteria. In addition, the effectiveness of our proposed solution will be discussed.

Finally Chapter 7 concludes the thesis and its key outcomes and discusses the possible improvements to the proposed solution in future work.

Chapter 2

Intrusion Detection Systems

This chapter provides the necessary background in the field of intrusion detection. In the following sections, the objective of intrusion detection is described briefly. In addition, different types of intrusion detection systems are discussed in terms of their source of information as well as their detection technique. Finally, anomaly detection systems are elaborated a bit as a subcategory of intrusion detection systems.

2.1 Definition and Purpose

In the field of network security, intrusion is defined as a set of malicious activities against the integrity, confidentiality and availability of information on a system or network that make it vulnerable against future attacks [22]. In order to counter such threats, intrusion detection systems are deployed to detect and identify the attempted intrusions into a system or network. Using a set of hardware and software resources, IDS tries to detect intrusions by monitoring the data collected from a single host or network and generate alarm in case of detecting attempted intrusions.

2.2 IDS Classification

Intrusion detection systems can be divided into different categories according to their source of information and detection technique. Each of these categories is described briefly in the following subsections.

2.2.1 Detection Technique

Considering their detection method, intrusion detection systems can be classified into two major types of signature-based and anomaly-based systems [7, 13, 23, 39, 48].

Signature-based

Signature-based IDSs try to identify malicious activities in a system by comparing the observed behavior to known attack patterns. In this type of systems, a database is used to store the behavior of previously known attacks which are encoded as signatures [59]. Therefore, these systems are able to detect attacks by finding similar signatures in the database.

Signature-based IDSs can operate reliably when the observed attack pattern is already known [51]. As a result, they have a very low rate of false positives. However, the missing ability in detecting unknown attacks is a major drawback of this type of systems which makes it an undesirable solution comparing to other types.

Anomaly-based

Anomaly-based IDSs create a profile by monitoring the normal activities on a system [29]. This profile is used to find deviations of the observed activities from the normal behavior of the system and report them as anomalies. In this type of systems, it is of great importance to define the normal profile of the system properly in an offline or online manner [20]. Otherwise, any activity that differ from the constructed profile can be identified as an intrusion and hence, the system may suffer from a high false positive rate which is caused by marking any non-intrusive abnormal behavior as malicious. In contrast with signature-based systems, as a key advantage anomaly-based systems are capable of detecting novel types of attacks.

2.2.2 Source of Information

The data used by an intrusion detection system may come from different sources on a network or system [50]. Depending on the source of this data, intrusion detection systems can be divided in two categories of host-based and network-based which are complementary solutions [28].

Host-based

In host-based IDSs, the information is collected from a single host and may consist of audit data, log files and resource usage of the machine [28]. Host-based IDSs have the advantage of detecting the malicious activities aimed toward the host's operating system which cannot be detected by network-based IDSs [5]. However, they also cause performance degradation on the host machine due to the background computational activities [12]. In addition, they may be disabled by the attacker as part of system compromise [8].

Network-based

Network-based IDSs monitor network traffic data and try to find malicious activities using the data coming from a single or multiple sources on the network [13]. This type of IDS can be deployed as a separate entity in the network and therefore it has its own resources. As a result, it addresses the performance degradation problem of the host-based solutions.

On the other hand, network-based IDSs do not provide any information regarding the impact of an attack on network hosts. Therefore, it is necessary to use some complementary solutions to find out whether an attack was successful in compromising the targeted host or not [5]. This can be done e.g. by collecting the audit logs of the network hosts.

2.3 Anomaly Detection System

An intrusion detection system which uses anomaly-based techniques for finding malicious activities is called an anomaly detection system. Anomaly detection systems are constructed based on the assumption that intrusions are a subset of anomalies [33]. In the ideal case, each detected anomaly is in fact an intrusion which means to have zero false positives in the system. Under this assumption, anomaly detection systems are capable of detecting intrusive activities even if the attacks are launched from a legitimate user account. As mentioned earlier, in contrast with signature-based systems, anomaly detection systems are capable of detecting unknown intrusions. However, the major drawback of these systems is the high rate of false alarms triggered on observing any legitimate abnormal activity on the monitored system or network. In real applications, the burden of false alarms may lead to the crash of the IDS [49]. On the other hand, upon detection of abnormal activities, the system administrator must spend a lot of time to identify the root cause of generated alarms [53]. As the number of alarms grows, the system administrator may not be able to monitor all the triggered alerts or may simply disable the alarm generator. Therefore, a number of true alarms may be lost among the large volume of false ones in system traces.

The common process of an anomaly detection system can be divided in two phases of training and testing. In the training phase, the normal profile of the system is constructed by the IDS. As discussed in earlier sections, this profile represents the normal behavior of the system in terms of defined features and metrics. In the testing phase, the IDS monitors ongoing activities of the system or network and tries to find deviations by comparing the trained profile to the new data [51].

Anomaly detection systems may use different techniques with respect to the available data features. These include statistics-based and machine learning-based techniques. In the following subsections, each of these techniques is discussed briefly.

2.3.1 Statistics-based

In statistics-based techniques, the system maintains two profiles for normal and current behavior of the system or network. In each of these profiles, the system behavior is modeled by calculating some statistical measures such as mean, standard deviation, distribution of the data, etc [54]. When a new set of activities is launched in the system or network, the IDS tries to find deviations between the two profiles according to the mentioned measures. By using some statistical tests, the system calculates an anomaly score for the current profile which indicates the degree of abnormality. If the anomaly score is higher than a certain threshold, the system triggers an alarm as a sign of an anomaly [51].

Statistics-based methods have a number of advantages. Firstly, they do not need any prior knowledge about the system or network since the normal profile can be built only by observing the activities in the system for a period of time. Furthermore, as these approaches use statistical models they are particularly effective in detecting the attacks occurring over long periods of time [17].

On the other hand, statistics-based approaches have two major drawbacks. In these methods, the system can be trained by an intruder so that the intrusive behavior can be assumed to be normal. Moreover, a primary principle of these approaches is to model data points as stochastic distributions assuming to be quasi-stationary process [17]. However, in this thesis, we take a frequentist approach to this problem and therefore, do not rely on such assumptions.

2.3.2 Machine learning-based

In machine learning-based approaches, the system tries to learn the patterns of the data and construct a model for categorizing the analyzed patterns [17]. Using this model, machine learning-based anomaly detection systems have the ability to improve their performance according to the previous results. Machine learning-based techniques [9] may use supervised or unsupervised learning.

In supervised learning, a set of labeled data is used by the system in the training phase. Each label represents a class of activities such as malicious or normal. After establishing a model on the basis of data features and their corresponding labels, the system is able to make predictions for categorizing the new data which is available in the testing phase.

On the other hand, in unsupervised learning, the available data set is not labeled. In this method the system tries to analyze the structure of the data and identify

similar data points which makes it quite effective in solving clustering problems.

Artificial Neural Networks

Among different machine learning models, artificial neural networks have been quite popular during the recent years. The basic goal of neural networks is to model the human brain and perform a particular task in a similar way as a human by learning from the input data. A neural network is made up of a number of interconnected neurons acting as processing units where each connection has a strength called weight [21].

In a neural network, computation takes place on a layer-by-layer basis where each layer contains a number of neurons. The basic goal of this property is to model the parallel computing capability of human brain [21]. The number of layers and the number of neurons in each layer can be adjusted according to the application. The primary task of a neural network is to learn from the input data and adjust the connection weights to achieve the desired output. This can be done by using a learning algorithm.

Neural networks may use different architectures and learning algorithms. A popular architecture of neural networks is Multi-layer Perceptron (MLP) which is particularly used for supervised learning. MLP consists of a single input and output layer containing the input and output data respectively. In addition, it contains a number of intermediate layers called hidden layers. In this type of neural network, all the neurons of a layer are connected to all the neurons of the previous layer. The weights can be optimized on a layer-by-layer basis using a training algorithm.

Among different training algorithms, Backpropagation (BP) has been widely used in many applications [36][30]. In this algorithm, after a random initialization of weights the actual output of the neural network is compared to the desired output. Starting from the output layer the local error of each layer is calculated. Considering the error values, the weights of each layer are adapted accordingly. This procedure continues until the network reaches a desired level of accuracy [21].

BP algorithm has a major drawback of having a long convergence time which makes it a bottleneck in real applications. Therefore, some other methods have been proposed to improve the convergence time of this algorithm. Extreme Learning Machine (ELM) is one of the algorithms proposed recently as an alternative solution which offers shorter training time. The details of this algorithm will be discussed in Chapter 4.

2.4 Summary

This chapter provided the required background in the field of intrusion detection containing different categories of intrusion detection systems considering their in-

formation sources as well as method of detection. In addition, the advantages and drawbacks of different methods of intrusion detection has been discussed. The provided knowledge about the existing solutions and their shortcomings gives a better understanding of the tackled problem of this thesis which is formulated in Chapter 3.

Chapter 3

Problem Statement

This chapter provides a description of the problem which is tackled in this thesis. In addition, the criteria used in this work for evaluation of the proposed solution are introduced. Finally, the data set which is used for assessing the effectiveness of the proposed method is described.

3.1 Problem Description

In the evolving nature of today's world of cyber-security, the deployment of intrusion detection systems has become a critical task. As the number of security threats increases, developing adaptive solutions for intrusion detection is becoming a major challenge. Despite the variety of intrusion detection approaches proposed in the recent years, there are several challenges remaining to be tackled in this field.

In the context of intrusion detection, the proper selection of the detection technique is a matter of discussion. While signature-based techniques provide a high detection rate for the well-known attacks, they are not capable of detecting novel ones. Although the majority of commercial intrusion detection systems mainly use signature-based techniques [51], with new types of attacks emerging continually, this missing ability disfavors the use of these systems in many applications.

On the other hand, anomaly-based techniques provide the capability of detecting unknown or "zero day" attacks by tracking the behavior of the system or network and comparing it with a well-defined normal profile. However, as mentioned in earlier chapters, the major drawback of these techniques is the high rate of false alarms which affects the performance of the IDS and causes difficulties for the administrators to identify the root cause of the triggered alarms. Therefore, providing an adaptive approach for reducing the likelihood of false alarms can be considered as a key requirement in designing an effective solution of intrusion detection.

Unavailability of labeled data set for training and evaluation of intrusion detection systems is another major challenge in real applications. This can be particularly

noticed in network-based intrusion detection systems. Since strict privacy-preserving laws are issued in many countries, sharing network traffic traces of the users for public use is banned [10]. As a result, since the labeled network data is not available for training IDSs, the use of supervised learning methods has become very limited. This makes unsupervised learning methods to be a proper choice for intrusion detection systems [34] as they do not need any prior labeling of the data set. However, even in unsupervised learning, the system decisions must be monitored and evaluated by security experts.

Evaluation of intrusion detection systems is a critical task. In every intrusion detection system, the decisions made by the system must be verified by human security experts. This verification consists in checking of all the system decisions and correcting the possible errors in classification of the data. In real systems, this evaluation must be performed manually. Therefore, it is a time consuming and error prone task and needs a lot of human effort. A proper intrusion detection solution should provide a mechanism for simplifying the task of evaluation and reducing the likelihood of human errors.

As new patterns and classes of attacks appear in the world of cyber-crimes, updating the existing intrusion detection systems according to new patterns of data has become an essential requirement. Such update requires the IDS to be retrained using the new data set beside the old existing one. As discussed in earlier chapters, training of an IDS is a costly operation in terms of computational time and resources. Therefore, performing such costly updates is not desired by system administrators although it has a great impact on the adaptivity of the designed solution according to new trends of cyber-attacks. Considering this fact, providing a timely and cost-effective update mechanism can be noted as a critical requirement in designing intrusion detection systems.

The main objective of this thesis is to propose a new intrusion detection system which can address the challenges and requirements of real life IDSs as discussed above. In subsequent chapters, the proposed IDS is constructed and evaluated using the criteria and the data set introduced in the following sections.

3.2 Evaluation Criteria

Evaluation of intrusion detection systems is a challenging task. Although a large amount of research has been conducted in this field, there is no globally accepted metric for measuring the performance of intrusion detection systems [51]. Generally, the evaluation methods of intrusion detection systems can be divided in two categories of offline and online evaluation. In offline evaluation, the system is tested by using an existing data set which simulates the behavior of real systems or networks. The most widely used data sets in the field of network intrusion detection are DARPA

1998 and 1999 data sets [42, 41]. Although these data sets have been used for evaluating many intrusion detection systems, they have been criticized as they cannot simulate real network environments [51]. On the other hand, in online evaluation, the intrusion detection system can be tested using the actual traffic of the network containing both normal and malicious activities in a real time manner [12]. While online evaluation provides a more realistic tailored test case, setting up such an evaluation is challenging as it needs similar experimental environment as in real networks.

Despite its shortcomings, in this thesis, offline method is chosen to evaluate the proposed intrusion detection system as a network-based solution using the data set introduced in the next section. The evaluation is conducted for two main criteria as follows.

Evaluation of Computational time

The main focus of this evaluation is to provide a study of the computational time required for updating the system according to the new information available as the input. In this study, the speed of the update mechanism of the proposed system is compared to the existing solution where the system is fully retrained. This comparison is performed by evaluating the computational time of each method while varying the parameters of the problem context such as the size of existing data, the number of traffic classes, etc. This criteria is relevant as one desired requirement of the system is its online adaptivity.

Evaluation of Accuracy

In this evaluation, the accuracy of the decisions made by the proposed IDS is studied. In order to measure such accuracy, a number of metrics may be taken into consideration. The most commonly used evaluation metrics in the field of intrusion detection are Detection Rate and False Positive Rate which can be defined as

$$\text{Detection Rate} = \frac{\text{Number of Detected Intrusions}}{\text{Total Number of Injected Intrusions}}$$

and

$$\text{False Positive Rate} = \frac{\text{Number of Normal Patterns Classified as Intrusions}}{\text{Total Number of Normal Patterns}}.$$

In the context of intrusion detection, an effective system must have a high detection rate while having a low rate of false positives. Using these two accuracy evaluation metrics, the adaptability of the proposed system to new trends of input data is studied.

3.3 Data Set Description

In this thesis, in order to evaluate the performance of the proposed intrusion detection system NSL-KDD data set is used which is a variant of KDD 99 data set.

KDD 99 is one of the publicly available network intrusion detection data sets which is widely used in several studies [19, 15, 6, 31, 16, 57]. The original data set was produced in DARPA Intrusion Detection Program [2] with the purpose of evaluation of research in intrusion detection. The data were collected over nine weeks of TCP dump data on a Local Area Network (LAN) which was processed into seven million records of connections. KDD 99 is a derivative of this data [3].

Despite its popularity, KDD 99 data set has been reported to suffer from various deficiencies discussed in [43]. A major problem of this data set is the huge number of redundant records in both train and test sets. This makes the machine learning model to be biased towards the most frequent records in the training phase so that it cannot learn well the less frequent samples. On the other hand, having too many repeated samples in the test set prevents the model to be properly evaluated.

NSL-KDD data set is a modified version of KDD 99 and has a number of advantages over the original KDD 99 data set [60]. The redundant records have been removed from both training and test sets so that the performance of classifiers can be evaluated properly in both phases. In addition, as the number of records in training and test sets are reasonable, there is no need to make a random selection of the records for evaluation. Therefore, the performance of different classifiers can be compared according to the results obtained from experimenting on the whole data set.

In NSL-KDD data set, each record represents a single connection corresponding to a sequence of TCP packets flowing between a source and destination IP addresses under a predefined protocol. This data set contains 41 number of features for each record. These features include 6 discrete fields as well as 35 continuous ones. The full description of features can be seen in Appendix A.

In addition, each record has a label which indicates the traffic class of that record. This label may correspond to either a normal or malicious traffic class. The available attack classes of NSL-KDD data set fall into four main categories as follows [3].

- DoS: Denial of Service, e.g. syn flood;

- R2L: Remote to Local - Unauthorized access from a remote machine e.g. guessing password;
- U2R: User to Root - Unauthorized access to local superuser (root) privileges, e.g., buffer overflow attacks;
- Probe: Surveillance and other probing, e.g., port scanning.

Each of the aforementioned attack categories consists of a number of example attacks in the data set. The full list of attacks can be seen in table 3.1. A detailed description of different attack types can be found in [58].

Table 3.1: List of Attacks in NSL-KDD Data Set

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

3.4 Summary

This chapter described the existing challenges in the field of intrusion detection and stated the problem which is tackled in this thesis. Furthermore, the criteria used for evaluating the effectiveness of the proposed solution as well as the data set which is used for evaluation of this work was presented. In the following chapter, the mathematical background which is required in constructing the proposed solution will be introduced with regards to the stated problem.

Chapter 4

Mathematical Methods and Algorithms

This chapter provides the theoretical machine learning background that is needed for constructing the proposed intrusion detection system. The machine learning methods which are used in this research are mainly based on the Extreme Learning Machine algorithm. In this chapter, the original Extreme Learning Machine (ELM) algorithm and two of its variants are discussed in detail.

4.1 Extreme Learning Machine

The original algorithm of ELM was proposed by Huang et al. in [25][26] for training Single hidden Layer Feedforward Neural networks (SLFN). In contrast with other existing learning methods such as Backpropagation (BP) and Levenberg-Marquardt (LVM) [47], while using a random initialization of input weights; ELM only updates the output weights in a single iteration without tuning the input weights. As a result, it provides a fast [61] and robust [45] [46] learning capability. The original ELM algorithm works as follows.

Suppose that we have a SLFN with n number of hidden neurons and a training set of N samples $(\mathbf{x}_i, \mathbf{y}_i)$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in \mathbb{R}^c$. The output of such SLFN can be represented as

$$\sum_{j=1}^{n} \beta_j G(\mathbf{w}_j \mathbf{x}_i + b_j), \qquad i \in [1, N], \tag{4.1}$$

where $\mathbf{w}_j$ and b_j are the learning parameters, β_j is the output weight of node j and $G : \mathbb{R} \to \mathbb{R}$ is the activation function.

A perfect approximation of SLFN with zero error shows that with random $\mathbf{w}_j$ and b_j there exists β_j such that

$$\sum_{j=1}^{n} \beta_j G(\mathbf{w}_j \mathbf{x}_i + b_j) = \mathbf{y}_i, \qquad i \in [1, N], \tag{4.2}$$

which can be written as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{Y}, \tag{4.3}$$

where

$$\mathbf{H} = \begin{bmatrix} G(\mathbf{w}_1 \mathbf{x}_1 + b_1) & ... & G(\mathbf{w}_n \mathbf{x}_1 + b_n) \\ . & ... & . \\ . & ... & . \\ . & ... & . \\ G(\mathbf{w}_1 \mathbf{x}_N + b_1) & ... & G(\mathbf{w}_n \mathbf{x}_N + b_n) \end{bmatrix}, \tag{4.4}$$

and

$$\boldsymbol{\beta} = (\beta_1^T \dots \beta_n^T)^T, \ \mathbf{Y} = (\mathbf{y}_1^T \dots \mathbf{y}_N^T)^T. \tag{4.5}$$

Assuming that the number of samples is greater than the number of hidden neurons, output weights can be calculated using the following formula

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{Y}, \tag{4.6}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse [55] of matrix $\mathbf{H}$. The calculation of $\mathbf{H}^{\dagger}$ can be performed in a single step without going through lengthy training iterations. This makes ELM a computationally cost-effective learning method.

The detailed theoretical proofs of ELM algorithm can be found in the original paper [26].

4.2 Online Sequential ELM

In many of the applications, SLFNs are trained using batch learning algorithms. Most of these algorithms suffer from lengthy training times caused by the large number of iterations needed to choose the proper parameters for the model. In these algorithms, it is assumed that the entire training set is available at once. On the arrival of new data, batch learning algorithms require the model to be fully retrained using the past data along with the new one and hence consuming a lot of time. Considering this, in many of the real applications the usage of online sequential learning algorithms is taken into consideration rather than batch learning algorithms as they can handle newly arrived chunks of data without requiring to perform full retraining [40].

The original ELM is a batch learning algorithm. In order to handle the cases where the training data arrives sequentially, Online Sequential Extreme Learning Machine (OS-ELM) [40] was proposed as a variant of ELM. As a sequential learning algorithm, on the arrival of new data, OS-ELM is able to update the model without requiring the previously trained data to be available. In addition, unlike many of the sequential learning algorithms such as Stochastic Gradient Descent Backpropagation (SGBP) [37] and Resource Allocation Network (RAN) [52], OS-ELM can learn data on a one by one or chunk by chunk basis while having fixed or varying chunk size.

The main idea of OS-ELM is to update the output weights of hidden layer on the arrival of new data without retraining the whole model. For this purpose, OS-ELM uses a special case of Sherman-Morrison-Woodbury formula [18] to update the generalized inverse of $\mathbf{H}$.

OS-ELM consists of two phases of initial and sequential learning. In the initial phase, the model is trained using N_0 number of distinct samples and the hidden layer output matrix $\mathbf{H_0}$ is constructed according to these samples. As a fundamental assumption of ELM and OS-ELM, N_0 must be equal or greater than the number of hidden neurons n, so that $rank(\mathbf{H_0}) \geq n$. After the initial phase, the model can be updated sequentially on the arrival of new data.

Assuming $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_0}$ as the available training set at the initial phase, $\mathbf{H}_0$ and $\boldsymbol{\beta_0}$ are the hidden layer output matrix and the output weights of ELM respectively and can be formulated as follows.

$$\mathbf{H_0} = \begin{bmatrix} G(\mathbf{w}_1\mathbf{x}_1 + b_1) & ... & G(\mathbf{w}_n\mathbf{x}_1 + b_n) \\ . & ... & . \\ . & ... & . \\ . & ... & . \\ G(\mathbf{w}_1\mathbf{x}_{N_0} + b_1) & ... & G(\mathbf{w}_n\mathbf{x}_{N_0} + b_n) \end{bmatrix}, \tag{4.7}$$

$$\boldsymbol{\beta_0} = \mathbf{P_0}\mathbf{H_0^T}\mathbf{Y_0}, \tag{4.8}$$

where $\mathbf{P_0} = (\mathbf{H_0^T}\mathbf{H_0})^{-1}$ and $\mathbf{Y_0} = (\mathbf{y}_1^T \dots \mathbf{y}_{N_0}^T)^T$.

After this initial phase, on the arrival of $(k+1)-th$ chunk of data with N_{k+1}number of samples, the partial hidden layer output matrix can be constructed as

$$\mathbf{H_{k+1}} = \begin{bmatrix} G(\mathbf{w}_1\mathbf{x}_{(\sum_{j=0}^{k} N_j)+1} + b_1) & \dots & G(\mathbf{w}_n\mathbf{x}_{(\sum_{j=0}^{k} N_j)+1} + b_n) \\ \vdots & \dots & \vdots \\ \vdots & \dots & \vdots \\ \vdots & \dots & \vdots \\ G(\mathbf{w}_1\mathbf{x}_{(\sum_{j=0}^{k+1} N_j)+1} + b_1) & \dots & G(\mathbf{w}_n\mathbf{x}_{(\sum_{j=0}^{k+1} N_j)} + b_n) \end{bmatrix}, \tag{4.9}$$

and the output weights can be updated using the following formula.

$$\boldsymbol{\beta_{k+1}} = \boldsymbol{\beta_k} + \mathbf{P_{k+1}}\mathbf{H_{k+1}^T}(\mathbf{Y_{k+1}} - \mathbf{H_{k+1}}\boldsymbol{\beta_k}), \tag{4.10}$$

with

$$\mathbf{P_{k+1}} = \mathbf{P_k} - \mathbf{P_k}\mathbf{H_{k+1}^T}(\mathbf{I} + \mathbf{H_{k+1}}\mathbf{P_k}\mathbf{H_{k+1}^T})^{-1}\mathbf{H_{k+1}}\mathbf{P_k}. \tag{4.11}$$

A comparison of OS-ELM with other well-known sequential learning algorithms indicates that OS-ELM has a lower training time while producing a better generalization performance [40].

4.3 CLUS-ELM

The smooth approximation of ELM makes it a proper solution for clustering problems. A clustering algorithm is proposed in [44] which assumes that the number of clusters and the size of each cluster are known as prior knowledge and tries to find the appropriate cluster for each sample. In this method, the ELM is used in a reversed way i.e. the cluster centers are projected into the input data space. In other words, the cluster centers are the inputs and the original input samples are the output of ELM.

The algorithm is initialized by random assignment of samples to different clusters and training the model using ELM. The number of hidden neurons is adjusted in a way to achieve the lowest possible training error.

After the initial training where the output weights are calculated as $\boldsymbol{\beta}^{\text{old}}$, the proposed method keeps swapping the rows of samples to obtain a lower training error.

Considering $\boldsymbol{\beta}^{\text{old}} = \mathbf{H}^{\dagger}\mathbf{X}^{\text{old}}$ as the output weights matrix calculated by ELM, the proposed method keeps swapping the rows of the desired ELM output to find a better cluster assignment for the samples i.e. achieving a lower training error.

Assuming rows i and j are swapped, the new output weights matrix can be updated using the following formula.

$$\boldsymbol{\beta}^{\text{new}} = \boldsymbol{\beta}^{\text{old}} - \mathbf{H}^{\dagger(i)}\mathbf{X}_{(i)} - \mathbf{H}^{\dagger(j)}\mathbf{X}_{(j)} + \mathbf{H}^{\dagger(i)}\mathbf{X}_{(j)} + \mathbf{H}^{\dagger(j)}\mathbf{X}_{(i)}, \tag{4.12}$$

where $\mathbf{H}^{\dagger(i)}$ is column i in $\mathbf{H}^{\dagger}$ and $X_{(i)}$ is row i in X^{old}.

Considering the updated output weights, the algorithm evaluates the success of swapping decision with respect to the new training error which is $\frac{1}{N}||\mathbf{H}\boldsymbol{\beta}^{\text{new}} - \mathbf{X}^{\text{new}}||^2$. In case of achieving a smaller training error, the proposed methodology keeps the current cluster assignment for the swapped samples.

4.4 Summary

This chapter provided a detailed description of the original algorithm of ELM and two of its variants i.e. OS-ELM and CLUS-ELM. In this thesis, the proposed intrusion detection system is constructed on the basis of these algorithms. It is necessary to note that a critical requirement of an effective design for intrusion detection systems is the fast learning capability and real time detection which are supported by the aforementioned algorithms. The fundamental learning method of the proposed IDS in this research is based on the clustering approach of CLUS-ELM. Although this algorithm is originally proposed as an unsupervised learning method, in this thesis, it can be used both for unsupervised and supervised problems. In addition, in the proposed methodology, the prior assumptions of CLUS-ELM i.e. the number of clusters and the number of samples in each cluster can be modified after the initial convergence of the model. In such cases, two novel approaches are proposed in order to update the learning model in a fast adaptive manner. On the other hand, in order to provide the capability of sequential training for the proposed IDS, the OS-ELM algorithm is used. In the following Chapter 5, the proposed intrusion detection

system will be introduced and the novel update approaches will be presented in detail.

Chapter 5

The Proposed Intrusion Detection System

In this chapter, the proposed intrusion detection system is described. Firstly, an overview of the system is presented and explained briefly. Secondly, the function of different components of the system is discussed in detail. In addition, the operation of the system is exemplified using toy data.

5.1 System Overview

The proposed intrusion detection system in this thesis is constructed on the basis of anomaly-based machine learning techniques. Thus, it is capable of detecting novel attacks. The proposed IDS uses an adaptive approach to reduce the number of false alarms over time. This is done by providing the capability of having the input from human experts and updating the learning model in accordance with that input. This reduces the likelihood of generating repeating false alarms for similar data in an adaptive manner.

On the other hand, the proposed IDS can be used both as an unsupervised and supervised learning method. As a result, there is no requirement of having labeled data in the training phase. While having the number of traffic classes as the only required prior knowledge, the proposed solution is capable of finding the correct labels for unlabeled data. In addition, it provides correction proposals in cases where the training data is labeled manually by human experts.

The proposed intrusion detection system provides a scoring mechanism to simplify the evaluation of decisions for human security experts. In this mechanism, the system is capable of detecting errors by means of predefined measurements. In addition, in case of supervised learning, the system has the ability to detect human errors in labeling the data and provide correction proposals to the expert. On the other hand, using such scoring mechanism, the system is capable of detecting new classes of traffic.

Moreover, the proposed intrusion detection solution provides a fast update mechanism which addresses the adaptability challenge of the existing solutions. The proposed mechanism can be used to update the learning model according to the new data as well as new attack types with a very low computational cost.

Fig. 5.1 represents an overview of the proposed intrusion detection system. The proposed IDS consists of three main components of Clustering Manager (CM), Evaluation Engine (EE) and Update Manager (UM). Clustering Manager is responsible for constructing a learning model by mapping the training data into different clusters where each cluster represents a class of traffic. Evaluation Engine is responsible for evaluating the clustering decisions made by the trained model. This component consists of two elements called Scorer and Fault Detector (FD). The Scorer tries to measure the accuracy of clustering decisions on different data samples by assigning a set of score values to each sample. These scores are used by the Fault Detector to detect possible errors in the current clustering decisions and provide modification proposals to the learning model. These proposals can be used by Update Manager component which has three procedures for updating the model in different cases.

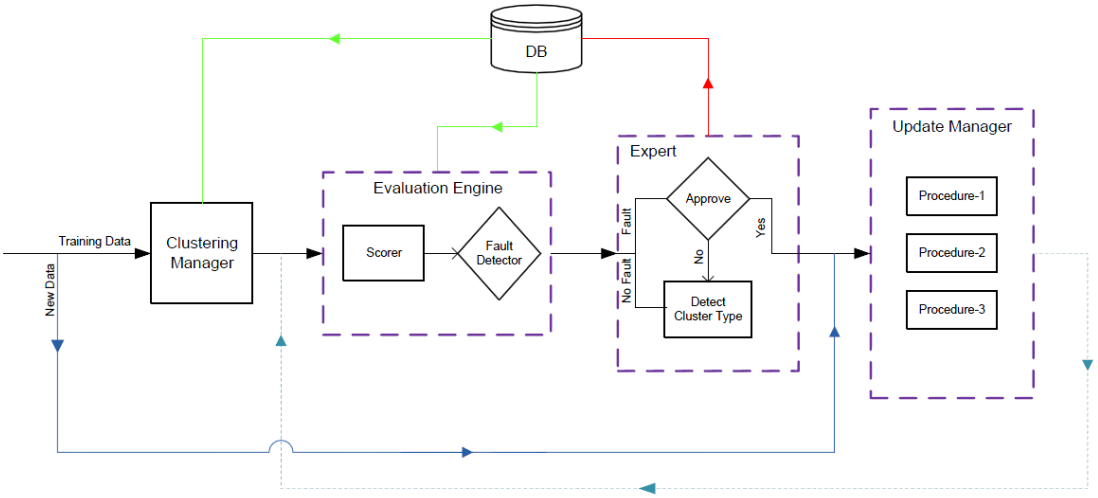


Figure 5.1: The Proposed IDS Overview

In addition to these three components, the proposed system is able to interact with an external human expert. This expert is responsible for firstly approving or rejecting the correction proposals provided by the Evaluation Engine and secondly labeling each class of traffic by checking the cluster centers.

To clarify the relations between different components of the system, assume that the model is constructed by the Clustering Manager according to some training data. The Evaluation Engine tries to validate the constructed model by verifying

the current clustering decisions and providing proposals for modifying the model. If there is no fault reported by the Evaluation Engine, the expert can label the current clusters by looking at the cluster centers. Otherwise, the expert may reject or approve the proposals. On the approval, the proposals must be sent to the Update Manager to apply the required changes to the model. On the other hand, on arrival of new data, the Update Manager is responsible for updating the model according to the newly arrived samples. In case of any changes applied to the system, the updated learning model must be evaluated by the Evaluation Engine.

5.2 Clustering Manager

The Clustering Manager is the primary component of the proposed IDS which is responsible for clustering data samples. This component uses a modified version of the algorithm introduced in CLUS-ELM. In the proposed approach, the number of clusters and the probability of data samples being in each cluster are known as prior assumptions of the expert. As a key advantage of our model, such assumptions can be tuned later according to the clustering evaluation results produced by the Evaluation Engine.

In order to highlight the importance of such an advantage, an example scenario can be considered. Suppose a situation in which the expert assumes the traffic being categorized into two different classes of normal and malicious where their occurrence probabilities are set as 0.8 and 0.2 respectively according to the general status of the network. In this case, if these probabilities are not correctly set by the expert, our proposed model is capable of modifying the expert knowledge and tuning the clustering according to that.

Now suppose that the number of clusters and the probability of data being in cluster j (c_j) are formulated as nc and $Pr(c_j)$ respectively. Therefore, the number of samples being in cluster j can be shown as

$$size(c_j) = NPr(c_j), \tag{5.1}$$

where N is the total number of samples in the training set.

In order to have independent and equally spaced cluster centers, the cluster assignment is formulated by a 1-in-all code i.e. cluster j can be coded as

$$\mathbf{c}_j = \left[\ c_k\ \right]_{1\times nc} \quad c_k = \begin{cases} 1 & k = j \\ 0 & else \end{cases}. \tag{5.2}$$

Considering the adjusted number of clusters and the number of samples in each cluster, the cluster assignment matrix $\mathbf{C}$ can be constructed as

$$\mathbf{C} = \begin{bmatrix} 0 & . & . & . & 1 & 0 & 0 & . & . & . & 0 \\ . & & & & . & & & . & & & \\ . & & & & . & & & . & & & \\ . & & & & . & & & . & & & \\ 0 & . & . & . & 0 & 1 & 0 & . & . & . & 0 \end{bmatrix}_{N\times nc}. \tag{5.3}$$

To find a proper clustering, it is required to map the input matrix $\mathbf{X}$ to clusters matrix $\mathbf{C}$. However, according to CLUS-ELM, another solution is to perform clustering in a reversed fashion by projecting cluster centers into the original input data space. This is particularly important to avoid the errors in distances between data samples [44].This can be done by using ELM with $\mathbf{C}$ and $\mathbf{X}$ being the input and output matrices respectively.

In the initial step, the samples in $\mathbf{X}$ must be assigned to the rows in clusters matrix $\mathbf{C}$ randomly. This assignment can be modified later to achieve a smaller training error. Also the number of hidden neurons in ELM must be adjusted in a way to obtain the smallest possible training error. This can be done by performing k-fold cross-validation test [32] on the initial cluster assignment.

Now suppose that n is the number of hidden neurons and $\mathbf{W}$ is the input weights matrix of ELM which can be denoted as

$$\mathbf{W} = \begin{bmatrix} w_{11} & ... & w_{n1} \\ . & ... & . \\ . & ... & . \\ . & ... & . \\ w_{1nc} & ... & w_{nnc} \end{bmatrix}_{nc\times n}. \tag{5.4}$$

Considering the 1-in-all property of rows in clusters matrix and assuming the r^{th} sample being assigned to cluster j, the corresponding row of the hidden layer output

matrix of ELM will be

$$\mathbf{H}_r^{nonlinear} = \left[\begin{array}{ccc} G(w_{1j} + b_1) & ... & G(w_{nj} + b_n) \end{array} \right], \tag{5.5}$$

where b_i is the bias value of neuron i and G is the non-linear activation function.

Therefore, under the assumption of first and last sample being assigned to clusters j and k respectively, hidden layer output matrix can be represented as

$$\mathbf{H}^{nonlinear} = \left[\begin{array}{ccc} G(w_{1j} + b_1) & ... & G(w_{nj} + b_n) \\ . & ... & . \\ . & ... & . \\ . & ... & . \\ G(w_{1k} + b_1) & ... & G(w_{nk} + b_n) \end{array} \right]. \tag{5.6}$$

The original version of the ELM proposed in [24] did not include linear neurons in addition to the non-linear ones. While it is theoretically possible to approximate a linear behavior using only non-linear neurons, it is much easier if linear neurons are present in the network. Miche et al. in [45] have proposed to use such linear neurons for such reasons.

Therefore, in addition to the n hidden neurons with non-linear activation function, we also consider using of nc additional linear neurons in the hidden layer. The new output matrix of the hidden layer will be

$$\mathbf{H} = [\mathbf{H}^{nonlinear} \quad \mathbf{C}] = \left[\begin{array}{ccccccccccc} G(w_{1j} + b_1) & ... & G(w_{nj} + b_n) & 0 & ... & 1 & 0 & 0 & ... & 0 \\ . & ... & . & & ... & & & & & . \\ . & ... & . & & ... & & & & & . \\ . & ... & . & & ... & & & & & . \\ G(w_{1k} + b_1) & ... & G(w_{nk} + b_n) & 0 & ... & 0 & 1 & 0 & ... & 0 \end{array} \right]. \tag{5.7}$$

As mentioned in Chapter 4, the output weights matrix $\boldsymbol{\beta}$ of the ELM can be computed as

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{X}.$$

After obtaining the initial $\boldsymbol{\beta}$, the algorithm keeps swapping the samples in $\mathbf{X}$ as in CLUS-ELM to achieve a smaller training error which is formulated as

$$\varepsilon_{\text{train}} = \frac{1}{N}||\mathbf{H}\boldsymbol{\beta} - \mathbf{X}||^2. \tag{5.8}$$

In case of a successful swap which lowers the training error, the weights matrix $\boldsymbol{\beta}$ is updated using Eq. 4.12.

As a key improvement to the convergence time of CLUS-ELM approach, we propose two methods for selecting the best swapping candidates among the data samples. In the first method, the swapping candidates can only be selected from a subset of samples with the highest Mean Squared Error (MSE). This provides a better chance of selecting the samples with a larger distance from their cluster centers. On the other hand, another approach is to select a proportion of candidates from the subset with highest MSE while choosing the rest randomly.

A comparison of the two proposed method and the original swapping algorithm of CLUS-ELM with regards to their convergence time is presented in Fig. 5.2. The experiments of this comparison are conducted on a synthetic data set of 2000 samples where the data points are randomly generated into two clusters with the same size. The clusters are composed of normally distributed points in a 2-dimensional feature space, with means $(1, 1)$ and $(2, 2)$ and standard deviations 0.2. The number of hidden neurons is set as 100. The results are averaged over 20 runs using Matlab on a Windows machine with 8 GB of RAM and CPU of 2.30 GHz.

According to the obtained results, both of the proposed methods perform faster than the original CLUS-ELM in terms of convergence time. Among these methods, the approach which selects all the candidates from the subset with highest MSE outperforms the other method as it gives a lower total MSE in a shorter time.

5.3 Evaluation Engine

The Evaluation Engine is responsible for assessing the clustering assignments and detecting the possible errors in such assignments. This component provides some measurements for the human expert and suggests some modifications to the model to improve the accuracy of clustering. EE contains two components of Scorer and Fault Detector which are discussed in detail in the following subsections.

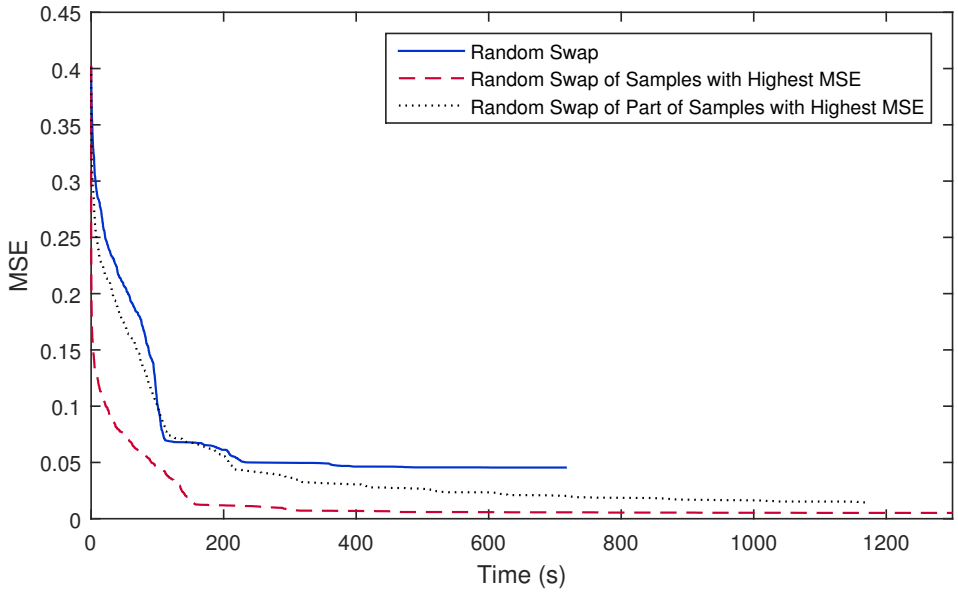


Figure 5.2: Comparison of swapping algorithms with regards to their convergence time. The algorithm which selects the swapping candidates from the samples with highest MSE achieves the lowest error in the shortest time.

5.3.1 Scorer

The Scorer component is responsible for providing the evaluation results of the current clustering to other components such as Fault Detector. These results can be studied by Fault Detector to detect the possible errors and suggest modifications to the clustering.

Assuming that the current number of clusters is nc, the Scorer calculates nc number of scores for each sample. These scores are driven directly from the MSE and can be interpreted as the scores of each sample being in any of the clusters. The scores can be measured by rotating the clusters matrix $\mathbf{C}$ by one column for nc times, constructing the hidden layer output matrix $\mathbf{H}_{rotated}$ of the rotated matrix and simply calculating the per sample training error for each matrix using the following formula

$$\text{MSE}_{rotated} = \frac{1}{nc}||\mathbf{H}_{rotated}\boldsymbol{\beta} - \mathbf{X}||. \tag{5.9}$$

Therefore, after normalizing the obtained training errors with respect to the maximum and minimum available values the output of Scorer can be denoted as

$$\mathbf{Sc} = \{sc_{ij}\}_{1\leq j\leq nc}^{1\leq i\leq N},$$

where $sc_{ij} = 1 - \mathrm{MSE_{ij}}$ and can be interpreted as the score of i^{th} sample being in cluster j.

5.3.2 Fault Detector

The Fault Detector (FD) component is responsible for finding possible errors in the current clustering and providing correction proposals to the human expert. This component utilizes the scores calculated by the Scorer to find suspected clustering decisions.

There are two different scenarios where the current clustering may not be accurate enough. First case happens when the ratio of different types of traffic which is imported to the model by the expert does not suit the real training data. And the other one occurs when a new type of attack is detected, i.e. a new cluster is needed. In order to handle these cases, the proposed FD uses two criteria for detecting the samples suspected of not being in the proper cluster.

For the first scenario, FD prepares a new clusters matrix only based on the calculated scores. Based on that, each sample will be assigned to the cluster which has the highest score in the corresponding row of scores matrix i.e. the cluster with the lowest error. Comparing the new clustering assignments to the original ones, FD finds the unsimilar rows and sends an alarm to the expert along with the suspected samples and their likely clusters.

As an example, assume that in a three-cluster problem the cluster matrix $\mathbf{C}_{example}$ and the corresponding scores matrix $\mathbf{Sc}_{example}$ are driven as follows.

$$\mathbf{C}_{example} = \begin{bmatrix} 1 & 0 & 0 \\ . & . & . \\ . & . & . \\ . & . & . \\ 0 & 0 & 1 \end{bmatrix}_{N\times 3}, \tag{5.10}$$

and

$$\mathbf{Sc}_{example} = \begin{bmatrix} 0.5 & 0.6 & 0.9 \\ . & . & . \\ . & . & . \\ . & . & . \\ 0.44 & 0.5 & 0.93 \end{bmatrix}_{N\times 3}. \tag{5.11}$$

Considering the values in the score matrix, the new clusters matrix constructed by FD will be

$$\mathbf{C}^{FD}_{example} = \begin{bmatrix} 0 & 0 & 1 \\ . & . & . \\ . & . & . \\ . & . & . \\ 0 & 0 & 1 \end{bmatrix}_{N\times 3} . \tag{5.12}$$

Comparing the two matrices of $\mathbf{C}_{example}$ and $\mathbf{C}^{FD}_{example}$, FD proposes to change the cluster assignment of the first sample in the data set.

The second scenario focuses on the ability of the system to detect new types of attacks as an essential duty of the proposed IDS. In the proposed methodology, it is obvious that the samples belonging to a new traffic type do not suit any of the existing clusters. However, even such samples will finally find the best cluster among the bad ones although having very low scores. Considering this fact, we use the difference of two highest scores on each row of the scores matrix as a metric to find suspected samples. While a larger difference means that the new sample fits into its current cluster properly, a small difference shows that the sample cannot be placed into any of the existing clusters or simply fit into two or many of them. As a result, assigning such sample to a newly added cluster may increase the accuracy of clustering.

Suppose that the clusters matrix and the corresponding scores matrix is given as follows.

$$\mathbf{C}_{example} = \begin{bmatrix} 0 & 0 & 1 \\ . & . & . \\ . & . & . \\ . & . & . \\ 0 & 0 & 1 \end{bmatrix}_{N\times 3} , \tag{5.13}$$

and

$$\mathbf{Sc}_{example} = \begin{bmatrix} 0.61 & 0.59 & 0.62 \\ . & . & . \\ . & . & . \\ . & . & . \\ 0.93 & 0.93 & 0.95 \end{bmatrix}_{N\times 3} . \tag{5.14}$$

Considering the score matrix, it is obvious that the values in the 2^{nd} column of the matrix is too close to the values in the 3^{rd} column for the first and last row. So the corresponding samples in the data set are suspected to be from a new class of attacks. However, the value of the highest score in the last row shows a small clustering error meaning that the last sample can fit into all three existing clusters. On the other hand, the value of the highest score in the first row of $\mathbf{Sc}_{example}$ shows a large clustering error which means the first sample may be a proper candidate for being in a new cluster. As a result, in such example the FD proposes to add a new cluster to the model and place the first sample in that cluster.

5.4 Update Manager

The main responsibility of the Update Manager (UM) component is to modify the clustering model according to new changes applied to cluster assignment. These changes may be applied based on the human expert's decision or due to the faults detected by Fault Detector component. As a requirement of real time attack detection, it is essential that the model update can be performed as fast as possible. The proposed UM fulfills such requirement by avoiding the full retraining of the model.

In general, there are three different cases where the clustering model needs to be updated. These cases include the availability of new incoming data, adding a new cluster and change of cluster assignment for a group of samples. In the following subsections, the proposed update mechanism for each of the cases is discussed in detail. In all the cases, it is assumed that the model is already trained and verified by the expert at the start of the update procedure.

5.4.1 UM-Procedure-1: New Data

The basic capability of the UM is to update the clustering model according to the new incoming data. The cluster assignment of such data may be known or unknown which can be interpreted as supervised or unsupervised learning. In the proposed model, we take advantage of the same idea as in OS-ELM to handle both types of updates. In case of unsupervised learning, the data samples are labeled in a random manner initially by assigning the samples to one of the existing clusters. Later, this assignment will be evaluated and modified according to the available scores provided by EE. The same procedure will be applied to the case of supervised learning with a slight difference as the cluster is already known. In this case, the Fault Detector component may provide some correction proposals for modifying the initial labels in order to enhance the accuracy of clustering.

Let N_1 be the size of the newly arrived chunk of data where all the data samples fit best into cluster j. Therefore, the output matrix of the hidden layer of ELM for the new data can be formulated as

$$\mathbf{H}^{chunk} = \begin{bmatrix} G(w_{1j}+b_1) & ... & G(w_{nj}+b_n) & 0 & ... & 1 & 0 & 0 & ... & 0 \\ . & ... & . & & ... & & & & & . \\ . & ... & . & & ... & & & & & . \\ . & ... & . & & ... & & & & & . \\ G(w_{1j}+b_1) & ... & G(w_{nj}+b_n) & 0 & ... & 1 & 0 & 0 & ... & 0 \end{bmatrix}_{N_1\times(n+nc)} . \tag{5.15}$$

Assuming $\mathbf{H}^{old}$ to be the current output matrix of the hidden layer, the $\mathbf{H}^{new}$ matrix can be constructed as

$$\mathbf{H}^{new} = \begin{bmatrix} \mathbf{H}^{old} \\ \mathbf{H}^{chunk} \end{bmatrix}. \tag{5.16}$$

Thus, according to the proposed method in OS-ELM the new output weights can be calculated using the following formulas.

$$\mathbf{P}^{new} = \mathbf{P}^{old} - \mathbf{P}^{old}\mathbf{H}^{newT}\left(\mathbf{I} + \mathbf{H}^{new}\mathbf{P}^{old}H^{newT}\right)^{-1}\mathbf{H}^{new}\mathbf{P}^{old}, \tag{5.17}$$

and

$$\boldsymbol{\beta}^{new} = \boldsymbol{\beta}^{old} + \mathbf{P}^{new}\mathbf{H}^{newT}\left(\mathbf{X}^{chunk} + \mathbf{H}^{new}\boldsymbol{\beta}^{old}\right), \tag{5.18}$$

where $\mathbf{P}^{old} = \left(\mathbf{H}^{oldT}\mathbf{H}^{old}\right)^{-1}$ and $\mathbf{X}^{chunk}$ is the arrived chunk of data.

After updating the output weights matrix $\boldsymbol{\beta}^{new}$, the clustering assignment must be evaluated by the EE. Upon detection of errors in cluster assignments, clustering will be modified automatically by the system using UM-Procedure-3 for the unsupervised learning. On the other hand, if the learning type is supervised, EE sends the results to the expert who is responsible for verifying the correctness of clustering for the new samples.

5.4.2 UM-Procedure-2: New Cluster

In case of new type of incoming traffic, the Fault Detector sends an alarm to inform the UM about some samples not being a proper match to any of the current clusters.

In this case, it is necessary to add a new cluster to the model so that it can be able to find a better cluster assignment for the mismatched samples. Such update can be performed as to first add a new cluster to the model and then to map the mismatched samples to the new cluster. In order to adapt the model to the change in cluster numbers, some of the parameters such as $\boldsymbol{\beta}$ and $\mathbf{C}$ must be updated. This can be done by going through the following procedure.

The new cluster can be added to the model by simply adding a column of zeros to the current clusters matrix $\mathbf{C}$ meaning that currently there is no sample in this cluster. In addition, a new input weight vector of $\mathbf{w}_d$ must be initialized randomly where d is the new number of clusters. Adding a column of zeros to the clusters matrix does not affect the values in $\mathbf{H}$ and $\mathbf{H}^{\dagger}$ except for adding a new column and row of zeros to each respectively.

Now suppose that the i^{th} sample is a mismatched row and must be assigned to the newly added cluster. Assuming this sample to be in j^{th} cluster originally, the corresponding row in $\mathbf{H}$ can be formulated as

$$\mathbf{H}_i^{old} = \begin{bmatrix} G(w_{j1} + b_1) & ... & G(w_{jn} + b_n) & 0 & ... & 1 & 0 & 0 & ... & 0 \end{bmatrix}. \tag{5.19}$$

And assigning this sample to the new cluster only changes the values in i^{th} row of $\mathbf{H}$ to

$$\mathbf{H}_i^{new} = \begin{bmatrix} G(w_{d1} + b_1) & ... & G(w_{dn} + b_n) & 0 & ... & 0 & 0 & 0 & ... & 1 \end{bmatrix}. \tag{5.20}$$

Considering these values, $\mathbf{H}^{new}$ can be written as

$$\mathbf{H}^{new} = \mathbf{H}^{old} + \mathbf{H}_{i(j)}^{update}, \tag{5.21}$$

where $\mathbf{H}_{i(j)}^{update}$ is a whole zero matrix except for the i^{th} row which equals $\mathbf{H}_i^{new} - \mathbf{H}_i^{old}$.

To update output weights $\boldsymbol{\beta}$, the pseudo-inverse of the $\mathbf{H}^{new}$ matrix must be calculated. In the proposed methodology, we take advantage of the work of Campbell and Meyer on generalized inverses of linear transformations [11] to speed up this process.

According to [11], pseudo-inverse of any matrix of the form $\mathbf{H}_1 = \left(\mathbf{H} + \mathbf{uv^T}\right)$ with $\mathbf{u}$ and $\mathbf{v}$ being vectors can be formulated as

$$\mathbf{H}_1^\dagger = \mathbf{H}^\dagger + \mathbf{M} \tag{5.22}$$

where $\mathbf{M}$ is a matrix made up of sums and products of matrices $\mathbf{H}$, $\mathbf{H}^\dagger$, $\mathbf{u}$, $\mathbf{v}$ and their conjugate transposes. Therefore, rather than the lengthy calculation of pseudo-inverse of $\mathbf{H}_1$ the following procedure can be utilized with a lower computational cost.

From [11], Theorem 3.1.3, we first calculate the intermediate step vectors $\mathbf{q}$, $\mathbf{r}$, $\mathbf{s}$, $\mathbf{t}$ and the scalar γ as follows.

$$\mathbf{q} = \mathbf{H}^\dagger\mathbf{u}$$

$$\mathbf{r} = \mathbf{v^T}\mathbf{H}^\dagger$$

$$\mathbf{s} = \left(\mathbf{I} - \mathbf{H}\mathbf{H}^\dagger\right)\mathbf{u}$$

$$\mathbf{t} = \mathbf{v^T}\left(\mathbf{I} - \mathbf{H}^\dagger\mathbf{H}\right)$$

$$\gamma = 1 + \mathbf{v^T}\mathbf{H}^\dagger\mathbf{u}$$

Then, and denoting by $\mathbf{u}^\dagger = \mathbf{u}/||\mathbf{u}||^2$, the six following cases give the form of the pseudo inverse $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger$:

1. If $\mathbf{s} \neq \mathbf{0}$ and $\mathbf{t} \neq \mathbf{0}$, then $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger = \mathbf{H}^\dagger - \mathbf{q}\mathbf{s}^\dagger - \mathbf{t}^\dagger\mathbf{r} + \gamma\mathbf{t}^\dagger\mathbf{s}^\dagger$.

2. If $\mathbf{s} = \mathbf{0}$, $\mathbf{t} \neq \mathbf{0}$ and $\gamma = 0$, then $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger = \mathbf{H}^\dagger - \mathbf{q}\mathbf{q}^\dagger\mathbf{H}^\dagger - \mathbf{t}^\dagger\mathbf{r}$.

3. If $\mathbf{s} = \mathbf{0}$ and $\gamma \neq 0$, then $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger = \mathbf{H}^\dagger + \frac{1}{\bar{\gamma}}\mathbf{t}^T\mathbf{q}^T\mathbf{H}^\dagger - \frac{\bar{\gamma}}{\sigma_1}\mathbf{m}_1\mathbf{n}_1^T$, with $\mathbf{m}_1 = -\left(\frac{||\mathbf{q}||^2}{\bar{\gamma}}\mathbf{t}^T + \mathbf{q}\right)$, $\mathbf{n}_1^T = -\left(\frac{||\mathbf{t}||^2}{\bar{\gamma}}\mathbf{q}^T\mathbf{H}^\dagger + \mathbf{r}\right)$, and $\sigma_1 = ||\mathbf{q}||^2||\mathbf{t}||^2 + \gamma^2$.

4. If $\mathbf{s} \neq \mathbf{0}$, $\mathbf{t} = \mathbf{0}$, and $\gamma = 0$, then $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger = \mathbf{H}^\dagger - \mathbf{H}^\dagger\mathbf{r}^\dagger\mathbf{r} - \mathbf{q}\mathbf{s}^\dagger$.

5. If $\mathbf{t} = \mathbf{0}$ and $\gamma \neq 0$, then $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger = \mathbf{H}^\dagger + \frac{1}{\bar{\gamma}}\mathbf{H}^\dagger\mathbf{r}^T\mathbf{s}^T - \frac{\bar{\gamma}}{\sigma_2}\mathbf{m}_2\mathbf{n}_2^T$, with $\mathbf{m}_2 = -\left(\frac{||\mathbf{s}||^2}{\bar{\gamma}}\mathbf{H}^\dagger\mathbf{r}^T + \mathbf{q}\right)$, $\mathbf{n}_2^T = -\left(\frac{||\mathbf{r}||^2}{\bar{\gamma}}\mathbf{s}^T + \mathbf{r}\right)$ and $\sigma_2 = ||\mathbf{r}||^2||\mathbf{s}||^2 + \gamma^2$.

6. If $\mathbf{s} = \mathbf{0}$, $\mathbf{t} = \mathbf{0}$ and $\gamma = 0$, then $\left(\mathbf{H} + \mathbf{uv^T}\right)^\dagger = \mathbf{H}^\dagger - \mathbf{qq}^\dagger\mathbf{H}^\dagger - \mathbf{H}^\dagger\mathbf{r}^\dagger\mathbf{r} + \left(\mathbf{q}^\dagger\mathbf{H}^\dagger\mathbf{r}^\dagger\right)\mathbf{qr}$.

Now that we are able to calculate the pseudo-inverse of the updated matrix in the mentioned form, we need to find the proper $\mathbf{u}$ and $\mathbf{v}$ for our case which can meet the requirement of $\mathbf{H}_i^{update} = \mathbf{uv}^T$.

Remembering the fact that $\mathbf{H}_{i(j)}^{update}$ is an all-zero matrix except for the i^{th} row, it is clearly a $rank(1)$ matrix. Therefore it can be decomposed as the product of a column and a row vector which can be denoted as

$$\mathbf{u} = \begin{bmatrix} u_k \end{bmatrix} \quad u_k = \begin{cases} 1 & k = i \\ 0 & else \end{cases}$$

and $\mathbf{v} = \mathbf{H}_i^{new} - \mathbf{H}_i^{old}$.

As an important fact, it is necessary to note that the updates of moving samples from cluster j to cluster d can be performed in a single iteration of the six-case method mentioned above. This can be done by using the same $\mathbf{v}$ and simply modifying $\mathbf{u}$ to contain the value "1" in all the rows which their corresponding rows in $\mathbf{H}^{new}$are updated. As a result, $\mathbf{u}$ can be formulated as

$$\mathbf{u} = \begin{bmatrix} u_k \end{bmatrix} \quad u_k = \begin{cases} 1 & k^{th} \text{ sample placed from cluster } j \text{ to cluster } d \\ 0 & \text{else} \end{cases}$$

Utilizing such $\mathbf{u}$ and $\mathbf{v}$ and the previously mentioned method, the pseudo-inverse of the updated matrix and hence $\boldsymbol{\beta}$ can be simply calculated.

In order to generalize the proposed update method to be used in different cases, it should be considered that $\mathbf{H}_{total}^{update}$ which contains the updates of all clustering changes can be denoted as

$$\mathbf{H}_{total}^{update} = \sum_{j=1}^{d-1} \mathbf{H}_{i(j)}^{update}$$

Therefore, the entire update procedure can be completed in $(d-1)$ iterations with a low computational cost comparing to full retraining of the model.

5.4.3 UM-Procedure-3: Change of Cluster Assignment

There are two cases where the clustering decisions for some of the samples must be changed due to the errors in cluster assignments. Firstly, the clustering update may be requested by the FD component which shows a mismatching between the sample's current assignment and its highest scored cluster. This means that according to the calculated scores, there is a more appropriate cluster for the specified sample in terms of MSEs. Secondly, the update request may come from a human expert. An example scenario of this case may be that the expert decides that a new type of traffic being normal rather than malicious after studying the status of the network. Therefore, similar traffic must be considered as normal by the model in future clustering decisions. In any of these cases, the parameters of the model must be updated in order to improve the clustering accuracy for future samples.

The problem of cluster assignment modification is an expanded case of adding the new cluster problem where in the latter the preferred cluster for all the samples is the d^{th} one. Therefore, here we use a similar technique as in the previous problem along with a small modification.

Suppose i^{th} and l^{th} samples which are being in clusters j and k originally are requested to be replaced in clusters k and j respectively. As a result. the corresponding rows in $\mathbf{H}^{old}$ and $\mathbf{H}^{new}$ matrices can be represented as

$$\mathbf{H}_i^{old} = \mathbf{H}_l^{new} = \Big[\ G(w_{j1} + b_1) \quad ... \quad G(w_{jn} + b_n) \quad 0 \quad ... \quad 1 \quad 0 \quad 0 \quad ... \quad 0\ \Big], \tag{5.23}$$

$$\mathbf{H}_i^{new} = \mathbf{H}_l^{old} = \Big[\ G(w_{k1} + b_1) \quad ... \quad G(w_{kn} + b_n) \quad 0 \quad ... \quad 0 \quad 0 \quad 1 \quad ... \quad 0\ \Big]. \tag{5.24}$$

Considering these values, $\mathbf{H}^{new}$ can be written as

$$\mathbf{H}^{new} = \mathbf{H}^{old} + \mathbf{H}_{i,l(j,k)}^{update}, \tag{5.25}$$

where $\mathbf{H}^{update}_{i,l(j,k)}$ is a whole zero matrix except for the i^{th} and l^{th} row respectively being equal to $(\mathbf{H}^{new}_i - \mathbf{H}^{old}_i)$ and $(\mathbf{H}^{old}_i - \mathbf{H}^{new}_i)$.

To calculate the pseudo-inverse of $\mathbf{H}^{new}$ we take advantage of the same idea as in the previous subsection to be able to use Campbell and Meyer solution for updated matrix pseudo-inverse.

As in the previous case $\mathbf{H}^{update}_{i,l(j,k)}$ can be decomposed as the product of column and row vectors of $\mathbf{u}$ and $\mathbf{v}$ which can be denoted as

$$\mathbf{u} = \begin{bmatrix} u_m \end{bmatrix} \quad u_m = \begin{cases} 1 & m = i \\ \text{-1} & m = l\,, \\ 0 & else \end{cases} \tag{5.26}$$

$$\mathbf{v} = \mathbf{H}^{new}_i - \mathbf{H}^{old}_i. \tag{5.27}$$

Therefore, all the updates of replacing samples from cluster j to cluster k and vice versa can be performed in a single iteration of Campbell and Meyer solution utilizing the following parameters.

$$\mathbf{u} = \begin{bmatrix} u_m \end{bmatrix} \quad u_m = \begin{cases} 1 & m^{th} \text{ sample moved from cluster } j \text{ to cluster } k \\ \text{-1} & m^{th} \text{ sample moved from cluster } k \text{ to cluster } j\,, \\ 0 & \text{else} \end{cases} \tag{5.28}$$

$$\mathbf{v} = \mathbf{H}^{new}_i - \mathbf{H}^{old}_i. \tag{5.29}$$

Again as in the previous case, $\mathbf{H}^{update}_{total}$ consisting of all the updates can be represented as

$$\mathbf{H}_{total}^{update} = \sum_{j=1}^{nc} \sum_{k=1}^{nc} \mathbf{H}_{(\mathbf{j},\mathbf{k})}^{update}. \tag{5.30}$$

Thus, considering the proposed method for decomposition of the update matrices, the entire update procedure can be completed in $\mathbf{O}(nc^2)$ number of iterations which is significantly cost effective comparing to the case where the model is fully retrained.

5.5 Example Story

An example of system operation is presented in Fig. 5.3. In this example, the same data set as in Fig. 5.2 is used. However, in the initial setup of these experiments, it is assumed that there are more number of samples in cluster 2 comparing to cluster 1. This is considered as a prior knowledge injected to the system by the human expert. Under this assumption, some of the samples which are originally from cluster 1 are wrongly labeled as being in cluster 2. In this case, the operation of system in finding the mislabeled samples and modifying them is presented in Figs. 5.3a-5.3f. In addition, Figs. 5.3h-5.3l present the system operation in discovering the new classes of input.

Fig. 5.3a indicates the initial labeling of the data points according to the available information from the expert. As it can be seen, a number of samples which are similar to cluster 1 samples are labeled as being in cluster 2. As discussed in earlier chapters, the clustering is started by shuffling the data points and assigning the samples to different clusters in a random manner as in Fig. 5.3b. In the next phase, the algorithm keeps swapping the samples between different clusters in order to achieve a lower training error. The outcome of this phase can be seen in Fig. 5.3c which represents the current clustering of the trained model. After the initial training phase, the clustering decisions of the system must be evaluated by the EE. Fig. 5.3d, demonstrates the group of samples detected by EE as being in the wrong cluster. As a result, the cluster assignments of these samples are modified as in Fig. 5.3e and the system is updated in accordance with that. The clustering assignment of the updated system is presented in Fig. 5.3e. Considering the evaluation results of EE in Fig. 5.3g, although a number of samples are detected as being suspected to be in the other cluster, the correction proposals of the Fault Detector are ignored as these samples are at the border of the two clusters and can be mapped to any of them.

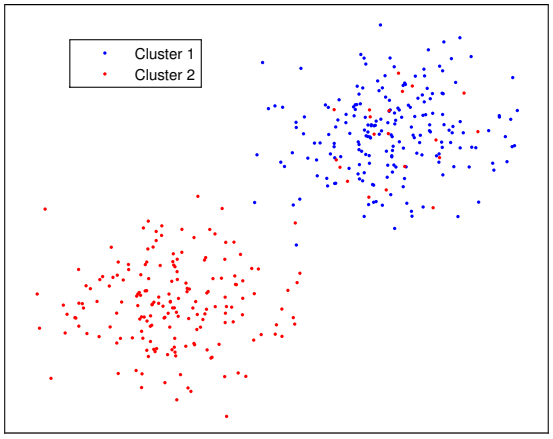
Now assume a case where a number of data points from a new class are served into the system as the input. The system first tries to find the best cluster for the newly arrived samples among the existing classes. Thus, the new samples are mapped into cluster 1 as in Fig. 5.3h. However, the Fault Detector component detects these samples as the candidates of being in a new cluster (Fig. 5.3i). Therefore, the model is updated by adding a new cluster and assigning the new sample to that cluster as in

Fig. 5.3j. The evaluation results of the system decisions provided by EE shows that a number of data points from cluster 1 and cluster 2 can also be assigned to the newly added cluster 3 (Fig. 5.3k). However, as in the previous case, these samples can be ignored as they are at the borders of the three clusters. The outcome of the update procedure is the new model containing three classes of data points as in Fig. 5.3l.

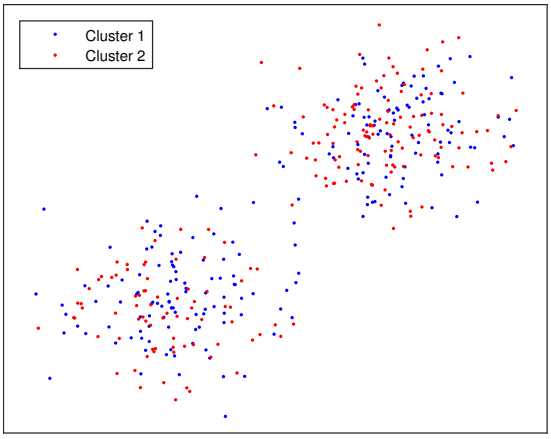
5.6 Summary

This chapter provided a detailed description of the intrusion detection system proposed in this research. The proposed IDS consists of three main components of Clustering Manager, Evaluation Engine and Update Manager. The Clustering Manager is responsible for constructing the initial learning model of the system. The Evaluation Engine evaluates the clustering decisions of the system and provides correction proposals to the current decisions. These proposals may be offered upon detection of errors in the current clustering as well as observing new classes of data in the input. The Update Manager is responsible for updating the model according to new patterns and information provided by the human expert or the Evaluation Engine. This chapter also introduced two novel update approaches which assist Update Manager to speed up the update process.

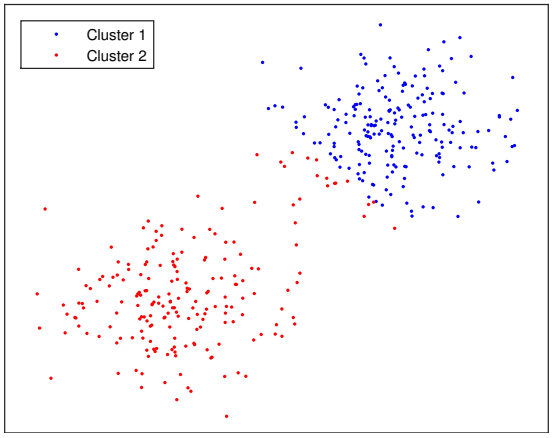
In the following chapter, the performance of the proposed intrusion detection system will be evaluated in different circumstances.



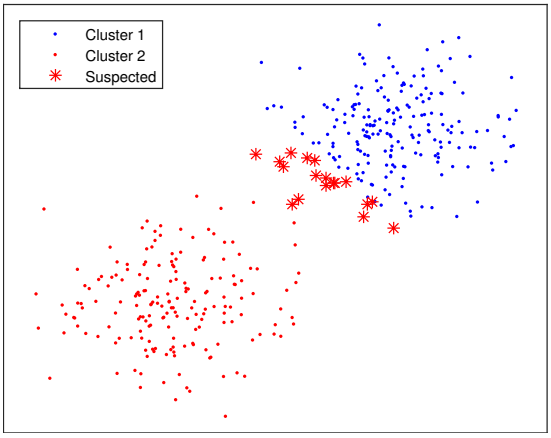
(a) The initial clustering is provided by the expert.



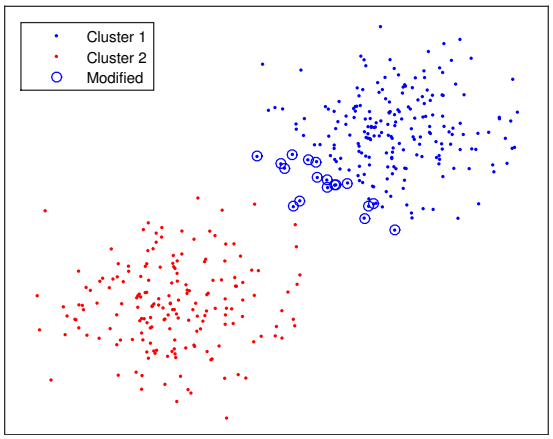
(b) Samples are assigned to different clusters randomly.



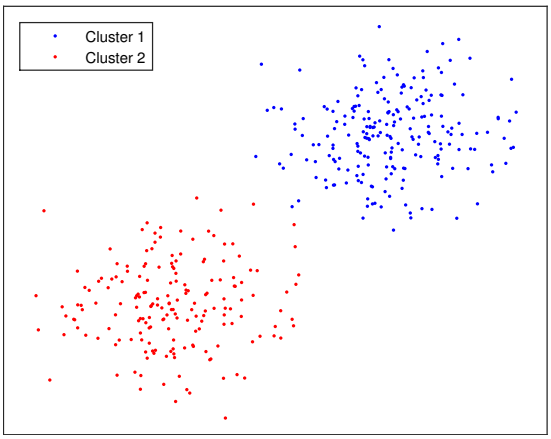
(c) The trained model finds the best cluster assignment for all the samples with the current cluster sizes set by the expert.



(d) A group of misclassified samples are detected by FD.

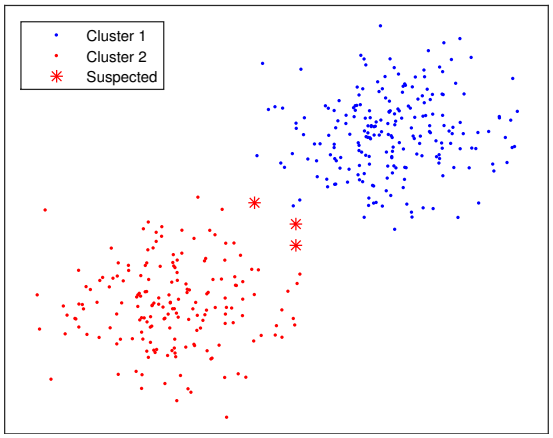


(e) The cluster assignments of misclassified samples are modified.

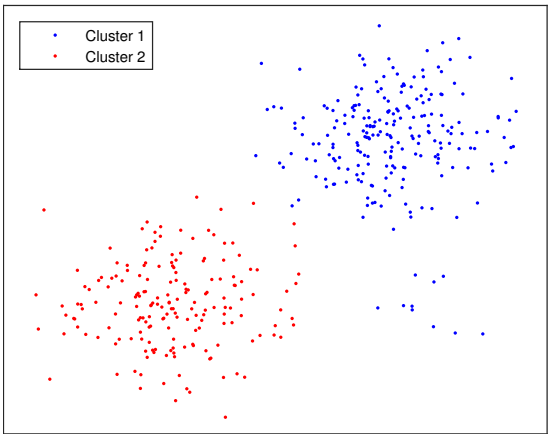


(f) The system is updated according to the modified cluster assignments.

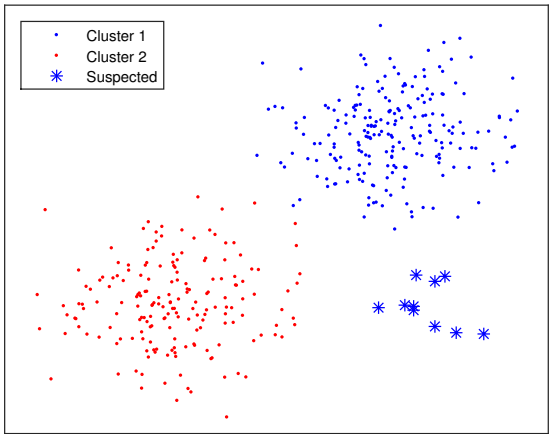
Figure 5.3: The Operation of The Proposed IDS



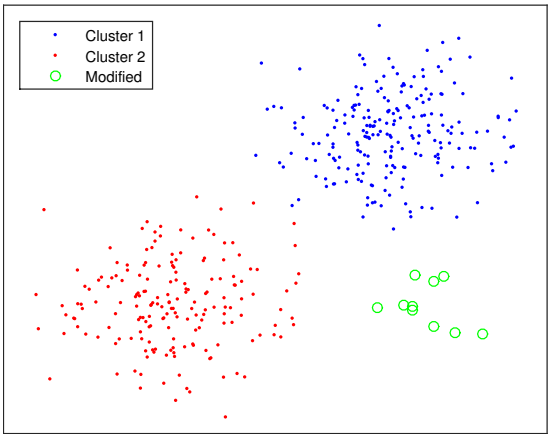
(g) A number of samples are detected as being suspected to be in the other cluster. These samples are marked to be ignored by the expert as they are at the border of the two clusters.



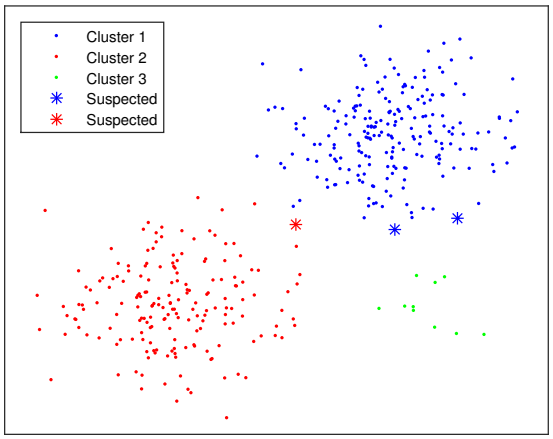
(h) New input data arrives into the system. The system classifies them in the best cluster among the existing ones.



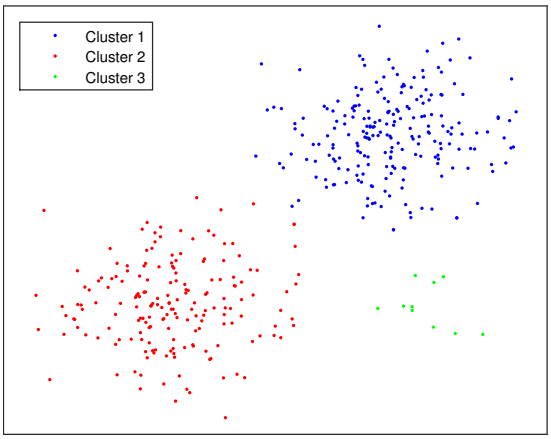
(i) The system detects the group of new samples as the candidates for being in a new cluster.



(j) The system is updated by adding a new cluster and assigning the new samples to that cluster.



(k) A number of samples from the initial data set are detected to be similar to the newly added cluster. These samples are ignored by the expert as they are at the borders of multiple clusters.



(l) The updated system contains three classes of data where the samples are correctly assigned to these classes.

Figure 5.2: The Operation of The Proposed IDS

Chapter 6

Results and Evaluation

This chapter provides an evaluation of the proposed solutions presented in Chapter 5. Firstly, the performance of the two novel update approaches of UM-Procedure-2 and UM-Procedure-3 are evaluated. In this evaluation, the computational time of the proposed methods is compared to the existing update method of full retraining. Secondly, the performance of the proposed intrusion detection system is evaluated using the criteria defined in earlier chapters. In addition, the suitability of the used test data for the purpose of this research is discussed through some examples.

6.1 Evaluation of Novel Approaches

This section provides an evaluation of the novel approaches proposed in Chapter 5. Firstly, the performance of the UM-Procedure-2 for adding a new cluster will be studied. Secondly, the approach of UM-Procedure-3 for modifying the cluster assignment for part of the samples is evaluated. The performance of these methods is studied with regards to the parameters of the problem context such as the size of the training data, the number of clusters and the number of mislabeled samples in the data set. The main focus of this evaluation is to provide a comparison of the proposed methods and the traditional method of full retraining in terms of computational time.

For each of the proposed approaches, we consider two types of evaluations which are defined as follows. The experimental results of both types are presented separately for each update procedure.

Type A: How is the computational time affected by the variation in the size of the training set? In this case, the computational time of the proposed method is measured in response to the change in the size of the training set while the number of mislabeled samples is adjusted to 400. The number of clusters is fixed and chosen from $\{5, 10\}$ and the size of the training set is varied between $[1000, 24000]$ number of data points.

Type B: How do the approaches react to the change in the number of mislabeled samples? In this case, the variation of the computational time of the update procedure in response to the change in the number of updated samples is studied. In the corresponding experiments, the number of clusters is set to 5 and the size of the training set is fixed and chosen from $\{12000, 20000\}$ while having the number of mislabeled samples from as 10, 20 and 30 % of the size of the training set.

All the experiments are done on a synthetic data set where the data points are randomly generated into different clusters and normalized with zero mean and unit variance. For simplicity, the number of data features is chosen to be 2 and the number of hidden neurons in the learning model is set as 150.

The experiments are repeated 50 times using Matlab code on a Windows machine with 8 GB of RAM and CPU of 2.30 GHz and the average results over those number of runs are presented.

6.1.1 UM-Procedure-2

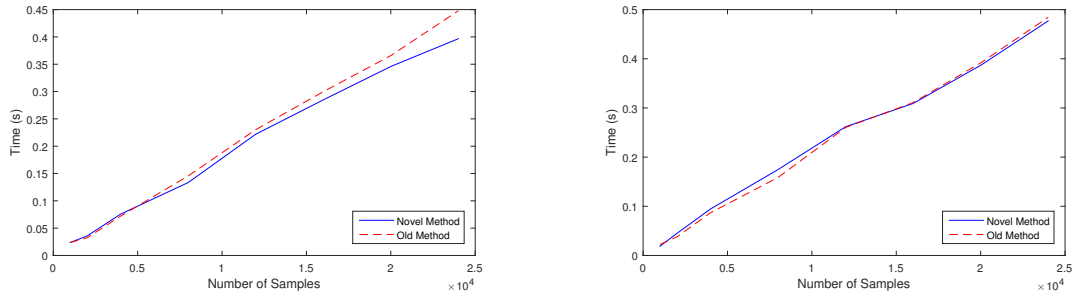
UM-Procedure-2 performs the task of updating the learning model in case of detecting new classes of traffic. As discussed in Chapter 5, in the proposed mechanism first a new cluster is added to the model, and then the suspected data samples are mapped into the new cluster. The main focus of the following experiments is to study the computational time of the proposed update procedure in comparison with the case where the model is fully retrained from the initial stage.

Type A

In the following experiments, we consider two problems of having the initial number of clusters as 5 and 10. In each of these problems, the data points are randomly generated and assigned into different clusters. After the initial training, each data point is mapped into its correct cluster. The data points of the new class are generated in a way to be of the same distance from all the existing clusters i.e. having equal chance of being mapped to any of them. As a result, in the update procedure the current cluster of the newly added data points can come from all the existing clusters rather than a single one.

Fig. 6.1a represents the results of the experiments on the $5 - cluster$ problem. It can be seen that as the size of the training set grows, the proposed method tends to perform slightly faster than the traditional method of full retraining. The same phenomenon can be observed in Fig. 6.1b which represents the results for the $10 - cluster$ problem although the difference is less noticeable.

Although we expected to see a significant improvement of the computational time using the proposed update procedure, the above experiments only offer a minor



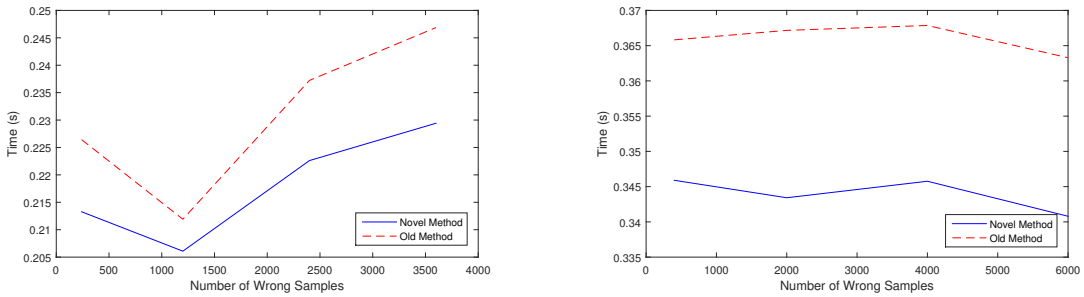
(a) 5-cluster (b) 10-cluster

Figure 6.1: Time comparison of the novel update method and old method of full retraining using ELM for new cluster case. The number of hidden neurons and the number of updated samples are set as 150 and 400 respectively.

improvement comparing to the old method as the number of samples in the training set exceeds a specific data point.

Type B

Supposing to have the same problem of 5 clusters as before, the effect of the change in the number of newly added samples on the computational time of the proposed method is studied in the following experiments. In the following experiments, we consider two cases of having the size of training set as 12000 and 20000.



(a) 12000 Samples (b) 20000 Samples

Figure 6.2: Reaction of novel update method in response to changing the number of updated samples in comparison with the old method of full retraining using ELM for new cluster case. The number of hidden neurons and the number of clusters are set as 150 and 5 respectively.

As it can be seen in Fig. 6.2a, as the number of mislabeled samples increases, the computational time of the update procedure changes in a similar way for both the proposed method and the full retraining method. This fact can also be observed in Fig. 6.2b which represents the results of the similar experiments on a larger data

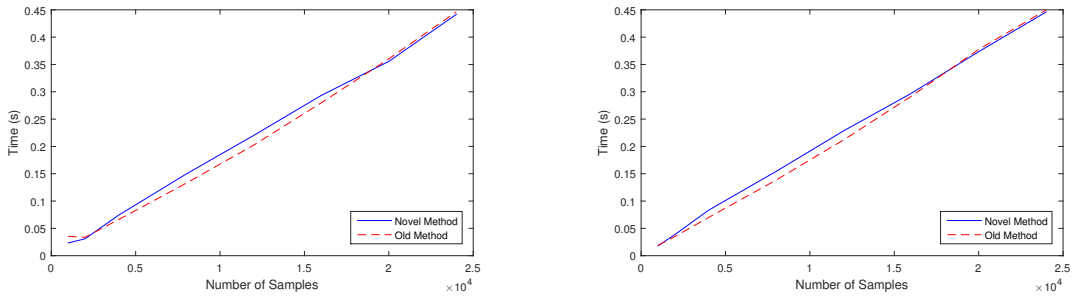
set. Considering these results, it can be concluded that the varying the number of mislabeled samples does not have a major impact on the computational time of the update procedure.

6.1.2 UM-Procedure-3

As discussed in Chapter 5, the Update Manager uses UM-Procedure-3 for modifying the cluster assignment for a group of samples and updating the learning model according to that. The following experiments are performed in order to provide an evaluation of the proposed method in comparison with the method of full retraining in terms of computational time.

Type A

As in the previous subsection, we consider two problems of having 5 and 10 clusters. In each of the problems, the data points are generated randomly into the existing clusters. The data points are mapped into different clusters in a random manner. After the initial training, a number of samples which are suspected of being in the wrong cluster are chosen by the Fault Detector component. Among these samples, 400 data points are selected as the correct candidates for being updated in the following experiments.



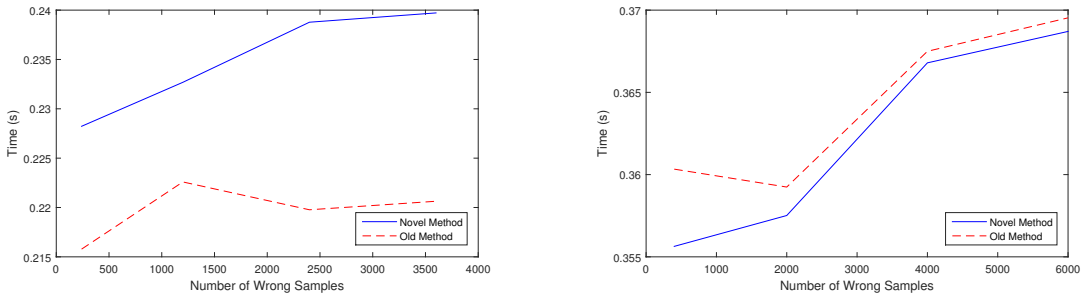
(a) 5-cluster (b) 10-cluster

Figure 6.3: Time comparison of the novel update method and old method of full retraining using ELM for cluster modification case. The number of hidden neurons and the number of modified samples are set as 150 and 400 respectively.

Fig. 6.3a and Fig. 6.3b represent the results for the $5-cluster$ and $10-cluster$ problems respectively. As it can be seen in the figures, in both of the problems the proposed method performs slightly faster than the traditional method as the number of the training samples increases. Considering the above plots, our proposed method provides a minor improvement to the computational time of update procedure where the size of the training set exceeds 16000 samples.

Type B

In order to study the effect of the number of mislabeled data points on the computational time of the update procedure, we consider a number of assumptions. Considering two cases of having the previously mentioned $5-cluster$ problem, we run the experiments on the training sets of 12000 and 20000 samples while changing the number of mislabeled samples.



(a) 12000 Samples (b) 20000 Samples

Figure 6.4: Reaction of novel update method in response to changing the number of modified samples in comparison with the old method of full retraining using ELM for cluster modification case. The number of hidden neurons and the number of clusters are set as 150 and 5 respectively.

As it can be seen in Fig. 6.4a, for the problem with 12000 samples the computational time of the proposed update mechanism grows slowly as the number of mislabeled samples increases. This follows a similar pattern as in the full retraining method. On the other hand, a similar sketch can be observed in Fig. 6.4b where the results of experiments on the larger data set is presented. In either case, the growth in the number of mislabeled samples does not affect the computational time in a considerable manner.

6.1.3 Discussion

As the results of type A experiments indicate, our novel method offers a minor improvement to the computational time of the model update comparing to the traditional update method. The same results can be observed for both the methods of UM-Procedure-2 and UM-Procedure-3 which start to perform faster with a slight difference as the number of samples exceeds a certain point. This is against our expectation to see a noticeable difference in the computational time of the proposed methods comparing to the full retraining of ELM.

The immediate outcome of the aforementioned observations is the excellence of ELM as a fast learning method which can be fully retrained using thousands

of samples with a low computational cost. In spite of this fact, it is necessary to mention that in real life cases, the IDSs may be trained using millions of samples rather than a few thousands. As a result, it is very likely that even a fast learning method like ELM cannot provide the required speed for fully retraining the system in such cases. This is where the two proposed update methods can be considered as alternative approaches for updating the model without retraining. In this thesis, due to the lack of computational resources, conducting the experiments with such large data sets was not possible. Otherwise, the difference between the computational time of the proposed methods and the traditional method could be more noticeable.

In addition, it is possible that improving the code used to perform the above experiments can make the difference between the novel and old approaches more evident.

6.2 Evaluation of the IDS

This section provides an evaluation of the proposed IDS according to the defined accuracy measures in the field of intrusion detection. The results of this evaluation is presented using the criteria introduced in Chapter 3 which are the detection rate and false positive rate of the system. The primary objective in this evaluation is to provide an assessment of the adaptivity of the proposed solution in different conditions. In other words, the main focus of this assessment is to study how the accuracy of the system decisions is affected when the learning model is updated according to the new patterns appearing in the input data.

In the following experiments, we consider two cases for updating the system. In the first case, the system is updated according to the new data from an existing traffic class. In this case, the system first categorizes the data using the current learning model and then updates the model according to the predicted labels. On the other hand, in the second case, the system is tested using the new data from a new traffic class. As in the previous case, the data is initially classified by the system and then the system is updated by adding a new cluster considering the correction proposals provided by the Fault Detector component. Each of these cases can be considered on the presence or absence of a human expert monitoring the system decisions.

In the following subsections, the experimental setup of the test environment is discussed in detail and the evaluation results of the proposed intrusion detection system are presented.

6.2.1 Experimental Setup

In this thesis, performance of the system is evaluated using NSL-KDD data set [60]. As discussed in Chapter 3, this data set consists of five main traffic categories of

normal, DoS, Probe, R2L and U2R. For the purpose of testing the proposed IDS for different scenarios, we consider to include the samples from four classes of normal, DoS, Probe and U2R in the training phase. It is necessary to note that in the original NSL-KDD data set the proportion of each class (and also malicious against normal samples) are in no way realistic when compared to real network data. However, in this thesis, we use the original proportions in order to be able to evaluate the proposed system in a similar way as the existing IDSs in the state of the art.

In order to choose the proper parameters of the learning model i.e. the number of hidden neurons, k-fold cross-validation method is used where k is adjusted to be 10. The results of model selection phase averaged over 50 runs of repeated experiments are presented in Fig. 6.5. The number of hidden neurons is varied between $[20, 400]$.

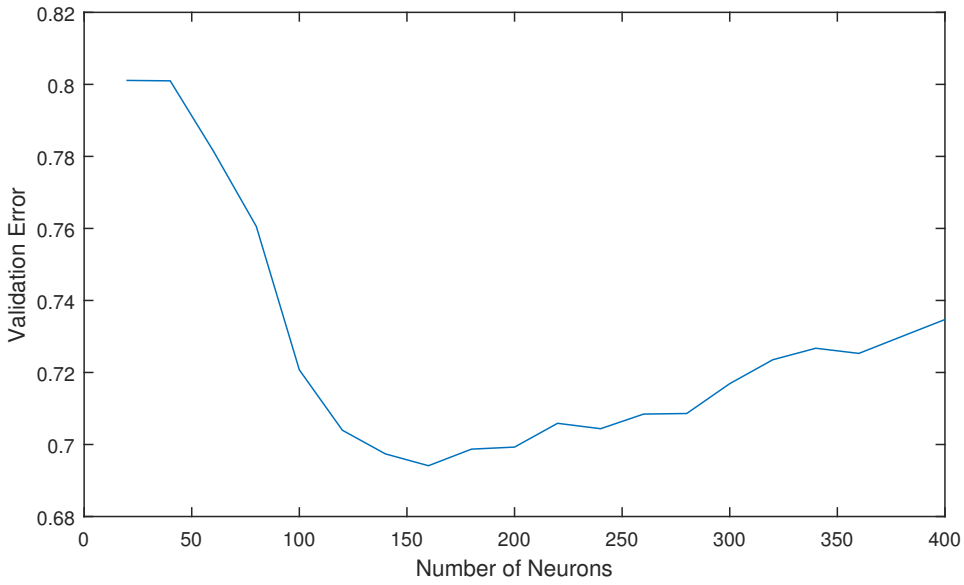


Figure 6.5: Results of model selection phase using k-fold validation method. The model achieves the smallest validation error where the number of hidden neurons equals 160.

As it can be seen in Fig. 6.5, the model achieves the lowest validation error when the number of neurons in the hidden layer equals 160. As a result, the same parameter is utilized for training the intrusion detection system with the defined training set.

After the initial training, the trained model must be validated by a human expert. The validation includes the task of verifying the correctness of the assigned labels and choosing the appropriate thresholds which will be used by Fault Detector to detect possible errors in future decisions. As a result, the validated model can be

used to classify the future test data. In this thesis, the verification of the assigned labels are performed based on the labels provided by NSL-KDD data set i.e. the current training samples are classified according to their original labels.

In order to evaluate the performance of the intrusion detection system we consider two types of testing as supervised and unsupervised. In supervised testing, it is assumed that the system update is performed under the supervision of a human security expert. In this case, the labeled data is served into the system as the input and the correction proposals offered by the Fault Detector can be accepted or rejected by the human expert. On the other hand, unsupervised testing is conducted in the absence of the expert's supervision while having unlabeled data as the input. As a result, all the corrections proposed by the Fault Detector are applied to the model.

In each of the aforementioned experiments, we consider two scenarios for testing the intrusion detection system as follows.

In the first scenario, the trained model is evaluated using part of NSL-KDD test set containing the samples of known attack types which are normal, DoS, Probe and U2R. In this case, we are able to assess the accuracy of system decisions where there is no novel attack in the input data.

The second scenario focuses on the case where the input data comes from an unknown attack class which is R2L. In this case, the model is updated using the instances of R2L class in NSL-KDD train set.

The experimental results of these scenarios for supervised and unsupervised mode are presented in the following subsections.

6.2.2 Unsupervised Testing

The unsupervised testing of the system is conducted in the absence of the human expert. As a result, in this evaluation all the corrections proposed by Fault Detector are assumed to be accepted and applied in system update. Assuming the model is already trained using the samples of four traffic classes of normal, DoS, Probe and U2R, the results of the system evaluation are presented for each scenario as follows.

Scenario 1: Known Traffic Type

In this scenario, the performance of the system in categorizing the data which comes from existing traffic types is evaluated. In addition, the adaptivity of the system in response to the injection of new data into the learning model is taken into consideration.

For this purpose, an unlabeled data set which contains a number of samples from three classes of normal, DoS and Probe is used as the input. As described in Chapter 5, the system tries to discover the best cluster for each sample in the data set. The results of the initial classification of data are presented in tables 6.1 and 6.2.

		Actual			
		Normal	DoS	Probe	U2R
Predicted	Normal	9406	2310	276	0
	DoS	11	2023	234	0
	Probe	275	3125	1911	0
	U2R	19	0	0	0

Table 6.1: Confusion Matrix of Multiclass Classification by The Initial System

		Actual	
		Normal	Malicious
Predicted	Normal	9406	2586
	Malicious	305	7293

Table 6.2: Confusion Matrix of Binary Classification by The Initial System

Considering the values in tables above, the total rates of detection (recall) and false alarms for all the malicious samples can be depicted as in table 6.3

Detection Rate	False Positive Rate
0.76	0.0314

Table 6.3: Evaluation Results of The Initial System

As the system is being tested in an unsupervised state, all the labeling proposals provided by the Fault Detector are assumed to be accepted. As a result, the system is updated by incorporating the newly arrived samples to the learning model while being labeled according to table 6.1.

In order to discover how the applied updates regarding the new samples affected the accuracy of system decisions, the above experiment is repeated by serving the same data set into the updated system as unlabeled input. The confusion matrices of the new experiment are presented in tables 6.4 and 6.5

According to the values presented in the above tables, the new rates of detection (recall) and false positives are calculated as in table 6.6

A comparison of the values in the confusion matrices of the initial and updated system presented in tables 6.1 and 6.4 shows that the new system provides a higher

		Actual			
		Normal	DoS	Probe	U2R
Predicted	Normal	9418	1733	55	0
	DoS	10	5623	206	0
	Probe	264	102	2160	0
	U2R	19	0	0	0

Table 6.4: Confusion Matrix of Multiclass Classification by The Updated System

		Actual	
		Normal	Malicious
Predicted	Normal	9406	1788
	Malicious	293	8091

Table 6.5: Confusion Matrix of Binary Classification by The Updated System

detection rate in predicting the correct labels for all the three classes of normal, DoS and Probe. This can be observed specially for DoS samples where the detection rate has a great improvement.

Moreover, the same improvement can be seen in the total rates of detection and false alarms of the system. Considering the results presented in tables 6.3 and 6.6, it can be seen that the updated system provides a higher detection rate comparing to the initial system. In addition, the updated model offers an improvement to the rate of false alarms as well. Considering these observation, it is obvious that the system was able to adapt to the new data coming from existing traffic classes.

Scenario 2: Unknown Traffic Type

The main focus of this scenario is to assess the effectiveness of the system in detecting novel attacks. In addition, the ability of the system for adapting to the new traffic class is evaluated. For this purpose, a data set containing the samples of the new traffic class of R2L is served into the system as unlabeled input. As discussed in Chapter 5, the system first tries to find the best cluster among the existing ones for the unlabeled samples. The results of the initial decisions of the system can be found in table. 6.7.

Among the classified samples, the Fault Detector reported that 81 % of the newly arrived samples have a very similar likelihood of being in any of the existing clusters. In addition, about 8 % of the sample have very low scores for being in any of the existing classes. Finally, a proportion of 89 % of all the new samples are considered to be the candidates of being in a new cluster.

As the system is being tested in an unsupervised mode, all the proposals offered by the Fault Detector are assumed to be approved and applied in system update. As

Detection Rate	False Positive Rate
0.84	0.0302

Table 6.6: Evaluation Results of The Updated System

Predicted Label	Ratio (%)
Normal	91
DoS	1.5
Probe	6
U2R	1.5

Table 6.7: The Results of The Initial Classification for R2L Samples

a result, the learning model is updated by adding a new cluster and mapping 89 % of the new samples into the newly added cluster.

In order to evaluate the effectiveness of the updates applied to the learning model regarding the new cluster, the accuracy of system decisions is taken into consideration by using an unlabeled data set containing both normal and R2L instances as the test set. The results of this evaluation can be seen in table 6.8

		Actual	
		Normal	R2L
Predicted	Normal	1776	557
	R2L	56	1966

Table 6.8: Confusion Matrix of Binary Classification by The Updated System

According to the numbers in table 6.8, the rates of detection and false alarms for R2L class can be calculated as below.

Considering the results presented above, it can be seen that the system is able to detect 89 % of the samples of the new attack type. In addition, as depicted in table 6.9, the system provides an acceptable detection rate of 77 % as well as a low false positive rate in categorizing the samples of the newly added class. This approves that the system was able to adapt to the updates regarding the new attack type.

6.2.3 Supervised Testing

As discussed earlier, the supervised testing of the system is performed in the presence of a human expert who can approve or reject the corrections proposed by Fault Detector and provide input to the learning model. Assuming the model is already trained using the samples of four traffic classes of normal, DoS, Probe and U2R, the results of the system performance are presented for each scenario as follows.

Detection Rate	False Positive Rate
0.77	0.0305

Table 6.9: Evaluation Results of The Updated System

Scenario 1: Known Traffic Type

As mentioned earlier, the main objective of this scenario is to evaluate the ability of the system in adapting to the new data coming from existing traffic classes. As in the unsupervised case, here we use a data set which contains samples from three classes of normal, DoS and Probe as the input to the system.

As this experiment is being conducted in the presence of a human expert, it is assumed that the new samples are already labeled according to their original labels provided by NSL-KDD data set. In this case, the Fault Detector tries to modify the current labels by offering corrections as in the unsupervised case presented in table 6.1. However, the correction proposals are rejected as the expert trusts NSL-KDD labeling. As a result, the system is updated using the new samples labeled as their original classes.

In order to assess the adaptivity of the system in response to the applied updates, the experiment is repeated by serving the same data set into the updated system as unlabeled input. The confusion matrices of the updated system decisions are presented in tables 6.10 and 6.11.

		Actual			
		Normal	DoS	Probe	U2R
Predicted	Normal	9412	2084	6	0
	DoS	11	3598	48	0
	Probe	269	1776	2367	0
	U2R	19	0	0	0

Table 6.10: Confusion Matrix of Multiclass Classification by The Updated System

		Actual	
		Normal	Malicious
Predicted	Normal	9406	2090
	Malicious	299	7789

Table 6.11: Confusion Matrix of Binary Classification by The Updated System

In this case, the rates of detection and false alarms of the updated system are depicted in table 6.12

Detection Rate	False Positive Rate
0.81	0.0308

Table 6.12: Evaluation Results of The Updated System

Considering the results of the repeated experiment, it can be seen that the system was able to update part of its decisions in categorizing the data set. This can be observed by comparing the values of the confusion matrix of the initial system in table 6.1 to the matrix presented in 6.10. For example, the number of samples correctly detected as DoS had a great improvement after the system update.

In addition, the total values of detection false positive rates shows the same improvement in system performance which verifies the effectiveness of system in adapting to the modifications added from an external source.

Scenario 2: Unknown Traffic Type

In this scenario, the adaptivity of the system in response to adding a new class of traffic is evaluated. For this purpose, a data set containing the instances of the new class of R2L is added to the system. This data set is the same as the set used in the unsupervised case. As a result, the Fault Detector generates the same results as in the unsupervised case i.e. only 89 % of the new samples are the candidates of being in a new cluster.

However, as this evaluation is being conducted in the presence of a security expert, it is assumed that the original labels of the samples provided by NSL-KDD data set are trusted. Therefore, the correction proposals of the Fault Detector can be assumed to be ignored. In this case, upon expert's request, first a new cluster is added to the system and secondly all the samples of the new data set are mapped into that cluster. As a result, the output of this phase is an updated model containing five classes of normal, DoS, Probe, U2R and newly added R2L.

In order to discover how the system adapts itself to the new updates, the same unlabeled data set as in the unsupervised case is served into the system for testing purpose. As mentioned before, this data set contains samples of both normal and R2L classes. The evaluation results of this phase can be seen in table 6.13.

		Actual	
		Normal	R2L
Predicted	Normal	1734	737
	R2L	98	1786

Table 6.13: Confusion Matrix of Binary Classification by The Updated System

Therefore, the rates of detection and false alarms for R2L class can be calculated

as in table 6.14.

Detection Rate	False Positive Rate
0.70	0.05

Table 6.14: Evaluation Results of The Updated System

Considering the results of the above experiment, it can be seen that after being updated according to the expert input regarding the new traffic class, the system provides an acceptable detection rate of 70 % and 5% of false positives in the test phase. This shows that the system is adapted to the changes regarding the new cluster.

6.2.4 Discussion

A comparison of the results obtained in supervised and unsupervised testings indicates that in many cases, the system achieves a better performance after being updated in the unsupervised mode. For instance, in the first scenario where the system is being tested using the new data from existing classes, the detection rate of the system is improved by 8 % when it is updated according to the corrections provided by Fault Detector. On the other hand, in the same scenario in supervised mode, the system offers only 3 % of improvement to the detection rate comparing to the detection rate of the initial system. A similar phenomenon can also be observed in the second test scenario where the system is tested using data from a new attack class. In this case, the system achieves a higher detection rate and lower rate of false alarms when the new samples are labeled according to the proposals of Fault Detector.

Plausible Explanation

As mentioned earlier, in supervised testing, the expert uses the labels of the original NSL-KDD data set in order to classify the data into different classes. Considering this fact and assuming that the original labels are trusted, we expected to achieve a higher performance in supervised testing rather than unsupervised. However, the obtained results do not meet this expectation. The above observations can be interpreted as if in supervised testing, the human expert tries to bias the learning model by injecting wrongly labeled data into the system. As a result, the system which tries to learn from the wrong input data achieves a lower detection rate comparing to the unsupervised case. In addition to these observations, in the initial training of the system, we faced a similar challenge. Although in the initial phase of training, the correct labels of the samples were used, lots of samples were detected as being in the wrong traffic class by the Fault Detector. This happened for many of the samples in all the existing classes.

From the aforementioned remarks, we may conjecture that the samples of NSL-KDD data set are not correctly labeled or the data set does not fit into our problem context for evaluating the performance of the proposed system.

Separability of Classes in the Data Set

In order to verify the validity of the decisions of Fault Detector in correcting the labels of the training data, a new clustering is conducted on the same training set using k-means clustering method. The new clustering provides similar results e.g. many of the samples of DoS class can already be labeled as normal and vice versa.

In addition, the similarity of samples in different classes is taken into consideration by measuring Euclidean distance between each pair of samples in the data set. The distance matrix of the data set is presented in Fig. 6.6 as a heatmap where the larger distances are mapped into warmer colors. In the constructed matrix, outlier distances over 60 are removed.

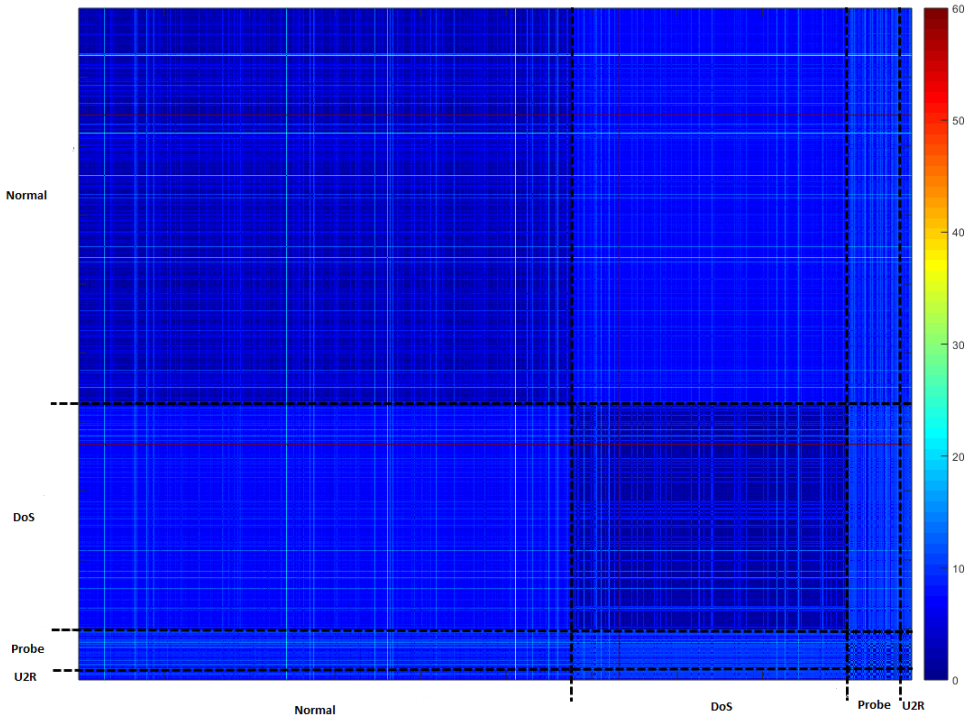


Figure 6.6: Heatmap of the pairwise Euclidean distance matrix of Samples in NSL-KDD data set. Outlier distances over 60 are removed

Although the resolution of Fig. 6.6 does not permit to see individual lines, the trends are visible enough. In spite of the fact that the distances inside a class are expected to be smaller comparing to the distances between samples of different classes, the above heatmap depicts slightly different results. As it can be seen in Fig. 6.6, although the majority of samples in normal and DoS classes have similar inner-class distances, there are still many samples which are the same distance from the samples of other classes as well. This can be derived by observing many straight lines of the same color sketched from one class to the other. In addition, many of the points in the inner-class squares have the same color as the inter-class points i.e. they have similar distances.

A similar phenomenon can also be observed in the points corresponding to the distances in classes of Probe and U2R. As it can be seen in the heatmap, these classes have warmer colors in their inner class points comparing to the two aforementioned classes. This means that it is even more difficult to distinguish the samples of these classes from others as they are very similar.

Concluding Remarks on NSL-KDD Data Set

The aforementioned remarks verify the obtained results that many samples of one class are as similar to each other as to the samples of other classes in NSL-KDD data set. It is necessary to note that any learning method relying on Euclidean distances to discriminate between classes will likely be affected by this fact. In addition, manual checking of individual records from the data set shows that samples belonging to different classes (DoS and normal, e.g.), have absolutely identical feature values. As a result, they have a similar likelihood to be classified into different classes and hence in our case, they can be detected as wrongly labeled by Fault Detector in the training phase. Considering this fact, it can be concluded that NSL-KDD data set does not properly fit into the requirements of the proposed IDS to be evaluated although it may represent the behavior of real network traffic.

In addition, in the context of this thesis, each traffic class in the data set is modeled by a single cluster. However, in real networks and possibly in NSL-KDD data set as well there may be different clusters for a single traffic type e.g. multiple clusters can be labeled as normal. Therefore, we may get slightly different results in the evaluation phase using this new scheme.

6.3 Summary

This chapter provided an evaluation of the proposed intrusion detection system. Firstly, the performance of the two novel update approaches of UM-Procedure-2 and UM-Procedure-3 were studies in comparison with the traditional method of full retraining in terms of computational time. The results indicated that the novel update approaches perform faster than the old method with a slight difference when

the number of samples in the training set exceeds a certain point. In addition, the experiments showed that the size of the updated samples does not have a major impact on the computational time of the update procedures.

Further, the evaluation results of the system performance in terms of accuracy measures of intrusion detection were presented in this chapter. The experimental results indicated that the system is capable of detecting novel attacks and being updated according to the new patterns of data available as the input. In addition, the system is able to provide proper correction proposals for modifying the labels of the input data. The obtained results indicated that the system achieves a higher detection rate as well as lower rate of false alarms while being updated in an unsupervised mode rather than the supervised mode. In other words, the system provides a higher accuracy when the human input data is modified according to the corrections proposed by the Fault Detector prior to system update. Considering such counter-intuitive observations, this chapter also provided a discussion about the NSL-KDD data set and its suitability the purpose of evaluating this work.

The following chapter concludes this thesis by providing a summary of key outcomes and possible improvements for future work.

Chapter 7

Summary and Conclusion

This thesis addressed the problem of adaptability in the field of intrusion detection by proposing a new intrusion detection system. The proposed IDS is an adaptive solution which provides the capability of detecting known and novel attacks as well as being updated according to the new input from human experts in a cost-effective manner.

Two novel approaches were proposed for updating the system according to the new available information with a low computational cost. Each of these approaches was introduced as a method which can update the learning model without being fully retrained. The first approach called "UM-Procedure-2" focuses on cases where a new class of traffic is detected in the input data. In such cases, this approach modifies the system by adding a new cluster to the existing ELM model and updates the output weights with regards to that. The second approach which is called "UM-Procedure-3" is used for cases where a human expert requests for modifying the cluster assignment of existing data. In these cases, the proposed approach updates the model without performing a full retraining. The experiments done as part of this thesis indicate that both of these methods perform slightly faster than the traditional method of full retraining when the size of the training set exceeds a certain point, although they do not offer a significant improvement to the computational time of the update process. In addition, the experimental results showed that the number of the updated samples in any of the proposed approaches does not have a major impact on the computational time of the update procedure.

Further, a scoring mechanism was presented in this thesis which enables the proposed IDS to detect novel traffic types. In addition, using such mechanism, the proposed system is capable of detecting human errors and providing correction proposals for labeling the data. Applying the scoring mechanism along with the two aforementioned update approaches assists the system to adapt to the new information available as input. In order to evaluate the performance of the proposed IDS in this thesis, the experiments were done in two modes of supervised and unsupervised. In supervised mode, the information arriving to the system is based on the knowledge of a human expert who ignores the correction proposals offered by the IDS. On the

other hand, in the unsupervised mode all the corrections proposed by the IDS are assumed to be approved and applied to the system.

The experiments conducted in this thesis using NSL-KDD data set indicated that the system was capable of detecting novel traffic types while offering acceptable rates of detection and false positives. In addition, using the update approach of "UM-Procedure-2", the proposed IDS was able to adapt itself in response to adding the new traffic type to the learning model.

On the other hand, the experimental results showed that the performance of the system was improved after being updated according to the newly arrived information. This could be inferred from the enhanced rates of detection and false positives achieved by the updated system. These results were obtained for both supervised and unsupervised cases. However, in all the conducted experiments, the system achieved a higher performance while being tested in an unsupervised mode.

Future Work

As mentioned earlier, the two proposed update approaches perform slightly faster than the traditional retraining method of ELM when the size of the training data exceeds a specific point. In spite of this fact, we expected to see a more noticeable improvement to the computational time of the update procedure using the proposed approaches. One possible area of future work can be to explore the variation of time difference of the novel and old approaches in cases where the number of samples grows toward the size of training data in real world applications of intrusion detection.

As discussed in previous chapter, the suitability of NSL-KDD data set for the purpose of evaluating this work cannot be assured. Considering this fact, a potential area for continuing this work in future is to evaluate the performance of the proposed system using real network traffic data. For this purpose, various traffic analysis tools such as Argus [1] can be used for extracting required features from network flows. In addition, in order to enhance the relevance of the extracted features to the problem domain the use of appropriate feature selection techniques could be taken into consideration.

Bibliography

[1] Argus-Auditing Network Activity. Online: http://www.qosient.com/argus/index.shtml. Accessed: 2015-10-30. 7

[2] DARPA Intrusion Detection Evaluation. Online: https://www.ll.mit.edu/ideval/data/1998data.html. Accessed: 2015-09-02. 3.3

[3] KDD Cup 1999 Data Set. Online: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: 2015-08-08. 3.3

[4] Estimating the Global Cost of Cybercrime. Technical report, McAfee, Centre for Strategic & International Studies, 2014. 1.1

[5] H. Alene. Graph Based Clustering for Anomaly Detection in IP Networks. Master's thesis, Aalto University, Department of Information and Computer Science, 2011. 2.2.2, 2.2.2

[6] N. B. Amor, S. Benferhat, and Z. Elouedi. Naive Bayes vs Decision Trees in Intrusion Detection Systems. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 420–424. ACM, 2004. 3.3

[7] S. Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical report, Technical report Chalmers University of Technology, Goteborg, Sweden, 2000. 2.2.1

[8] R. Bace and P. Mell. NIST Special Publication on Intrusion Detection Systems. Technical report, DTIC Document, 2001. 2.2.2

[9] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 2.3.2

[10] D. Brauckhoff. *Network Traffic Anomaly Detection and Evaluation*. PhD thesis, ETH Zurich, 2010. 3.1

[11] S. L. Campbell and C. D. Meyer. *Generalized Inverses of Linear Transformations*. SIAM, 2009. 5.4.2, 5.4.2

[12] J. Corsini. Analysis and Evaluation of Network Intrusion Detection Methods to Uncover Data Theft. Master's thesis, Napier University, 2009. 2.2.2, 3.2

[13] H. Debar, M. Dacier, and A. Wespi. Towards a Taxonomy of Intrusion Detection Systems. *Computer Networks*, 31(8):805–822, 1999. 2.2.1, 2.2.2

[14] D. E. Denning. An Intrusion-Detection Model. *Software Engineering, IEEE Transactions on*, 13(2):222–232, 1987. 1.1

[15] O. Depren, M. Topallar, E. Anarim, and M. K. Ciliz. An Intelligent Intrusion Detection System (IDS) for Anomaly and Misuse Detection in Computer Networks. *Expert Systems with Applications*, 29(4):713–722, 2005. 3.3

[16] C. Elkan. Results of the KDD'99 Classifier Learning. *ACM SIGKDD Explorations Newsletter*, 1(2):63–64, 2000. 3.3

[17] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers & Security*, 28(1):18–28, 2009. 2.3.1, 2.3.2

[18] G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012. 4.2

[19] J. Gomez and D. Dasgupta. Evolving Fuzzy Classifiers for Intrusion Detection. In *Proceedings of the 2002 IEEE Workshop on Information Assurance*, volume 6, pages 321–323. New York: IEEE Computer Press, 2002. 3.3

[20] F. Gong. Deciphering Detection Techniques: Part ii Anomaly-based Intrusion Detection. *White Paper, McAfee Security*, 2003. 2.2.1

[21] S. Haykin. *Neural Networks A Comprehensive Foundation*. Prentice Hall International, inc., 1999. 2.3.2

[22] R. Heady, G. F. Luger, A. Maccabe, and M. Servilla. The Architecture of a Network Level Intrusion Detection System. Technical Report CS90-20, Department of Computer Science, University of New Mexico, August 1990. 2.1

[23] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion Detection Using Sequences of System Calls. *Journal of Computer Security*, 6(3):151–180, 1998. 2.2.1

[24] G. B. Huang, L. Chen, and C. K. Siew. Universal Approximation Using Incremental Constructive Feedforward Networks with Random Hidden Nodes. *Neural Networks, IEEE Transactions on*, 17(4):879–892, 2006. 5.2

[25] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004. 4.1

[26] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme Learning Machine: Theory and Applications. *Neurocomputing*, 70(1):489–501, 2006. 4.1, 4.1

[27] K. Ingham and S. Forrest. A History and Survey of Network Firewalls. Technical report, University of New Mexico, 2002. 1.1

[28] S. A. Iqbal. Automatic Rule Tunning in Intrusion Detection Systems: Online and Offline. Master's thesis, University of New Brunswick, 2009. 2.2.2, 2.2.2

[29] B. Jain. Intrusion Prevention and Vulnerability Assessment in BCEFHP Intrusion Detection System. Master's thesis, Indian Institute of Technology, Kanpur, June 2005. 1.1, 2.2.1

[30] J. S. R. Jang. Self-learning Fuzzy Controllers based on Temporal Backpropagation. *Neural Networks, IEEE Transactions on*, 3(5):714–723, 1992. 2.3.2

[31] H. G. Kayacik, A. N. Zincir-Heywood, and M. I. Heywood. Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets. In *Proceedings of the Third Annual Conference on Privacy, Security and Trust*. Citeseer, 2005. 3.3

[32] R. Kohavi. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *International Joint Conference on Artificial Intelligence*, 1995. 5.2

[33] S. Kumar and E. H. Spafford. An Application of Pattern Matching in Intrusion Detection. Technical report, Department of Computer Science, Purdue University, 1994. 2.3

[34] P. Kumpulainen, M. Kylväjä, and K. Hätönen. Importance of Scaling in Unsupervised Distance-Based Anomaly Detection. In *Proceedings of IMEKO XIX World Congress. Fundamental and Applied Metrology, Lisbon, Portugal, September*, pages 6–11, 2009. 3.1

[35] K. Labib. Computer Security and Intrusion Detection. *Crossroads*, 11(1):2–2, 2004. 1.1

[36] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten Digit Recognition with a Back-propagation Network. In *Advances in neural information processing systems*. Citeseer, 1990. 2.3.2

[37] Y. LeCun, L. Bottou, G. B. Orr, and K-R. Müller. Efficient BackProp. In *Neural networks: Tricks of the trade*, pages 9–48. 2012. 4.2

[38] W. Lee and S. J. Stolfo. Data Mining Approaches for Intrusion Detection. In *Usenix Security*, 1998. 1.1

[39] W. Lee, S. J. Stolfo, and K. W. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120–132. IEEE, 1999. 2.2.1

[40] N. Y. Liang, G. B. Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *Neural Networks, IEEE Transactions on*, 17(6):1411–1423, 2006. 4.2, 4.2

[41] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer Networks*, 34(4):579–595, 2000. 3.2

[42] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. 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, pages 12–26. IEEE, 2000. 3.2

[43] J. McHugh. Testing Intrusion Detection Systems: A Critique of The 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000. 3.3

[44] Y. Miche, A. Akusok, D. Veganzones, K. M. Björk, E. Séverin, P. du Jardin, M. Termenon, and A. Lendasse. SOM-ELM-Self-Organized Clustering Using ELM. *Neurocomputing*, 165:238–254, 2015. 4.3, 5.2

[45] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse. OP-ELM: Optimally Pruned Extreme Learning Machine. *Neural Networks, IEEE Transactions on*, 21(1):158–162, 2010. 4.1, 5.2

[46] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, and A. Lendasse. TROP-ELM: A Double-regularized ELM Using LARS and Tikhonov Regularization. *Neurocomputing*, 74(16):2413–2421, 2011. 4.1

[47] J. J. Moré. The Levenberg-Marquardt Algorithm: Implementation and Theory. In *Numerical Analysis*, pages 105–116. Springer, 1978. 4.1

[48] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network Intrusion Detection. *Network, IEEE*, 8(3):26–41, 1994. 2.2.1

[49] D. Newman, J. Snyder, and R. Thayer. Crying wolf: False alarms hide attacks. *Network World*, 24, 2002. 2.3

[50] A. Niemelä. Traffic Analysis for Intrusion Detection in Telecommunications Networks. Master's thesis, Tampere University of Technology, Finland, 2011. 2.2.2

[51] A. Patcha and J. M. Park. An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. *Computer Networks*, 51:3448–3470, 2007. 1.1, 2.2.1, 2.3, 2.3.1, 3.1, 3.2

[52] J. Platt. A Resource-Allocating Network for Function Interpolation. *Neural Computation*, 3:213–225, 1991. 4.2

[53] P. E. Proctor. Marketscope for Network Behavior Analysis. *Gartner Research Report G00144385*, 144385, 2006. 2.3

[54] A. Qayyum, M.H. Islam, and M. Jamil. Taxonomy of Statistical Based Anomaly Detection Techniques for Intrusion Detection. In *Emerging Technologies, 2005. Proceedings of the IEEE Symposium on*, pages 270–276, 2005. 2.3.1

[55] C. R. Rao and S. K. Mitra. *Generalized Inverse of Matrices and its Applications*, volume 7. Wiley New York, 1971. 4.1

[56] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *LISA*, volume 99, pages 229–238, 1999. 1.1

[57] M. Sabhnani and G. Serpen. Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context. In *MLMTA*, pages 209–215, 2003. 3.3

[58] Sophos. The A-Z of Computer and Data Security Threats. Online: https://www.sophos.com/en-us/medialibrary/PDFs/other/sophosthreatsaurusaz.pdf?la=en.pdf. Accessed: 2015-10-05. 3.3

[59] Aurobindo Sundaram. An Introduction to Intrusion Detection. *Crossroads*, 2(4):3–7, 1996. 2.2.1

[60] M. Tavallaee, E. Bagheri, W. Lu, and A. Ghorbani. A Detailed Analysis of the KDD CUP 99 Data Set. In *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pages 53–58, 2009. 3.3, 6.2.1

[61] M. van Heeswijk, Y. Miche, E. Oja, and A. Lendasse. GPU-accelerated and Parallelized ELM Ensembles for Large-scale Regression. *Neurocomputing*, 74(16):2430–2437, 2011. 4.1

[62] Y. Zhang and L. Wenke. Intrusion Detection in Wireless Ad-hoc Networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 275–283, 2000. 1.1

Appendix A

Feature Description of The NSL-KDD Data Set

Feature Name	Description
duration	length of the connection
protocol_type	type of the protocol, e.g. tcp, udp, etc.
service	network service on the destination, e.g., http, telnet, etc.
flag	normal or error status of the connection
src_bytes	number of bytes from source to destination
dst_bytes	number of bytes from destination to source
land	1 if connection is from/to the same host/port; 0 otherwise
wrong_fragment	number of wrong fragments
urgent	number of urgent packets
hot	number of hot indicators
num_failed_logins	number of failed login attempts
logged_in	1 if successfully logged in; 0 otherwise
num_compromised	number of compromised conditions
root_shell	1 if root shell is obtained; 0 otherwise
su_attempted	1 if "su root" command attempted; 0 otherwise
num_root	number of "root" accesses
num_file_creations	number of file creation operations
num_shells	number of shell prompts
num_access_files	number of operations on access control files
num_outbound_cmds	number of outbound commands in an ftp session
is_host_login	1 if the login belongs to the "hot" list; 0 otherwise
is_guest_login	1 if the login is a "guest"login; 0 otherwise
count	number of connections to the same host as the current connection in the past two seconds

srv_count	number of connections to the same service as the current connection in the past two seconds
serror_rate	% of connections that have “SYN” errors
srv_serror_rate	% of connections that have “SYN” errors
rerror_rate	% of connections that have “REJ” errors
srv_rerror_rate	% of connections that have “REJ” errors
same_srv_rate	% of connections to the same service
diff_srv_rate	% of connections to different services
srv_diff_host_rate	% of connections to different hosts
dst_host_count	number of connections between destination and host
dst_host_srv_count	number of connections between destination and host with the same service
dst_host_same_srv_rate	% of connections between destination and host with the same service
dst_host_diff_srv_rate	% of connections between destination and host with a different service
dst_host_same_src_port_rate	% of connections with ”SYN” rate between destination and host with the same source port
dst_host_srv_diff_host_rate	% of connections between destination and host with different host and service
dst_host_serror_rate	% of connections with ”SYN” rate between destination and hos
dst_host_srv_serror_rate	% of connections with ”SYN” rate between destination and host with the same service
dst_host_rerror_rate	% of connections with ”REJ” rate between destination and host
dst_host_srv_rerror_rate	% of connections with ”REJ” rate between destination and host with same service