

---

# A Comparison Of Machine Learning Algorithms to Detect Network Intrusions

---

Jack Anderson -  
40208539

Submitted in partial fulfilment of  
the requirements of Edinburgh Napier University  
for the Degree of  
BEng (Hons) Software Engineering

School of Computing

28th March 2018

### **Authorship Declaration**

I, Jack Anderson, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

*Signed:*

*Date:*

*Matriculation no:*

**Data Protection Declaration**

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

A handwritten signature in black ink, consisting of a stylized 'J' and 'A' followed by a horizontal line.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

## **Abstract**

## Contents

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Background . . . . .	10
1.2	Research Questions . . . . .	10
1.3	Aims and Objectives . . . . .	11
1.4	Scope and Limitations . . . . .	12
1.4.1	Deliverables . . . . .	12
1.4.2	Boundaries . . . . .	12
1.4.3	Constraints . . . . .	12
1.5	Structure of this Dissertation . . . . .	12
<b>2</b>	<b>Literature Review</b>	<b>13</b>
2.1	Search Strategy . . . . .	13
2.2	Databases . . . . .	14
2.3	Paper Selection . . . . .	14
2.4	Network Intrusion Detection . . . . .	15
2.5	Machine Learning Algorithms . . . . .	16
2.5.1	k-Nearest Neighbours . . . . .	16
2.5.2	Artificial Neural Network . . . . .	16
2.5.3	Self Organising Map . . . . .	18
2.5.4	Recurrent Neural Network . . . . .	19
2.5.5	Negative Selection . . . . .	20
2.6	Support Vector Machine . . . . .	22
2.7	Research Contribution . . . . .	22
2.8	Dataset . . . . .	23
2.9	Literature Conclusion . . . . .	24
<b>3</b>	<b>Existing Software</b>	<b>25</b>
3.1	WEKA . . . . .	25
3.2	SPLUNK . . . . .	25
<b>4</b>	<b>Software</b>	<b>25</b>
4.1	Overall Description . . . . .	25
4.1.1	Product Perspective . . . . .	25
4.1.2	Product Functions . . . . .	25
4.1.3	User Characteristics . . . . .	25
4.1.4	Operating Environment . . . . .	25
4.2	Specific Requirements . . . . .	26
4.2.1	User Interface . . . . .	26
4.2.2	Functional Requirements . . . . .	28

4.2.3	Non-Functional Requirements . . . . .	33
4.3	Testing . . . . .	34
<b>5</b>	<b>Methodology</b>	<b>38</b>
5.1	Implementation . . . . .	38
5.1.1	PyQt . . . . .	38
5.1.2	Scikit-Learn . . . . .	38
5.1.3	Pandas . . . . .	39
5.1.4	Matplotlib . . . . .	39
5.1.5	Software . . . . .	39
5.1.5.1	MainWindow . . . . .	40
5.1.5.2	DatasetWindow . . . . .	40
5.1.5.3	UIObject . . . . .	40
5.1.5.4	Classifier . . . . .	40
5.1.5.5	QClfSelector . . . . .	40
5.1.5.6	QBarChart . . . . .	41
5.1.5.7	ErrorMessage . . . . .	41
5.2	Datasets . . . . .	41
5.2.1	NSL-KDD . . . . .	41
5.2.2	UNSW-NB15 . . . . .	42
5.3	Data Pre-processing . . . . .	44
5.4	Classifier Configurations . . . . .	45
5.5	Single Stage Classification . . . . .	46
5.6	Two Stage Classification . . . . .	47
5.7	Data Collection . . . . .	47
5.8	Metrics . . . . .	48
<b>6</b>	<b>Results</b>	<b>49</b>
6.1	NSL-KDD . . . . .	49
6.2	UNSW-NB15 . . . . .	52
<b>7</b>	<b>Evaulation</b>	<b>54</b>
7.1	Software . . . . .	54
7.2	Single vs Two-stage Classifiers Performance . . . . .	54
<b>8</b>	<b>Conclusion</b>	<b>55</b>
8.1	Research Questions . . . . .	55
8.2	Has the Project met it's Aims and Objectives? . . . . .	55
8.3	Reflection . . . . .	55
8.4	Learning Outcomes . . . . .	55
8.5	Further Research . . . . .	55

<b>Appendices</b>	<b>60</b>
<b>A Project Overview</b>	<b>60</b>
A.A Project Timeline . . . . .	63
<b>B Second Formal Review Output</b>	<b>64</b>
<b>C Diary Sheets</b>	<b>66</b>
<b>D MoSCoW Analysis</b>	<b>70</b>
<b>E Class Diagram</b>	<b>71</b>
<b>F Use Case Diagram</b>	<b>72</b>
<b>G NSL-KDD Features</b>	<b>73</b>
<b>H UNSW-NB15 Features</b>	<b>75</b>
<b>I NSL-KDD Results</b>	<b>78</b>
I.1 NSL-KDD Classification Data . . . . .	78
I.2 NSL-KDD Precision . . . . .	80
I.3 NSL-KDD Recall . . . . .	81
I.4 NSL-KDD f1-Score . . . . .	82
<b>J UNSW-NB15 Results</b>	<b>83</b>
J.1 UNSW-NB15 Classification Data . . . . .	83
J.2 UNSW-NB15 Precision . . . . .	87
J.3 UNSW-NB15 Recall . . . . .	88
J.4 UNSW-NB15 f1-Score . . . . .	89

**List of Tables**

1	Test Case Descriptions . . . . .	34
2	NSL-KDD Dataset Attack Samples. . . . .	41
3	NSL-KDD Dataset Categories Samples. . . . .	42
4	UNSW-NB15 Dataset Attack Samples. . . . .	43
5	Classifier Parameters . . . . .	45
7	MoSCoW Analysis of Requirements . . . . .	70
8	NSL-KDD Features . . . . .	73
9	UNSW-NB15 Features . . . . .	75
10	NSL-KDD Classification Data . . . . .	78
11	NSL-KDD Classifiers Precision Table . . . . .	80
12	NSL-KDD Classifiers Recall Table . . . . .	81
13	NSL-KDD Classifiers f1-Score Table . . . . .	82
14	UNSW-NB15 Classification Data . . . . .	83
15	UNSW-NB15 Classifiers Precision Table . . . . .	87
16	UNSW-NB15 Classifier Recall Table . . . . .	88
17	UNSW-NB15 Classifiers f1-Score Table . . . . .	89



**List of Figures**

1	Artificial Neural Network Layers . . . . .	17
2	Recurrent Neural Network . . . . .	20
3	Main Window Result Tab Wireframe . . . . .	26
4	Main Window Graph Tab Wireframe . . . . .	27
5	Dataset Window Wireframe . . . . .	28
6	One-hot Encoding Example . . . . .	44
7	Feature Scaling Formula . . . . .	45
8	k-Fold Cross Validation . . . . .	48
9	Precision Formula . . . . .	48
10	Recall Formula . . . . .	49
11	f1-Score Formula . . . . .	49
12	Classifiers Precision Graph . . . . .	50
13	Classifiers Recall Graph . . . . .	51
14	Classifiers f1-Score Graph . . . . .	51
15	Classifiers Precision Graph . . . . .	52
16	UNSW-NB15 Classifiers Recall Graph . . . . .	53
17	UNSW-NB15 Classifiers f1-Score Graph . . . . .	53
18	Project Timeline Gantt Chart . . . . .	63
19	UML Class Diagram . . . . .	71
20	UML Use Case Diagram . . . . .	72

## **Acknowledgements**

Insert acknowledgements here

## 1 Introduction

### 1.1 Background

For any major company or business security is of the utmost concern, with a study finding that in the UK in 2016 an estimated 46% of all businesses experienced a cyber security breach or attack K, Jayesh N S, Tom R and Mark B, 2017. These breaches are particularly dangerous as even if a network is compromised a single time a company can have its entire database destroyed, or customer data leaked, leading to legal repercussions. When it comes to preventing these intrusions firewalls alone are insufficient for anything but the most rudimentary of attacks and so a Network Intrusion Detection System (NIDS) is employed to further bolster the security of the network. Traditional NIDS are placed strategically on a network to monitor all incoming traffic. It analyses the passing traffic and then compares it to a large library of known attacks and if it matches will flag the traffic. While these systems do provide some degree of protection they are unable to detect novel attacks or zero-day vulnerabilities and so some other method of identifying suspicious traffic is required. Introducing machine learning to a NIDS is one way of attempting to solve this problem first proposed by Denning, 1987. In using machine learning to detect network intrusions the system can be trained to recognise patterns of intrusive behaviour, allowing it to detect attacks which it may not have seen before but have characteristics of similar attacks. Machine learning also allows systems to be easily retrained to accommodate for new data on attacks as it emerges. There are two main categories of NIDS: misuse detection, and anomaly detection; both of which have their own advantages and disadvantages. This literary review aims to discuss the different kinds of network intrusion detection systems, the algorithms that these systems employ, and the gap in papers which directly compare the performance of single stage and two-stage classifiers in the domain of network intrusion detection.

### 1.2 Research Questions

For this honours thesis there are a number of research questions which have been collated, and an attempt made to answer them. These questions in which I am interested in answering are the following:

- What is the rate of accurate detection and classification of network intrusions by single stage machine learning classification methods?
- What is the rate of accurate detection and classification of network intrusions by two stage machine learning classification methods?

- Which method of classification i.e. single stage or two stage, is more accurate and by what amount?
- Which configuration of algorithms in the two stage classifier produces the most accurate results?

In this context, accuracy is defined as a high number of true positives and true negatives, and a low number of false positives and false negatives when classifying network intrusions.

First research will be completed in order to gain an understanding of the history and current state of machine learning for network intrusion detection, through reading relevant research papers and articles. Next the individual algorithms and methods which go into detecting network intrusions will be researched and understood. This knowledge will then be put into developing a piece of software capable of running these algorithms and extracting metrics which will be used to answer these research questions.

### **1.3 Aims and Objectives**

The aim of this project is to create a piece of software which is capable of running both single and two stage classification algorithm arrangements on a number of datasets and displaying the results in a clear manner. In doing so this will assist in answering the research questions put forward.

These research questions, specifically whether or not two stage classification provides more accurate results than single stage classification will be answered by meeting the following objectives:

- Perform a review of existing literature.
- Perform a review of existing software.
- Select appropriate machine learning classification algorithms.
- Select appropriate datasets for use in network intrusion detection.
- Implementation of software to assist in comparison of classification algorithms.
- Evaluation of software with regards to specification and other software.
- Evaluation of classification algorithm performance.

## **1.4 Scope and Limitations**

### **1.4.1 Deliverables**

The list of deliverables for this project are the following:

- A review of literature on related topics and a report detailing the findings of such reports.
- A review of existing software which performs a similar role to the proposed software.
- A software requirement specification.
- A description of all testing which will be carried out.
- A report detailing the results of each classification algorithm and configuration of two stage classification algorithms
- An evaluation of the implemented software with regards to how well it meets the proposed specification, and how it compares to existing software.

### **1.4.2 Boundaries**

There exists a large number of different classification algorithms and methods to perform network intrusion detection, therefore this project must focus on a few specifically. This project will focus on performing misuse detection rather than anomaly detection, and will limit the classification algorithms used to: k-Nearest Neighbour, Multi-layer Perceptron, and a Support Vector Machine. This investigation will also only be assessing the performance of misuse detection, and not anomaly detection.

### **1.4.3 Constraints**

The largest constraint facing this project is that of time. There is a large amount of literature to be reviewed, and also a considerable amount of results which must be collected and results drawn from in a short amount of time.

## **1.5 Structure of this Dissertation**

This dissertation can be divided into three main parts: part one is a review of the relevant literature and related works carried out in the field of network intrusion detection, as well as any existing software; Part two details the planning, design, implementation and testing of the software which is to be created to aid in the investigation; finally, part three details the results of

the experiments carried, an evaluation of the created software, an analysis of gathered results, and the conclusions drawn from the project as a whole.

Part one (Sections 2-3) involves a review of the existing literature within the field of machine learning for network intrusion detection and examines results of related works and individual machine learning algorithms. A review will also be done of existing software to see what features they offer in what aspects they might be improved.

Part two (Sections 4-5) describes the detailed software requirements, specific functionality which should be implemented, user interface designs, and a listing of what testing should take place. The methodology for the project is also detailed in this part which includes a description of the technologies and libraries used, the datasets used any the processing performed upon them, and finally how data will be collected and what metrics will be drawn from the results.

Part three (Sections 5-8) is a listing of results gathered in experiments carried out and an evaluation of the final software product against its initial design and any existing software. An analysis of the results gathered will then take place and conclusions drawn, as well as a review of research questions, learning outcomes, and suggestions for future work. also has an evaluation of the overall project how well it met its aims and suggestions for future work.

## 2 Literature Review

### 2.1 Search Strategy

While searching for relevant papers on the subject of network intrusion detection, a number of key terms were identified which could be used to find papers within the field. The main search term which was used was '*Network Intrusion Detection*' which was used to find papers which were generally related to the topic. A number of supplemental search terms were identified and used in conjunction with the main search term when attempting to find papers which used a specific technology within network intrusion detection. These included: 'Nearest Neighbor', 'k-NN', 'Neural', 'Self Organising Map', 'SOM', 'Recurrent', 'Hybrid', 'Stage', and 'Ensemble'. Using a combination of the main search term and these additions search terms allowed the discovery of papers directly relevant to the project. If two terms were required to be included within the search then the **AND** operator would be used to ensure that both were matched. Similarly if there were multiple terms for the same technique or technology, i.e. 'Nearest Neighbor' and 'k-NN' then the **OR** operator would be used, to reduce the number of individual searches

which were required to be carried out. When papers were found and deemed relevant to the project, a review of citations included within those papers would also be carried out, which allows the finding of papers which are directly related to the subject but which may have been missed by the search terms specified. A separate search was also performed when researching a relevant dataset by searching for the title of each dataset, such as: 'KDD Cup', 'NSL-KDD', 'DARPA', etc.

## **2.2 Databases**

To find appropriate research papers, Google Scholar and the Edinburgh Napier University Library Search were used to locate articles and other databases which could also be searched. The databases from which the papers were retrieved were:

- ScienceDirect
- IEEE Xplore
- ACM Digital Library
- Society for Industrial and Applied Mathematics

## **2.3 Paper Selection**

At first a large range of papers were collected by title and held on to that could be relevant to the subject matter. Then the abstract and conclusion were read through to determine whether or not the paper was relevant to the research questions. Once the pool of papers had been reduced to a manageable size the entire paper was read through to attempt to make links between the content of the paper and the work which would be carried out and how they could assist in answering the research questions.

During this process, papers would be selected for use if they met all of the following criteria:

- Peer reviewed.
- If the paper is older than five years and has at least 50 citations.
- The paper is directly relevant to the research questions.

Papers would be rejected if they met any of the following criteria:

- Too broad.

- Cannot be directly applied to the research questions.
- The paper has less than fifty citations and is older than five years.
- A newer more relevant paper on topic was found.

## 2.4 Network Intrusion Detection

The field of network intrusion detection was conceived by Denning, 1987, a paper in which the author describes a model for a "real-time intrusion-detection expert system" capable of discerning between normal and abnormal network activity. NIDS can be broadly categorized as performing either: misuse detection or anomaly detection. Misuse detection systems are first trained using some kind of learning algorithm on a set of labelled data where each entry is defined as being either 'normal' or 'intrusive'. Using this the system can create a sophisticated model of attacks, with a very high accuracy in detecting previously observed attacks and variations of such attacks. However, these types of systems perform poorly when faced with novel attacks being unable to accurately detect and classify them.

Anomaly detections systems are trained on a set of data in which every entry is an example of 'normal' network traffic. Training the system in such a way means that *"behavior is flagged as a potential intrusion if it deviates significantly from expected behavior"* Javitz, Valdes, Breen and Patton, 1994. This allows such systems to easily detect novel attacks which have not been observed before. Although these systems perform well at detecting novel attacks they also have a much higher false positive rate than misuse detection systems. The reason for this is that *"previously unseen (yet legitimate) system behaviors are also recognized as anomalies, and hence flagged as potential intrusions"* Lazarevic, Ertoz, Kumar, Ozgur and Srivastava, 2003. Anomaly detection systems also suffer from a so called *"semantic gap"* where anomalies can be detected yet there is no further information on what type of attack has been performed Sommer and Paxson, 2010.

In the beginning of network intrusion detection research methods were proposed for an expert system NIDS Ilgun, Kemmerer and Porras, 1995, in which took knowledge from experts within the network security field and encoded them into rules with which the system could use to check traffic with to determine if it was intrusive. Then in W. Lee, Stolfo and Mok, 1999 a framework was proposed for the use of data mining for building NIDS. This was the first to employ machine learning in its approach to detecting intrusions, a method which today has been explored extensively. In using this approach to network intrusion detection the NIDS can detect attacks which have not been seen before, and can also be retrained quickly on data



of new which have appeared, whereas an expert system would need updated in an expensive and slow process.

More recently hybrid approaches to network intrusion detection have been proposed in papers such as; Powers and He, 2008 and, Panda, Abraham and Patra, 2012, which employ a two stage classification approach. This two stage classification functions by first classifying network traffic as either 'normal' or 'intrusive'. Once intrusive traffic has been identified it is fed into a second stage which then determines the type of intrusion. Performing the detection and classification in this manner is beneficial as two separate classifiers can be used, allowing each to specialise, producing a higher rate of intrusion detection and a lower false positive rate.

The research questions which this dissertation aims to answer are surrounding the comparison of these hybrid approaches in comparison to single stage monolithic detectors, which is discussed in more detail in section 2.7.

## **2.5 Machine Learning Algorithms**

This section will detail the Machine learning algorithms which have been chosen to be evaluated and compared in this thesis.

### **2.5.1 k-Nearest Neighbours**

The k-nearest neighbours (k-NN) is one of the most simple machine learning classification and regression algorithms. k-NN is a lazy algorithm meaning that it uses the training dataset directly and therefore does not require any training time. The algorithm functions by first taking a training set of labelled vectors. A vector which is to be classified is then input, and the algorithm calculates the distance from the input vector to each other point in the training set and selects the k nearest points from the training set. For each of these k nearest points their classifications are totalled and the classification which makes up the most of these neighbours is then output as the class of the input vector. In both Liao and Vemuri, 2002, and in Hautamaki, Karkkainen and Franti, 2004, it has been demonstrated that k-NN based network intrusion detection methods can produce good results. The k-NN algorithm has been selected for this thesis mainly because of its simplicity and according to Jain, Duin and Mao, 2000 it *"can be conveniently used as a benchmark for all the other classifiers since it appears to always provide a reasonable classification performance in most applications."*

### **2.5.2 Artificial Neural Network**

According to Caudill, 1987 an artificial neural network (ANN) is *"a computing system made up of a number of simple, highly interconnected processing*

*elements, which process information by their dynamic state response to external inputs*". The ANN is inspired by and aims to recreate the operation of a biological neural network, found within the brains of living animals. The neural network consists of several layers of neurons or nodes, which are interconnected by axioms each of which has its own associated weight which is altered over the course of the training of the network. These weights are adjusted through a method called backpropagation. A network will typically consist of three layers; the input layer, the hidden layer(s), and the output layer, shown in Figure 1.

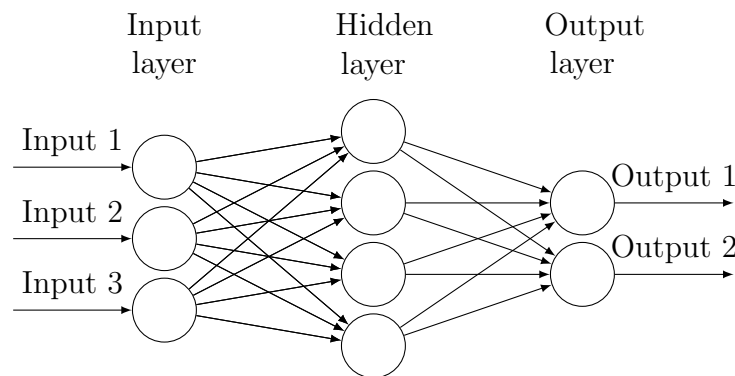


Figure 1: Artificial Neural Network Layers

The input layer is where the neural network receives the data which it is to process. This input layer is then connected via weighted axioms to the hidden layer(s). The hidden layer may consist of many layers and is responsible for the recognition of patterns within the input data. These hidden layers are then connected to the output layer, which gives the result or classification of the input data. The ANN is trained by feeding in a labelled dataset containing inputs and expected output(s). Each data entry which is fed into the network slightly alters the weights of all axioms within the network depending on the inputs and outputs, which when performed a large number of times allows the network to recognise patterns within input data and attempt to predict the correct output. Neural networks were first proposed for use within the field of network detection intrusion by Herve Debar, Becker and Siboni, 1992, who found them to be a promising method. ANNs are especially good for use with data classification problems and in turn with network intrusion detection as they can recognise complex patterns within datasets with a large number of features, such as network traffic Sung, 1998. Some early methods of network intrusion detection were also unable to

make any sort of classification beyond a binary one, i.e. normal or intrusive, whereas ANNs can classify data into any number of categories allowing for a greater amount of information about an attack to be gained from the detection Moradi and Zulkernine, 2004.

However, while neural networks excel at classifying information and recognising complex patterns within data, they do suffer from a semantic gap. Garcia-Teodoro, Diaz-Verdejo, Maciá-Fernández and Vázquez, 2009 explain that *"they do not provide a descriptive model that explains why a particular detection decision has been taken."* meaning that the network is essentially a black box, and without the details of the operation of the network it is difficult to determine what features of a connection identify it as intrusive and how effectively it may perform on other tasks and in other areas. The remainder of this section will discuss several different kinds of ANN which have been successfully applied to network detection intrusion.

This basic form of neural network is also called a feed forward neural network (FFNN) and is the most simple class of neural network available in which the connections between neurons do not form a cycle unlike in a RNN, i.e. information only travels forward from input neurons to output neurons. These kinds of neural networks have been employed in a number of research papers to great effect. Mukkamala, Janoski and Sung, 2002 found that with the use of a neural network intrusions could be accurately detected more than 99% of the time. The same result was also found in Z. Zhang, Li, Manikopoulos, Jorgenson and Ucles, 2001 with a 99% detection rate. Linda, Vollmer and Manic, 2009, S. C. Lee and Heinbuch, 2001, and Moradi and Zulkernine, 2004 also all achieved the same results as other studies. Using these results it is clear that using a FFNN is a viable method detecting network intrusions, as well as being simple to implement within a limit time frame. S. C. Lee and Heinbuch, 2001 also found that these networks are extremely adept at detecting novel attacks which is a desirable trait in a NIDS. It is for these reasons that a FFNN has been chosen to be implemented in this thesis.

### 2.5.3 Self Organising Map

The self organising map (SOM) is a type of ANN first proposed in the paper Kohonen, 1982 and is used to map high dimensional data into lower dimensions. The SOM is unlike the other neural networks which are to be examined as it can learn to classify data without supervision. This means that whereas a regular neural network will require an input vector and an output vector, the SOM will learn to classify data without the need for an output vector. This lack of need for supervised training can be extremely

useful in the context of network intrusion detection as described by Rhodes, Mahaffey and Cannady, 2000 *"This approach is particularly powerful because the self-organizing map never needs to be told what intrusive behaviour looks like. By learning to characterize normal behaviour, it implicitly prepares itself to detect any aberrant network activity."* both in anomaly detection and especially in misuse detection as normal behaviour can be continuously fed into it without the need for examples of intrusive behaviour.

The SOM has been implemented and tested within a number of research papers with great success. In Powers and He, 2008 and Depren, Topalilar, Anarim and Ciliz, 2005 it was found that the use of an SOM showed favourable false positive rates and attack classification results over other intrusion detection methods using the KDD 1999 Cup dataset. Similarly, in Lichodziejewski, Zincir-Heywood and Heywood, 2002 the researchers found that a SOM produced good results on the DARPA 1998 dataset with a low false positive rate and a correct classification rate of more than 95%. Kayacik, Zincir-Heywood and Heywood, 2003 too found SOM to be an effective method of classification with results much similar to the previous studies performed on the KDD 1999 Cup dataset however with a much higher rate of false positives.

These papers demonstrate that the SOM is a viable method for the detection of network intrusions giving high rate of correct classifications and low false positive rate, however this algorithm will not be implemented as part of this thesis. This algorithm has been chosen to not be implemented due to an unfamiliarity on the part of the researcher and also time constraints of the project associated with this unfamiliarity regarding the research and implementation it, with other methods being less time consuming to implement.

#### **2.5.4 Recurrent Neural Network**

A recurrent neural network (RNN) is a class of artificial neural network where the connections within the network connect back to previous neurons in order to form a cycle of nodes as shown in Figure 2. Having the neural network be structured in this manner allows it to process a sequence of input vectors allowing it to process data which relies on other vectors for context. RNNs are typically applied to problems such as handwriting recognition or speech recognition, however they can be particularly effective in network intrusion detection in recognising sequences of network traffic which alone are not suspicious but in a specific order can then be classified as intrusive behaviour.

The first paper to advocate for the use of RNNs within the field of network intrusion detection was Hervé Debar and Dorizzi, 1992 in which it was found

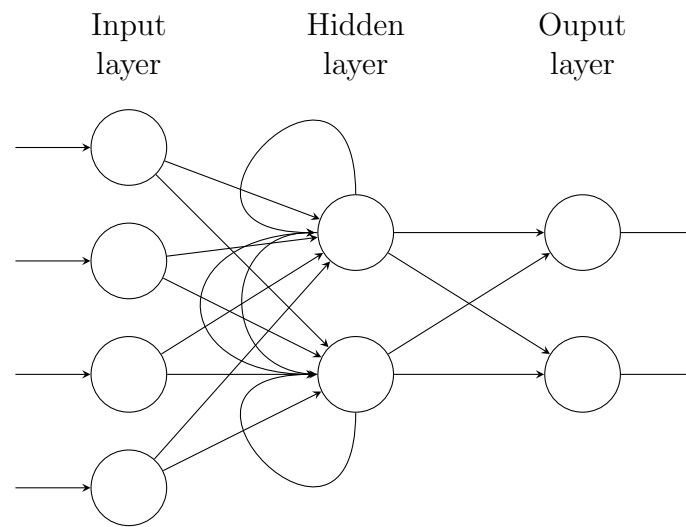


Figure 2: Recurrent Neural Network

RNNs to be a promising method of detecting intrusions. A later paper Ryan, Lin and Miikkulainen, 1998 re-enforces this statement finding this method to be very effective when employed on real worlds data. Two papers using RNNs in order to detect network intrusions, Tong, Wang and Yu, 2009 and Ghosh and Schwartzbard, 1999 found this approach to have high rate of correct detection of 90%-100%, however with a slightly higher false positive rate than other methods of intrusion detection such as the SOM.

There are however some issues in using RNNs for network intrusion detection. In order to train the network for use in detection, sequenced data such as timed network traffic is required. In this thesis, the data set used is the KDD 1999 Cup dataset in which none of the data is sequenced therefore making it unsuitable for testing with a RNN, which is the main reason why this type of neural network will not be implemented.

### 2.5.5 Negative Selection

Negative selection is an algorithm in the field of artificial immune systems. Artificial immune systems are a class of algorithm which seek to imitate the immune system of a living creature. De Castro and Timmis, 2002 describes artificial immune systems as *"adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are*

*applied to problem solving.*". There are several algorithms related artificial immune systems such as: Clonal Selection, Immune Network, and Dendritic Cell, however the algorithm which will be examined is the negative selection algorithm.

The negative selection algorithm takes its inspiration from the generation of T cells in the immune system. These T cells are capable of distinguishing between the bodies own cells and foreign cells, and are created pseudo-randomly. These cells then undergo a censoring process called negative selection in which cells that recognise the body's own cells are destroyed leaving only ones which detect foreign cells. This process is the basis for the negative selection algorithm where detectors (T cells) are generated by some method, randomly or otherwise, and detectors which detect self are deleted Forrest, Perelson, Allen and Cherukuri, 1994.

In the negative selection algorithm detectors must be represented by some means. In papers such as Dasgupta and Forrest, 1996 and Kim and Bentley, 2001b detectors are represented by fixed length bit string where each portion of the string is a binary representation of some feature of the input data. In order to determine whether or not these detectors match self, the detector is checked against each entry in the training set and if both of the strings contain the same stretch of  $r$  uninterrupted bits then the strings are said to match Powers and He, 2008. The detectors which do not match self are then used to detect intrusive behaviour by attempting to match incoming data represented as bit strings. If the incoming bit string matches any of the detectors it is flagged as intrusive. The use of binary strings as detectors however presents similar problems as found in neural networks. When a string is flagged as being intrusive it is difficult to extract semantic meaning from the detectors, i.e. it is apparent that the string is intrusive but information about what features of the connection made it intrusive are not readily available González and Dasgupta, 2003. Kim and Bentley, 2001a also found that the use of bit strings for detectors becomes infeasible when using a dataset which contains a lot of different features for each entry, such as in the KDD 1999 Cup dataset. A solution to this problem lies in the use of real-valued detectors in the place of bit string detectors. Through the use of real values when defining detectors this allows a great amount of domain knowledge to be extracted from subsequent results. For example when a detector is activated and a connection flagged as being intrusive the detector can be examined and the exact features and values of the connection which triggered the detector can be viewed and higher level information can be extracted. In comparison it can be extremely difficult and time consuming to try and extract semantic knowledge from a bit string as it can be unclear what triggered the detector. In order to evolve detectors, there are two main methods of doing so: through

random generation, or by means of a genetic algorithm. In the random generation method, a set of self is required, in the form of a dataset of non intrusive behaviour only. Detectors are then generated at random and tested to see whether or not they match the set of self using some method, i.e. r-contiguous bits, r-chunk matching, hamming distance etc. If the generated detector matches self it is discarded and a new detector is generated until the detector does not match self and it is stored. Generating detectors by means of a genetic algorithm is described by Powers and He, 2008. In this paper a population of detectors is initialised where each feature of the detector has a 50% chance to be initialised as blank and ignored in detection. Leaving fields blank in this way increases the generality of the detector, allowing it to cover as large an area of non-self as possible and to detect more kinds of attacks. The fields which were not left blank during the initialisation are then randomly assigned a value from a list of allowed values for that feature. When evolving the population two parents are bred using a uniform crossover to produce a single child which then has a small probability of mutation. This mutation takes a field from the child and replaces its value with another value which is randomly chosen from the list of allowed values. This newly created child will then replace the parent which it is most similar to if it has a greater fitness. At the end of the evolution process detectors which match self are removed by comparing each detector with the self-set. Generating detectors in such a manner is advantageous as it allows detectors to be created quickly when compared to a purely random generation process.

This method of intrusion detection through the use of the negative selection algorithm has been implemented in a number of papers to great effect. Powers and He, 2008 shows that the negative selection algorithm evolved using a genetic algorithm can achieve detection rates of up to 98%. Dasgupta and González, 2002 found similar results similarly using a genetic algorithm with a self detection rate of 96% and a non-self detection rate of up to 86%. This demonstrates that negative selection is an effective and viable method of network intrusion detection.

## **2.6 Support Vector Machine**

TODO

## **2.7 Research Contribution**

During this dissertation there are a number of goals which are aimed to be met and questions answered in order to ultimately contribute to the research of network intrusion detection in some way. The main question which is to be answered is what is the difference in performance between a single stage classifier and a two stage classifier in the context of network intrusion detection, and to determine what the configuration of machine learning clas-

sifiers discussed within this project produces the highest accuracy network intrusion detection and classification of intrusive network connections, where accuracy is defined as a high number of true positives and a low number of false positives.

Within the field of network intrusion detection there have been numerous papers describing methods of detecting intrusion using single stage classifiers such as citations wherein network traffic data is fed into the classifier and the single classifier will determine whether or not the connection is intrusive, and if it is intrusive what class of intrusion it is. Hybrid intrusion detection systems function similarly however the main difference is that there are two stages to the classification process. The first stage will be some form of anomaly detector which reads in network data and determines whether or not a connection is intrusive or not without specifying the type of attack that the intrusive connection is. Then the first stage of the classifier will send all data deemed as intrusive to the second stage of the classifier which will then determine the type of attack. Building an intrusion detection system in a manner such as this allows each stage of the system to specialise in detecting different aspects of a connection and therefore producing a higher detection and classification accuracy. While there have been several papers on hybrid classifiers in the domain of network intrusion detection such as Powers and He, 2008, Panda et al., 2012, and J. Zhang and Zulkernine, 2006, there is an apparent distinct lack of papers which directly compare the performance of a single stage monolithic classifier to that of a two-stage stage classifier and investigate different arrangements of machine learning algorithms to evaluate their performance. This gap in research papers is what this dissertation aims to fill.

The project will achieve that goal in the following manner: A number of machine learning algorithms will be chosen to be implemented, i.e. k-nearest neighbors, artificial neural networks, and negative selection. These algorithms will be housed by a peice of software which will be written to run them, obtain metrics and display information on the performance of each algorithm, in the form of tables and graphs, etc. Once this information has been extracted from each algorithm, an analysis and evauluation will be performed on the accuracy of each method, and a conclusion drawn.

## **2.8 Dataset**

TODO: Change to NSLKDD and UNSWNB15

The dataset which was to perform the evaluation of the network intrusion detection methods is the KDD Cup 1999 dataset. This dataset comes from the Third International Knowledge Discovery and Data Mining Tools Competition and is based on data captured in the DARPA'98 IDS evaluation



program Lippmann et al., 2000, Tavallae, Bagheri, Lu and Ghorbani, 2009.

This dataset has been widely criticized by researchers and experts due to several factors such as; having a large number of redundant records *"which cause the learning algorithm to be biased towards the most frequent records, thus prevent it from recognizing rare attack records"* Panda et al., 2012 , and in Vasudevan, Harshini and Selvakumar, 2011, for being outdated meaning that it does not account for new developments in network attacks. This is not of concern during this project as the dataset is used purely as a proof of concept and will not be deployed into an actual network intrusion detection role. There are other datasets which have been proposed for use such as the NSL-KDD which is a revised version of the KDD99 dataset which has been shown to have eliminated many of the faults of the KDD99 dataset while also achieving better performance in the training of network intrusion detection systems Dhanabal and Shantharajah, 2015. While all of the criticisms against the KDD Cup 1999 dataset are well founded and a legitimate cause for concern it has been chosen as it is the single most used dataset and researched dataset in the entire network intrusion detection field. With this large amount of research having been performed on this dataset the results from such studies can be used as confirmation that the algorithms which are to be implemented are performing correctly, and also can be used to make an accurate comparisons of results. The KDD Cup 99 dataset also has a version released which has all redundant entries removed while maintaining the correct ratio of non-intrusive entries to attacks, allowing for much faster training of algorithms and collection of data, which is a priority as there is a time constraint which must be followed during this project.

## **2.9 Literature Conclusion**

The research field of Network Intrusion Detection is one of upmost important due to the widespread prevalence of network security breaches and attacks. There have been a large number of studies performed in this area, with each resulting in varying levels of success. The algorithms chosen for this investigation have been proven to be effective at detecting network intrusions, and studies have also shown that through the use of hybrid network intrusion detection systems which make use of multiple classification stages and methods an increase in accurate classification rate. However, machine learning techniques have not yet proven accurate enough in their classification and detection rates to be deployed in a commercial setting, without substantial supervision. There is also a distinct lack within the research in making direct comparisons between different configurations of multiple stage classifiers, and also in making direct comparisons to their single stage counterparts. As a result of this, the conclusion has been reached that conducting

an investigation into this area may prove beneficial for future studies, and may provide an insight into what configurations of multiple stage classifiers are effective in the context of network intrusion detection.

### **3 Existing Software**

#### **3.1 WEKA**

#### **3.2 SPLUNK**

### **4 Software**

This section of the report will provide a high level description of the functionality and requirements of the software as well as describing the role of the user which will interact with the system. Detailed descriptions of functionality will also be included, and a list of hardware and software constraints, dependencies and assumptions made about the system and users. A detailed testing description will then be given outlining any and all unit testing and functional testing which will take place. Finally an evaluation will take place assessing how well the produced software meets its initial requirements and design.

#### **4.1 Overall Description**

##### **4.1.1 Product Perspective**

##### **4.1.2 Product Functions**

##### **4.1.3 User Characteristics**

This software is primarily focused towards researchers and students interested in comparing machine learning classification algorithms with a single stage against those with multiple stages. The software may be used by them in order to quickly make comparisons between different configurations of classifiers and to assess the effectiveness of these configurations. They will be able to enter their own datasets and classifiers and receive results in a standard format, and to view these results in a graph.

##### **4.1.4 Operating Environment**

In order to run the software the following requirements must be met:

- Python 3.6.4+ with the following packages:
  - NumPy 1.14.0+
  - Pandas 0.22.0+
  - Pandas-ml 0.5.0+
  - Scikit-learn 0.19.1+

- Scipy 1.0.0
  - Matplotlib 2.1.1+
  - PyQt 5.9.1+
- One of the following operating systems:
  - Windows 7 (x86) or greater
  - macOS 10.11 or greater
  - Linux with X11
- A system capable of running one of the above operating systems.

## 4.2 Specific Requirements

### 4.2.1 User Interface

TODO: Write about UI

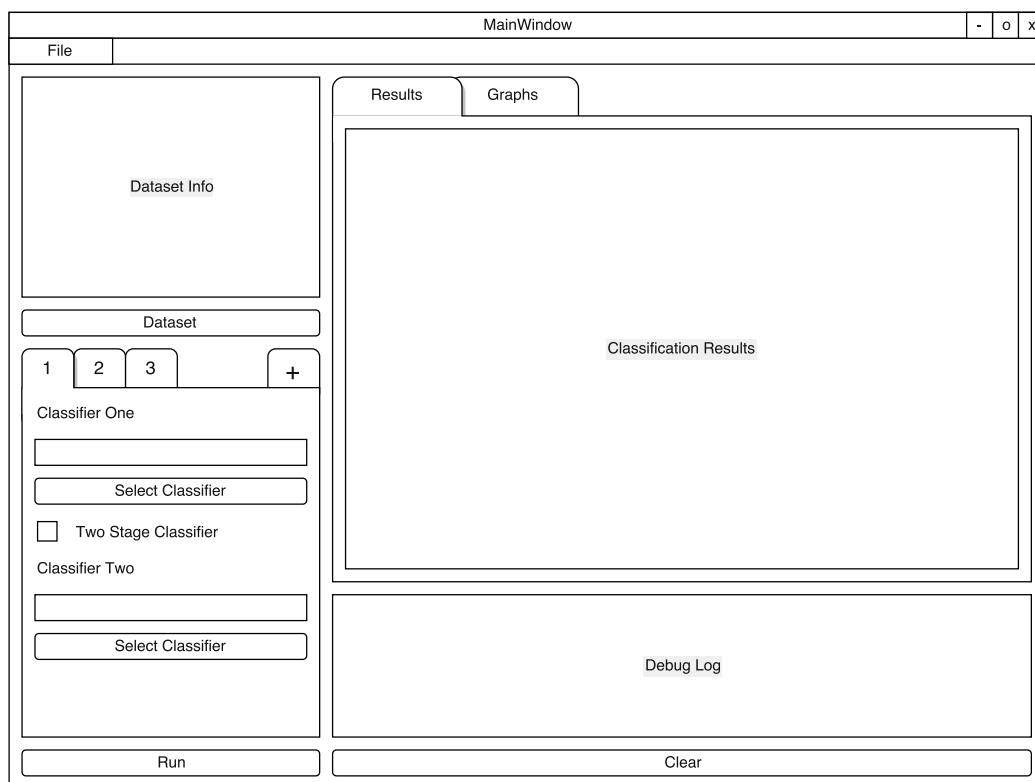


Figure 3: Main Window Result Tab Wireframe

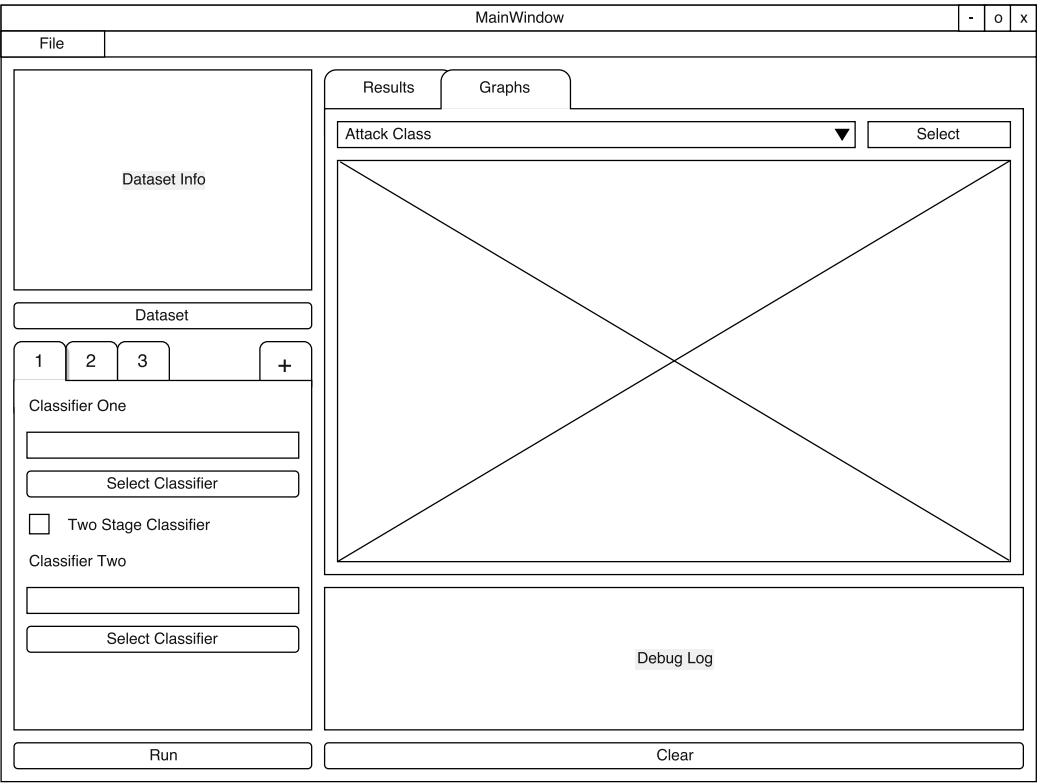


Figure 4: Main Window Graph Tab Wireframe

Dataset

-

o

x

Training Dataset

Select File

Testing Dataset

Select File

☐ k-Fold Cross Validation

Folds

Attack Categories

Select File

Stochastic Classifier Runs

Nominal Columns

Column 1

Column 3

Column 4

Column 7

Binary Columns

Column 2

Column 5

Column 6

Numeric Columns

Column 8

Column 9

Column 10

Column 11

Column 12

>

<

>

<

Ok

Figure 5: Dataset Window Wireframe

4.2.2 Functional Requirements

TODO: Fill in descriptions

FR1 - Select Classifiers

ID: FR1

TITLE: Select Classifiers

PRIORITY: High

DESC:

DEP: N/A

FR2 - Two Stage Classification

ID: FR2

TITLE: Two Stage Classification

PRIORITY: High

DESC:

DEP: N/A

28

**FR3 - Run Classifiers**

**ID:** FR3

**TITLE:** Run Classifiers

**PRIORITY:** High

**DESC:**

**DEP:** FR1, FR2

**FR4 - Create New Classifier**

**ID:** FR4

**TITLE:** Create New Classifier

**PRIORITY:** Low

**DESC:**

**DEP:** N/A

**FR5 - Debug Output**

**ID:** FR5

**TITLE:** Debug Output

**PRIORITY:** Medium

**DESC:**

**DEP:** N/A

**FR6 - Multiple Classifier Configurations**

**ID:** FR6

**TITLE:** Multiple Classifier Configurations

**PRIORITY:** Medium

**DESC:**

**DEP:** FR1, FR2

**FR7 - Aggregate Results by Attack**

**ID:** FR7

**TITLE:** Aggregate Results by Attack

**PRIORITY:** High

**DESC:**

**DEP:** FR3

**FR8 - Write Results to File**

**ID:** FR8

**TITLE:** Write Results to File

**PRIORITY:** Medium

**DESC:**

**DEP:** N/A

**FR9 - Get Classification Results**

**ID:** FR9

**TITLE:** Get Classification Results

**PRIORITY:** High

**DESC:**

**DEP:** FR3

**FR10 - Graph Results**

**ID:** FR10

**TITLE:** Graph Results

**PRIORITY:** High

**DESC:**

**DEP:** FR9

**FR11 - Select Training Dataset**

**ID:** FR11

**TITLE:** Select Training Dataset

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR12 - Select Testing Dataset**

**ID:** FR12

**TITLE:** Select Testing Dataset

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR13 - k-Fold Cross Validation**

**ID:** FR13

**TITLE:** k-Fold Cross Validation

**PRIORITY:** High

**DESC:**

**DEP:** FR11

**FR14 - Select Dataset Column Labels**

**ID:** FR14

**TITLE:** Select Dataset Column Labels

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR15 - Select Dataset Attack Categories**

**ID:** FR15

**TITLE:** Select Dataset Attack Categories

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR16 - Categorise Dataset Fields**

**ID:** FR16

**TITLE:** Categorise Dataset Fields

**PRIORITY:** High

**DESC:**

**DEP:** FR14

**FR17 - One Hot Encoding**

**ID:** FR17

**TITLE:** One Hot Encoding

**PRIORITY:** High

**DESC:**

**DEP:** FR18

**FR18 - Load Data**

**ID:** FR18

**TITLE:** Load Data

**PRIORITY:** High

**DESC:**

**DEP:** FR11, FR12, FR14, FR15

**FR19 - k-Nearest Neighbour Classifier**

**ID:** FR19

**TITLE:** k-Nearest Neighbour Classifier

**PRIORITY:** High



**DESC:**

**DEP:** N/A

**FR20 - Multi-layer Perceptron Classifier**

**ID:** FR20

**TITLE:** Multi-layer Perceptron Classifier

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR21 - Negative Selection Classifier**

**ID:** FR21

**TITLE:** Negative Selection Classifier

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR22 - Support Vector Machine Classifier**

**ID:** FR22

**TITLE:** Support Vector Machine Classifier

**PRIORITY:** High

**DESC:**

**DEP:** N/A

**FR23 - Stochastic Classifier Averaging**

**ID:** FR23

**TITLE:** Stochastic Classifier Averaging

**PRIORITY:** High

**DESC:**

**DEP:** FR1, FR2

**FR24 - Feature Selection**

**ID:** FR24

**TITLE:** Feature Selection

**PRIORITY:** Low

**DESC:**

**DEP:** N/A

### **4.2.3 Non-Functional Requirements**

#### **QR1 - Robustness**

**ID:** QR1

**TITLE:** Robustness

**PRIORITY:** High

**DESC:**

#### **QR2 - Responsiveness**

**ID:** QR2

**TITLE:** Responsiveness

**PRIORITY:** Medium

**DESC:** UI should not hang during stuff.

#### **QR3 - Usability**

**ID:** QR3

**TITLE:** Usability

**PRIORITY:** Medium

**DESC:**

#### **QR4 - Maintainability**

**ID:** QR4

**TITLE:** Maintainability

**PRIORITY:** High

**DESC:**

#### **QR5 - Portability**

**ID:** QR5

**TITLE:** Portability

**PRIORITY:** Low

**DESC:** Work on different systems and OS's

**QR6 - Correctness****ID:** QR6**TITLE:** Correctness**PRIORITY:** High**DESC:** Results MUST be correct.**4.3 Testing**

TODO: Paragraph about testing, unit testing and why

Table 1: Test Case Descriptions

ID	Objective	Precondition	Steps	Test Data	Expected Result
TC1	Create new classifier tab	There is only one tab in the classifier tab box.	- Press the '+' button above the classifier tab box.	N/A	A new tab is created with the number 2 on it.
TC2	Open dataset window.	N/A	- Press the 'Select Dataset..' button in the main window.	N/A	Dataset window is shown.
TC3	Select training dataset.	Dataset window is open.	- Press the 'Select Training Set..' button. - Select a file.	Any .csv file.	Absolute path of the file appears in the textbox below 'Training Dataset' label.
TC4	Deselect k-Fold Cross Validation.	Dataset window is open and k-Fold Cross Validation checkbox is checked.	- Press the k-Fold Cross Validation checkbox.	N/A	Testing dataset label, textbox and button become enabled and folds spinbox is disabled.
TC5	Select testing dataset.	Dataset window is open.	- Press the 'Select Testing Set...' button. - Select a file.	Any valid .csv file.	Absolute path of the file appears in the textbox below 'Training Dataset' label.
TC6	Select dataset fields.	Dataset window is open.	- Press the 'Select Dataset Fields' button. - Select a file.	A .csv file containing two columns, the second of which is one of the values: Numeric, Nominal, Binary.	Absolute path of the file appears in the textbox below 'Dataset Fields' label, and the 'Numeric', 'Nominal', and 'Binary' list views are populated.

TC7	Select attack categories.	Dataset window is open.	<ul style="list-style-type: none"> <li>- Press the 'Select Attack Categories' button.</li> <li>- Select a file.</li> </ul>	Any valid .csv file.	Absolute path of the file appears in the textbox below 'Attack Categories' label.
TC8	Display dataset info in main window.	Valid dataset files have been selected and dataset window is open.	<ul style="list-style-type: none"> <li>- Press the OK button in the bottom right of the dataset window.</li> </ul>	N/A	The dataset window closes, showing no error message and the textbox in the top left of the main window is populated with information regarding the dataset.
TC9	Select classifier.	N/A	<ul style="list-style-type: none"> <li>- Press the 'Select Classifier' button below the 'Classifier One' label.</li> <li>- Select a file.</li> </ul>	Any .py file.	Absolute path of the file appears in the textbox below the 'Classifier One' label.
TC10	Select two stage classification.	Two stage classification checkbox is unchecked.	<ul style="list-style-type: none"> <li>- Press the 'Two Stage Classification' checkbox.</li> </ul>	N/A	The 'Second Classifier' label, textbox, and 'Select Classifier' button are enabled.
TC11	Select second classifier.	The 'Two Stage Classification' checkbox is checked.	<ul style="list-style-type: none"> <li>- Press the 'Select Classifier' button below the 'Classifier Two' label.</li> <li>- Select a file.</li> </ul>	Any .py file.	Absolute path of the file appears in the textbox below the 'Classifier One' label.
TC12	Error message on invalid training dataset filename.	N/A	<ul style="list-style-type: none"> <li>- Enter a filepath which does not exist into the textbox below the 'Training Dataset' label.</li> <li>- Supply every other textbox with valid filepaths.</li> <li>- Press the OK button in the bottom right of the dataset window.</li> </ul>	N/A	An error message is shown and the window does not exit.

TC13	Error message on invalid testing dataset filename.	N/A	<ul style="list-style-type: none"> <li>- Enter a filepath which does not exist into the textbox below the 'Testing Dataset' label.</li> <li>- Supply every other textbox with valid filepaths.</li> <li>- Press the OK button in the bottom right of the dataset window.</li> </ul>	N/A	An error message is shown and the window does not exit.
TC14	Error message on invalid field names filename.	N/A	<ul style="list-style-type: none"> <li>- Enter a filepath which does not exist into the textbox below the 'Dataset Fields' label.</li> <li>- Supply every other textbox with valid filepaths.</li> <li>- Press the OK button in the bottom right of the dataset window.</li> </ul>	N/A	An error message is shown and the window does not exit.
TC15	Error message on invalid attack categories filename.	N/A	<ul style="list-style-type: none"> <li>- Enter a filepath which does not exist into the textbox below the 'Attack Categories' label.</li> <li>- Supply every other textbox with valid filepaths.</li> <li>- Press the OK button in the bottom right of the dataset window.</li> </ul>	N/A	An error message is shown and the window does not exit.
TC16	Error message on invalid training dataset.	Valid dataset files have been selected, except from training dataset which is invalid. A valid classifier configuration has been selected.	<ul style="list-style-type: none"> <li>- Press the 'Run' button on the main window.</li> </ul>	N/A	An error message is displayed and the program does not crash.
TC17	Error message on invalid testing dataset.	Valid dataset files have been selected, except from testing dataset which is invalid. A valid classifier configuration has been selected.	<ul style="list-style-type: none"> <li>- Press the 'Run' button on the main window.</li> </ul>	N/A	An error message is displayed and the program does not crash.

TC18	Error message on invalid field names.	Valid dataset files have been selected, except from dataset fields which do not correspond to the dataset. A valid classifier configuration has been selected.	- Press the 'Run' button on the main window.	N/A	An error message is displayed and the program does not crash.
TC19	Error message on invalid attack categories.	Valid dataset files have been selected, except from attack categories which do not correspond to the dataset. A valid classifier configuration has been selected.	- Press the 'Run' button on the main window.	N/A	An error message is displayed and the program does not crash.
TC20	Dataset window is populated with existing information.	Dataset file paths have been previously selected and the OK button pressed on the dataset window.	- Press the 'Select Dataset..' button on the main window.	N/A	The text boxes containing the paths to files should all be filled with the information which was entered previously, as well as fold count and stochastic classifier run count.
TC21	Error message on invalid classifier.	N/A	- Press the 'Select Classifier' button the main window. - Select an invalid .py file. - Press the 'Run' button.	An invalid .py file.	An error message is displayed and the program does not crash.
TC22	Delete classifier tab.	There is more than one tab in the classifier tab.	- Press the 'x' button next to the tab number in the classifiers tabs in any tab except the first.	N/A	The tab in which the 'x' button was pressed should close.
TC23	Show results for all classifiers run.	Valid dataset has been selected and valid classifier configuration has been selected.	- Press the 'Run' button.	N/A	A tab should be created on the main tab container for each classifier configuration run, and populated with classification information.

---

TC24	Show graph of classification results.	Classifiers have been run and results retrieved.	- Select the 'Graphs' tab on the main window. - Select a class from the drop down in the 'Graphs' tab. - Press the 'Show' button.	N/A	A graph is shown which has a series for each classifier configuration specified.
------	---------------------------------------	--	---	-----	--

---

## 5 Methodology

### 5.1 Implementation

#### 5.1.1 PyQt

PyQt is a set of Python bindings for the Qt application framework, which facilitates the writing of cross platform Qt applications which are capable of running on Windows, Mac, and Linux (PSF, 2018).

PyQt was chosen as the framework of choice for creating the application GUI over other frameworks such as Java Swing for a number of reasons, the greatest of which being that it would allow all development to be carried out in Python as the libraries which were already selected for machine learning were written in Python. Writing the entire project in one language comes with extra benefits such as being able to have a single development environment, simplifying application design as there is no need for interfaces between each language, and having the ability to write all unit tests in a single language. PyQt also has powerful tools which aid in the design and development of user interfaces in the form of Qt Creator, which increased the speed of development substantially, as well as an extensive set of online documentation.

#### 5.1.2 Scikit-Learn

Scikit-Learn is a Python library consisting of a set of tools for easily performing machine learning and data analysis (Pedregosa et al., 2011). The library consists of a large number of pre-implemented classification, and pre-processing algorithms, as well as others such as clustering, regression etc. These pre-implemented algorithms allowed for a much faster development cycle and also guaranteed the correctness of all algorithms which were used, again speeding up development time by removing the need for testing individual algorithms. Sci-kit also provides some basic preprocessing in the form of scaling of inputs so that they are all within a normalised range, as well as some reporting which can be used to quickly extract a lot of information in the form of confusion matrices and classification reports from the results.

Some other libraries were considered for implementing the machine learning algorithms such as Tensorflow due to its highly configurable nature and high performance due to having GPU acceleration, however the implementations of similar algorithms are considerably more complex using Tensorflow, and as time was a major factor in this project Scikit-learn was the preferable library. One final factor which influenced this choice is Scikit-learn's interoperability with Matplotlib allowing for graphs to be created quickly and easily.

### **5.1.3 Pandas**

Pandas is a library which provides high performance, simple data structures and data analysis tools for Python (McKinney, 2010). Pandas provides to the programmer the DataFrame object type which allows for highly complex data manipulation to be carried out easily, such as data alignment and handling of missing data, dataset slicing, indexing and subsetting, merging and joining of multiple sets of data, etc. and is highly optimised for speed as the critical code within the library is written in C, allowing for much higher performance when compared to Python. This means that when working with very large datasets, such as the NSL-KDD and UNSW-NB15 which both have around 100,000 rows of data each, manipulations can be carried out quickly. The documentation for Pandas is also extensive and there exists a large online community of users which is indispensable during the learning of and development within the library.

### **5.1.4 Matplotlib**

Matplotlib (Hunter, 2007) is a Python 2D plotting library which produces high quality graphs and charts in both hardcopy and interactive formats. Matplotlib makes creating complex charts extremely simple as well as offering a massive number of charts, including, histograms, bar charts, scatterplots, confusion matrices, etc., with as little lines of code as possible. One large reason for selecting Matplotlib for use is its interoperability with the other libraries and frameworks which were selected. It works with Scikit-learn with very little effort and also contains custom widgets for use with PyQt which allows for extremely fast and easy integration of Matplotlib charts into the user interface of the software.

### **5.1.5 Software**

The software which was written is described with a class diagram, seen in Appendix E specifying each distinct class created along with its methods,



properties and relationship with other classes, and with a use case diagram showing the general function of the software from the perspective of the user, as seen in Appendix F. A description of all of the software classes functions is as follows.

#### **5.1.5.1 MainWindow**

The MainWindow class is the class which is responsible for all of the logic behind the user interface of the main window. It inherits the QMainWindow class; a PyQt type which provides a main application window, and from Ui\_MainWindow, which provides the specification and initialisation of the user interface. The classes main function is handling user input, creating new user interface elements and windows, and allowing the user to view classification results.

#### **5.1.5.2 DatasetWindow**

The DatasetWindow class is primarily responsible for defining what dataset the user wishes to use, to specify the data type of each column of said dataset. The window also allows users to choose between a testing dataset or k-fold cross validation and specify a number of folds, and to specify the number of runs a stochastic classifier should be run for. This class inherits from QDialog which provides the class with buttons allowing the user to accept or cancel the information they have input. It also inherits from Ui\_DatasetWindow which like Ui\_MainWindow provides the user interface for the window.

#### **5.1.5.3 UIObject**

UIObject is an interface which is used by other classes which implement a user interface for some window. It is required as when a user interface is inherited by a window class a number of methods are expected to be available in order to create and render the user interface at runtime.

#### **5.1.5.4 Classifier**

Classifier is an interface which any classification algorithm which is to be used must implement. It has one property, 'stochastic', which determines whether or not the classifier must be run multiple times, and a 'run' method which carries out the classification and returns the results.

#### **5.1.5.5 QClfSelector**

QClfSelector is a class which inherits from QWidget, the base class of all Qt user interface objects, and who's main function is to act as a widget

that can be included within MainWindow multiple times. It provides the functionality of accepting user classifier arrangements and then fetching and running those classifiers, using whatever method has been specified by the user, be it k-fold cross validation or using a test dataset, or running stochastic classifiers multiple times and gathering the results.

#### 5.1.5.6 QBarChart

QBarChart is a class which inherits from 'FigureCanvas', a Matplotlib class which interfaces with PyQt in order to create widget which can be displayed within a PyQt application. The class accepts a set of classification results and then constructs a Matplotlib chart which can be displayed in the MainWindow class for the user to view.

#### 5.1.5.7 ErrorMessage

The ErrorMessage Class is the class responsible for displaying an error message popup to users, in a standard manner and encapsulates boiler plate code required for creating and showing a message box. The class inherits from QMessageBox which is a PyQt class provides a modal dialog for informing the user.

### 5.2 Datasets

#### 5.2.1 NSL-KDD

Within the KDD-NSL dataset, traffic is categorised into either normal network traffic for a military network or into one of the following four attack categories:

- Denial of Service (DoS) - Attacker tries to prevent legitimate users from using a service.
- Remote to Local (r2l) - Attacker does not have an account on the victim machine, hence tries to gain access.
- User to Root (u2r) - Attacker has local access to the victim machine and tries to gain super user privileges.
- Probe - Attacker tries to gain information about the target host.

A complete listing of all attacks and their categories can be seen in Table 2, as well as the sample count of each category of attack in Table 3.

Table 2: NSL-KDD Dataset Attack Samples.

Attack	Samples	Category
back	956	dos
buffer_overflow	30	u2r
ftp_write	8	r2l
guess_passwd	53	r2l
imap	11	r2l
ipsweep	3599	r2l
land	18	dos
loadmodule	9	u2r
multihop	7	r2l
neptune	41214	dos
nmap	1493	probe
normal	67343	normal
perl	3	u2r
phf	4	r2l
pod	201	dos
portsweep	2931	probe
rootkit	10	u2r
satan	3633	probe
smurf	2646	dos
spy	2	r2l
teardrop	892	dos
warezclient	890	r2l
warezmaster	20	r2l

Table 3: NSL-KDD Dataset Categories Samples.

DoS	Probe	u2r	r2l	Normal
45927	11656	52	995	67343

The dataset itself is also comprised of 41 features per connection recorded, a detailed listing of each and their type can be seen in Appendix G.

### 5.2.2 UNSW-NB15

Similarly, within the UNSW-NB15 dataset, as with the NSL-KDD dataset, traffic is also categorised into either normal network traffic for a military network or into one of the following seven attack categories:

- Fuzzers - Attempting to cause a program or network suspended by feeding it the randomly generated data. Analysis 2,677 It contains different attacks of port scan, spam and html files penetrations.
- Backdoors - A technique in which a system security mechanism is bypassed stealthily to access a computer or its data.
- DoS - A malicious attempt to make a server or a network resource unavailable to users, usually by temporarily interrupting or suspending the services of a host connected to the Internet.
- Exploits - The attacker knows of a security problem within an operating system or a piece of software and leverages that knowledge by exploiting the vulnerability.
- Generic - A technique works against all block-ciphers (with a given block and key size), without consideration about the structure of the block-cipher.
- Reconnaissance - Contains all Strikes that can simulate attacks that gather information.
- Shellcode - A small piece of code used as the payload in the exploitation of software vulnerability. Worms 174 Attacker replicates itself in order to spread to other computers. Often, it uses a computer network to spread itself, relying on security failures on the target computer to access it.

Table 4: UNSW-NB15 Dataset Attack Samples.

Attack	Samples
Analysis	677
Backdoor	583
DoS	4089
Exploits	11132
Fuzzers	6062
Generic	18871
Reconnaissance	3496
Shellcode	378
Worms	44
Normal	37000

The UNSW-NB15 dataset is comprised of 42 features per connection recorded, a detailed listing of each feature, their type and a description of each can be seen in Appendix H.

### 5.3 Data Pre-processing

Before classification can take place it is necessary to perform some pre-processing upon both datasets, so that when they are used they are in a form which is most useful for classification, allowing for more accurate, and faster classification. The first form of pre-processing which will take place upon the datasets is in the form of one hot encoding. One hot encoding is a process wherein categorical features of a dataset are converted into a binary representation. This process can be seen in Figure 6, which shows a small snippet from the NSL-KDD dataset before, and after one hot encoding.

flag	src_bytes	dst_bytes		flag_SF	flag_S0	flag_REJ	src_bytes	dst_bytes
SF	491	0		1	0	0	491	0
SF	146	0		1	0	0	146	0
S0	0	0	→	0	1	0	0	0
SF	232	8153		1	0	0	232	8153
SF	199	420		1	0	0	199	420
REJ	0	0		0	0	1	0	0

Figure 6: One-hot Encoding Example

One hot encoding is necessary as many classification algorithms rely on either some kind of internal averaging, or some form of activation function which require numerical input in order to extract meaning from. Simply converting each of these categories from a nominal to a numeric representation is not sufficient as the categories lose meaning when represented by a series of numbers, i.e. when calculating distance from one category to another any number selected will be arbitrary, and the machine learning algorithm will attempt to extract a relationship between the numbers where there is none, therefore reducing accuracy in classification.

One side effect of one hot encoding is that when processing both the training and testing dataset it is possible that the testing set is missing some categories present within the training set, leading to the algorithms being unable to perform a categorisation. To rectify this the next form of pre-processing which must take place is the insertion of missing features from the training set into the testing set. To do this the training set is simply scanned to find features which are not present in the testing set, and then inserts them into the testing set at the same index thereby making both sets features identical.

The final form of pre-processing to be performed on the dataset is feature scaling. Feature scaling is where a range of data is normalised by scaling it to be within some range, in this case between -1 and 1. The formula used for doing this is seen below in Figure 7:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Figure 7: Feature Scaling Formula

where  $x$  is the original value and  $x'$  is the normalised value. Representing a range of values between a normalised range is important as a large number of classification algorithms compute the euclidian distance between two points and will therefore not work correctly if two different features have wildly different scales. For example if one feature has a large range of numbers and another a very narrow range, the distance will be dependant only on the feature with the larger range, so each feature should be normalised to ensure that each contributes the same amount to the final distance which is computed. Feature scaling also allows classifiers which use gradient descent, such as the multi-layer perceptron, to converge much faster allowing for faster collection of results.

#### 5.4 Classifier Configurations

The following is a listing of the parameters used in the running of each of the classifiers implemented within the scikit-learn library, with the descriptions of each taken from the official documentation:

Table 5: Classifier Parameters

	Parameter	NSL-KDD Value	UNSW-NB15 Value	Description
k-Nearest Neighbour	algorithm	'kdtree'	'kdtree'	Algorithm used to compute the nearest neighbors.
	weights	'uniform'	'uniform'	Uniform weights. All points in each neighborhood are weighted equally.
	leaf_size	40	40	Leaf size used for the KDTree.
	p	2	2	Power parameter for the Minkowski metric.

	n_neighbors	3	2	The number of nearest neighbours to search for, i.e. $k$ .
Multi-Layer Perceptron	hidden_layer_sizes	100	100	The number of neurons within the hidden layer.
	activation	'relu'	'relu'	Activation function for the hidden layer, rectified linear unit function, returns $f(x) = \max(0, x)$ .
	solver	'adam'	'adam'	The solver for weight optimization, 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma and Ba, 2014.
	learning_rate	'invscaling'	'invscaling'	The learning schedule, 'invscaling' gradually decreases the learning rate at each time step.
Support Vector Machine	kernel	'rbf'	'rbf'	Specifies the kernel type to be used in the algorithm, Radial Basis Function kernel.
	gamma	$1e - 1$	$1e - 1$	How much influence a single training example has. The larger gamma is, the closer other examples must be to be affected.
	C	10	10	Trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly.

### 5.5 Single Stage Classification

The first form of classification which will take place is single stage classification. In single stage classification only a single machine learning technique and stage is implemented in order to classify network traffic. The dataset is first read and preprocessed, and the entire set is used in order to train the

machine learning classifier. Once the classifier has been trained, the classifier will make predictions about the testing set, classifying it into normal traffic or any one of the attack categories for the dataset it was trained upon, e.g. the attacks seen in Table 2.

### 5.6 Two Stage Classification

In second form of classification to take place is two stage classification. In two stage classification, the classification processes is split into two separate stages, the first indentifying normal traffic, and the second identifying abnormal. Before the first stage is trained, the dataset must first be flattened, with all attack types being reduced to simply 'attack', resulting in a set of data labelled only 'normal' and 'attack'. This new dataset is then used to train the first classifier, and then predictions made about the testing set, consisting of only 'normal' and 'attack' classifications. The second classifier is now trained on the original, unflattened dataset with all of the normal traffic removed, resulting in a dataset consisting of only attacks. Once the classifier has been trained, the traffic which was classified as being an attack by the first classifier is then fed into the second classifier, and is then labelled as a specific attack. The total result is then the combination of the traffic identified as normal within the first classifier and the attacks from the second classifier.

Performing classification in this manner is expected to give more accurate results than a single stage classifier as it allows each stage to specialise, with the first only needing to differentiate between two classes, 'normal' and 'attack', and the second stage focussed on specific attack types, in theory increasing the overall rate of accurate detection by reducing domain size.

### 5.7 Data Collection

When performing data collection a number of things were done to ensure the greatest consistency within the results gathered, that comparisons between classification methods are valid, and that the results portray the real world performance of such a classifier as closely as possible. The first thing done to ensure validity of results gathered is that classifiers which are stochastic, i.e. are non-deterministic, are run for a total of 30 times and the results are averaged to reduce the amount of randomness within the results and lower the chance that a classifier may perform abnormally well or otherwise.

To best represent the real world performance of the classifiers, k-fold cross validation was used as described by Refaeilzadeh, Tang and Liu, 2009. In k-fold cross validation the training dataset is partitioned into  $k$  equal parts, with one part being used as the testing dataset the classifier will make predictions about, and the remaining  $k - 1$  parts used as the training set.



This process is repeated  $k$  times with a different fold used each time as the testing set, a diagram of which can be seen in Figure 8.

Iteration	Dataset									
1	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_8$	$k_9$	$k_{10}$
2	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_8$	$k_9$	$k_{10}$
3	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_8$	$k_9$	$k_{10}$
$\vdots$					$\vdots$					
9	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_8$	$k_9$	$k_{10}$
10	$k_1$	$k_2$	$k_3$	$k_4$	$k_5$	$k_6$	$k_7$	$k_8$	$k_9$	$k_{10}$

Figure 8: k-Fold Cross Validation

The results from all of these folds can then be combined and used to obtain a single result for the performance of the classification method. Using this method for testing the performance of classifier is preferable to the use of a testing set as the all of the data available is used as both training data and testing data, therefore giving a more accurate estimation of how well the classifier will generalize to any independent dataset, i.e. it gives a more accurate idea of how well the classifier will perform in practice.

The  $k$  value selected for the k-fold cross validation to be carried out in this investigation is  $k = 10$  as a study carried out by Kohavi et al., 1995 found ten-fold cross validation to be the most effective in producing accurate results. The same  $k$  value was used throughout for each classifier, as well as the using the same folds for gathering results without randomising to ensure that results collected were comparable.

Finally the same parameters were used for each of the classifier arrangements, as described in Table 5, to ensure that results across classifier arrangements were consistent with one and other.

### 5.8 Metrics

The following section describes the metrics which will be extracted from each of the classification results.

Precision is a measure of how many of the classifications made are relevant, the formula for which can be seen below in Figure 9:

$$precision = \frac{tp}{tp + fp}$$

Figure 9: Precision Formula

where  $tp$  is the number of true positives, and  $fp$  is the number of false positives. For example, if a classifier were to have a precision of 1.0 that would mean that only relevant traffic was classified, there were no false positives. This however does not tell us what percentage of the relative traffic was actually classified. To do this another metric must be used called recall.

Recall, also known as sensitivity, is a measure of what percentage of the total relevant records were classified. The formula for recall can be seen below in Figure 10:

$$recall = \frac{tp}{tp + fn}$$

Figure 10: Recall Formula

where  $tp$  is the number of true positives, and  $fn$  is the number of false negatives. For example, if a classifier were to have a recall of 1.0, that would mean that every single record of that class had been identified and classified correctly, however even with a recall of 1.0 there may be a large number of false positives which is why it is important to also take a measure of precision. To combine both of these metrics into a single value, f1-score can be used.

f1-Score is a metric which takes both the precision and recall into account, the formula for which can be seen below in Figure 11:

$$f_1 = \frac{2tp}{2tp + fp + fn}$$

Figure 11: f1-Score Formula

where  $tp$  is the number of true positives,  $fp$  the number of false positives, and  $fn$  the number of false negatives. This metric is extremely useful if false positives and false negatives are weighted the same in terms of negative effects. Note that f1-score does not take true negatives into account, however in this case the sheer volume of true negatives for each class would cause a metric taking them into account yield no useful insight.

## 6 Results

### 6.1 NSL-KDD

The following section lists the results gathered running each classifier configuration on the NSL-KDD dataset. All of the metrics calculated using these results were first calculated per class, then combined and averaged for each category. This is done to ensure that classes with a lower number

of samples are equally represented as the number of samples for each class within a category can vary greatly. The total number of true positives, true negatives, false positives, and false negatives for each category can be found in Appendix I.1.

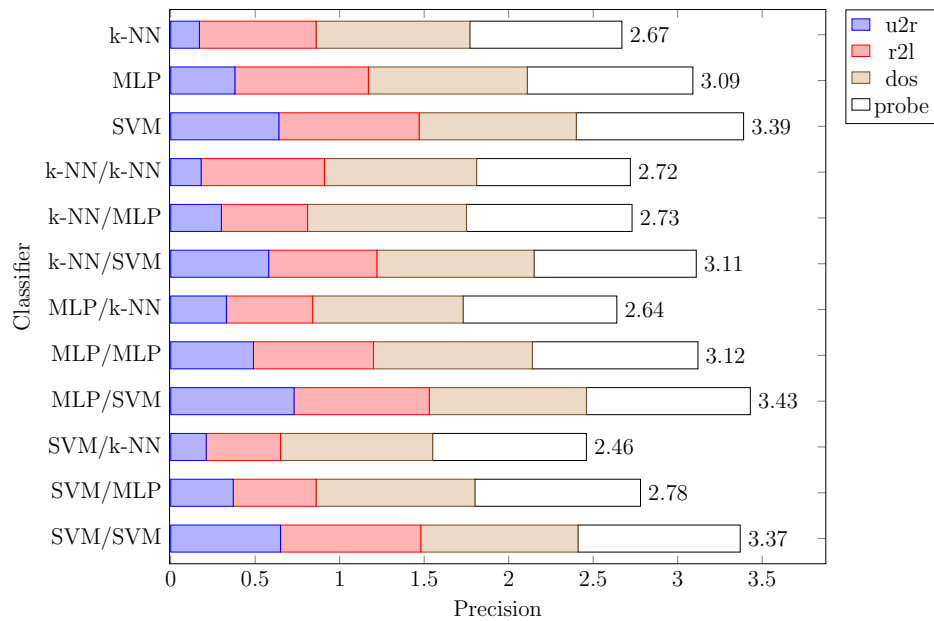


Figure 12: Classifiers Precision Graph

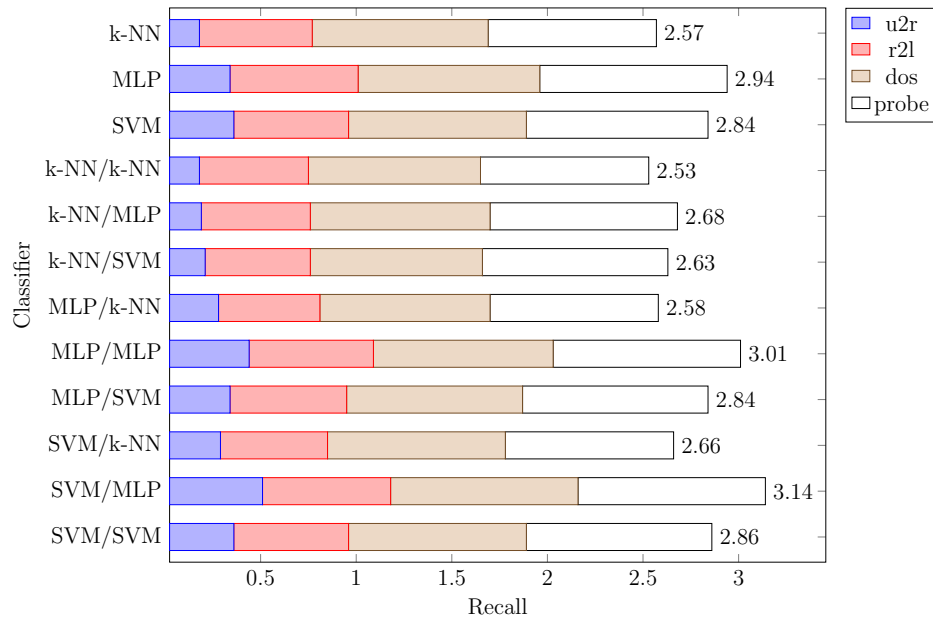


Figure 13: Classifiers Recall Graph

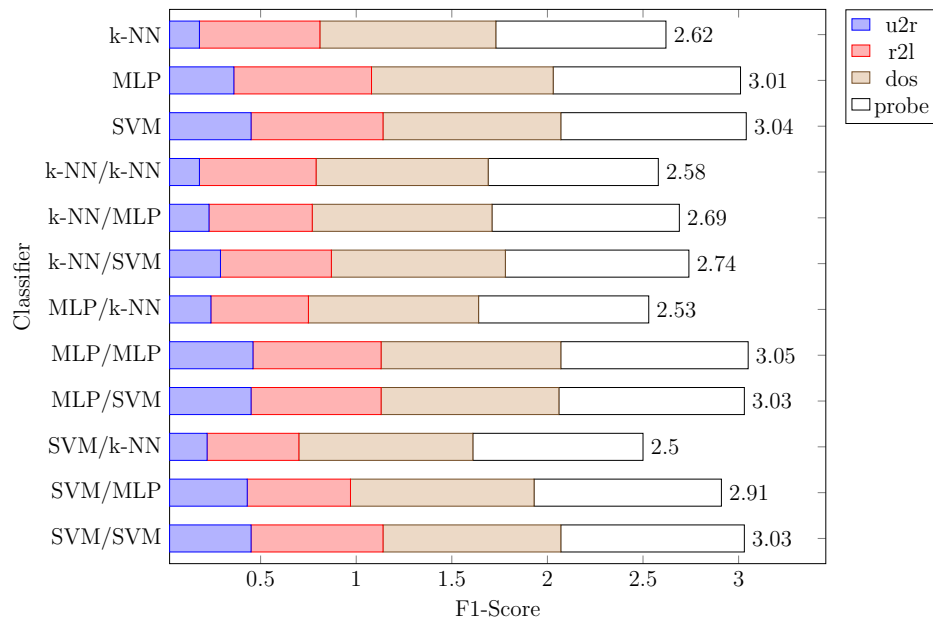


Figure 14: Classifiers f1-Score Graph

The values used in the creation of Figures 12, 13, and 14 can be found in Appendices I.2, I.3, and I.4 respectively.

## 6.2 UNSW-NB15

The following section lists the results gathered running each classifier configuration on the UNSW-NB15 dataset. In this dataset, attacks are already grouped into categories and so no averaging is necessary. The total number of true positives, true negatives, false positives, and false negatives for each category can be found in Appendix J.1.

TODO : Re-test results, I think I need to make changes to classifier configurations, thats why second set is getting real bad results.

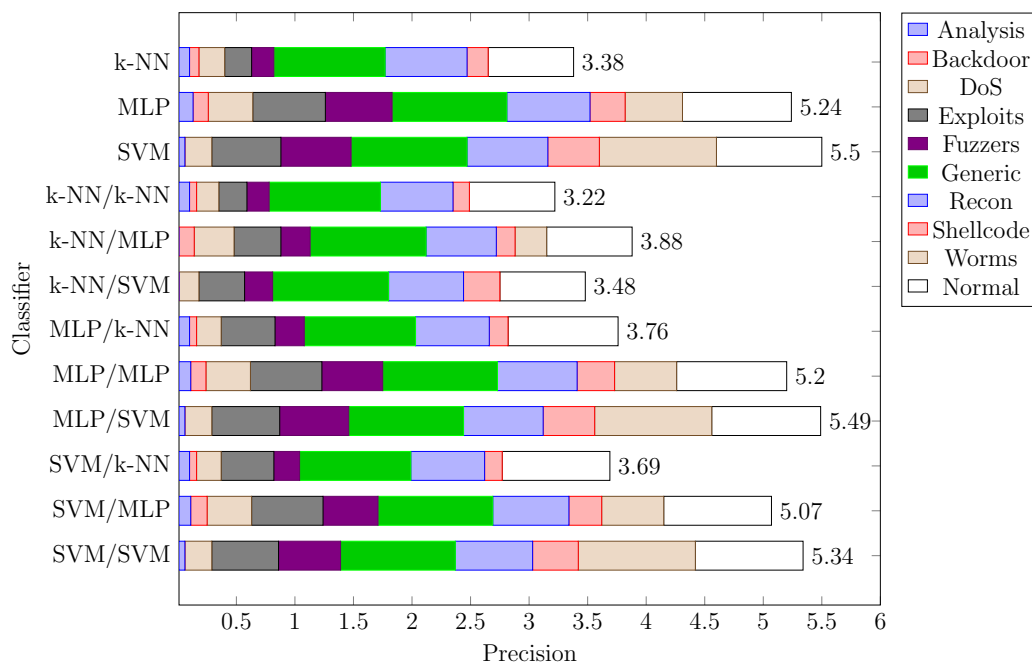


Figure 15: Classifiers Precision Graph

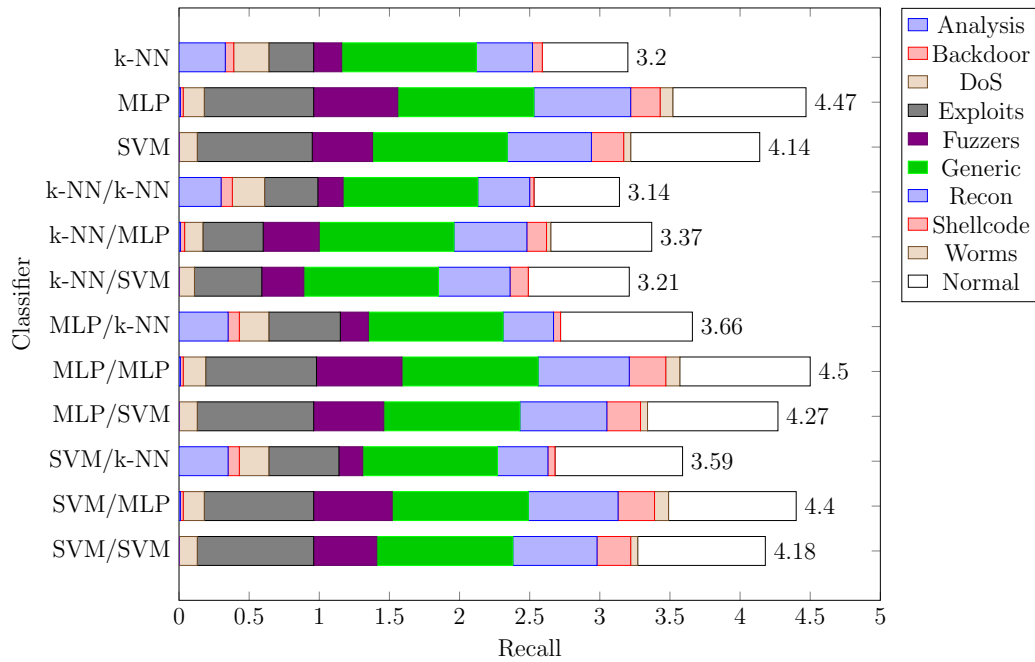


Figure 16: UNSW-NB15 Classifiers Recall Graph

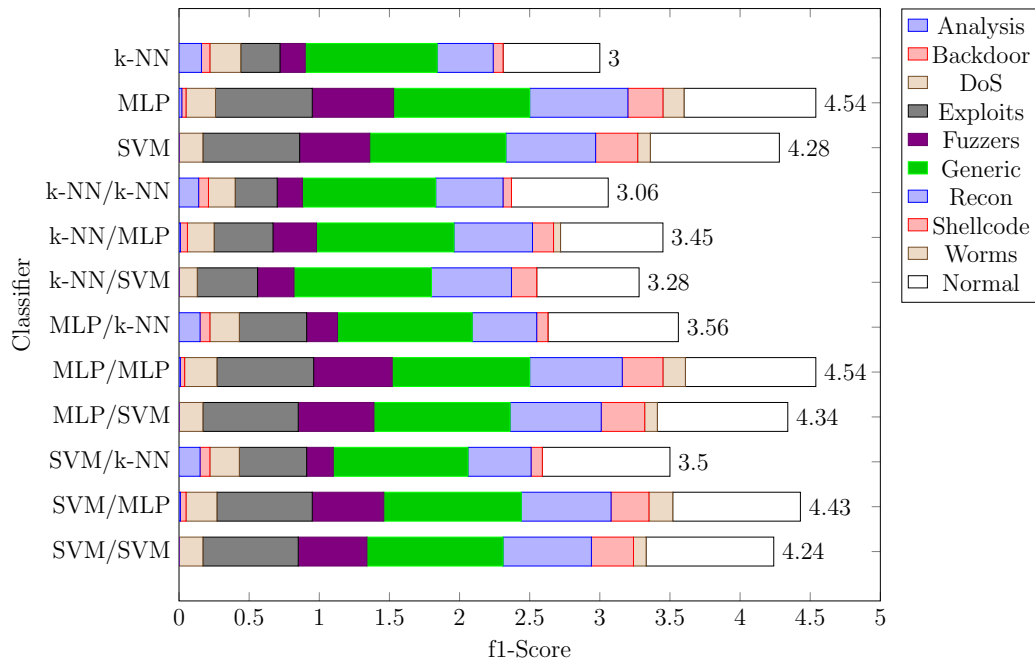


Figure 17: UNSW-NB15 Classifiers f1-Score Graph

The values used in the creation of Figures 15, 16, and 17 can be found in

Appendices J.2, J.3, and J.4 respectively.

## **7 Evaluation**

### **7.1 Software**

Evaluate the software How many of the function requirements were met  
How many of the non functional requirements were met Is the user interface  
like it was designed How does it compare to existing software

### **7.2 Single vs Two-stage Classifiers Performance**

- knn-knn second stage for unsw results in same normals but lower attack classification. Attacks require normals to better differentiate between them.
- svm-svm normal has less true positives meaning that attacks will all have higher false positive rates. Also means that individual attacks are important in defining boundaries between normal and attack, not just as a whole.
- Combining classifiers with less performant classifiers will always reduce performance
- in NSL KDD, adding a more performant will generally increase performance
- in NSL KDD combining with self will produce better results, except in the case of SVM

## 8 Conclusion

### 8.1 Research Questions

What is the rate of accurate detection and classification of network intrusions by single stage machine learning classification methods?

What is the rate of accurate detection and classification of network intrusions by two stage machine learning classification methods?

Which method of classification i.e. single stage or two stage, is more accurate and by what amount?

Which configuration of algorithms in the two stage classifier produces the most accurate results?

### 8.2 Has the Project met it's Aims and Objectives?

### 8.3 Reflection

### 8.4 Learning Outcomes

LO1 - Manage a substantial individual project, including planning, documentation and control

LO2 - Construct a focussed problem statement and conduct a suitable investigation, including literature or technology review, into the context of that problem

LO3 - Demonstrate professional competence by applying appropriate theory and practice to the analysis, design, implementation and evaluation of a non-trivial set of deliverables

LO4 - Show a capacity for self-appraisal by analysing the strengths and weaknesses of the project process and outcomes with reference to the initial objectives and to the work of others

### 8.5 Further Research



## References

- Caudill, M. (1987). Neural networks primer, part i. *AI expert*, 2(12), 46–52.
- Dasgupta, D. & Forrest, S. (1996). Novelty detection in time series data using ideas from immunology. In *Proceedings of the international conference on intelligent systems* (pp. 82–87).
- Dasgupta, D. & González, F. (2002). An immunity-based technique to characterize intrusions in computer networks. *IEEE transactions on Evolutionary Computation*, 6(3), 281–291.
- De Castro, L. N. & Timmis, J. (2002). *Artificial immune systems: A new computational intelligence approach*. Springer Science & Business Media.
- Debar, H. [Herve], Becker, M. & Siboni, D. (1992). A neural network component for an intrusion detection system. In *Research in security and privacy, 1992. proceedings., 1992 ieee computer society symposium on* (pp. 240–250). IEEE.
- Debar, H. [Hervé] & Dorizzi, B. (1992). An application of a recurrent network to an intrusion detection system. In *Neural networks, 1992. ijcnn., international joint conference on* (Vol. 2, pp. 478–483). IEEE.
- Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on software engineering*, (2), 222–232.
- Depren, O., Topallar, M., Anarim, E. & Ciliz, M. K. (2005). An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks. *Expert systems with Applications*, 29(4), 713–722.
- Dhanabal, L. & Shantharajah, S. (2015). A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(6), 446–452.
- Forrest, S., Perelson, A. S., Allen, L. & Cherukuri, R. (1994). Self-nonsel self discrimination in a computer. In *Research in security and privacy, 1994. proceedings., 1994 ieee computer society symposium on* (pp. 202–212). Ieee.
- Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G. & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1), 18–28.
- Ghosh, A. K. & Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Usenix security symposium* (Vol. 99, p. 12).
- González, F. A. & Dasgupta, D. (2003). Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4), 383–403.

- Hautamaki, V., Karkkainen, I. & Franti, P. (2004). Outlier detection using k-nearest neighbour graph. In *Pattern recognition, 2004. icpr 2004. proceedings of the 17th international conference on* (Vol. 3, pp. 430–433). IEEE.
- Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3), 90–95. doi:10.1109/MCSE.2007.55
- Ilgun, K., Kemmerer, R. A. & Porras, P. A. (1995). State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3), 181–199. doi:10.1109/32.372146
- Jain, A. K., Duin, R. P. W. & Mao, J. (2000). Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1), 4–37.
- Javitz, H. S., Valdes, A., Breen, G. & Patton, R. D. (1994). The nides statistical component description and justification.
- K, R., Jayesh N S, P. S., Tom R, G. P. & Mark B, V. W. (2017). *Cyber security breaches survey*. Ipsos MORI Social Research Institute.
- Kayacik, H. G., Zincir-Heywood, A. N. & Heywood, M. I. (2003). On the capability of an som based intrusion detection system. In *Proceedings of the international joint conference on neural networks, 2003.* (Vol. 3, 1808–1813 vol.3). doi:10.1109/IJCNN.2003.1223682
- Kim, J. & Bentley, P. J. (2001a). An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the 3rd annual conference on genetic and evolutionary computation* (pp. 1330–1337). Morgan Kaufmann Publishers Inc.
- Kim, J. & Bentley, P. J. (2001b). Towards an artificial immune system for network intrusion detection: An investigation of clonal selection with a negative selection operator. In *Evolutionary computation, 2001. proceedings of the 2001 congress on* (Vol. 2, pp. 1244–1252). IEEE.
- Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, 2, pp. 1137–1145). Montreal, Canada.
- Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1), 59–69.
- Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A. & Srivastava, J. (2003). A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 siam international conference on data mining* (pp. 25–36). SIAM.

- Lee, S. C. & Heinbuch, D. V. (2001). Training a neural-network based intrusion detector to recognize novel attacks. *IEEE Transactions on systems, man, and Cybernetics-Part A: Systems and Humans*, 31(4), 294–299.
- Lee, W., Stolfo, S. J. & Mok, K. W. (1999). A data mining framework for building intrusion detection models. In *Proceedings of the 1999 ieee symposium on security and privacy (cat. no.99cb36344)* (pp. 120–132). doi:10.1109/SECPRI.1999.766909
- Liao, Y. & Vemuri, V. R. (2002). Use of k-nearest neighbor classifier for intrusion detection. *Computers & security*, 21(5), 439–448.
- Lichodziejewski, P., Zincir-Heywood, A. N. & Heywood, M. I. (2002). Dynamic intrusion detection using self-organizing maps. In *The 14th annual canadian information technology security symposium (citss)*.
- Linda, O., Vollmer, T. & Manic, M. (2009). Neural network based intrusion detection system for critical infrastructures. In *Neural networks, 2009. ijcnn 2009. international joint conference on* (pp. 1827–1834). IEEE.
- Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., ... Zissman, M. A. (2000). Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Darpa information survivability conference and exposition, 2000. discecx '00. proceedings* (Vol. 2, 12–26 vol.2). doi:10.1109/DISCEX.2000.821506
- McKinney, W. (2010). Data structures for statistical computing in python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th python in science conference* (pp. 51–56).
- Moradi, M. & Zulkernine, M. (2004). A neural network based system for intrusion detection and classification of attacks. In *Proceedings of the ieee international conference on advances in intelligent systems-theory and applications* (pp. 15–18).
- Mukkamala, S., Janoski, G. & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In *Neural networks, 2002. ijcnn'02. proceedings of the 2002 international joint conference on* (Vol. 2, pp. 1702–1707). IEEE.
- Panda, M., Abraham, A. & Patra, M. R. (2012). A hybrid intelligent approach for network intrusion detection. *Procedia Engineering*, 30, 1–9.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Dubourg, V. et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct), 2825–2830.
- Powers, S. T. & He, J. (2008). A hybrid artificial immune system and self organising map for network intrusion detection. *Information Sciences*, 178(15), 3024–3042.

- PSF. (2018). Pyqt5. Retrieved March 14, 2018, from <https://pypi.python.org/pypi/PyQt5>
- Refaeilzadeh, P., Tang, L. & Liu, H. (2009). Cross-validation. In *Encyclopedia of database systems* (pp. 532–538). Springer.
- Rhodes, B. C., Mahaffey, J. A. & Cannady, J. D. (2000). Multiple self-organizing maps for intrusion detection. In *Proceedings of the 23rd national information systems security conference* (pp. 16–19).
- Ryan, J., Lin, M.-J. & Miikkulainen, R. (1998). Intrusion detection with neural networks. In *Advances in neural information processing systems* (pp. 943–949).
- Sommer, R. & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Security and privacy (sp), 2010 ieee symposium on* (pp. 305–316). IEEE.
- Sung, A. (1998). Ranking importance of input parameters of neural networks. *Expert Systems with Applications*, 15(3), 405–411.
- Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *Computational intelligence for security and defense applications, 2009. cisda 2009. ieee symposium on* (pp. 1–6). IEEE.
- Tong, X., Wang, Z. & Yu, H. (2009). A research using hybrid rbf/elman neural networks for intrusion detection system secure model. *Computer physics communications*, 180(10), 1795–1801.
- Vasudevan, A., Harshini, E. & Selvakumar, S. (2011). Ssenet-2011: A network intrusion detection system dataset and its comparison with kdd cup 99 dataset. In *Internet (ah-ici), 2011 second asian himalayas international conference on* (pp. 1–5). IEEE.
- Zhang, J. & Zulkernine, M. (2006). A hybrid network intrusion detection technique using random forests. In *Availability, reliability and security, 2006. ares 2006. the first international conference on* (8–pp). IEEE.
- Zhang, Z., Li, J., Manikopoulos, C., Jorgenson, J. & Ucles, J. (2001). Hide: A hierarchical network intrusion detection system using statistical pre-processing and neural network classification. In *Proc. ieee workshop on information assurance and security* (pp. 85–90).

# Appendices

## A Project Overview

Jack Anderson

40208539

### Initial Project Overview

#### SOC10101 Honours Project (40 Credits)

#### Title of Project: A Comparison of Machine Learning Algorithms to Detect Network Intrusions

##### Overview of Project Content and Milestones

For this project, the main objective is to research different machine learning strategies for detecting network intrusions and to develop a piece of software which can obtain metrics from these algorithms. The focus of this project will be on a comparison of two-stage classifiers versus single-stage classifiers for detecting and classifying network intrusions. The dataset upon which these methods will be tested is the KDD Cup 1999 dataset, with the possibility of testing upon other data sets should time allow it.

The different strategies for machine learning will be gathered by reading relevant research papers and articles within the field of machine learning and network intrusion detection and selecting appropriate algorithms/strategies which are applicable for the chosen data set, and which are also feasible to implement within the timescale.

The software will be a desktop application and will take either a predetermined algorithm or a user submitted algorithm, and a data set of network traffic. The software should then run the algorithm and extract metrics from it such as rates for false positives, true positives, false negatives, true negatives, overall accuracy of classification, etc. These results can then be used to plot graphs and charts to visualise this information to a user in a useful way.

A dissertation will be delivered at the end of the project and should contain a detailed design and plan of the software as well as complete testing strategy and results. Also included in the final report should be a comparison of several of the implemented algorithms to determine which is the most suitable for use if any at all.

A list of milestones for this project goes as the following:

- Initial Project Overview Submitted
- Relevant Algorithms Selected
- Project timescale Completed
- Literature Review Complete
- Software Design Completed
- Algorithms Implemented
- Software Implemented
- Software Testing Plan
- Software Tested
- Algorithm Comparison Completed
- Dissertation Written
- Poster Presentation Completed
- Project Submitted

Jack Anderson

40208539

**The Main Deliverable(s):**

A list of the main deliverables for the project is as follows:

- Initial Project Overview
- Gantt Chart
- Literary Review
- Interim Report
- Requirement Specification
- Software Design Document
- Software Test Plan
- Software Test Results
- Software Implementation
- Algorithm Implementations
- Meeting Diary
- Algorithm Experimental Results
- Algorithm Comparisons
- Software User Documentation
- Dissertation
- Poster Presentation

**The Target Audience for the Deliverable(s):**

The target audience for this project could include, machine learning and network intrusion researchers, and computer science students. Researchers may find the comparison of algorithms to be highly useful when carrying out preliminary research and could save time on selecting or discounting an algorithm. Computer science students may also find this project of use for experimenting with different network intrusion methods and different machine learning methods, giving them an insight on what they can be used for and how effective they are.

**The Work to be Undertaken:**

During this project, the work which must be undertaken is first extensive research of the subject area and collection of relevant sources. The project must then be planned and a timeline of work set out to be completed, with deadlines for each deliverable. Algorithms such as k-nearest neighbour, Artificial neural network, negative selection genetic algorithm, etc, will be implemented, compared and contrasted, specifically the performance of single stage against two-stage classifiers using these algorithms. At the same time as implementing these algorithms a requirement specification and then a design document will be created for the software package as well as a test plan. The design will then be implemented and then be tested according to the test plan. Once fully tested and proven correct the software can then be used to compare the algorithms and obtain metrics from then. The final dissertation will then be written which will include an analysis of the results for each algorithm.

**Additional Information / Knowledge Required:**

Additional research is required on network intrusion and two stage classifiers to best select the methods which will be implemented and explored. Research into similar software products such as WEKA ("Weka 3 - Data Mining with Open Source Machine Learning Software in Java", 2017) will also be conducted to source ideas and to see in which areas these pieces of software are lacking. Research on each individual algorithm to be implemented will also be required to ensure a correct implementation. And finally, an investigation into relevant libraries which may be used to assist with GUI creation, Graphing, and algorithm implementations.

Jack Anderson

40208539

**Information Sources that Provide a Context for the Project:**

1. Tsai, C. F., Hsu, Y. F., Lin, C. Y., & Lin, W. Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications*, 36(10), 11994-12000.
2. Sommer, R., & Paxson, V. (2010, May). Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on* (pp. 305-316). IEEE.
3. Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on software engineering*, (2), 222-232.
4. Powers, S. T., & He, J. (2008). A hybrid artificial immune system and Self Organising Map for network intrusion detection. *Information Sciences*, 178(15), 3024-3042.
5. Shon, T., & Moon, J. (2007). A hybrid machine learning approach to network anomaly detection. *Information Sciences*, 177(18), 3799-3821.
6. Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on* (Vol. 2, pp. 1702-1707). IEEE.
7. Frank, J. (1994, October). Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th national computer security conference* (Vol. 10, pp. 1-12).
8. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. (2017). Cs.waikato.ac.nz. Retrieved 21 September 2017, from <http://www.cs.waikato.ac.nz/ml/weka/>

**The Importance of the Project:**

This project has importance as there has not been many papers directly comparing implementations of different machine learning approaches to network intrusion detection. While there are software packages which deal with gaining metrics from and comparing algorithms there is not one which is focused solely on network intrusion detection. Making a piece of software which is focused on one area of research may prove to provide greater insights, through more focussed results or through ease of use regarding comparing single and multiple stage classifiers, whereas a more complicated piece of software may take a long time to become acquainted with and to produce results.

**The Key Challenge(s) to be Overcome:**

The key challenges to be overcome in this project are the implementations of the machine learning algorithms themselves. This is due to a personal lack of experience in implementing machine learning algorithms. Experience is also lacked in understanding formal descriptions of algorithms which may hinder my understanding of techniques when reading research papers. Another challenge will be creating a method of accommodating algorithms by creating interfaces which they will communicate with the main piece of software allowing for any algorithm to be entered by a user.

A.A Project Timeline

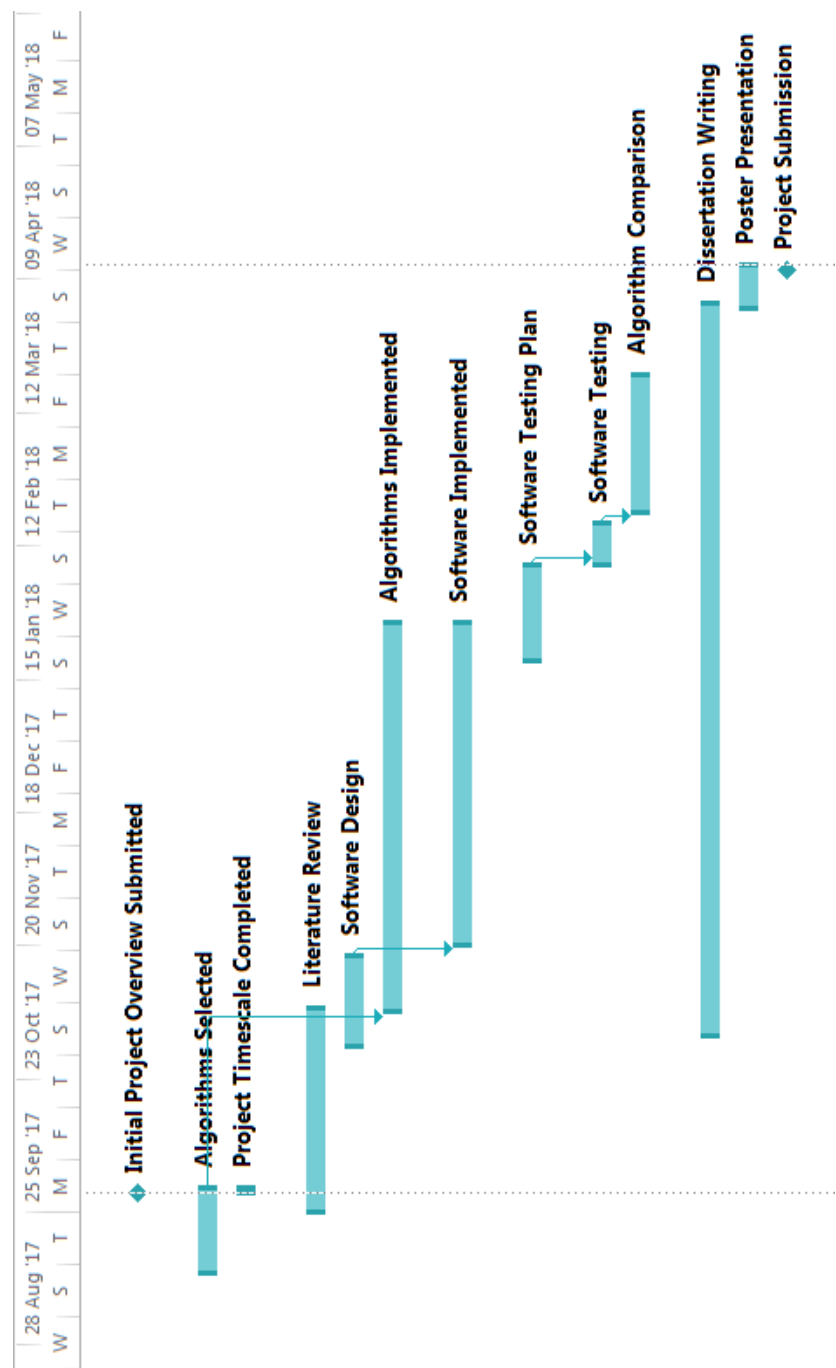


Figure 18: Project Timeline Gantt Chart



## B Second Formal Review Output

## SOC10101 Honours Project (40 Credits)

## Week 9 Report

Student Name: JACK ANDERSON

Supervisor: SIMON POWERS

Second Marker: NAGHMEH MORADPOUR

Date of Meeting: 9/11/17

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? ☒ yes ☐ no\*

If not, please comment on any reasons presented

Please comment on the progress made so far

- weekly meetings &amp; keeping track of them

Is the progress satisfactory? ☒ yes ☐ no\*Can the student articulate their aims and objectives? ☒ yes ☐ no\*

If yes then please comment on them, otherwise write down your suggestions.

- More recent publication.
- Introduction section: more recent attack
- Justification on: ML-based IDS
  - : chosen algorithms
  - : chosen dataset:
    - NSL-KDD
    - UNSW-NB15
    - PU-IDS
    - ADFA-Linux
- Expansion on search term
- study: Splunk
- Research question: hypothesis to add

\* Please circle one answer; if no is circled then this must be amplified in the space provided

Does the student have a plan of work? ☒ yes ☐ no\*

If yes then please comment on that plan otherwise write down your suggestions.

*- work plan has been discussed during the meeting*

Does the student know how they are going to evaluate their work? ☒ yes ☐ no\*

If yes then please comment otherwise write down your suggestions.

*- comparison with correct products: splunk  
interns at performan for 3 algorithms*

Any other recommendations as to the future direction of the project

N/A

Signatures: Supervisor *Simon Powers*

Second Marker

*Naghmi  
Moradpour*

Student *Jack Anderson*

Please give the student a photocopy of this form immediately after the review meeting; the original should be lodged in the School Office with Leanne Clyde

\* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

C Diary Sheets

EDINBURGH NAPIER UNIVERSITY  
SCHOOL OF COMPUTING  
PROJECT DIARY

Student: Jack Anderson  
Date: 15/09/2017

Supervisor: Simon Powers  
Last diary date: N/A

Objectives:

- Complete first draft of IPO.
- Continue reading about Network Intrusion Detection and Two Stage Classifiers
- Investigate WEKA software.

Progress:

- Decided upon language
- Picked several algorithms to implementation
- Read some research papers

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY  
SCHOOL OF COMPUTING  
PROJECT DIARY

Student: Jack Anderson  
Date: 21/09/2017

Supervisor: Simon Powers  
Last diary date: 15/09/2017

Objectives:

- Correct issues with IPO
- Continue reading about network intrusion detection and two stage classifiers
- Investigate WEKA software more
- Begin outlining structure of the literature review

Progress:

- Wrote first draft of IPO
- Explored WEKA
- Read research papers

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY  
SCHOOL OF COMPUTING  
PROJECT DIARY

Student: Jack Anderson  
Date: 28/09/2017

Supervisor: Simon Powers  
Last diary date: 21/09/2017

Objectives:

- Continue collecting citations
- Begin writing literature review

Progress:

- Finalized IPO
- Collected more citations to do with intrusion detection and ensemble classifiers.
- Investigated WEKA
- Outlined literature review structure

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY  
SCHOOL OF COMPUTING  
PROJECT DIARY

Student: Jack Anderson  
Date: 05/10/2017

Supervisor: Simon Powers  
Last diary date: 28/09/2017

Objectives:

- Continue writing literature review

Progress:

- Collected more citations
- Began writing literature review

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 12/10/2017

Last diary date: 05/10/2017

Objectives:

- Continue writing literature review
- Begin implementing the k-nn algorithm

Progress:

- Made progress on literature review about history of intrusion detection
- Collected references regarding specific algorithms

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 20/10/2017

Last diary date: 12/10/2017

Objectives:

- Go into more detail regarding the negative selection algorithm and neural networks in literature review
- Back up some points made within the literature review

Progress:

- Began section regarding algorithms within the literature review

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 26/10/2017

Last diary date: 20/10/2017

Objectives:

- Add a conclusion to literature review
- Finish search strategy section in literature review
- Begin implementation of k-nn algorithm

Progress:

- Finished literature review section on algorithms
- Added a section in literature review about research contribution

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 02/11/2017

Last diary date: 26/10/2017

Objectives:

- Finish first draft of literature review
- Create the outline for the entire dissertation
- Update project schedule Gantt Chart

Progress:

- No progress was made this week as I had two coursework deadlines at the conclusion of the week

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 09/11/2017

Last diary date: 02/11/2017

Objectives:

- At some point in the future update my literature review using all of the feedback within the interim report this includes
  - More up to date references
  - Justifications for different algorithms and machine learning in general
  - Switching datasets from KDD 99 to a more recent/amended one
- Begin work on a k-nn classifier
- General research on python machine learning and GUI libraries

Progress:

- Finished first draft of the literature review
- Updated Gantt Chart to better reflect current project schedule.
- Created outline for dissertation

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 16/11/2017

Last diary date: 09/11/2017

Objectives:

- Improve k-nn classifier by finding alternative to onehot encoding.
- Begin creating the GUI

Progress:

- Set up Github repository
- Researched and implemented preparing a dataset for processing
- Implemented a basic k-nn classifier
- Researched different GUI options

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 23/11/2017

Last diary date: 16/11/2017

Objectives:

- Continue to attempt to improve knn performance
- Continue work on creating GUI
- Implement multi-layer perceptron classifier

Progress:

- Began creating a simple GUI to start learning PyQt
- Began writing simple unit tests

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 09/01/2018

Last diary date: 23/11/2017

Objectives:

- Implement a multi-layer perceptron classifier
- Continue creation of a GUI

Progress:

- No progress was made over the holiday period

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 16/01/2018

Last diary date: 09/01/2018

Objectives:

- Implement a support vector machine instead of negative selection classifier due to time constraints
- Continue work on the GUI
- Research feature selection and improving classifier performance

Progress:

- Created a simple multi-layer perception classifier
- Implemented a large section of the GUI
  - Dataset selection
  - Field type categorising
  - Custom classifier selection
  - Basic window layouts

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 23/01/2018

Last diary date: 16/01/2018

Objectives:

- Research statistical significance testing
- Implement ability to run and compare several classifiers
- Finish graphing and displaying results

Progress:

- Implemented support vector machine classifier
- Added basic graph to GUI
- Added K fold cross validation
- User creation of template classifiers
- Linked back end computation to the GUI
- General bug fixes

Supervisor's Comments:

Version 2

Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Jack Anderson

Supervisor: Simon Powers

Date: 30/01/2018

Last diary date: 23/01/2018

Objectives:

- Finish displaying results in GUI
- Implement ability to run classifiers several times and average results for stochastic classifiers such as MLP
- Begin gathering results

Progress:

- Added ability to run several classifier configurations at once
- Added graphing of classification results and selection of specific classes
- Fixed major bug regarding two stage classification results

Supervisor's Comments:

Version 2

Edinburgh Napier University

## D MoSCoW Analysis

Table 7: MoSCoW Analysis of Requirements

Requirement		Priority			
ID	Title	Must	Should	Could	Wont
FR1	Select Classifiers	x			
FR2	Two Stage Classification	x			
FR3	Run Classifiers	x			
FR4	Create New Classifier			x	
FR5	Debug Output		x		
FR6	Multiple Classifier Configurations		x		
FR7	Aggregate Results By Attack	x			
FR8	Write Results to File		x		
FR9	Get Classification Results	x			
FR10	Graph Results		x		
FR11	Select Training Dataset	x			
FR12	Select Testing Dataset	x			
FR13	k-Fold Cross Validation	x			
FR14	Select Dataset Column Labels	x			
FR15	Select Dataset Attack Categories	x			
FR16	Categorise Dataset Fields	x			
FR17	One Hot Encoding	x			
FR18	Load Data	x			
FR19	k-Nearest Neighbour Classifier	x			
FR20	Multi-layer Perceptron Classifier	x			
FR21	Negative Selection Classifier			x	
FR22	Support Vector Machine Classifier	x			
FR23	Stochastic Classifier Averaging		x		
FR24	Feature Selection				x
QR1	Robustness	x			
QR2	Responsiveness			x	
QR3	Usability		x		
QR4	Maintainability	x			
QR4	Correctness	x			
QR5	Portability			x	

## E Class Diagram

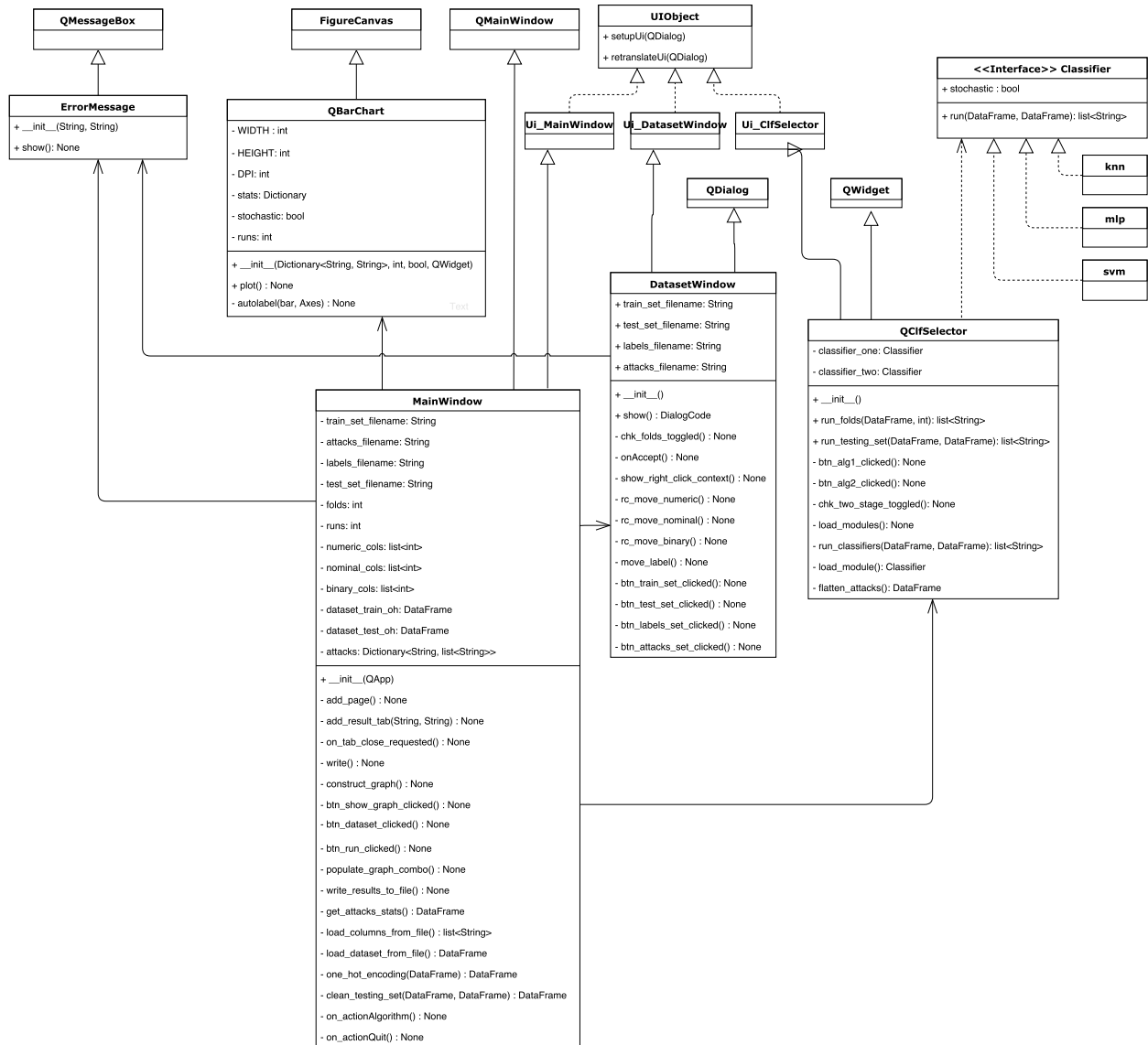


Figure 19: UML Class Diagram



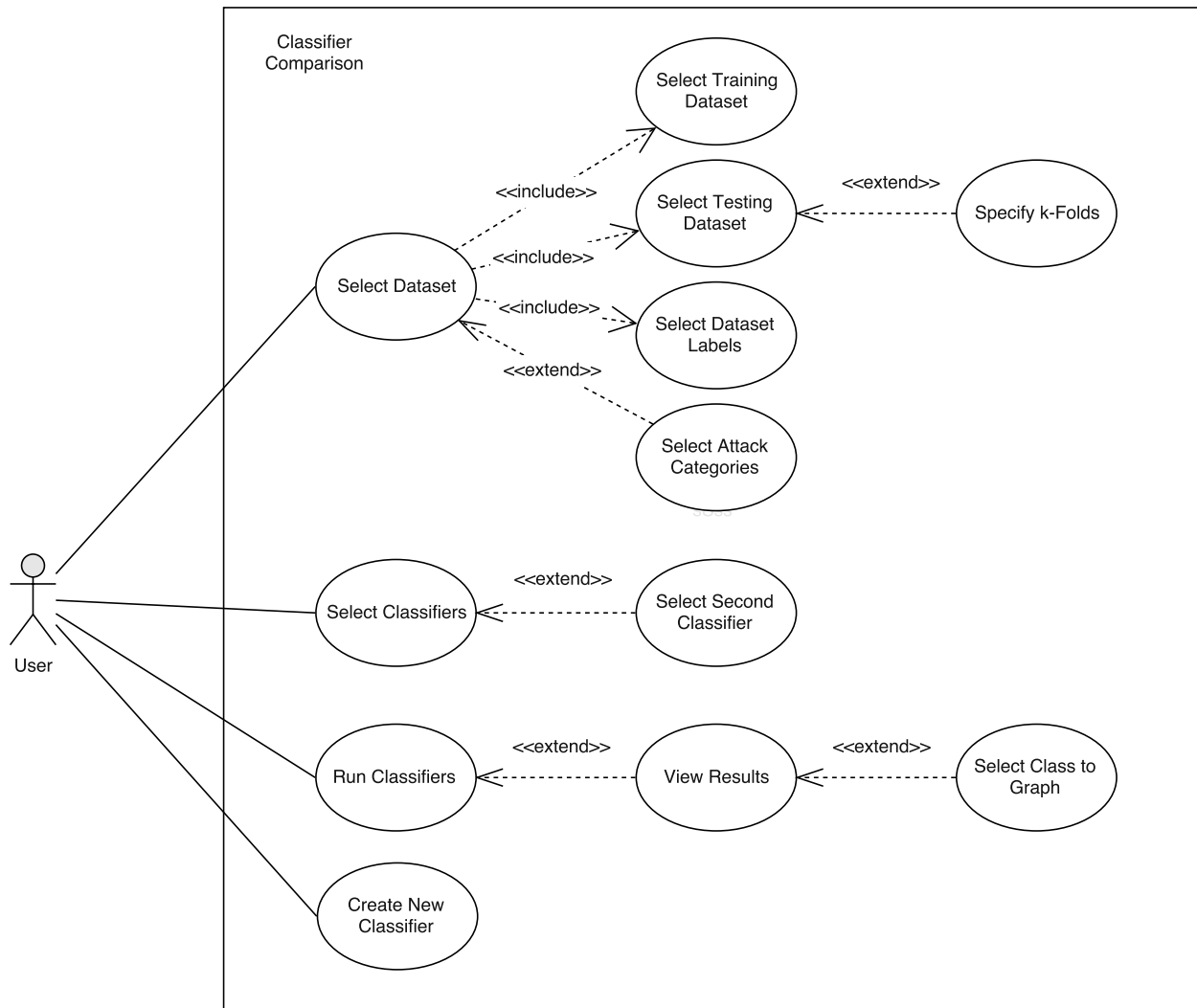
**F Use Case Diagram**

Figure 20: UML Use Case Diagram

**G NSL-KDD Features**

Table 8: NSL-KDD Features

Feature	Type
duration	continuous
protocol_type	symbolic
service	symbolic
flag	symbolic
src_bytes	continuous
dst_bytes	continuous
land	binary
wrong_fragment	continuous
urgent	continuous
hot	continuous
num_failed_logins	continuous
logged_in	binary
num_compromised	continuous
root_shell	binary
su_attempted	binary
num_root	continuous
num_file_creations	continuous
num_shells	continuous
num_access_files	continuous
num_outbound_cmds	continuous
is_host_login	binary
is_guest_login	binary
count	continuous
srv_count	continuous
error_rate	continuous
srv_error_rate	continuous
error_rate	continuous
srv_error_rate	continuous
same_srv_rate	continuous
diff_srv_rate	continuous
srv_diff_host_rate	continuous
dst_host_count	continuous
dst_host_srv_count	continuous
dst_host_same_srv_rate	continuous
dst_host_diff_srv_rate	continuous

dst_host_same_src_port_rate	continuous
dst_host_srv_diff_host_rate	continuous
dst_host_serror_rate	continuous
dst_host_srv_serror_rate	continuous
dst_host_rerror_rate	continuous
dst_host_srv_rerror_rate	continuous

---

**H UNSW-NB15 Features**

Table 9: UNSW-NB15 Features

Field	Type	Description
dur	Float	Record total duration
proto	nominal	Transaction protocol
service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service
state	nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, REQ, RST, TST, TXD, URH, URN, and (-) (if not used state)
spkts	integer	Source to destination packet count
dpkts	integer	Destination to source packet count
sbytes	Integer	Source to destination transaction bytes
dbytes	Integer	Destination to source transaction bytes
rate	Float	Connection rate
sttl	Integer	Source to destination time to live value
dttl	Integer	Destination to source time to live value
sload	Float	Source bits per second
dload	Float	Destination bits per second
sloss	Integer	Source packets retransmitted or dropped
dloss	Integer	Destination packets retransmitted or dropped
Sintpkt	Float	Source interpacket arrival time (mSec)
Dintpkt	Float	Destination interpacket arrival time (mSec)
sjit	Float	Source jitter (mSec)
djit	Float	Destination jitter (mSec)
swin	integer	Source TCP window advertisement value
stcpb	integer	Source TCP base sequence number
dtcpb	integer	Destination TCP base sequence number
dwin	integer	Destination TCP window advertisement value
tcprtt	Float	TCP connection setup round-trip time, the sum of synack and ackdat.
synack	Float	TCP connection setup time, the time between the SYN and the SYN_ACK packets.
ackdat	Float	TCP connection setup time, the time between the SYN_ACK and the ACK packets.
smean	integer	Mean of the flow packet size transmitted by the src

dmean	integer	Mean of the flow packet size transmitted by the dst
trans_depth	integer	Represents the pipelined depth into the connection of http request/response transaction
res_bdy_len	integer	Actual uncompressed content size of the data transferred from the servers http service.
ct_srv_src	integer	No. of connections that contain the same service (14) and source address (1) in 100 connections according to the last time (26).
ct_state_ttl	Integer	No. for each state (6) according to specific range of values for source/destination time to live (10) (11).
ct_dst_ltm	integer	No. of connections of the same destination address (3) in 100 connections according to the last time (26).
ct_src_dport_ltm	integer	No of connections of the same source address (1) and the destination port (4) in 100 connections according to the last time (26).
ct_dst_sport_ltm	integer	No of connections of the same destination address (3) and the source port (2) in 100 connections according to the last time (26).
ct_dst_src_ltm	integer	No of connections of the same source (1) and the destination (3) address in in 100 connections according to the last time (26).
is_ftp_login	binary	If the ftp session is accessed by user and password then 1 else 0.
ct_ftp_cmd	integer	No of flows that has a command in ftp session.
ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and Post in http service.
ct_src_ltm	integer	No. of connections of the same source address (1) in 100 connections according to the last time (26).
ct_srv_dst	integer	No. of connections that contain the same service (14) and destination address (3) in 100 connections according to the last time (26).
is_sm_ips_ports	binary	If source (1) and destination (3)IP addresses equal and port numbers (2)(4) equal then, this variable takes value 1 else 0

attack_cat	nominal	The name of each attack category. In this data set , nine categories e.g. Fuzzers, Analysis, Backdoors, DoS Exploits, Generic, Reconnaissance, Shellcode and Worms
labels	binary	0 for normal and 1 for attack records

---

## I NSL-KDD Results

### I.1 NSL-KDD Classification Data

Table 10: NSL-KDD Classification Data

	Class	TP	TN	FP	FN
KNN	dos	45811	709649	262	116
	u2r	22	503827	13	30
	r2l	949	1006750	39	46
	probe	10625	491238	998	1031
	normal	67068	58444	186	275
MLP	dos	45896	709862	49	31
	u2r	25	503831	9	27
	r2l	897	1006731	58	98
	probe	11479	492073	163	177
	normal	67171	58404	226	172
SVM	dos	45884	709860	51	43
	u2r	20	503836	4	32
	r2l	886	1006756	33	109
	probe	11163	492087	149	493
	normal	67208	58055	575	135
KNN/KNN	dos	45809	709648	263	118
	u2r	22	503825	15	30
	r2l	948	1006751	38	47
	probe	10629	491238	998	1027
	normal	67067	58446	184	276
KNN/MLP	dos	45890	709846	65	37
	u2r	20	503828	12	32
	r2l	947	1006689	100	48
	probe	11477	492025	211	179
	normal	67067	58446	184	276
KNN/SVM	dos	45882	709862	49	45
	u2r	18	503836	4	34
	r2l	938	1006728	61	57
	probe	11322	491788	448	334
	normal	67067	58446	184	276
MLP/KNN	dos	45811	709676	235	116
	u2r	21	503773	67	31
	r2l	861	1006730	59	134
	probe	10597	491321	915	1059

---

MLP/MLP	normal	67132	58355	275	211
	dos	45890	709840	71	37
	u2r	23	503825	15	29
	r2l	882	1006694	95	113
	probe	11479	492043	193	177
MLP/SVM	normal	67082	58387	243	261
	dos	45889	709847	64	38
	u2r	19	503837	3	33
	r2l	870	1006709	80	125
	probe	11310	491830	406	346
SVM/KNN	normal	67086	58384	246	257
	dos	45819	709661	250	108
	u2r	21	503739	101	31
	r2l	886	1006682	107	109
	probe	10618	491291	945	1038
SVM/MLP	normal	67026	58430	200	317
	dos	45901	709838	73	26
	u2r	26	503790	50	26
	r2l	893	1006683	106	102
	probe	11476	492014	222	180
SVM/SVM	normal	67026	58430	200	317
	dos	45895	709852	59	32
	u2r	20	503837	3	32
	r2l	886	1006763	26	109
	probe	11330	491708	528	326
	normal	67026	58430	200	317

---



## I.2 NSL-KDD Precision

Table 11: NSL-KDD Classifiers Precision Table

	dos	u2r	r2l	probe	normal
K-NN	0.91	0.17	0.69	0.9	0.99
MLP	0.94	0.38	0.79	0.98	0.99
SVM	0.93	0.64	0.83	0.99	0.99
K-NN/K-NN	0.9	0.18	0.73	0.91	0.99
K-NN/MLP	0.94	0.3	0.51	0.98	0.99
K-NN/SVM	0.93	0.58	0.64	0.96	0.99
MLP/K-NN	0.89	0.33	0.51	0.91	0.99
MLP/MLP	0.94	0.49	0.71	0.98	0.99
MLP/SVM	0.93	0.73	0.8	0.97	0.99
SVM/K-NN	0.9	0.21	0.44	0.91	0.99
SVM/MLP	0.94	0.37	0.49	0.98	0.99
SVM/SVM	0.93	0.65	0.83	0.96	0.99

### I.3 NSL-KDD Recall

Table 12: NSL-KDD Classifiers Recall Table

	dos	u2r	r2l	probe	normal
K-NN	0.92	0.18	0.59	0.88	0.99
MLP	0.95	0.34	0.57	0.98	0.99
SVM	0.93	0.36	0.6	0.95	0.99
K-NN/K-NN	0.9	0.18	0.58	0.88	0.99
K-NN/MLP	0.94	0.19	0.57	0.98	0.99
K-NN/SVM	0.9	0.21	0.55	0.97	0.99
MLP/K-NN	0.89	0.28	0.53	0.88	0.99
MLP/MLP	0.94	0.44	0.65	0.98	0.99
MLP/SVM	0.92	0.34	0.61	0.97	0.99
SVM/K-NN	0.93	0.29	0.56	0.88	0.99
SVM/MLP	0.98	0.51	0.57	0.98	0.99
SVM/SVM	0.93	0.36	0.6	0.97	0.99

#### I.4 NSL-KDD f1-Score

Table 13: NSL-KDD Classifiers f1-Score Table

	dos	u2r	r2l	probe	normal
K-NN	0.92	0.18	0.63	0.89	0.99
K-NN/K-NN	0.9	0.18	0.51	0.89	0.99
K-NN/MLP	0.94	0.23	0.54	0.98	0.99
K-NN/SVM	0.91	0.29	0.58	0.96	0.99
MLP	0.95	0.36	0.72	0.98	0.99
MLP/K-NN	0.89	0.24	0.51	0.89	0.99
MLP/MLP	0.94	0.46	0.67	0.98	0.99
MLP/SVM	0.93	0.45	0.68	0.97	0.99
SVM	0.93	0.45	0.69	0.97	0.99
SVM/K-NN	0.91	0.22	0.48	0.89	0.99
SVM/MLP	0.96	0.43	0.54	0.98	0.99
SVM/SVM	0.93	0.45	0.69	0.96	0.99

**J UNSW-NB15 Results****J.1 UNSW-NB15 Classification Data**

Table 14: UNSW-NB15 Classification Data

	Class	TP	TN	FP	FN
KNN	Analysis	223	79699	1956	454
	Backdoor	37	81204	545	546
	DoS	1022	74140	4103	3067
	Exploits	3600	59893	11307	7532
	Fuzzers	1187	70162	6108	4875
	Generic	18145	61858	1603	726
	Reconnaissance	1381	76733	2103	2115
	Shellcode	26	81562	392	352
	Worms	0	82250	38	44
	Normal	22678	39454	5878	14322
MLP	Analysis	2	81639	16	675
	Backdoor	5	81711	38	578
	DoS	427	77489	754	3662
	Exploits	8905	65394	5806	2227
	Fuzzers	3511	73422	2848	2551
	Generic	18271	62979	482	600
	Reconnaissance	2397	78026	810	1099
	Shellcode	82	81748	206	296
	Worms	5	82283	5	39
	Normal	34931	42501	2831	2069
SVM	Analysis	1	81638	17	676
	Backdoor	4	81717	32	579
	DoS	800	75907	2336	3289
	Exploits	8658	65292	5908	2474
	Fuzzers	2860	74563	1707	3202
	Generic	18264	62874	587	607
	Reconnaissance	2517	77986	850	979
	Shellcode	94	81785	169	284
	Worms	2	82279	9	42
	Normal	34588	42403	2929	2412
KNN/KNN	Analysis	203	79670	1985	474
	Backdoor	49	80915	834	534
	DoS	946	73280	4963	3143
	Exploits	4285	57926	13274	6847

KNN/MLP	Fuzzers	1090	71343	4927	4972
	Generic	18133	62259	1202	738
	Reconnaissance	1287	78306	530	2209
	Shellcode	13	81899	55	365
	Worms	0	82288	0	44
	Normal	22678	39454	5878	14322
	Analysis	1	81217	438	676
	Backdoor	6	81707	42	577
	DoS	349	77379	864	3740
	Exploits	5073	42	7256	6059
KNN/SVM	Fuzzers	63944	10210	7373	3590
	Generic	7256	23	205	686
	Reconnaissance	6059	19	1095	1717
	Shellcode	46	81712	242	332
	Worms	2	82286	2	42
	Normal	26790	35220	10112	10210
	Analysis	1	81471	184	676
	Backdoor	0	81747	2	583
	DoS	445	76114	2129	3644
	Exploits	5323	62928	8272	5809
MLP/KNN	Fuzzers	1828	70357	5913	4234
	Generic	18159	63251	210	712
	Reconnaissance	1786	77821	1015	1710
	Shellcode	50	81841	113	328
	Worms	0	82288	0	44
	Normal	26790	35220	10112	10210
	Analysis	235	79521	2134	442
	Backdoor	47	81069	680	536
	DoS	874	74930	3313	3215
	Exploits	5685	64706	6494	5447
MLP/MLP	Fuzzers	1202	72640	3630	4860
	Generic	18124	62581	880	747
	Reconnaissance	1277	78108	728	2219
	Shellcode	20	81852	102	358
	Worms	0	82288	0	44
	Normal	34508	42933	2399	2492
	Analysis	2	81640	15	675
	Backdoor	18	81641	108	565
	DoS	444	77396	847	3645
	Exploits	8978	65321	5879	2154
	Fuzzers	3750	73057	3213	2312

MLP/SVM	Generic	18304	63142	319	567
	Reconnaissance	2252	77716	1120	1244
	Shellcode	99	81709	245	279
	Worms	5	82285	3	39
	Normal	34412	43013	2319	2588
	Analysis	1	81640	15	676
	Backdoor	0	81747	2	583
	DoS	550	76358	1885	3539
	Exploits	9261	64580	6620	1871
	Fuzzers	3043	74085	2185	3019
SVM/KNN	Generic	18231	63166	295	640
	Reconnaissance	2113	77833	1003	1383
	Shellcode	91	81829	125	287
	Worms	2	82288	0	42
	Normal	34398	42820	2512	2602
	Analysis	235	79515	2140	442
	Backdoor	47	81063	686	536
	DoS	865	74938	3305	3224
	Exploits	5598	64417	6783	5534
	Fuzzers	1034	72506	3764	5028
SVM/MLP	Generic	18129	62557	904	742
	Reconnaissance	1244	78106	730	2252
	Shellcode	20	81838	116	358
	Worms	0	82288	0	44
	Normal	33701	42301	3031	3299
	Analysis	2	81637	18	675
	Backdoor	22	81602	147	561
	DoS	746	76997	1246	3343
	Exploits	8511	66106	5094	2621
	Fuzzers	3380	72380	3890	2682
SVM/SVM	Generic	18282	63069	392	589
	Reconnaissance	2264	77521	1315	1232
	Shellcode	91	81763	191	287
	Worms	5	82284	4	39
	Normal	33701	42301	3031	3299
	Analysis	1	81639	16	677
	Backdoor	3	81696	53	583
	DoS	656	75815	2428	4089
	Exploits	8647	64669	6531	11132
	Fuzzers	2880	74154	2116	6062
	Generic	18264	52841	620	18871

Reconnaissance	2350	77984	852	3496
Shellcode	114	81668	286	378
Worms	2	82267	21	44
Normal	33835	42675	2657	37000

---

## J.2 UNSW-NB15 Precision

Table 15: UNSW-NB15 Classifiers Precision Table

	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Recon	Shellcode	Worms	Normal
K-NN	0.1	0.08	0.22	0.23	0.19	0.95	0.7	0.18	0	0.73
MLP	0.13	0.13	0.38	0.62	0.57	0.98	0.71	0.3	0.49	0.93
SVM	0.06	0	0.23	0.59	0.6	0.99	0.69	0.44	0.1	0.9
K-NN/K-NN	0.1	0.06	0.19	0.24	0.19	0.95	0.62	0.14	0	0.73
K-NN/MLP	0.01	0.13	0.34	0.4	0.25	0.99	0.6	0.16	0.27	0.73
K-NN/SVM	0.01	0	0.17	0.39	0.24	0.99	0.64	0.31	0	0.73
MLP/K-NN	0.1	0.06	0.21	0.46	0.25	0.95	0.63	0.16	0	0.94
MLP/MLP	0.11	0.13	0.38	0.61	0.52	0.98	0.68	0.32	0.53	0.94
MLP/SVM	0.06	0	0.23	0.58	0.59	0.98	0.68	0.44	1	0.93
SVM/K-NN	0.1	0.06	0.21	0.45	0.22	0.95	0.63	0.15	0	0.92
SVM/MLP	0.11	0.14	0.38	0.61	0.47	0.98	0.65	0.28	0.53	0.92
SVM/SVM	0.06	0	0.23	0.57	0.53	0.98	0.66	0.39	1	0.92



### J.3 UNSW-NB15 Recall

Table 16: UNSW-NB15 Classifier Recall Table

	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Recon	Shellcode	Worms	Normal
k-NN	0.33	0.08	0.24	0.26	0.14	0.96	0.36	0.03	0	0.72
MLP	0.01	0.02	0.15	0.78	0.6	0.97	0.69	0.21	0.09	0.95
SVM	0	0	0.13	0.82	0.43	0.96	0.6	0.23	0.05	0.92
k-NN/k-NN	0.35	0.08	0.19	0.26	0.14	0.96	0.37	0.03	0	0.72
k-NN/MLP	0.01	0.03	0.13	0.43	0.4	0.96	0.52	0.14	0.03	0.72
k-NN/SVM	0	0	0.11	0.48	0.3	0.96	0.51	0.13	0	0.72
MLP/k-NN	0.35	0.08	0.21	0.51	0.2	0.96	0.36	0.05	0	0.94
MLP/MLP	0.01	0.02	0.16	0.79	0.61	0.97	0.65	0.26	0.1	0.93
MLP/SVM	0	0	0.13	0.83	0.5	0.97	0.62	0.24	0.05	0.93
SVM/k-NN	0.35	0.08	0.21	0.5	0.17	0.96	0.36	0.05	0	0.91
SVM/MLP	0.01	0.02	0.15	0.78	0.56	0.97	0.64	0.26	0.1	0.91
SVM/SVM	0	0	0.13	0.83	0.45	0.97	0.6	0.24	0.05	0.91

**J.4 UNSW-NB15 f1-Score**

Table 17: UNSW-NB15 Classifiers f1-Score Table

	Analysis	Backdoor	DoS	Exploits	Fuzzers	Generic	Recon	Shellcode	Worms	Normal
k-NN	0.16	0.08	0.23	0.25	0.16	0.95	0.48	0.05	0	0.73
MLP	0.02	0.03	0.21	0.69	0.58	0.97	0.7	0.25	0.15	0.94
SVM	0	0	0.17	0.69	0.5	0.97	0.64	0.3	0.09	0.92
k-NN/k-NN	0.15	0.07	0.19	0.25	0.16	0.96	0.46	0.05	0	0.73
k-NN/MLP	0.01	0.05	0.19	0.42	0.31	0.98	0.56	0.15	0.05	0.73
k-NN/SVM	0	0	0.13	0.43	0.26	0.98	0.57	0.18	0	0.73
MLP/k-NN	0.15	0.07	0.21	0.48	0.22	0.96	0.46	0.08	0	0.93
MLP/MLP	0.01	0.03	0.23	0.69	0.56	0.98	0.66	0.29	0.16	0.93
MLP/SVM	0	0	0.17	0.68	0.54	0.97	0.65	0.31	0.09	0.93
SVM/k-NN	0.15	0.07	0.21	0.48	0.19	0.96	0.45	0.08	0	0.91
SVM/MLP	0.01	0.04	0.22	0.68	0.51	0.98	0.64	0.27	0.17	0.91
SVM/SVM	0	0	0.17	0.68	0.49	0.97	0.63	0.3	0.09	0.91