PhD Thesis

Enhanced Classification of Network Traffic Data Captured by Intrusion Prevention Systems

Author: Reem Aljoufi

A thesis submitted in partial fulfilment of the requirement for the degree of Doctor of Philosophy, PhD.

Department of Computer Science
Middlesex University
London, United Kingdom

September 2022

Declaration and List of Publication

Signed: _____

Date: _____

Publication:

A research paper titled *Multi-task Learning for Intrusion Detection and Analysis of Computer Network Traffic* summarising some of the work carried out as part of this thesis was published at the *The 3rd International Conference of Computer Science and Renewable Energies (ICCSRE2020)*.

Abstract

A common practice in modern computer networks is the deployment of Intrusion Prevention Systems (IPSs) for the purpose of identifying security threats. Such systems provide alerts on suspicious activities based on a predefined set of rules. These alerts almost always contain high percentages of false positives and false negatives, which may impede the efficacy of their use. Therefore, with the presence of high numbers of false positives and false negatives, the analysis of network traffic data can be ineffective for decision makers which normally require concise, and preferably, visual forms to base their decisions upon. Machine learning techniques can help extract useful information from large datasets. Combined with visualisation, classification could provide a solution to false alerts and text-based outputs of IPSs.

This research developed two new classification techniques that outperformed the traditional classification methods in accurate classification of computer network traffic captured by an IPS framework. They are also highly effective. The main purpose of these techniques was the effective identification of malicious network traffic and this was demonstrated via extensive experimental evaluation (where many experiments were conducted and results are reported in this thesis). In addition, an enhancement of the principal component analysis (PCA) was presented as part of this study. This enhancement proved to outperform the classical PCA on classification of IPS data.

Details of the evaluation and experiments are provided in this thesis. One of the classification methods described in this thesis achieved accuracy values of 98.51% and 99.76% on two computer network traffic dataset settings, whereas the Class-balanced Similarity Based Instance Transfer Learning (CB-SBIT) algorithm achieves accuracy values of 93.56% and 96.25% respectively on the same dataset settings. This means the proposed method outperforms the state-of-the-art algorithm.

As for the PCA enhancement mentioned above, using its resulting principal components as inputs to classifiers leads to improved accuracy when compared to the classical PCA.

Acknowledgements

I would like to thank my family and friends who supported me and offered their encouragement during difficult times.

I would also like to express my gratitude to my supervisor, Dr Aboubakr Lasebae, who guided me throughout my PhD journey.

This Research is Dedicated

I am dedicating this thesis to some beloved people who have meant and continue to mean the world to me. First and foremost, to my beloved parents whose love and support for me knew no bounds and, who taught me the value of hard work.

Next, my brothers and sisters for their unconditional patience, love, support and encouragement.

Last but not least, to all my friends and teachers.

I love you all!

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

INTRODUCTION

This chapter provides an introduction to the work conducted in this thesis. It contains multiple aspects such as problem definition, the motivation and rationale of this work, research aims and objectives and more.

1.1 Introduction and Problem Definition

An Intrusion Prevention Systems (IPS) has a traditional role in computer systems of identifying security threats to a computer network (Mane and Rao, 2021). IPSs provide suspicious traffic alerts. The alerts are later used to analyse network events and act accordingly. Nevertheless, these alerts commonly contain relatively large numbers of false positives and false negatives, rendering the process of analysing them tedious to say the least. Consequently, the analysis of the alert files become insufficiently effective for decision makers to optimise their decisions on the threats faced by a network.

A more advanced way of considering security threats to a given network is by correctly and accurately classifying and visualising those threats. The notion here is to capture

network traffic, analyse it and decide whether it is safe or malicious. Security visualisation is the area of research and application whereby security threats are presented in visual manners to allow a more efficient way for security experts and network administrators to identify and analyse these threats.

Empowering IPS systems with accurate classification, threat identification techniques and visualisation makes them more capable of identifying harmful traffic that can be part of an attack. Attacks on computer networks can be highly costly.

This research focused on analysing network traffic data, in particular, IPS data. Such data are usually large. Hence, using an accurate classifier to perform data analysis is an advantage that not only saves time, but also increases chances of identifying attacks faster to avoid damage. The main aim is to provide accurate alerts to network administrators to react upon. This research therefore aims at developing new simple and easy to implement and yet highly accurate classification methods that are capable of identifying network attacks. These classification methods can be easily incorporated into any existing IPS platforms.

The use of data classification, or categorisation, will be the first step of the system. The importance of using classification stems from the large number of alerts an IPS usually produces. Using accurate classification methods will help reduce the number of false alerts (false positives and false negatives) and therefore produce better predictions and visualisation results. That will also assist security analysis in identifying security breaches.

As part of this research, various classification techniques such as RandomForest, NaiveBayes and Artificial Neural Networks were studied, analysed and tested and then compared with the developed classifiers. In addition to the new classification

methods, this study aims to develop a new variation method to the known principal component analysis (PCA) algorithm that is usually used for dimensionality reduction and data exploration. All implementations were carried out using Python and Scikit-learn (Pedregosa et al., 2011), which is an open-source Python package for machine learning.

1.2 Motivation and Rationale

The main motivation behind carrying out this research is the lack of fast, efficient and easy to implement classification methods that can accurately identify malicious network traffic. The situation becomes particularly harder when the amounts of available training data for such malicious attacks are scarce. In addition, cyber-attacks are a main concern for companies due to the high costs associated with these attacks (Morgan, 2020). Classical classifiers such as RandomForest, NaiveBayes, and Artificial Neural Networks (ANNs), to mention a few, do not perform well on network traffic data (this is explained in more detail in Chapter 3). Therefore, the main idea of this research is to enhance detection of attacks and risks aimed at computer networks through the integration of several factors: Intrusion detection and prevention, machine learning and visualisation.

Essentially, the Packet Capture (PCAP) files (or real-time captured network traffic) from an IPS of a Local Area Network (LAN) was analysed using machine learning and data mining algorithms. For this purpose, new classification methods and a new variation to an existing dimensionality reduction and data exploration techniques were proposed. Details of these methods are provided in the subsequent chapters in this thesis. The analysis provides a classification model that is used to classify new incoming network

traffic. Training of the model aims to minimise the numbers of false positives and false negatives generated by the IPS.

The PCA variation suggested in this research is used as a basis of a visualisation model to be adopted by network security experts to detect possible threats. The feedback obtained from the visualisation is then used to enhance the data mining classification model. A typical example scenario can be depicted as in Figure 1.1.

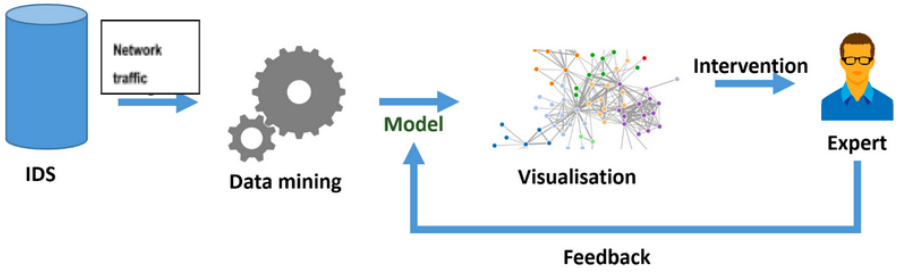


FIGURE 1.1: An improved attack detection method of computer network traffic using data mining and visualisation with human intervention

Figure 1.1 shows how to improve network attack analysis by implementing several facets. Here, the IPS, which provides the first level of data containing potential attacks and many other relevant and irrelevant details. This data will undergo data pre-processing and preparation techniques so that it is ready for data mining algorithms to generate a classification model. After the data have been prepared for analysis, a new classification method was implemented to accurately identify malicious traffic. Being able to accurately spot traffic that is unsafe is key in any IPS.

The variation to the classical PCA proposed in this thesis uses Spearman's rho (Spearman, 1904) instead of covariance coefficients. The variation helps in producing better visualisation as well as separation between network flows that belong to different categories (i.e. normal v. malicious traffic). This is demonstrated throughout the remaining chapters of this thesis.

The results of the above methods can be incorporated into an IPS and used as a basis for decision making as it can greatly support human observation in an intrusion detection/prevention system. This could help identify threats typically unnoticeable by machines.

1.2.1 Aims and Objectives

The aim of this research work is to improve network attack analysis by implementing several facets as described above. This includes developing new classification algorithms that can accurately identify malicious network traffic (especially when available training data is scarce). To achieve this aim, the following objectives are sought:

1. To investigate data mining techniques and tools available and evaluate their performance for network traffic data analysis (IPS data in particular).
2. To present and implement the two proposed classification techniques. Namely, the eigenvector-based classifier and the multi-task learning (MTL) classification method.
3. To investigate and evaluate existing the classical PCA algorithm and use it to perform classification and visualisation of network traffic data.
4. To test and evaluate the classification techniques using existing open-source machine learning platforms/libraries such as Weka or Scikit-learn.
5. To establish criteria for measuring system accuracy as a key step in the evaluation (choosing the correct evaluation metric is highly important) and then to conduct a comparison between the developed methods based on that.

A typical scenario of the system is explained in the following steps:

1. Data are captured by an IPS platform (such as Snort (Snort Documentation, 2022)).

2. Data are cleaned and prepared for data mining (e.g., transformed from PCAP to CSV or other formats).

3. Data are subjected to classification on a regular basis and a classification model is generated.

4. The classification criteria will be initially based on a training set of common attacks. The set will grow while more data are obtained.

5. The model is used to predict new network flows (i.e. a suitable evaluation procedure is followed and implemented)

6. Graphs are generated for visualising the data using the suggested PCA variation.

Demonstration of achieving the above-mentioned objectives is shown throughout the subsequent chapters of this thesis.

1.3 Research Questions

This research is focused on the accurate identification of malicious computer network traffic. This identification is done by analysing the traffic flowing through the network. This is possible after capturing the traffic, pre-processing and preparing it into a format suitable for further analysis by machine learning and data mining platforms and libraries. As such, the following are the main research questions the researched is poised to answer:

- The available network traffic data is suitable for multi-task learning (MTL). How would MTL improve the quality (i.e. accuracy) of predictive models? What would be a suitable similarity measure task in order to apply MTL?.

- Several well-known and commonly used classifiers exist (such as DecisionTree, NaiveBayes and Artificial Neural Networks). How would an MTL classifier perform compared to them?

- It is possible to develop a classification method based on matrix determinants. How would this classifier perform when used to classify network traffic data? How accurate is it when compared with existing traditional classifiers such as NaiveBayes and RandomForest?.

- The eigenvalues provide powerful information about their matrices (Mackey et al., 2005), which makes them a good foundation for a classification method. How would this method compare to the method in the previous point (i.e., determinants)? How would this classifier perform when used to classify network traffic? How accurate is it when compared with existing traditional classifiers such as NaiveBayes and RandomForest?.

- Similar to eigenvalues, eigenvectors provide useful information about their matrices and could be a good foundation for a classification method. When comparing vectors (flows) as matrix rows instead of scalars (eigenvalues), should this provide better accuracy as opposed to eigenvalues?

- Spearman correlation can be used in the PCA algorithm instead of Pearson correlation (Freedman et al., 2007). How would this improve PCA in terms of separating different classes? If the resulting PCA were used as input to a classifier,

how would the results improve when compared with the results of the original PCA?

1.4 Key Research Contributions

Here is a list of the key contributions this research offers:

1. Introducing a new variation to an existing regression method that is based on multi-task learning (MTL). The novel work in thesis uses the method not only in a classification context, but also in identifying shortcomings in the original method (which was published in late 2019 as can be found in (Sadawi et al., 2019)).
2. Developing a new effective classification method based on eigenvectors.
3. Introducing a variation to the PCA algorithm, which helps produce better clustered visualisation outcomes.

1.5 Methodology Overview

Research can be classified from three different perspectives:

1. **Field:** This is where a hierarchy of topics is built and the position of the research within it is identified. For example, the new classification technique(s) introduced in this thesis fall under the umbrella of *supervised machine learning*, which fall under the umbrella of *machine learning* which falls under the umbrella of *artificial intelligence*.
2. **Approach:** This is where the research method(s) employed as part of the research process are stated. Some examples include: case study, experiment, survey,

proof, quantitative/qualitative and so on. Work introduced in this thesis is an experimental and quantitative research approach.

3. **Nature:** Examples of the nature of research include: pure theoretical development, review of pure theory and evaluation of its applicability and applied research. This thesis falls under the umbrella of applied research.

The work explained throughout this thesis focuses on attaining two main goals, namely: enhanced classification methods and improved dimensionality reduction (via data visualisation and exploration) methods. Both methods aimed at the analysis of network traffic data captured by IPS platforms for the purpose of cyber-security.

This research is experimental based because it involves demonstrating the effectiveness of the proposed techniques by way of experimentation. In more detail, it falls under the umbrella of *Problem-solving studies* where a novel solution, or an improvement of an existing solution, is invented.

In addition, this research is quantitative because it uses numeric measurements (i.e. numeric features derived from computer network traffic data), it tests hypotheses and creates machine learning methods for induction (i.e. to make future predictions). Repeatability and accuracy in this research are of vital importance.

The first challenge was obtaining suitable data. Fortunately, a freely available dataset exists and, therefore, was used when running experiments and performing evaluation as part of this research thesis.

The second challenge was to develop robust classification approaches for efficiently and accurately identifying malicious network traffic, which could be used to indicate network attacks. The third challenge was to improve the classical principal component

algorithm (PCA) that is usually used for dimensionality reduction, and therefore allow data visualisation and exploration.

For those points, existing literature was reviewed, and limitations were listed and critically analysed. The findings indicated that there are existing approaches that try to address the above-mentioned problems, but not to the finest forms when it comes to network traffic data (more on this in Chapters 2 and 3). The methods developed by this research add new contributions to the field as explained in detail in this thesis.

For the development methodology, the agile software development methodology was used due to its flexibility in terms of producing a working model of the system at early stages, which is later enhanced by several iterations. The approach used was to test some of the existing techniques/algorithms for security visualisation and data mining classification and suggest possible improvements with regard to the capability of the individual technique to cater for growth of data.

The results of the tests were documented, and a comparative analysis was conducted as will be discussed in Chapters 3 and 6. At the next stage, selected techniques underwent enhancements. Once the enhanced technique(s) was obtained, it was tested with the same benchmark data and a comparative analysis was conducted. If the technique proved effective in terms of some threshold, it was implemented as the underlying algorithm for a security visualisation system, otherwise further enhancements were undertaken. The system was critically analysed based on extensive test results.

1.6 Thesis Structure

The following points provide a summary of the chapters that comprise this thesis:

- **Chapter 1:** This chapter provides an introduction to the work conducted in this thesis. It contains multiple aspects such as problem definition, research aims and objectives and more. In Chapter 2, some background and subject area review are presented.

- **Chapter 2:** This chapter provides an overview of the IPS systems, how to visualise and analyse the data they collect. The chapter also provides an overview of machine learning, multi-task learning, and why multi-task learning is a suitable method to tackle the problem at hand. In addition, it provides a review of existing literature on traditional machine learning methods, multi-task learning methods and principal components analysis based methods for network traffic analysis.

- **Chapter 3:** This chapter provides some initial experiments run to evaluate the performance of several traditional classifiers on network traffic data. The chapter starts with an overview of several metrics used for evaluation of classification methods. The main aim is to check whether some better classification methods are required to accurately classify data of this nature (i.e. network traffic data).

- **Chapter 4:** This chapter provides an overview of two new classification methods developed as part of this research. The chapter explains how they work and shows Python code snippets when possible. It starts with the multi-task learning classification method and then moves to the eigenvalue based classifier. Evaluation of these two techniques is detailed in Chapter 6.

- **Chapter 5:** In this chapter a proposed enhancement of the classical PCA algorithm is provided. Before doing this, an overview of the PCA algorithm and explanation of how it works is introduced. After that, an explanation of how PCA

can be enhanced and used for pre-processing and visualisation of network traffic data is detailed.

- **Chapter 6:** This chapter contains the experiments (in order to evaluate the performance of the introduced methods) and results as well as discussion of those results. Here the evaluation of the methods proposed as part of this thesis will demonstrated. The chapter begins by speaking about the data used in the evaluation process and then moves to a detailed evaluation of the proposed MTL method for classification. After that, the evaluation of the Eigenvector Perturbation (EV) classifier is explained. The experimental results show that the proposed methods are indeed effective.

- **Chapter 7:** This chapter provides a high level overview of the work carried out in this thesis and discusses the conclusions, lessons learned as well as lists several possible ways to extend and improve this work.

This chapter provided an introduction to the work conducted in this thesis. It contained multiple aspects such as problem definition, research aims and objectives and more. In the next chapter, some background and subject area review are going to be presented.

Chapter 2

BACKGROUND AND SUBJECT AREA REVIEW

Before delving into the details of the algorithms used and developed as part of this research, this chapter provides an overview of the data captured by IPS systems, how visualisation is used in them and how mining of the data they collect is performed. The chapter also provides an overview of machine learning and multi-task learning (and why multi-task learning is a suitable method to tackle the problem at hand). In addition, it provides a review of existing literature which is a significant part of this thesis.

2.1 Overview of Intrusion Prevention Systems

An intrusion prevention system (IPS), sometimes referred to as an intrusion detection system (IDS), is a software system that monitors an individual application, host, or an entire network for suspicious activities and provides alerts accordingly. Intrusion detection is a security technology that aims to identify threats against a target host or

application and does not actively block network traffic. If the system is provided with the ability to respond to a detected threat, it is known as an intrusion prevention system (IPS). A network-based IPS attempts to detect unauthorised or anomalous behaviour in network traffic. A network IPS typically uses a network Terminal Access Point (TAP) (Akashdeep et al., 2017), Switch Port Analysis (SPAN) (Smys et al., 2020), or a repeater (hub) to collect packets transferred across a given network and analyse the data (Cisco, 2019).

According to (Jacob and Wanjala, 2018), there are different types of intrusion prevention/detection systems. For example, there are signature-based systems that work by maintaining a database of signatures of known attacks and comparing newly captured data with those signatures to try and identify potential attacks. Another type is the anomaly-based systems, which use statistical models to build a description of what normal network traffic looks like. They use such models to spot traffic that deviates from them and label that traffic as *abnormal*. The focus in this thesis is on using IDS platforms because they are commonly used so they can be a reliable source of network traffic data (Sawant, 2018).

2.2 Visualisation of IPS Captured Data

The network traffic data captured by an IPS system are usually in a binary format known as of type PCAP (for Packet CAPture). These data require transformation into a textual format such as CSV (Character Separated Value) files which can be easily processed by data visualisation and analysis platforms. A practical and systematic way for transforming PCAP files into CSV files is reported in (Alothman, 2019*a*). PCAP files are usually huge in size and the amount of raw data generated by an IPS can quickly

overwhelm security analysts who work on analysing these data to identify malicious patterns in the data flow (Freet and Agrawal, 2017).

A potentially effective approach to examine possible risks in a large amount of multidimensional data provided by an IPS is PCAP content visualisation. The use of graphical illustrations of the network traffic could provide an insight into vulnerability and security issues that are not typically machine recognisable. The graphical approach to network security is usually referred to as security visualisation (Liu, Sun, Fang, Liu and Yu, 2014). Security visualisation is essentially the graphical representation of objects of interest, particularly the threats to a computer system. Some of the uses of security visualisation include malware analysis, reverse engineering, digital forensics, anomaly detection, and many others (Sarigiannidis et al., 2015). A simple example of a security visualisation use is a 2-dimensional graph representing data flow on a computer network against time, marking potential threats in a different colour.

There are third party applications that support reading the packet files and produce graphical representations of the captured data. Some examples of IPS visualisers include EtherApe (EtherApe, 2020), which is a Linux-based package that can read live traffic or packet capture files, and renders information in a graphical format. It uses colour coding to differentiate information of interest and supports name resolution. Another application developed by the University of Maryland is NetGrok (NetGrok, 2009). NetGrok is Java-based and supports real-time monitoring and reading of packet capture files, and performs colour coding to identify relevant information. TNV is another Java-based data visualiser. It provides relevant information by associating remote hosts to local hosts (Goodall, 2009).

How to Visualise PCAP Data Files

As mentioned previously, PCAP data are binary and need to be transformed into a tabular format such as CSV so it can be easily processed. An open-source tool to transform PCAP data into CSV is the CICFlowMeter (Draper-Gil et al., 2016) and (CICFlowMeter, 2018). This application extracts a number of useful features from the data. After obtaining the data in CSV format, one possible way to visualise it is to use principal component analysis (PCA) and then plot the scores (or the principal components, PCs) against each other. For example, it is common to generate a scatter plot of PC1 versus PC2. This way it is possible to colour instances based on their class (or category) and check if there exists any separation between instances from different classes. Figure 2.1 shows the first two principal components plotted against each other after applying PCA to part of the open-source IPS dataset available from the Canadian Institute for Cybersecurity (Canadian Institute for Cybersecurity, 2017). The plot shows that there is a significant overlap between instances from different classes. It is important to highlight that the variation to PCA suggested in this research aims to achieve a better separation between instances from different classes.

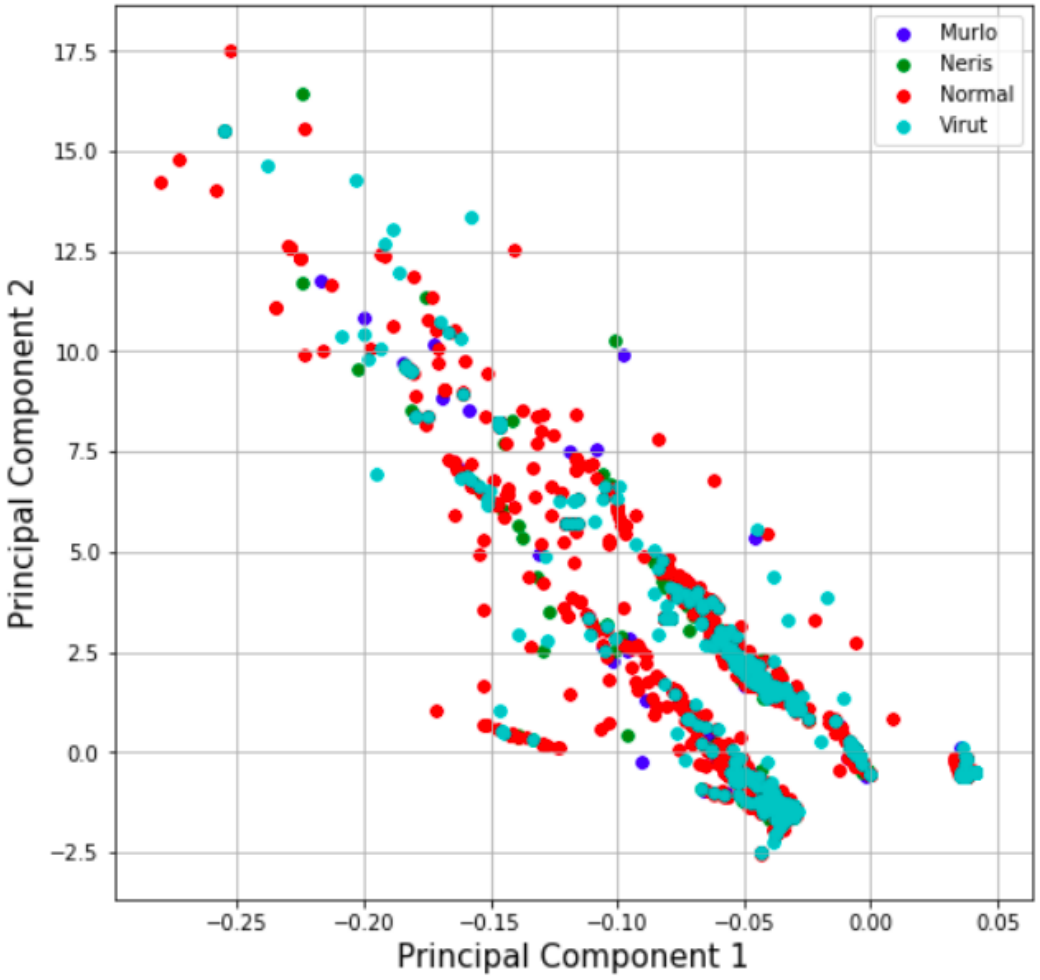


FIGURE 2.1: PCA of IPS data. The figure shows PC1 v PC2. The legend shows the types of attacks

2.3 High Level Overview of Snort PCAP Files

Snort is a popular IPS, therefore, this section provides a generic overview of the data it captures. It captures network traffic data in PCAP file format, which is a binary format that can be transformed into text-based, human-readable file format. These files could be manually scrutinised by a system admin to validate their content. An extract of a .pcap entry content is provided below:

2016-08-04 12:30:00 [1:2102924:4] GPL NETBIOS SMB-DS repeated logon

failure [Classification: Unsuccessful User Privilege Gain] [Priority: 1] TCP 192.168.229.123:445 -> 192.168.202.79:551

The alert above provides four pieces of information:

1. The date and time of the threat (in green);
2. The type of the threat, which is a repeated logon attempt from a DMZ machine to a LAN machine (in red);
3. Assigned priority is 1 (in orange); and
4. The attack was initiated by a machine with IP address 192.168.229.123 on port 445 to a machine with IP address 192.201.202.79 on port 551 (in magenta).

2.4 What is Data Mining?

Before delving into the details of how to apply data mining techniques for the analysis of PCAP files, it makes sense to define what data mining is. According to (Han et al., 2012), a reasonable definition can be as follows:
Data mining is a process that involves turning raw data into useful information by using techniques from several fields such as machine learning, statistics and databases. It can be considered an interdisciplinary branch of computer science.

The process can involve multiple steps such as data loading, transformation, cleaning, selection and integration. In addition, it can also include processes such as pattern evaluation and knowledge representation. Many of these steps are employed as part of the work developed in this thesis.

A definition of machine learning is provided in Section 2.6.1. As part of this thesis, an evaluation of several classical machine learning algorithms is performed in Chapter 3.

2.5 Data Mining of PCAP Files

There are various data mining techniques that can be used to analyse PCAP files. For example, classification can be used to classify attacks targeting certain ports. Another possible type of classification could be based on the geographical area where frequent attacks are launched.

Classification is the process of predicting a category of a given input. On the other hand, when a real value is being predicted the process is known as regression.

Classification is not the only technique that may be used for mining the PCAP files; other techniques may also be applied. For example, association-rule mining can be used to associate certain types of attack with certain ports. Pattern recognition may also be used to identify if certain attacks have common patterns, which can help recognise and prevent similar attacks in the future (Jakub and Branišová, 2015). The data mining techniques used help decrease the number of alerts by grouping similar alerts into one set and display it in a visual format. For example, the set of attacks from one source IP address can be represented by one object on a graph. There are various algorithms available for each data mining technique. For example, the C4.5 classifier based on the decision tree algorithm is widely used in practice. Decision trees form a straightforward and simple way for classifying data. A decision tree classifier makes its classification decision based on a series of questions about the attributes of the data. Each time an answer is received, a follow-up question is posed until a conclusion about a class label is reached (Cherfi et al., 2018). Classification training is performed to generate a model,

which is used to classify future data. This scenario is depicted in the diagram below (Figure 2.2).

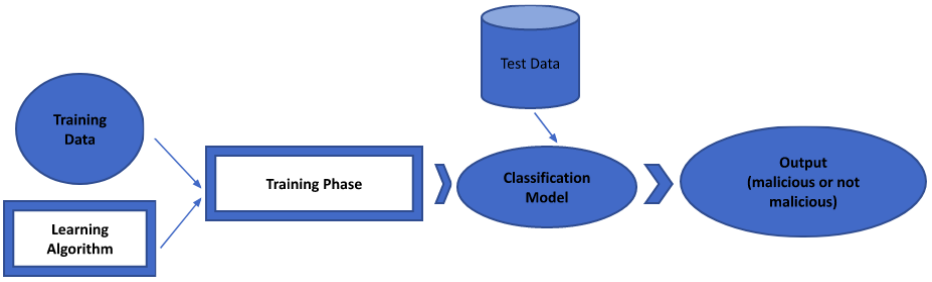


FIGURE 2.2: Classification procedure.

The model is created by using an initial dataset known as the training data. In our case the training data is the network traffic data pre-captured by an IPS and pre-processed so that it can used by machine learning platforms and algorithms. The classification model is created after the training process. In other words, the classification algorithm is trained on the training dataset to create a model that can be used to make future predictions when given data. Another dataset, called the test dataset, is used to test the quality of the model in terms of how good its predictions are. As mentioned, it should be remembered that data needs to be made ready for machine learning and data mining. This is because data captured by IPS platforms is not usually in a format understood by machine learning platforms and algorithms. A practical and systematic way for transforming data captured by IPS into machine learning suitable format is reported in (Alothman, 2019*a*).

2.6 Machine Learning and Multi-task Learning

It is logical to introduce what machine learning is about and the specific field that some of the work introduced in this thesis falls under. This section focuses on familiarising

the reader with these topics and providing useful resources for extra reading. How these methods and technique are used is explained in detail in the coming chapters of this thesis.

2.6.1 What is Machine Learning?

Machine-learning is the field that focuses on enabling machines (i.e. computers, hand-held devices and so on) to automatically learn and improve from experience without being manually, or explicitly, programmed to do so (Bruce et al., 2020). Experience here means data and usually learning is achieved via algorithms (of which there are plenty). Normally the learning is carried out for the purpose of explaining the past and/or predicting the future. Predicting the future means making predictions of new values or categories based on known values and categories of existing data.

Machine learning can be viewed as a method to achieve artificial intelligence (AI), which is the field that focuses on enabling machines to learn, reason and act for themselves.

It has to be said that machine learning is a large field, and the reader is referred to the book in (Kubat, 2015) for more details. In this thesis, the work is primarily focused on *classification*, which is the process of predicting a category of a given input (when predicting a real value the process is known as *regression*).

2.6.2 What is Multi-task Learning (MTL)?

According to Caruana (Caruana, 1997) which is the first known work on MTL as a machine learning technique, multi-task learning, or MTL for short, is a method for improving model generalisation by means of learning related tasks jointly. The core idea is to use a form of representation that is shared by these related tasks so that the

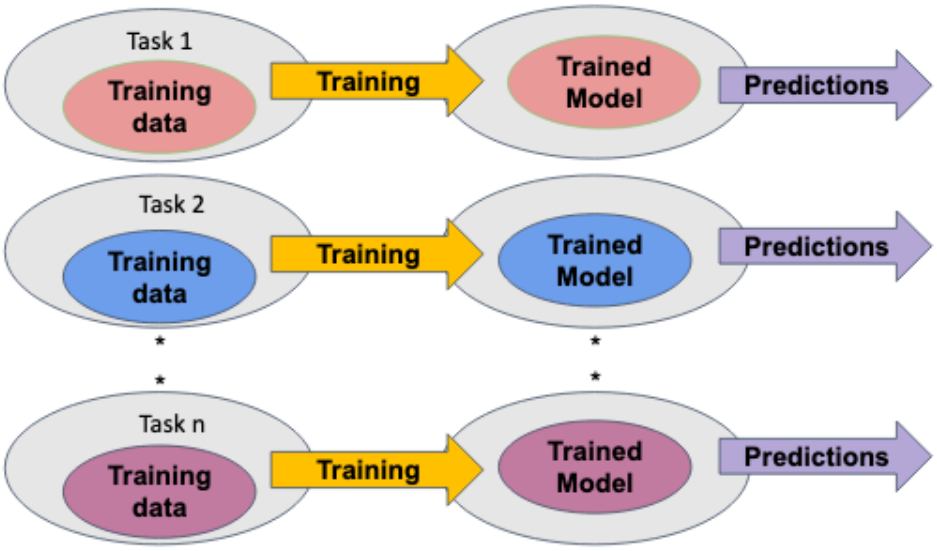
joint learning can occur. This has multiple benefits such as *inductive transfer*, which is the effect that happens internally when tasks learned together can benefit from each other and the ultimately resulting models are of better quality than when the tasks are learned individually (Sadawi et al., 2019) (which is known as single-task learning, or STL for short). A visual illustration of MTL and STL is shown in figure 2.3. Hence, MTL is by definition a branch of transfer learning (Weiss et al., 2016), or TL for short, where some form of learning is performed on tasks where there are large amounts of data (such tasks are known as the *source tasks*) and this learning is exploited to enhance learning for tasks where data is scarce (such tasks are known as the *target tasks*).

However, it is important to highlight that in MTL the tasks are learned together at the same time instead of learning from some tasks in one stage and then transferring the knowledge learned to other tasks in another stage. A good introduction to MTL with useful examples can be found at (Zhou1 et al., 2012) and in (Standley et al., 2020).

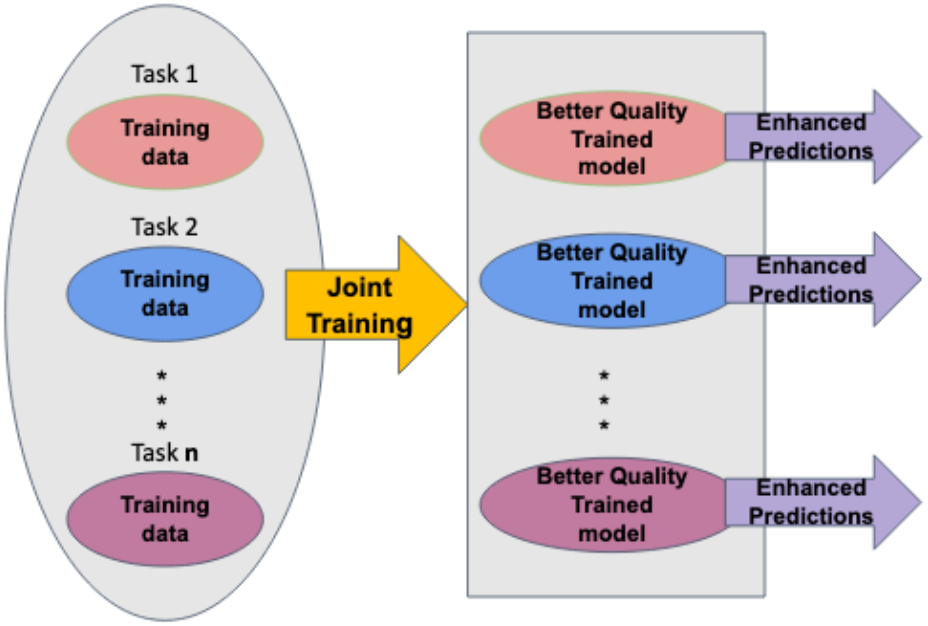
2.6.3 How MTL is used in this Work

There are several purposes of Cyber-attacks. These usually include the access, change or destruction of sensitive information. Such attacks can also be launched to extort money from people or interrupt businesses. Hence, attacks on computer networks can be highly costly and the number of approaches that attempt to identify malicious network traffic by performing traffic analysis is increasing as time goes on. This is because the number of cyber-attacks keeps increasing. This means there is a need for better techniques to be invented to protect computer devices and networks (Dasgupta et al., 2020).

Because it is possible to capture computer network traffic and transform it into a format suitable for machine learning, there exists some machine learning techniques for tackling



(A) Single-Task Learning (STL)



(B) Multi-Task Learning (MTL)

FIGURE 2.3: Visual Illustration of Single and Multi-task Learning

this problem. This thesis introduces a new machine learning method for the accurate detection of malicious traffic when existing training data is scarce. The method is a multi-task learning technique where datasets belonging to different network traffic types and attacks (i.e. tasks) are used together in the learning process instead of learning each task separately.

Network traffic data belonging to different types of traffic are related because they are from the same domain (the domain is network traffic and each attack can be considered a separate task). Therefore, and despite the possibility that they can have different distributions, it is intuitively appealing to take advantage of the relationships between various datasets belonging to different cyber-attacks in the machine learning process. This is one of the contributions that this thesis makes and this is explained in more detail in Chapter 4.

2.7 Contributions of this Multitask Learning Work

The proposed MTL method in this work is inspired by a recent work that can be found at (Sadawi et al., 2019). Although it was applied in the field of drug discovery, it is possible to adopt, enhance and apply it in other fields such as the field of automatic analysis of computer network traffic. It is important to highlight the differences between the method introduced in this paper and the above-mentioned method. The method introduced in this thesis differs in the following ways:

- The existing method measures the similarity between tasks using amino acid sequences whereas the method described in this thesis uses instance similarity.
- The existing method is used in a regression context (i.e. to predict a real number) whereas the method in this thesis is classification (i.e. to predict a category or class) and therefore the problem setting and evaluation metrics/procedures are different.
- As part of the research carried out in this thesis, a weakness in the existing method has been identified. Because it attempts to address a regression problem, it is not

straightforward to distinguish between instances from different tasks when the datasets are merged. Therefore, it adds a task ID column as a new feature (each task is given a task number) and this can mislead the classifier. This is because when an integer number (such as 1) is assigned to task A and another number (such as 2) to task B, mathematically 2 is greater than 1, but does this mean that task B is larger than task A? not necessarily. A classifier does not know that and it will use the task ID as a number and not as a category (which introduces order between tasks and that does not really exist).

- This is a weakness in the existing technique that the work introduced in this thesis does not suffer from because the class label is used to distinguish between instances from different tasks. Instead of assigning numerical values to categorical values, a technique called *one-hot encoding* should be used to correctly represent categorical data numerically (Brownlee, 2020)

Based on the above, the contributions of the work described in this thesis can be summarised as follows:

1. Develop a multi-task learning (MTL) approach for the accurate identification of malicious network traffic (even in case of scarce data)

2. Instead of learning the similarity between tasks, it is quantified in this work by measuring it using a well-known similarity metric (namely, the cosine similarity (Deza and Deza, 2009))

3. Identify a weakness in a recent existing method and demonstrate how the method described in this thesis avoids it

4. Experimentally show that the method introduced in this thesis outperforms existing widely used classifiers in different situations.

2.8 Related Work on using Traditional Machine Learning Methods for Network Traffic Analysis

The number of approaches that attempt to identify malicious network traffic by performing traffic analysis is increasing on a continuous basis. This is because the number of cyber-attacks is also increasing. A survey can be found in the work of (Silva et al., 2013). Below is a review of some of the approaches proposed in the literature.

A recent approach that uses machine learning to identify botnets is proposed by (Alothman, 2018*b*, Alothman et al., 2018). The approach uses transfer learning (Torrey and Shavlik, 2009), which is a branch of machine learning that attempts to learn from tasks with plenty of data to boost performance of machine learning models built for tasks with much less data. The approach is named SBIT (Similarity Based Instance Transfer). The approach exploits the similarity between instances from various types of network traffic to increase the number of instances in a dataset that has a low number of instances. The idea is to check if there are any highly similar instances in large datasets (similar to instances in the dataset at hand) and copy those instances over to a small dataset and assign them a new label (the same label of the instances in the dataset at hand). The large datasets are usually called the source datasets and the small dataset is usually called the target dataset. This way more data are obtained and the assumption of the idea is that it will lead to more accurate models.

The above model was extended in (Alothman et al., 2018), where a closer look at the data was taken to make sure the dataset resulting after instance transfer is class balanced. In other words, the SBIT approach copies instances without paying any attention to which class they belong to. On the other hand, the class-balanced SBIT (CB-SBIT) makes sure that the new dataset contains an approximately equal number of instances in each class. This way it is easy to avoid problems that result from using imbalanced datasets, such as overfitting and overoptimistic models. Some of the weaknesses of this approach is that it is very computationally expensive because it is iterative. In other words, it needs to loop through all input datasets (i.e. the source datasets) and check for the similarity between instances in them and instances in the dataset that should be improved (i.e. the target dataset). The loops go through all instances one by one, which could take a long time. Also, it is not clear which similarity measure is the most suitable one as there is a large number of possible similarity measures. Because this is a recent approach that has reported achieved high accuracy, it is going to be used in the evaluation stage of the MTL classification method developed as part of this thesis.

The approach proposed by (Jadidi et al., 2013) uses a multi-layer perceptron (MLP) to model and analyse network traffic data. Two different optimisation techniques were used in this method to optimise the neural network weights. Another technique that uses Artificial Neural Networks (ANNs) to analyse network traffic data is presented by (Beghdad, 2008).

In addition to the two previous methods, the method reported in (Abdulla et al., 2014) uses a two-stage method for the process of data analysis. The first stage focuses on predicting if the network traffic is safe or malicious. The second stage is concerned with identifying the type of attack in case of malicious traffic. ANNs can be seen as suitable techniques but they suffer from multiple drawbacks such as the need for high

computational power and the difficulty in explaining and interpreting their results. Also, because they start with random weights, they can converge to different values each time they are run.

The method reported in (Venkatesh and Anitha, 2016) also analyses network traffic to identify harmful traffic. According to the report, many high-impact features are extracted from network flow data and these features can be fed into commonly used classifiers such as decision trees, support-vector machine and NaiveBayes. Although the authors claim that their method is suitable for encrypted data because the features are not dependent on packet contents, it is not clear what these features are and how much impact they make.

The K-Nearest Neighbours (KNN) classifier has also been used for the analysis of network traffic. An example effort that employs the KNN classifier is described in (Costa et al., 2015). The reported technique is primarily a clustering technique that attempts to cluster similar traffic flows together (i.e., based on the assumption that similar traffic types will result in the same cluster and therefore malicious traffic can be identified).

Support vector machines (SVMs) are powerful classifiers, and it comes as no surprise that they have been used for network traffic. They are particularly powerful for binary classification (i.e., in identifying if traffic is safe or harmful). This is confirmed in the review reported in (Liao et al., 2013). The approach reported in (Yuan et al., 2010) is based on SVMs as it employs a feature selection algorithm and then trains an SVM classifier on the data to make future predictions. It is also possible to use a one-class SVM to perform outlier detection. This way malicious traffic can be spotted as outlier when compared to safe traffic and that was done in the approach reported in (Winter et al., 2011). The weaknesses of these approaches in general can be summarised in the

fact that it is not always possible to consider the malicious traffic identification problem as a binary classification problem and the fact that looking at the problem as an outlier detection problem can lead to unexpected results as it heavily relies on the training data, which can contain traffic from multiple attacks as well as safe traffic.

Evaluating the performance of a number of commonly used classifiers such as ANNs, SVM and Random Forest is reported in (Stevanovic and Pedersen, 2014). The idea is to use features extracted from network traffic flows and see which classifier exhibits the best performance. The results show that promising performance was obtained after using only two (of a total of eight) classifiers. In general, Random Forest and its base classifier, i.e., the decision tree classifier were the best performers.

Decision trees (DTs) are popular classifiers because they perform well in many tasks and because of their interpretability. One work that uses DTs is the study by (Zhuo et al., 2013), where several classifiers were experimented with and the general outcome was to use DTs. This is because they perform better than other classifiers when the data change in real time, which is the nature of network traffic data. Using decision trees also reduces the risk of having models of high complexity. In addition, DTs were also used in the work of (Kumar et al., 2012) where the way data is split at each node inside the DT is modified. The authors assert that they ensure that the classifier is not biased by performing feature selection that is based on information gain then choosing a value to split the data. Also, the work of (Haddadi et al., 2014) uses DTs (combined with genetic programming) to perform network traffic data analysis. An interesting aspect of this approach is that it only extracts features from packet headers which means no payload data is used (so the method is suitable for encrypted data).

Other works that use transfer learning techniques can be found in (Zhao et al., 2017)

and (Tan et al., 2018). The first approach uses what is known as feature transfer where a latent shared feature space was obtained by projecting features from input datasets into this space and then using the resultant features for analysis and predictions. The second approach evaluates the performance of an existing freely available transfer learning algorithm known as TrAdaBoost (Dai et al., 2007).

2.9 Existing Work on using MTL for Network Traffic Analysis

One of the most recent approaches is the work in (Sadawi et al., 2019) where MTL was used to obtain better predictive models in the field of drug discovery. The problem domain was focused on predicting the binding of small molecules (i.e. potential drugs) on human body proteins (each protein is considered a task). This binding is quantified using a real number, and therefore the problem setting was regression. As for task similarity, although there is existing work that focuses on learning similarity between tasks (Ben-David and Borbely, 2008, Shui et al., 2019), where attempts are made to automatically quantify the similarity between tasks instead of using manually provided similarity values, the reported method exploits the amino-acid sequence of each protein and uses it to compute the similarity between proteins. This similarity is then assumed to be the task similarity and the approach combines datasets from various proteins into one large dataset and adds the similarity values as new features. Although the method exhibits significantly improved results over single task learning, it suffers from weaknesses (and differs from the work introduced in this thesis) as listed in Section 2.7.

MTL is an active research area in deep learning (i.e. deep neural networks). An example configuration is known as *hard parameter sharing* where hidden layers are shared between

multiple tasks while task-specific output layers are kept separate (this is the setting designed by Caruana (Caruana, 1997)). This setting was shown to reduce overfitting as explained in (Baxter, 1997). Overfitting is the problem that occurs when a trained model performs well on the training data and poorly on unseen data. Another configuration is known as *soft parameter sharing* where each task keeps its own model (and parameters) and regularisation is used to ensure the parameters of separate models are similar (Duong et al., 2015, Yang and Hospedales, 2016). A recent survey of MTL in various deep learning configurations can be found in (Vandenhende et al., 2020).

A recent approach that uses MTL for Network Traffic Classification is reported in (Rezaei and Liu, 2020*a*). It is a deep learning technique that is developed for network bandwidth and duration prediction tasks. Another recent deep learning MTL technique for the classification of network traffic can be found in (Huang et al., 2018). The two methods are based on deep neural networks and they suffer from the usual deep learning drawbacks such as the need for large amounts of training data and the need for extensive compute power and resources. For example, in (Huang et al., 2018) a significant amount of data pre-processing is required as the data needs to be picturized before feeding it into the deep learning architecture (i.e. data must be transformed into a picture-like 2D representation to be suitable for convolutional neural networks CNNs). This is because CNNs are known to work well for image analysis and object recognition (Brownlee, 2019). Also, in (Rezaei and Liu, 2020*a*) the data is seen as a time-series and a 1D CNN architecture was used. The authors have selected only three features to use as inputs and no reason was provided for this. This is not the case with the approach described in this thesis as it simply concatenates datasets from different network traffic types and add the similarity values to obtain extra features (this is explained in detail in Section 2.6.2. It is worth mentioning here that deep learning architectures for multi-task and transfer

learning in the area of computer network traffic analysis are not difficult to exploit as they have been shown to contain security vulnerabilities (Rezaei and Liu, 2020*b*).

As stated in Section 2.6.2, MTL can be considered as a way of achieving transfer learning (TL). TL was used in the field of cyber-security for the identification of malicious network traffic. An example is the similarity-based instance transfer (SBIT) work in (Alothman, 2018*b*) and, its extension, the class-balanced similarity-based instance transfer (CB-SBIT) (Alothman et al., 2018). The reported approaches use a brute-force like method to copy instances from tasks with plenty of data to similar tasks with small amounts of data. The copying process is performed only for similar instances. In other words, the similarity of instances in a small dataset and a large dataset is measured (assuming each dataset belongs to a different type of network attack) and whenever highly similar instances (to the instances in the small dataset) are found in the large dataset are found, they are copied to the small dataset so that its size increases (the copied instances are assigned the class of the dataset they are copied to). This is performed and then learning is performed using the newly increased in size dataset. Experiments reported by the authors show improvement in model predictions. Work introduced in this thesis differs from this approach in more than one aspect. In more detail, the work introduced in this thesis employs MTL so that no instances are copied from large datasets to smaller datasets. This can give rise to a problem especially when no form of regularisation or weighting is used to control how much the copied instances should contribute to the target dataset (currently they are given the same weight as the original instances and this is not reliable). The approach in this thesis uses MTL to learn from the datasets as they are without changing them and this is advantageous. For more information about existing MTL approaches, a survey can be found in (Zhang and Yang, 2017).

2.10 Existing Work on using PCA and Eigenvectors for Network Traffic Analysis

This section provides a review of recent approaches involving the PCA algorithm and its uses in the context of computer network data analysis and other fields. It also summarises some approaches that involve using eigenvectors for several purposes. An overview of PCA, how it works and what eigenvectors are is provided in Sections 5.1 and 5.2.

The method explained in (Bhattacharya et al., 2020) uses the PCA algorithm with the XGBoost (XGBoost, 2021) classifier in the context of Intrusion Detection in Networks. The authors only apply the following techniques in this sequence: they apply the traditional PCA algorithm on the data to reduce its dimensionality, after this they apply the firefly optimisation method (Yang, 2008) for feature selection (i.e. to select a subset of the principal components generated by PCA). They train the XGBoost algorithm on the data and employ the trained model for predictions. Their results show improvement in accuracy when applying PCA and the firefly optimisation method as opposed to using the original features In (Murugan and Devi, 2019) PCA was combined with Logistic Regression and used as a feature extraction technique in order to increase classification accuracy. The application domain was the analysis of twitter data (i.e. text classification) and the reported results show better accuracy was obtained after applying PCA to reduce the number of input features.

PCA was also used for a similar purpose in (Nasution et al., 2018) where the idea was to increase the accuracy of decision trees by training them on a set of non-correlated features (i.e. the principal components generated by PCA). In addition, the authors in (Zhu et al., 2019) employ the classical PCA algorithms for dimensionality reduction

and then run several experiments to evaluate the performance of Logistic Regression for classification and k-means clustering for data point cluster analysis. The reported results in both works show accuracy improvement.

An interesting work is reported in (Gadekallu et al., 2021) where PCA was used in an optimisation experiment. The main aim of this work was to employ a machine learning model for tomato disease image classification, which can help in early identification of tomato disease in order to take proactive action. The technique combines PCA with the Whale optimisation algorithm (Mirjalili and Lewis, 2016) to select the most useful feature subset and optimise the classifier being built for image categorisation.

Eigenvectors were used as features for the purpose of machine learning model training and evaluation. For example, in (Wei et al., 2017), where a malicious Android app detection tool was built, eigenvectors were extracted from data collected about/from android apps and used as features to train several traditional machine learning classifiers. The authors report that, compared with existing work, their approach has a better false positive rate and accuracy.

The approach discussed in Jiang et al. (2017) proposes an optimised approach for solving the *eigenproblem* and determining the minimum finite eigenvalue and associated eigenvectors. On the other hand, based on the hypothesis that eigenvectors represent the principal components of the underlying data matrix, they were used to characterize dominant tremor sources in the work reported in (Soubestre et al., 2018). The work keeps track of daily tremor activity, and their corresponding eigenvectors, and uses these eigenvectors to construct a network (i.e. a mathematical graph) for further analysis.

Eigenvectors were also successfully used in spatial data filtering. One powerful method is the Moran eigenvector filter (Dray et al., 2006) that involves the spatial patterns

represented by maps of eigenvectors. The idea is to choose suitable orthogonal patterns and add them to a linear or generalised linear model. The residuals will result in having a form of spatial dependence that can be used as part of the model. An efficient implementation of this filter for large datasets is presented in (Griffith and Chun, 2019). The authors report that their implementation is computationally efficient for dealing with large spatial data.

In (Hu et al., 2018) eigenvectors are used as a basis to compute the centres of the radial basis function (RBF), hence, a clustering method is developed. The idea was to save time by calculating the principal components of the input data matrix instead of iteratively calculating cluster centres using the k-means clustering algorithm. The experiments reported by the authors show that their method leads to faster model training with similar accuracy result.

2.11 Summary

This chapter has provided a background account for the main subject areas involved in this thesis. Essentially, a review of how these subjects are interlinked and used together to improve network security has been introduced. Also, several machine learning methods have been described, which are commonly used in the analysis of network traffic data for the purpose of identifying malicious traffic. A close look at these methods showed that the state-of-the-art transfer learning algorithm for network traffic analysis (SBIT and its extension CB-SBIT) is a highly iterative algorithm that seems to use ad-hoc similarity measures to copy instances from large datasets to smaller datasets. A brief explanation and critical review of other recent existing techniques have also been provided.

In the next chapter some initial experiments run to evaluate the performance of several traditional classifiers on network traffic data are presented and discussed. The chapter provides a useful overview of several metrics used for evaluation of classification methods.

Chapter 3

DATA MINING OF IPS DATASETS: EXPERIMENTS AND RESULTS

This chapter provides some initial experiments run to evaluate the performance of several traditional classifiers on network traffic data. The chapter starts with an overview of several metrics used for evaluation of classification methods. The main aim is to have a comparison to check whether some more suitable classification methods are required to accurately classify data of this nature (i.e. network traffic data).

3.1 Classification Evaluation Metrics

Before running any experiments, a review of some of the main metrics used in evaluating the performance of classifiers had been conducted. When using classifiers, it is essential to obtain information about their performance. An evaluation metric quantifies the

performance of a predictive model (e.g., a classifier) and the classifier is only as good as the metric used to evaluate it. If one chooses the wrong metric to evaluate a predictive model, then they are likely to select a poor model, or in the worst case, they can be misled about the expected performance of the model. Therefore, choosing an appropriate metric is highly important in machine learning (Liu, Zhou, Wen and Tang, 2014). Several evaluation metrics are used by the machine learning community. All metrics make assumptions about the problem or about what is important in the problem they are used for. Hence, an evaluation metric must be chosen such that it best captures what is believed to be important for the model or predictions (Hossin and Sulaiman, 2019). Some of these metrics are reviewed below. The information provided in the remainder of this section is summarised from (Canbek et al., 2017) and (Japkowicz and Shah, 2011).

Classification Accuracy: Classification accuracy is a metric that summarises the performance of a classification model as the number of correct predictions divided by the total number of predictions (i.e., it is the proportion of the total number of predictions that were correct). To calculate it, one first uses a classification model to make a prediction for each example in a test dataset (data not used in training and with known class labels) and then the predictions are compared to the known labels for those examples in the test set (the numbers of correct and incorrect predictions are recorded).

Accuracy = Correct Predictions / Total Predictions

The value of accuracy is usually between 0 and 1, where 1 means the model is predicting every input correctly and 0 means the model is predicting every input incorrectly. It can also be reported as a percentage.

Error Rate: Another metric that is related to accuracy is the error rate. It is calculated as the as the number of incorrect predictions divided by the total number of

predictions: Error Rate = Incorrect Predictions / Total Predictions.

Based on that, it is straightforward to infer that:

Accuracy + Error Rate = 1

Hence:

Accuracy = 1 – Error Rate

and

Error Rate = 1 – Accuracy

It is important to note that accuracy is not always the best measure to employ especially when data is imbalanced (Fatourechi et al., 2008). Here is an example to clarify that. Consider the case of an imbalanced dataset with a 1:100 class where each example of the minority class (Class 1) will have a corresponding 100 examples in the majority class (Class 0). In problems of this type, the majority class represents "normal" and the minority class represents "abnormal" such as an intrusion or malicious network traffic. In this case: Good performance on the minority class is usually preferred over good performance on both classes. The issue in such situations arises from the fact that a model that predicts the majority class (Class 0) for all examples in the test set will have a classification accuracy of 99% and this is highly over-optimistic (i.e., results can be misleading).

Confusion Matrix: A confusion matrix is a summary of the predictions made by a classification model organised into a table by class (Ting, 2017). Each row of the table indicates the actual class, and each column represents the predicted class. A value in the cell is a count of the number of predictions made for a class that are actually for a given class. The cells on the diagonal represent correct predictions, where a predicted and expected class align. An example confusion matrix is shown in Figure 3.1.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

FIGURE 3.1: A confusion matrix showing the number of correctly and incorrectly predicted instances

Based on the counts in the confusion matrix several useful metrics can be calculated. Before explaining what these metrics are and how to calculate them, it is useful to introduce the following concepts:

- A true positive (TP) is an outcome where the model correctly predicts the positive class
- A true negative (TN) is an outcome where the model correctly predicts the negative class
- A false positive (FP) is an outcome where the model incorrectly predicts the positive class
- A false negative (FN) is an outcome where the model incorrectly predicts the negative class

These counts are useful to obtain an insight into the performance of a predictive model. They are based on the confusion matrix shown below:

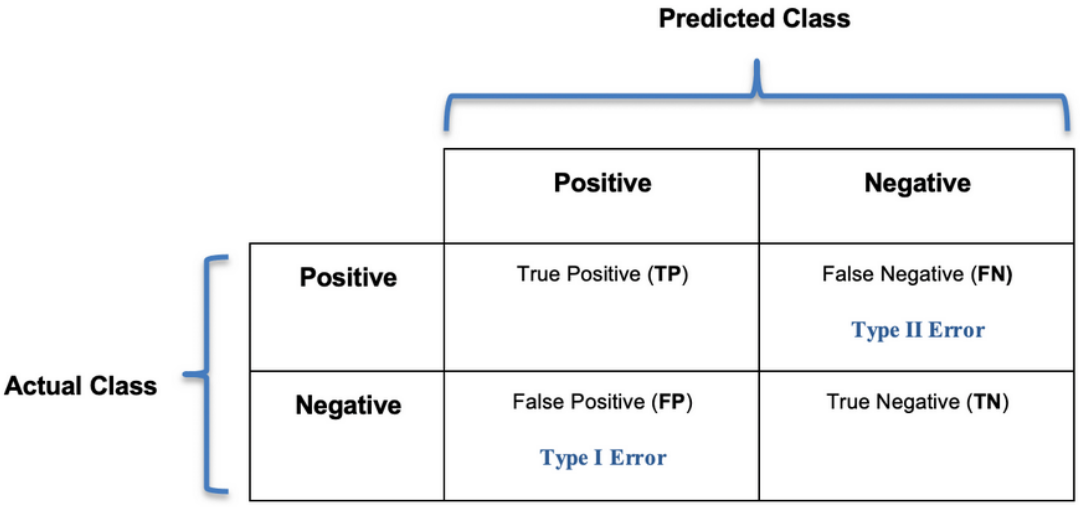


FIGURE 3.2: A confusion matrix showing predicted v actual classes and the two types of errors

Precision, Recall and F-Measure: Precision, also known as Positive Predictive Value, quantifies the number of positive class predictions that actually belong to the positive class (i.e., the proportion of positive cases that were correctly identified):

Precision = TruePositives / (TruePositives + FalsePositives)

This measure is appropriate when minimising false positives is the focus.

Recall, also known as Sensitivity, quantifies the number of positive class predictions made out of all positive examples in the dataset (i.e. the proportion of actual positive cases which are correctly identified):

Recall = TruePositives / (TruePositives + FalseNegatives)

This measure appropriate when minimising false negatives is the focus.

F-Measure provides a single score that balances both the concerns of precision and recall in one number:

*F-Measure = (2 * Precision * Recall) / (Precision + Recall)*

The value of each of the above three measures is always between 0 and 1 where a value as close as possible to 1 is desired.

Other useful metrics are Specificity, Sensitivity and Negative Predictive Value. Specificity is calculated as the fraction of TN divided by the total number of negative examples (i.e., the proportion of actual negative cases which are correctly identified). On the other hand, Sensitivity (which is the same measure as Recall) is calculated as the fraction of TP divided by the total number of positive examples. Negative Predictive Value is the proportion of negative cases that were correctly identified. These measures are illustrated on the confusion matrix as the following figure shows:

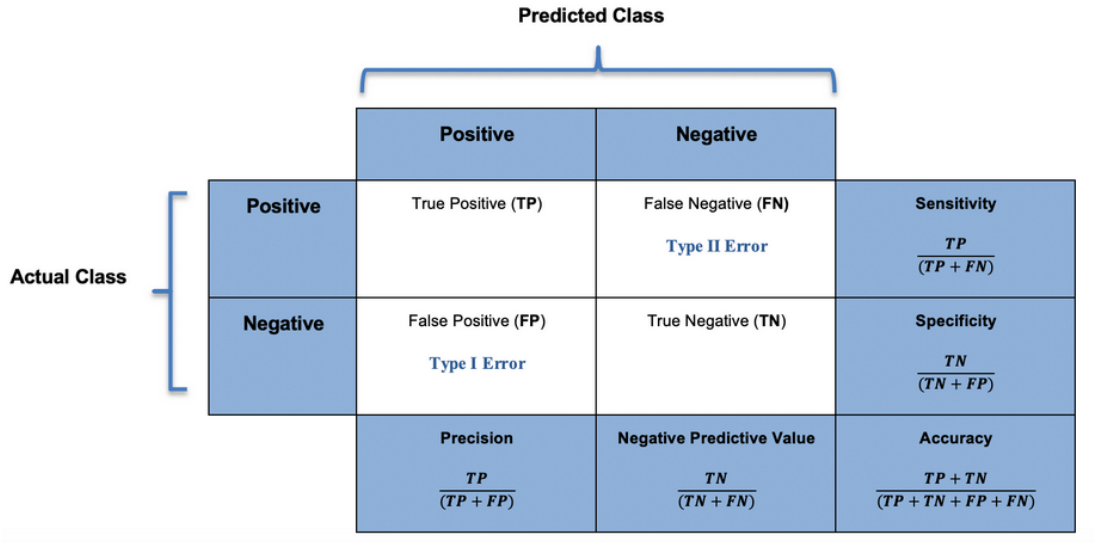


FIGURE 3.3: A confusion matrix showing more useful metrics

Receiver Operating Characteristic Curve and Area under the Curve: This is a useful tool when predicting the probability of a binary outcome instead of just predicting the class immediately. It is worth mentioning here that this is suitable for binary classifiers. The Receiver Operating Characteristic curve, or ROC curve, is a plot of the false positive rate (on the x-axis) versus the true positive rate (on the y-axis) for a number of different candidate threshold values between 0.0 and 1.0. In other words,

it plots the false alarm rate versus the hit rate. These are provided by the following formulas:

- *True Positive Rate (Sensitivity) = True Positives / (True Positives + False Negatives).*
- *Specificity = True Negatives / (True Negatives + False Positives).*
- *False Positive Rate (1 - Specificity) = False Positives / (False Positives + True Negatives).*

The ROC curve is a useful tool for a few reasons:

- The curves of different models can be compared directly in general or for different thresholds.
- The area under the curve (AUC) can be used as a summary of the model skill.

The shape of the curve contains plenty of information, including what we might care about most for a problem, the expected false positive rate, and the false negative rate.

- Smaller values on the x-axis of the plot indicate lower false positives and higher true negatives.
- Larger values on the y-axis of the plot indicate higher true positives and lower false negatives.

An example is shown in Figure 3.4. The red curve is for a classifier of interest where we wish the curve to go towards the top-left corner as much as possible. The area under the curve (AUC) is usually between 0-1 where 1 means a perfect classifier and 0 means the

classifier mis-classifies all the instances. The blue line is for a classifier that randomly picks classes for any input instance (gives a 50% chance to each class). Please observe that it is outside the scope of this thesis to explain how to generate such curves. For a detailed explanation on how to do that, the reader can refer to (Bradley, 1997).

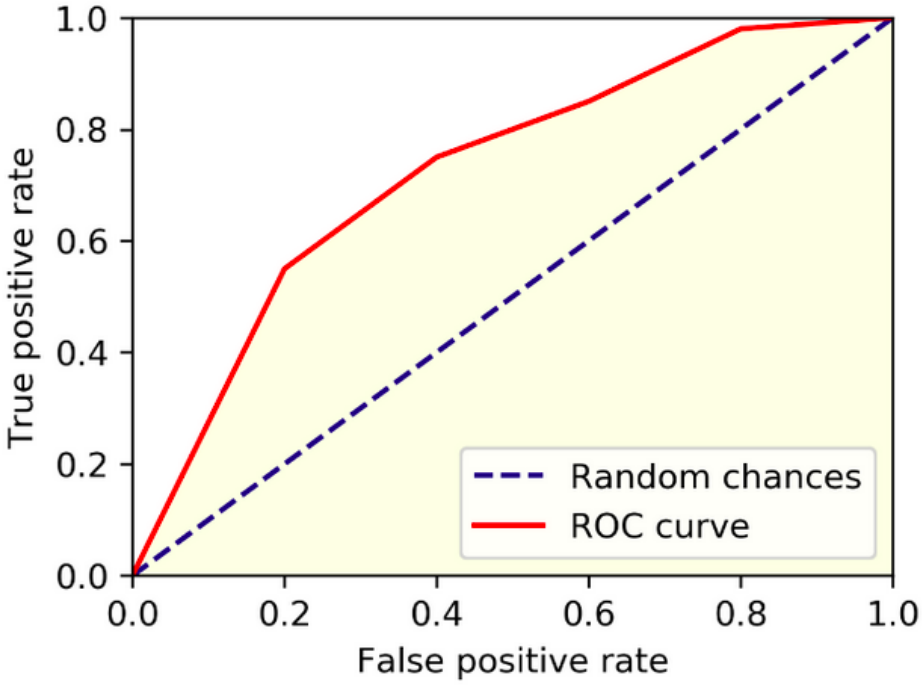


FIGURE 3.4: A ROC curve v random classes by a classifier (the dotted line y = x) giving 50% chance to each class

Matthews Correlation Coefficient (MCC): This metric is useful as it combines all four values in the confusion matrix. It is based on treating the actual class and the predicted class as two (binary) variables, and computing their correlation coefficient (just like computing the correlation coefficient between any two variables). A high value (close to 1) means that the classifier is predicting both classes well, even if the data is imbalanced. On the other hand, a low value (close to 0) means the classifier is misclassifying most of the classes. The formula to compute the MCC is:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.1}$$

Precision-Recall Curve: Especially in highly imbalanced data, and because of the high-class imbalance, one is usually less interested in the skill of the model at predicting Class 0 correctly, e.g., high true negatives. Key to the calculation of precision and recall is that the calculations do not make use of the true negatives. It is only concerned with the correct prediction of the minority class, Class 1.

A precision-recall curve (an example is shown in Figure 3.5) is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve. A very good model is represented by a curve that bows towards (1,1). As described above, the F-Measure summarises model skill for a specific probability threshold (e.g. 0.5), whereas the area under a precision-recall curve summarises the skill of a model across thresholds, just as ROC AUC.

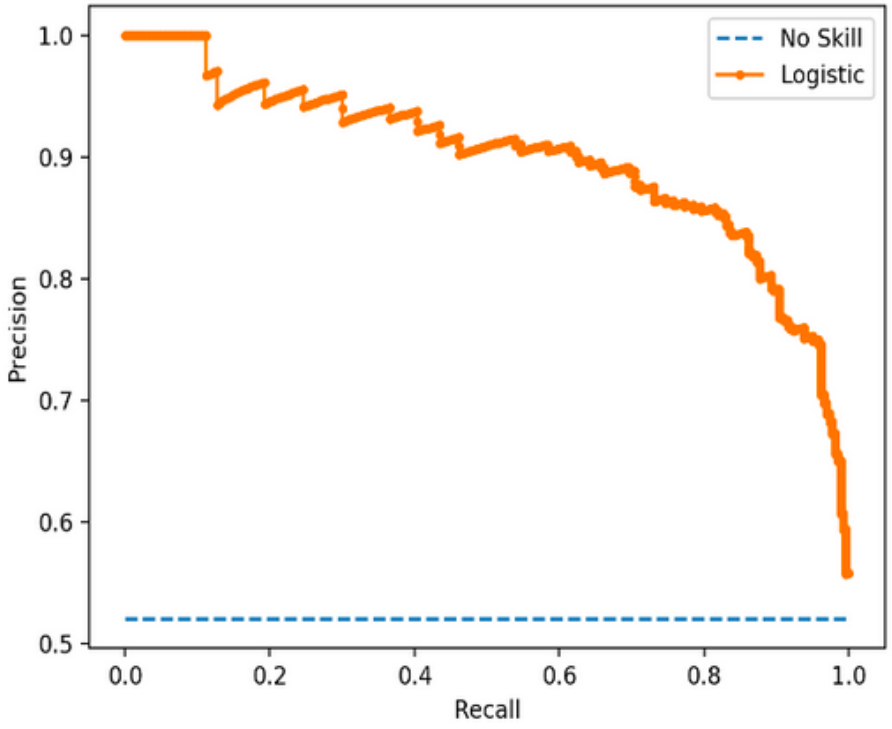


FIGURE 3.5: A precision-recall curve depicting the skill of a model

Please observe that it is outside the scope of this thesis to explain how to generate this

curve. For detailed explanation on how to do that, the reader can refer to (Saito and Rehmsmeier, 2015).

The above was a brief overview of these useful metrics. For a more detailed explanation, the reader can refer to the article in (Canbek et al., 2017).

K-Fold Cross Validation: The k-fold cross-validation procedure involves splitting the training dataset into k folds. The first $k-1$ folds are used to train a model, and the holdout kth fold is used as the test set. This process is repeated and each of the folds is given an opportunity to be used as the holdout test set. A total of k models are fitted and evaluated, and the performance of the model is calculated as the mean of these runs (Figure 3.6). This procedure ensures that every instance in the original dataset has the chance of appearing in training and test set which leads to having a good overview of classifier performance (Patil et al., 2021).

Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

FIGURE 3.6: Five-fold Cross Validation

In the standard CV procedure, the data is usually split into k-folds with a uniform probability distribution. This is not appropriate for evaluating imbalanced classifiers. It is likely that one or more folds will have few or no examples from the minority class.

This means that some or perhaps many of the model evaluations will be misleading, as the model needs only to predict the majority class correctly. The Solution: split a dataset randomly in a way that maintains the same class distribution in each subset. This is called stratification or stratified sampling and the target variable y, the class, is used to control the sampling process.

3.2 Classification Experiments

Some experiments were conducted on a freely available IPS dataset that contains network traffic data from various malicious activities as well as normal, or benign, traffic.

3.2.1 The Data

This dataset is freely available from the Canadian Institute for Cybersecurity (Canadian Institute for Cybersecurity, 2017). The dataset was in PCAP format, it was transformed by its publishers into a comma-separated value (CSV) format using a special tool. A practical and systematic way for transforming PCAP files into CSV files and pre-processing the resulting CSV files is reported in (Alothman, 2019*a*). The dataset used here contains 2830742 instances in total. The distribution of instances and class labels is as follows:

Traffic type	No of instances
Benign	2273096
DoS Hulk	231073
PortScan	158930
DDoS	128027
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS Slowloris	5796
DoS Slowhttptest	5499
Bot	1966
Web Attack Brute Force	1507
Web Attack XSS	652
Infiltration	36
Web Attack SQL Injection	21
Heartbleed	11

FIGURE 3.7: Distribution of instance classes in the IPS traffic dataset

The dataset has 78 columns in total, 77 features and a class label. The features, or attributes, include values such as the flow duration, the total forward packet, destination port and so on. The class label represents whether the flow results in an attack (attack name) or is benign. A view of some of the features is shown in Figure 3.8

Destination	Flow Duratic	Total Fwd P	Total Backw	Total Length	Total Length	Label
389	113095465	48	24	9668	10012	BENIGN
389	113473706	68	40	11364	12718	BENIGN
0	119945515	150	0	0	0	BENIGN
443	60261928	9	7	2330	4221	BENIGN
53	269	2	2	102	322	BENIGN
123	30352	1	1	48	48	BENIGN
59135	48	1	1	6	6	BENIGN
5353	89501767	223	0	26257	0	BENIGN

FIGURE 3.8: Some of the Features in the Data

3.2.2 Data Pre-processing

The data comes with several missing values and hence needed to be pre-processed and prepared for mining. It is usually useful to impute missing values if they do not represent a large fraction of the entire data. In this dataset, there was only a small fraction and they were replaced by the median of their respective columns (i.e. feature values).

Another important step to bear in mind is that the feature values are all numeric even for categorical values. An example is the Destination Port. The destination port represents a network protocol, which is categorical, but it is represented numerically in this data. The correct method to represent categorical values numerically is to use the one-hot encoding technique (Guo and Berkhahn, 2016).

(A) Categorical Column

Color
Red
Red
Yellow
Green
Yellow

(B) Corresponding One-hot Encoding Representation

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

TABLE 3.1: One-hot Encoding Example

Table 3.1 illustrates the idea of One-hot encoding. It works by splitting the column (which contains numerical categorical data) to many columns depending on the number

of categories present in that column. If a row contains a category, the corresponding column will have 1 and all other columns will have 0. However, if this is applied to a large dataset such as the one at hand, and with the fact that there is a possible 65535 number of ports, the dataset size would become too large for a desktop computer to process (the number of columns would hugely increase). Therefore, the choice made was to drop this column from the dataset.

In addition to the above, steps like removal of highly correlated features and outlier detection and removal could be applied (they are optional steps). However, they were not applied in these experiments as they might result in loss of relevant information.

3.2.3 Algorithms used in the Experiments

The algorithms used in this evaluation work were from Python's Scikit-learn machine learning package (Pedregosa et al., 2011). They were selected because they are commonly used for benchmarking in the literature (Hao & Ho, 2019). The following subsections provide a list of these algorithms and a brief introduction to each of them.

3.2.3.1 Naive Bayes

The Naive Bayes classifier (Choi et al., 2020) is based on the classical Bayes theorem which assumes the input variables (or the features) are independent and hence computes the posterior probability P(c—x), or the probability of the class given the data, from the following:

- $P(c)$ or probability of the class c (i.e. the class prior probability);

- $P(x)$ or probability of the data x (sometimes denoted the evidence or the predictor prior probability);
- and $P(x|c)$ or probability of the data x given the class c (i.e. the likelihood);

where $P(c|x)$ is given by the formula shown in the following equation.

$$P\left(c/x\right) = \frac{P\left(x/c\right)P\left(c\right)}{P\left(x\right)} \tag{3.2}$$

where:

$$P\left(c/X\right) = P\left(x_1/c\right) \times P\left(x_2/c\right) \times \cdots P\left(x_n/c\right) \times P\left(c\right)$$

3.2.3.2 Multilayer Perceptron (MLP)

This is the classical Artificial Neural Network (ANN). An ANN consists of a network of artificial neurons (also known as "nodes") that are connected to each other (Ramchoun et al., 2017). The strength (or weight) of their connections to one another is assigned a value based on how important that connection is when minimising the training error during the training process. These weight values are updated using an algorithm known as the Gradient Descent Algorithm. Each neuron contains a transfer function which translates the input signals to output signals. There are three types of neurons in an ANN: input nodes, hidden nodes, and output nodes.

3.2.3.3 Decision Trees

The Decision Tree algorithm works by creating classification or regression models in the shape of a tree structure (Molnar, 2022). The algorithm breaks down the training dataset into smaller and smaller sub-datasets and incrementally builds an associated

decision tree at the same time. When it finishes training, it results in a tree with two types of nodes: decision nodes and leaf nodes. A decision node is where the tree branches and it has two or more branches. A classification or a decision is made at a leaf node.

3.2.3.4 Random Forest

This algorithm is based on the Decision Trees algorithm. It works by creating a number of decision trees at training time and using them to make future predictions (Breiman, 2001). To develop each tree, random sampling is performed on the original training data to obtain a sample dataset (by using random sampling with replacement of both the instances and input features). Based on this (i.e., each new sample dataset is different), each tree is different as each tree focuses on a different aspect of the data that it is trained on. In the end, a classification is made by using a majority vote of predictions of all generated trees (or the mean is used when performing regression).

3.2.4 Experiments

The experiments reported here performed binary and multiclass classification. In binary classification, one is actually interested in a yes/no answer to whether an instance belongs to a given class. For example, whether a given flow of data constitutes an attack or not. Given the IPS dataset, it is common to use Naive Bayes and Random Forest classifiers. For example, Naive Bayes (Mukherjee and Sharma, 2012) and Random Forest (Farnaaz and Jabbar, 2016) as benchmarks. As for the multiclass classification common classifiers include the MultiLayer Perceptron (MLP) and the Decision Tree classifiers (Koklu and Ozkan, 2020). For the experiments, these multiclass classifiers were chosen.

The implementation was in Python mainly using the Scikit-learn and Pandas (McKinney et al., 2010) and (Wes McKinney, 2010) packages. For the following experiments, and based on previous explanation of classifier performance metrics, it is important to highlight the following regarding Precision and Recall as they are reasonable metrics especially when data is imbalanced: when one tries to maximise precision, they try to minimise the number false positives. On the other hand, when we maximise the Recall one tries to minimise the number of false negatives. Hence, precision is useful when the focus is to minimise the false positives, whereas recall is useful when the focus is to minimise the false negatives.

In problems like the one at hand, i.e., IPS data that might contain attacks, classifying mistakes can be highly costly. For example, misclassifying benign traffic as attack (called a false-positive) is often not desired, but less critical than classifying an attack as being benign traffic (called false negative). Therefore, minimising false-negatives can be the main focus but bearing in mind the importance of minimising false-positives. Therefore, a close attention was paid to several evaluation metrics to assess the performance of the classifiers in each setting.

3.2.4.1 Binary Classification

Before performing an evaluation of the Naive Bayes and Random Forest classifiers in a binary classification setting, it is essential to prepare the data so that they contain two classes (Attack and No Attack). Since the data contain several attacks, the choice to make was to select one attack as "Attack" and set everything else as "No Attack". The "Web Attack XSS" attack was chosen as the attack and everything else in the dataset was turned to "No Attack". As "Web Attack XSS" is the class of interest, it was assigned Class 1 and all other classes were Class 0. After doing this, the number of instances in

each class was as follows: In Class 0 there were 2830090 instances and in Class 1 there were 652 instances. This means the dataset was highly imbalanced. A testing procedure was conducted to split the data into two non-overlapping subsets: a training set and a testing set. The splitting was stratified based on the class variable to guarantee that the same percentage of the two classes existed in the training and test data. The training set was 65% of the original data whereas the test set was the remaining 35%.

Naive Bayes Results: After splitting the data as explained above and training a Naive Bayes classifier (with default parameters), the results were as shown in the following table:

Metric	Value
Accuracy	91%
Number of TruePositive	219
Number of TrueNegative	903055
Number of FalsePositive	87477
Number of FalseNegative	9
Recall	0.96
Precision	0.0025
F1	0.0049
ROC AUC	0.956
PRC AUC	0.477

FIGURE 3.9: Results of the Naive Bayes classifier

The results show a promising performance with 91% accuracy. However, this promising performance is not as good as it seems when one looks at the precision. As indicated in the definition of the precision measure above, the closer the precision is to 1, the better

the results are. The precision value in the results was as low as 0.0025 (despite the high recall value of 0.96). This poor performance is reflected in the F1-Measure, referred to as F1 in 3.9, which has a low value of 0.0049. The conclusion is therefore, for this dataset, the Naive Bayes classifier produced a high number of false positives but a low number of false negatives. Again, a promising performance may appear when looking at the ROC AUC value (0.956), but the actual, relatively low-quality of the performance is reflected by the PRC AUC which is 0.477.

Random Forest: For this experiment, the same train and test split used for the Naive Bayes experiment for Random Forest classifier was used. The results after training it and predicting the test set are shown below:

Metric	Value
Accuracy	99%
TruePositive	14
TrueNegative	990523
FalsePositive	9
FalseNegative	214
Recall	0.061
Precision	0.608
F1	0.111
ROC AUC	0.993
PRC AUC	0.463

FIGURE 3.10: Results of the Random Forest classifier

The results again show a promising performance with 99% accuracy. In a different behaviour to that of Naive Bayes, Random Forest showed a precision value of 0.608 and a low recall value of 0.061. This is reflected in the F1 value of 0.111. The Random Forest classifier produced a low number of false positives and a somewhat high number of false negatives. Again, a promising performance is observed when considering the ROC AUC value (0.993) but a low value of 0.463 for the PRC AUC was observed.

3.2.4.2 Multiclass Classification

For the multiclass classification, we have used the same train and test splits as the ones we have used in the binary classification. The difference is the class label as it now contains the 15 classes rather than only 2 classes.

Decision Trees: The Decision Tree classifier in the Scikit-learn package in Python was used with its default parameters except for max_depth which we set to 5 to save time. The results are reported in the following table:

Metric	Value
Accuracy	96%
Multiclass Precision	0.459
Multiclass Recall	0.329
Multiclass F1	0.383
Multiclass ROC AUC	0.658

FIGURE 3.11: Results of the Decision Tree classifier in the Multiclass Classification

Using the Decision Tree classifier for the multiclass experiment showed a high accuracy of 96%. However, both recall and precision metrics were low (with values 0.459 and 0.329

respectively). This clearly affected the F1 measure which dropped to 0.383. In addition to the above, the ROC AUC was much closer to 0.5 than to 1 and this is undesired. As explained earlier, the target is to have the values of the precision, recall, F1 and ROC AUC measures as close to 1 as possible to make sure the classifier introduced in this thesis is performing well on the test data.

Multilayer Perceptron: The Multilayer Perceptron classifier was used in Scikit-learn, with its default parameters except for the number of units in the hidden layer, which was set to 15. The results are reported in the following table:

Metric	Value
Accuracy	97%
Multiclass Precision	0.638
Multiclass Recall	0.489
Multiclass F1	0.554
Multiclass ROC AUC	0.741

FIGURE 3.12: Results of the Multilayer Perceptron classifier in the Multiclass Classification

Although the results obtained using Multilayer Perceptron were generally better than those obtained using Decision Tree, they were still undesired. This is despite the high accuracy value of 97%. The precision, recall, F1 and ROC AUC were not as high as desired.

3.3 Summary

This chapter has provided the main metrics used to assess and evaluate classifiers. These metrics were actually used to evaluate a number of tests conducted on a network traffic

dataset. These tests were divided into binary and multiclass classification methods. After performing binary and multiclass classification and employing different classifiers, and when looking at precision, recall, F1-Measure and ROC AUC, it was concluded that the performance was not as good as desired, and some improvements are required. The precision and recall metrics needed to be as close to 1 as possible in order to minimise false-negatives and false-positives. A large number of false positives would cause distraction to computer security teams and could unnecessarily waste time and exhaust resources. On the other hand, even a low number of false negatives could still be costly as it means malicious traffic has not been correctly identified. The next chapter provides an overview of the new classification methods that attempt to solve this problem.

Chapter 4

DEVELOPMENT OF NEW CLASSIFICATION METHODS

It was shown in the previous chapter of this thesis that better classifiers were required to deal with network traffic data. This chapter provides an overview of two new classification methods developed as part of this research. The chapter explains how they work and shows Python code snippets when possible. In the next chapter, results of thorough tests to evaluate their performance are presented.

4.1 Multi-Task Learning for Accurate Classification of Network Traffic

One of the classifiers developed as part of this research is based on the concept of multi-task learning (intuition behind MTL and why it is suitable for the purpose of the research project of this thesis were explained in Sections 2.6.2 and 2.6.3). As shown in previous chapters of this thesis, the IPS data show a significant difference between the

number of available instances that belong to different traffic types. This is a typical situation as datasets of this type are usually highly imbalanced. In a scenario like this, where plenty of data is available for some tasks and very small amounts of data is available for some other tasks related to these tasks, usually poor predictive models are obtained when learning the tasks with small amounts of data. It is important to notice that in this research, each network traffic category is considered a task (i.e., Benign is a task, DDoS attack is a task and so on). Because of the low performance of existing models, some improvement is required. This research introduces a multi-task learning approach to achieve such improvement. The idea is to learn tasks with large amounts of data together with tasks with small amounts of data in order to obtain better models for the latter tasks (i.e., with amounts of data).

The proposed MTL method in this work is inspired by a recent work by (Sadawi et al., 2019). Although it was applied in the field of drug discovery, it is possible to adopt, enhance and apply it in other fields such as the field of automatic analysis of computer network traffic. It is important to highlight the differences between the method introduced in this research and the above-mentioned method. The method this research adopts (referred to as the modified method) differs in the following ways:

- The existing method measures the similarity between tasks using amino acid sequences whereas the modified method uses instance similarity as will be explained in this chapter.
- The existing method is used in a regression context (i.e., to predict a real number) whereas the modified method is classification (i.e., to predict a category or class) and therefore the problem setting and evaluation metrics/procedures are different.

- A weakness has been identified in the existing method. Because it attempts to address a regression problem, it is not straightforward to distinguish between instances from different tasks when the datasets are merged. Therefore, it adds a task ID column as a new feature (each task is given a task number) and this can mislead the classifier as the task numbers are introducing something that does not exist. In more detail, this is because when an integer number (such as 1) is assigned to task A and another number (such as 2) to task B, mathematically 2 is greater than 1, but does this mean that task B is larger than task A? Not necessarily. A classifier does not know that and will use the task ID as a number and not as a category, which introduces order between tasks that does not really exist.

- The identified weakness in the existing technique has been overcome in the modified method. In the modified method, the class label is used to distinguish between instances from different tasks.

Based on the above, the contributions of this work for the multi-task learning approach can be summarised as follows:

- Develop a multi-task learning (MTL) approach for the accurate identification of malicious network traffic (even in case of scarce data).

- Instead of learning the similarity between tasks, it is quantified by measuring it using a well-known similarity metric (namely, the cosine similarity (Deza and Deza, 2009)).

- Identify a weakness in a recent existing method and demonstrate how the modified method avoids it.

- Experimentally show that the modified method outperforms existing widely used classifiers in various situations as will be explained in this Chapter 6.

The development of the MTL modified method has been documented in a paper (Aljoufi, Reem and Lasebae, Aboubaker, 2021). The paper describes the modified method in detail. It also shows a thorough experimental evaluation and comparison with a recently published successful method.

4.1.1 The Proposed Method

As stated previously, the MTL method introduced in this work is a novel extension of the recent work in (Sadawi et al., 2019). The idea is to consider each network traffic type as a task and then train a classifier in a multi-task learning setting (each task is represented by a corresponding dataset). Before running the algorithm, similarity between all tasks is computed (compute average instance similarity between instances of all datasets) as shown in algorithm 1.

Algorithm 1: How to Compute Similarity Values between Instances

Input : Two Datasets ($ds1$ and $ds2$)
Output: Vector of average similarity values

```
- Initialize empty similarity vector vsim;
for i ∈ ds1 do
    - Initialize temporary empty similarity vector vsim_temp;
    for j ∈ ds2 do
        - Compute s = cosine_similarity(i,j);
        - add s to vsim_temp;
    - Compute avg_sim = avg(vsim_temp);
    - add avg_sim to vsim;
- Return vsim;
```

Task similarity is required in multi-task learning because it is often used as a way of quantifying the relationship between tasks (Shui et al., 2019).

The result of the previous algorithm is a vector of the same length as $ds1$ that contains the similarity values of the instances of $ds1$ to instances in $ds2$. Using this algorithm, a vector of similarity values for each pair of datasets (i.e. tasks) can be easily computed.

Figure 4.1 shows example similarity values computed between instances from the PortScan attack and instances from DDoS attack, Benign traffic and Bot attach respectively.

Sim2DDoS	Sim2Benign	Sim2PortScan	Sim2Bot
0.045173	0.355133	1	0.384667
0.117410	0.598975	1	0.479856
0.283120	0.993953	1	0.198488
0.077537	0.827918	1	0.577897
0.050979	0.906212	1	0.386497
0.140248	0.892234	1	0.480951
0.116630	0.856175	1	0.672383
0.213279	0.816124	1	0.390222
0.143098	0.799398	1	0.486127
0.249391	0.809361	1	0.489147
0.153031	0.474504	1	0.488083
0.102993	0.994392	1	0.391403

FIGURE 4.1: Example Similarity Values Computed between Instances of Multiple Network Traffic Types (i.e. Tasks)

After the pairwise similarity values of all tasks (i.e. using their corresponding datasets) have been computed, these datasets are concatenated and similarity values are added as new features as shown in algorithm 2.

What the method introduced in this thesis does is: join all datasets together into one large dataset and then add similarity values as new features to the resulting large dataset.

Algorithm 2: How to Concatenate Datasets, add Similarity Values as new Features and Train a Classifier using the Resulting Dataset

Input : n related datasets (each contains data from one network traffic type)
Output: A Trained Model that can be used for Future Predictions

1 Merge the n datasets along the columns (i.e. stack them);
2 Add n extra variables to the dataset resulting from step 1: *SimToDS_1, SimToDS_2, ..., SimToDS_n*;
3 Populate the similarity values using the results from algorithm 1;
4 Train a classifier using the newly created dataset

After that a model can be trained using the large dataset and used for future predictions.

The dataset resulting after applying algorithm 2 should look like the example shown in figure 4.2. Notice the original features are on the left and the similarity values are on the right (this example has four new columns because this setting involves data from four tasks).

Task	f1	f2	.....	SimiToTask1	SimiToTask2	SimiToTask3	SimiToTask4
Task1				1	0.35	0.44	0.12
Task2				0.35	1	0.53	0.39
Task3				0.44	0.53	1	0.61
Task4				0.12	0.39	0.61	1

FIGURE 4.2: Contents of Dataset resulting after the Proposed MTL Method

A view of part of the merged data is shown in Figure 4.3. The figure shows the similarity values added as new columns (i.e. features after the label column).

Flow Packets/s	Bwd IAT Mean	Bwd IAT Std	Bwd IAT Max	Fwd Packets/s	Bwd Packets/s	label	Sim2DDoS	Sim2Benign	Sim2PortScan	Sim2Bot
0.797	61052.437	2398081.53	4900000.00	0.639	0.724	BENIGN	0.750253	1.000000	0.999135	0.552243
1235.980	40.200	1252.01	1958.83	618.000	1236.090	DDoS	1.000000	0.825122	0.099968	0.489020
336.000	102.803	2454.69	2997.33	216.000	288.000	BENIGN	0.245391	1.000000	0.000055	0.571679
1116.040	44.650	1105.03	1781.42	558.000	1116.150	DDoS	1.000000	0.694103	0.000006	0.583908
0.394	101666.667	3681055.16	7308333.33	0.303	0.454	BENIGN	0.562439	1.000000	0.997833	0.564032
...	...	...	...	...	...	...	...	...	...	...
12.300	4396.887	93802.56	183999.08	8.380	10.200	BENIGN	0.649460	1.000000	0.970705	0.598852
19.700	2530.078	81354.51	126445.08	9.880	19.800	DDoS	1.000000	0.226170	0.465622	0.193312
18.400	2497.742	86129.89	147147.33	8.340	19.500	DDoS	1.000000	0.202440	0.578942	0.370646
24.000	119.062	7395.82	43928.33	15.900	20.200	BENIGN	0.775083	1.000000	0.864287	0.451461
20.500	2690.827	75772.63	105892.33	11.600	19.400	DDoS	1.000000	0.289061	0.345324	0.258006

FIGURE 4.3: View of Merged Datasets with Added Similarity Columns

4.1.2 Instance Similarity

It was mentioned previously that instead of learning the similarity values between tasks in MTL, these values can be computed. This is applicable in the case of network traffic data belonging to different traffic types. The reason for this is that these different datasets are from the same domain and have the same feature space, with feature values being numeric. This makes it straightforward to quantify the similarity between instances in datasets.

The method employed for computing the similarity between tasks (i.e. datasets from different traffic types) is the same as in (Alothman, 2018*b*, 2019*b*, Alothman et al., 2018). Therefore, the following will briefly define how it is computed and refer the reader to those cited articles.

When there are two numeric vectors V_j and V_j, their similarity can be computed as a numeric value S such that this value is always between 0 and 1.

$$0 <= S <= 1$$

When $S = 0$, this means that V_j and V_j are not similar and there is no meaningful overlap between them (i.e. they have nothing in common or they are orthogonal). On the other hand, when $S = 0$, this means that V_j and V_j are the same (i.e. identical). As the value of S increases from 0 towards 1, it means V_j and V_j are more and more similar.

Now, the way to compute similarity between datasets (i.e. tasks) is illustrated in algorithm 1. The idea is to compute all similarity values between all pairs of instances in any two input datasets and then compute the average similarity value. It is important

to note that in the experiments, the cosine similarity has been used. There are other similarity types that can be employed (a useful resource on various types of similarity is the book in (Deza and Deza, 2009).

4.2 Classification Using Matrix Determinants

A new method for classifying the data is proposed by this research. The method is based on the idea of extracting linear equations from datasets and forming square matrices. The determinants of these matrices are then computed. Classification is based on the range of the determinant values. For example, determinants between a and b represent a given attack label, say 'Benign', while determinants between c and d represent another attack label, say 'DDoS', and so on.

4.2.1 Algorithm for Matrix Determinants

Since there are n numeric and one categorical attributes, the matrices formed are of size n *by* n, i.e., n rows and n columns. The algorithm (as sillustrated in Algorithm 3) entails forming matrices using the algorithm below for each single-label (attack/no attack based) dataset. Training will be attack based, so assuming attack a is considered, the following algorithm is used for training:

Algorithm 3: Algorithm for Matrix Determinants

Input : Dataset
Output: Matrix Determinants

1 Count the number of rows r in the a dataset, where a is a given attack;
2 Count the number of columns c with numerical values (i.e. exclude the label field);
3 Find $k = r - (r(mod(c)))$. k will be smaller than or equal to r;
4 Select arbitrary k rows from the dataset;
5 Form arbitrary $c \times c$ matrices from the datasets. We should get $t = k/c$ matrices. Call these matrices A_j;
6 Compute the determinants d_j of the A_j matrices, where $j = 1, 2, ..., t$;

The above steps are repeated for each attack a. When a new flow is encountered, the following algorithm (as illustrated in Algorithm 4) is used to test for a potential attack (or no attack) in the flow:

Algorithm 4: Prediction using Algorithm for Matrix Determinants

Input : New Flow
Output: Predicted Class

1 Form a row vector from a new data flow.;
2 Sum up the vector with each row of each of the matrices A_j formed in the training procedure for each attack a;
3 Each time the flow is added to one row of the matrix, compute the determinant d_{ij} . We will have k matrices with corresponding determinants for each attack dataset;
4 Find the $\{|d_{ij} - d_j|\}$ for $i = 1, 2, ..., k$, $j = 1, 2, ..., c$, where d_j is the determinant of the matrix A_j and $\{|d_{ij} - d_j|\}$ is the minimum of the absolute value of the difference $d_{ij} - d_j$. We will have k of these minimums;
5 Find the minimum of the k minimums calculated above, that is, $min\{\{|d_{ij} - d_j|\}\}$.Hence each attack set will have a number representing it. We call this number the dataset gist;
6 If the gist is zero for more than one attack dataset, count the number of zero gists for each attack;
7 Classify flow: the flow will belong to the attack set with the smallest gist or with the highest number of zero gists;

4.3 Classification Using Eigenvalue Perturbation

Although the use of the matrix determinant is useful to classify data as per the above algorithm, it could be problematic for certain datasets. For example, if the determinant is zero for more than one of the matrices being classified, it is not easy to choose in which matrix with the zero determinant the flow fits. One way of going around this is to count the number of zero occurrences of the determinant. However, this is not an effective way. Moreover, the occurrences may still coincide, and it would be difficult to choose the right dataset for the flow.

Another way of classifying the data, which is also similar to the way determinants work, is the use of the matrix eigenvalues. This is known as the eigenvalue perturbation problem.

Perturbation theory (Patkowski, 2020) works as follows: given a system represented by a matrix A and an operator λ acting on A, it would be highly useful to understand how a small perturbation ϵ to A affects λ. That is, to understand the relation between $\lambda(A)$ and $\lambda(A')$, where A' is the matrix after the perturbation. One of the interesting operators are the eigenvalues and eigenvectors of a matrix. So, the eigenvalues and eigenvectors of the original system are known, it becomes possible to find the eigenvalues and eigenvalues of the system after perturbation to determine how sensitive to change the system is. In theory, a small change of the matrix will result in a small change of its eigenvalues. However, it is not as straightforward as it sounds. Some matrices are more sensitive to perturbation than others.

This means a small change in one matrix could lead to a significant change in its eigenvalues. Such matrices are said to be sensitive to perturbation. Other matrices will have small change in their eigenvalues as a result of large perturbation. Therefore, a measure to identify what type of matrix one is dealing with is required. One way of detecting matrix sensitivity is through the use of the condition number. The condition number is a scalar value that measures how much a matrix is susceptible to change. There are various definitions of the condition number, but they all consider the maximum and the minimum of the eigenvalues of the matrix.

Before providing any formulae, it is worth mentioning here that formulae shown in this chapter are summarised from (Gezerlis, 2020).

One formula uses the 2-norm of the matrix, which is defined as:

$$\|A\| = \sqrt{max(\lambda_i)} \tag{4.1}$$

Here λ_i are the eigenvalues of the matrix AA^H, where A^H is the conjugate transpose of A.

Then the 2-norm of matrix A is given by the formula below:

$$cond(A) = \|A\|_2 \|A^{-1}\|_2 \tag{4.2}$$

Here A^{-1} is the inverse of matrix A.

So, in addition to considering the eigenvalues of a matrix, the condition number needs to also be considered. One way of using this result in classification is to combine the eigenvalues of a matrix with its condition number. It is possible to compute the condition number of the matrix after each perturbation and divide the eigenvalues by the condition number. The result will be a vector of ratios, call it τ, computed as follows:

$$\tau(A') = [\frac{\lambda_1}{cond(A')}] \tag{4.3}$$

where A' is the matrix A after perturbation, $cond(A')$ is its condition number and λ_i are its eigenvalues.

Then when it is possible compare the consecutive perturbations of the matrix, a factor can be used to measure the significance of the change. The perturbation factor of (A'), call it p, is the minimum of the elements of τ:

$$p(A') = min(\tau_i(A')) \tag{4.4}$$

The perturbation factor of the matrix can be used to compare a series of perturbations on a matrix and decide which perturbation affects the matrix the highest.

4.3.1 Algorithm for Eigenvectors Perturbation (EV)

A slightly better way of achieving the above classification is to replace the use of eigenvalues by eigenvectors. Doing such would provide better results as the entire flow is compared with an entire row of a matrix instead of comparing scalar values. This requires the use of the norm of the vectors (eigenvectors and matrix rows). Doing such would eliminate the need of relying on the condition number as it could result in inaccurate values if the matrix is ill-conditioned. The eigenvector perturbation method is therefore adopted for the classification of the IPS dataset. Given a dataset and a flow that is one row of data. The idea is to apply perturbation to the matrix by the flow and find the eigenvectors. This is achieved by replacing the first row of the matrix by the flow. Then the second row of the matrix is replaced by the flow and again find the eigenvectors. This is process is repeated until all the rows and all matrices are attempted. The algorithm is provided below in Algorithm 5.

Algorithm 5: Algorithm for Eigenvectors Perturbation (EV)

Input : Dataset
Output: Matrices A_j

1 Count the number of rows r in the dataset, where x_p is a given attack;
2 Count the number of c columns with numerical values (i.e. exclude the label field);
3 Find $k = r - (r(mod(c)))$ will be smaller than or equal to r;
4 Select arbitrary k rows from the dataset;
5 Form arbitrary $c \times c$ matrices from the datasets. We should get $t = k/c$ matrices. Call these matrices A_j;

The prediction algorithm is illustrated in Algorithm 6.

Algorithm 6: Prediction using Algorithm for Eigenvectors Perturbation (EV)

Input : New Flow

Output: Predicted Class

1 Take matrix A_j out of the generated matrix;

2 Take the eigenvectors of matrix A_j;

3 Form a row vector from a new data flow, call it F;

4 Take N_i the norm of order 1 row i divided elementwise by the flow F , where $Norm_i(r_i/F)$ is the vector norm of order 1 and r_i is the i_{th} row of the matrix A_j. Take the norm of the norms of eigenvectors of A_j, call it N_{Aj};

5 Replace r_i in A_j by F. Compute N_{Bj}, the norm of the norms of the eigenvectors of A_j after replacement;

6 Find D_{AB}, the absolute value of the difference of N_{Aj} and N_{Bj}, $D_{ABj} = |N_{Aj-Bj}|$;

7 Find $MinN_j$, the minimum of all N_i for matrix A_j;

8 Reinstate matrix A_j back to original;

9 Repeat from step 4, that is select the next row with $i \rightarrow i + 1$ until all rows of matrix A_j are used;

10 Repeat from step 1, that it, select the next matrix with $j \rightarrow i + 1$ until all matrices are covered;

11 Find the minimum of D_{ABj};

12 Find the minimum of $MinN_j$;

13 The result of the attack X dataset is then $Res(X) = min(D_{ABj} * min(MinN_j))$;

The above algorithm generates the number $Res(X_p)$ for each attack dataset X_p. To find out which attack dataset the flow belongs to, the minimum of these numbers needs to be determined, i.e., $min(Res(X_p))$. The following is a flowchart depicts the algorithm.

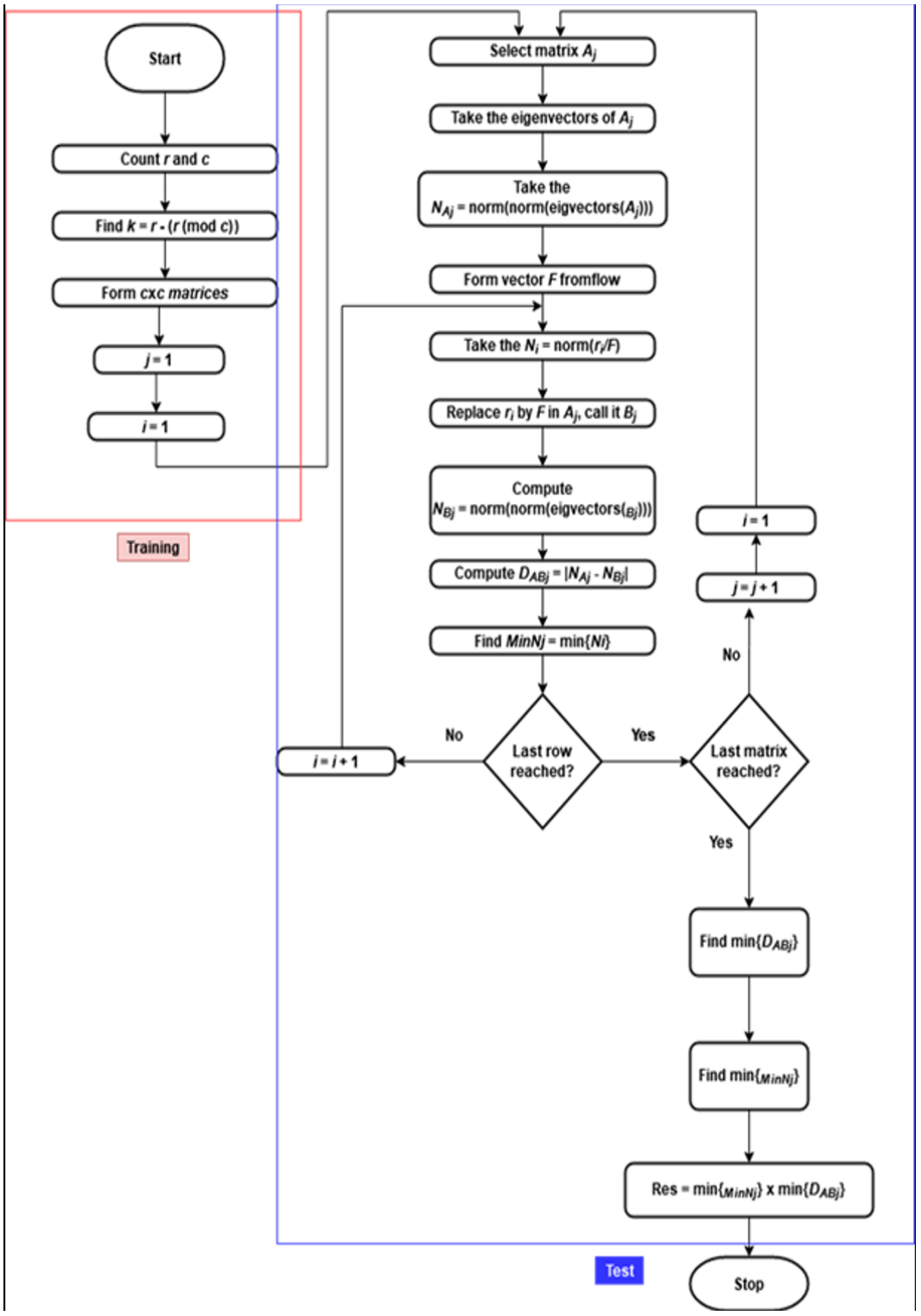


FIGURE 4.4: A flowchart depicting the Eigenvectors Perturbation algorithm

4.4 Summary

This chapter has provided the details of two new classification algorithms developed as part of this research. These algorithms outperform exiting ones for the dataset at hand and this is demonstrated via empirical evaluation. The algorithms are evaluated and implemented in Chapter 6 of this thesis. The next chapter contains an explanation of the work on improving the classical PCA algorithm for better PCAP data visualisation.

Chapter 5

VISUALISATION OF PCAP DATA USING PCA

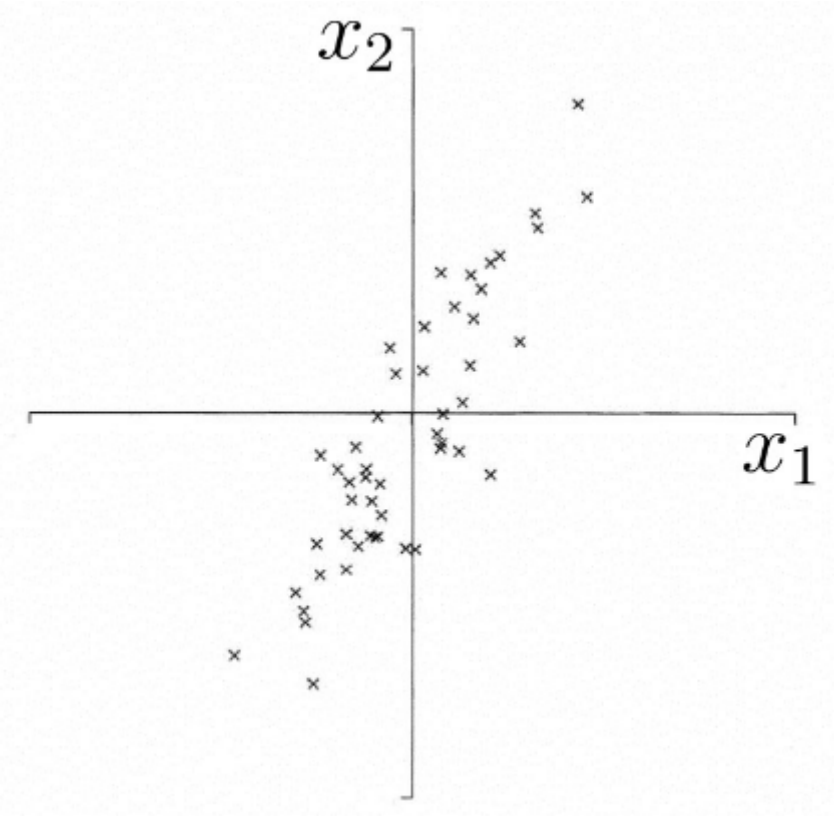
In this chapter a proposed enhancement of the classical PCA algorithm is provided. Before doing this, an overview of the PCA algorithm and explanation of how it works is introduced. After that, an explanation of how PCA can be enhanced and used for pre-processing and visualisation of network traffic data is detailed.

5.1 Principal Component Analysis (PCA)

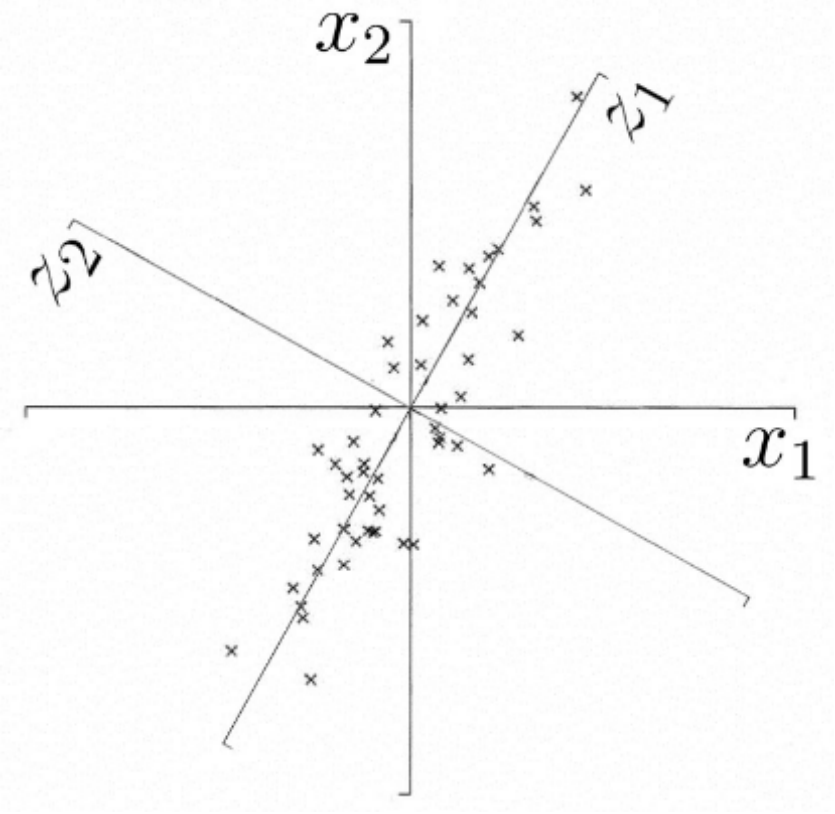
Principal Component Analysis (PCA) is a machine learning concept that is commonly used to reduce the dimensions of large datasets without losing the significance of the information in hand (Jolliffe, 2011). It is usually used to help visualise large datasets. The ideal is to transform possibly correlated variables into linearly uncorrelated variables called principal components. Here, the first principal component contains the largest amount of information, the second principal component contains the second largest

amount of information and so on. The amounts of information account for the variance of each of the variables. Hence, the PCA algorithm requires maximising the variance for each variable (or axis) and assigning it to the first principal component, second principal components and so on, such that axes with small variances may be ignored without loss of much information (Tipping and Bishop, 1999).

Geometrically, PCA is achieved by fitting an axis (or vector) across a set of points in every dimension. For example, if the data is two dimensional, two vectors are fitted across the data, which are perpendicular to ensure they are uncorrelated. This is shown in Figure 5.1.



(A) A scatter plot of two numeric variables



(B) Transformation of data by fitting orthogonal axes (Z1 and Z2 represent the first and second principal components respectively)

FIGURE 5.1: Visual Illustration of PCA (Diagram from (Masci, 2013))

It is obvious from the above figure that the variance is higher across the first axis, so it accounts for the first principal component, and the second axis is the second principal component. This approach is generalised to higher dimensions.

5.2 Computation of PCA

PCA has been explained in detail in the literature. Formulae illustrated in this chapter are summarised from (Jolliffe and Cadima, 2016) and (Brereton, 2009). Suppose the data are given in an n *by* m matrix representing m variables (dimensions), say $X_1, X_2, ..., X_m$ and n values. The PCA algorithm starts with subtracting from each data entry the mean of its variable, as shown below:

$$X = \begin{pmatrix} x_{11} - \mu_1 \cdots x_{1m} - \mu_m \\ \vdots \ddots \vdots \\ x_{n1} - \mu_1 \cdots x_{nn} - \mu_m \end{pmatrix} \tag{5.1}$$

Where x_{ij} is j^{th} value of the i^{th} variable and μ_i is the mean of the i^{th} variable. This generates a dataset of the same variables but with means all equal to zero. Sometimes it is also a common practice to standardise the dataset. The is achieved by subtracting the mean and dividing by the standard deviation for each entry in the matrix:

$$X = \begin{pmatrix} \frac{x_{11} - \mu_1}{\sigma_1} \cdots \frac{x_{1m} - \mu_m}{\sigma_m} \\ \vdots \ddots \vdots \\ \frac{x_{n1} - \mu_1}{\sigma_1} \cdots \frac{x_{nn} - \mu_m}{\sigma_m} \end{pmatrix} \tag{5.2}$$

Then it is required to calculate the covariance matrix for all pairwise combinations of the variable as below:

$$C = \begin{pmatrix} cov\,(x_1, x_1) \cdots cov\,(x_1, x_m) \\ \vdots \ddots \vdots \\ cov\,(x_m, x_1) \cdots cov\,(x_m, x_m) \end{pmatrix} \tag{5.3}$$

which is a symmetric m *by* m matrix.

It is also important to note that the correlation matrix can be used instead of the covariance matrix. The choice depends on the data in hand. Generally, the covariance matrix is chosen when the variables are measured in comparable units so the differences in the variance between variables make significant interpretation. In contrast, the correlation matrix is chosen when the variables are measured in different units so the differences in the variance between variables do not make much sense. The correlation matrix is given below:

$$Cr = \begin{pmatrix} corr\,(x_1, x_1) \cdots corr\,(x_1, x_m) \\ \vdots \ddots \vdots \\ corr\,(x_m, x_1) \cdots corr\,(x_m, x_m) \end{pmatrix} \tag{5.4}$$

Now eigenvectors and eigenvalues of the covariant matrix need to be computed. As the eigenvectors are orthogonal (given C is symmetric) and are taken for the covariance matrix, they will act as the axes in the transformation. Given matrix C is m *by* m, the result is m eigenvectors and m eigenvalues. The eigenvectors can be normalised by dividing each by its length. This yields eigenvectors each of length 1.

At this stage, all calculations needed to construct the principal components are available, which are the eigenvectors of the covariant matrix C. Moreover, the eigenvector with the highest eigenvalue is the first principal component, and that with the next highest is the second principal component and so on. Therefore, it is possible to ignore eigenvectors with less significance, and hence reduce the dimensionality of the original data. This is done by constructing the feature vector F containing all eigenvectors that should be kept. Therefore, F will be a matrix of size n *by* p, where $p \leq m$ depending on the number of eigenvectors ignored:

$$F = (e_1 \cdots e_p) \tag{5.5}$$

Here $e_1, ..., e_p$ are the eigenvectors decided to be kept. As these eigenvectors are ordered by significance based on the eigenvalues, the choice is typically made based on the first p out of n eigenvectors if no other criteria to do otherwise are involved.

The last step is to transpose the feature vector F and multiply it by the transpose of matrix X of the values. This will form the matrix Y of the final dataset:

$$Y = F^T X^T \tag{5.6}$$

The new dataset Y has dimension p *by* n, containing p variables and n values:

$$Y = \begin{pmatrix} y_{11} \cdots y_{1n} \\ \vdots \ddots \vdots \\ y_{p1} \cdots y_{pn} \end{pmatrix} \tag{5.7}$$

which can be transposed to get it in the usual form.

The first principal component Y_1 is the first row of matrix Y, the second principal component Y_2 is the second row of matrix Y and so on:

$$Y_1 = y_{11} + \cdots + y_{1n}, \tag{5.8}$$

$$\vdots$$

$$Y_p = y_{p1} + \cdots + y_{pn} \tag{5.9}$$

The original dataset, matrix X, can be retrieved from Y provided all eigenvectors have been selected to form Y. This is achieved by multiplying the inverse matrix of F (or its transpose) from the left on both sides:

$$(F^T)^{-1} Y = (F^T)^{-1} F^T X^T = I X^T = X^T \tag{5.10}$$

It is important to notice that the columns of F are unit orthogonal vectors, and so it is an orthogonal matrix. An orthogonal matrix has the property that its inverse is equal to its transpose. Therefore, it is enough to multiply F from the left on both sides to retrieve X:

$$YF = FF^T X^T = IX^T = X^T \tag{5.11}$$

And to retrieve the original dataset, the means initially subtracted from the values are added. If not all eigenvectors are considered, it is still possible to proceed the same but some of the data will not be retrieved.

5.3 How PCA Can Help Visualise Large Datasets

If there are large datasets to visualise, it is not usually possible to display all data in a visual format for inspection purposes. One of the ways enabling large dataset visualisation is PCA. Since data can be arranged in order of significance, it is possible to choose to display as many principal components as possible to visualise. It is therefore an efficient method of visualisation commonly used when faced with large datasets. Some example visualisations are included in the following sections of this chapter.

As for implementation, Python has a PCA implementation with the *sklearn.decomposition* class. For example, to use the PCA kernel in Python, the following lines of codes calls the PCA kernel in Python are used to call the PCA kernel and Fit the model from data in X and transform X:

$KernelPCA(n\_components = n, kernel = "linear")$

$fit\_transform(X)$

For the first experiment, the dataset will be visually classified based on the label, malicious or attack. Further classification results will also be attempted based on PCA.

5.4 Improvement of the PCA Algorithm

As detailed above, the use of the PCA algorithm requires forming the covariance matrix or the correlation matrix. Observe that we do not focus on Kernel PCA in this work. In Kernel PCA, the data is mapped into a higher-dimensional space before reducing the dimensionality (Wang, 2012) and this is not required in our case. The covariance and correlation are measures of a linear relation between two variables. The correlation is equal to the covariance divided by the product of the standard deviation of each variable. Geometrically, the correlation is the slope of the regression line of two variables. Given the dataset in hand, linear correlation will not be a good fit for the data. Therefore, nonlinear correlation would be more representative of the data and would consequently generate more representative principal components of the data. Spearman's rho assumes two variables are monotonically related (Ain et al., 2016), which is the case of the data being studied. Some tests were conducted to compare the results of using correlation coefficient and Spearman's rho and at each test Spearman's rho outperformed the correlation coefficient.

To provide an example, a set of 37 variables representing the time and 36 values of network parameters was used in an experiment. The data is a subset of the original data used for this research (i.e. the IPS dataset mentioned in previous chapters). The aim of this experiment is to show the advantage of using the Spearman matrix over the correlation matrix for the data at hand, assuming that the variables are monotonically related. Principal component analysis was conducted using two principal components

and results are illustrated in Figure 5.2 and 5.3. The horizontal axis represents the time of the measure and the vertical axis represent one of the other 36 variables of the dataset. PCA was conducted the principal components were generated using each of the matrices. The data was then transformed into a reduced dataset of two variables out of 36. The chosen variable's data points are in blue. An inverse PCA transform of the the new dataset was obtained to regain the original data.

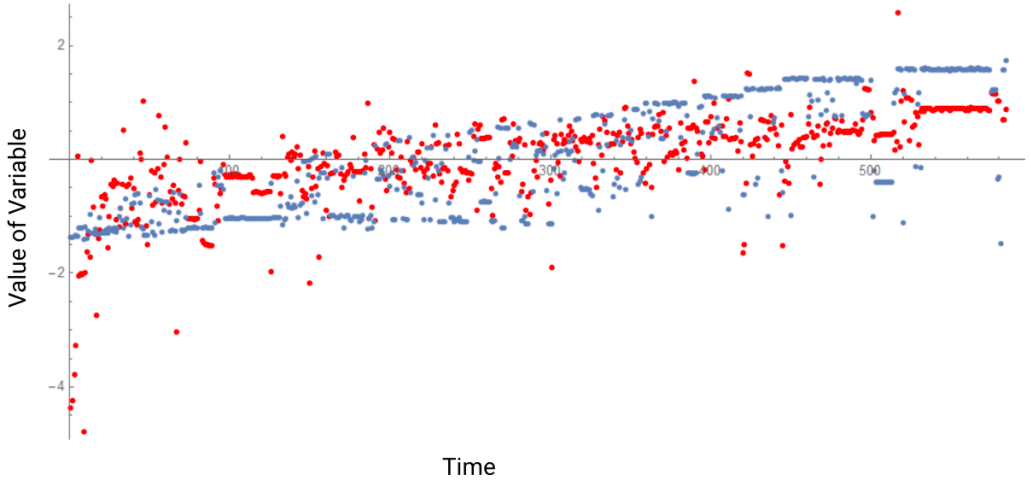


FIGURE 5.2: The actual data (blue) and the reconstructed data (red) after using Spearman's rho

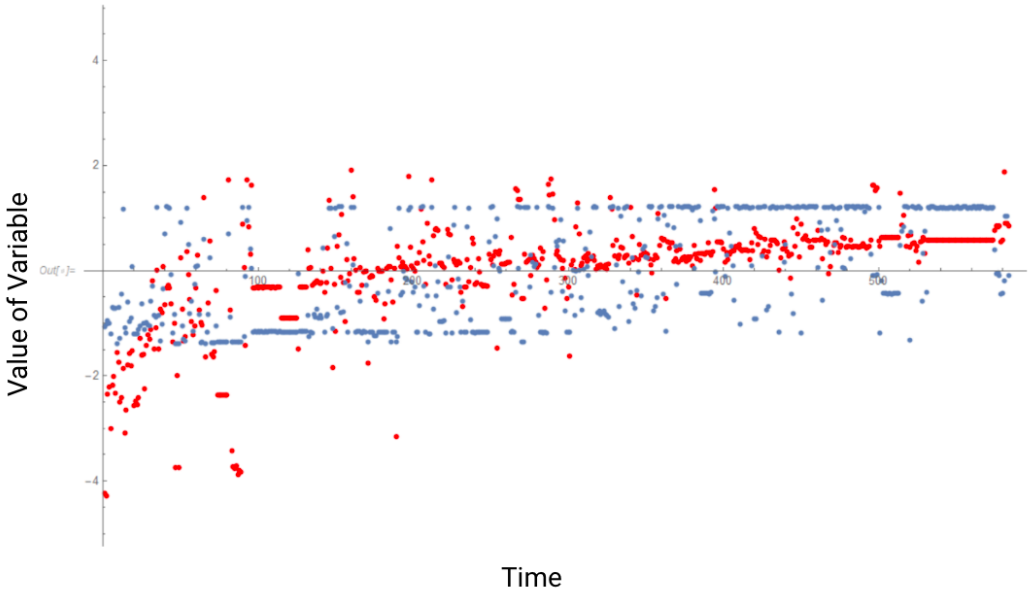


FIGURE 5.3: The actual data (blue) and the reconstructed data (red) after using the correlation coefficient

It can be noticed from Figure 5.2 and 5.3 that Spearman's rho results in data that is less sparse compared with the original. This result is also maintained by communality calculated for each test, which is 89.5% for the highest value variable when correlation is used and 96.7% when Spearman's rho is used. Communality is the total amount of variance that is shared between an original variable and all other variables included in the analysis (Watkins, 2018). The experiment was repeated for the other 35 variables and the Spearman's matrix performed better at each case.

5.5 Visualisation using PCA Spearman's Rho

The proposed PCA algorithm is used with the covariance matrix replaced by the Spearman's rho matrix. This is based on the fact that Spearman's rho matrix assumes two variables are monotonically related as it is the case of the datasets in hand. Since PCA involves plotting data points, the Python Matplotlib package (Hunter, 2007) is used. Here a comparison between three PCA algorithm implementations is carried out. Namely, the implementations are different because they use Pearson (Freedman et al., 2007), Spearman and covariance correlations respectively.

The data used in the comparison is network traffic data and samples are labelled based on the attack type: PortScan, DDoS, Bot or Benign. A Benign label means the flow does not generate an attack. The idea behind using PCA is to project the multidimensional data into a lower dimensional space in order to visualise it. In this case, the data has 76 dimensions (i.e. features) and is projected onto a two-dimensional space with two axes: Principal Component 1 and Principal Component 2.

Having the algorithms implemented, it becomes possible to visualise the resulting data after using PCA with different correlation matrices. Example results are provided below.

5.5.1 Visualising a Sample using the Covariance Matrix

Here Python's scikit-learn package, which has an implementation of PCA that uses covariance matrix by default, is used. A sample of 4000 flows drawn randomly from the data is shown as a scatter plot below (Figure 5.4):

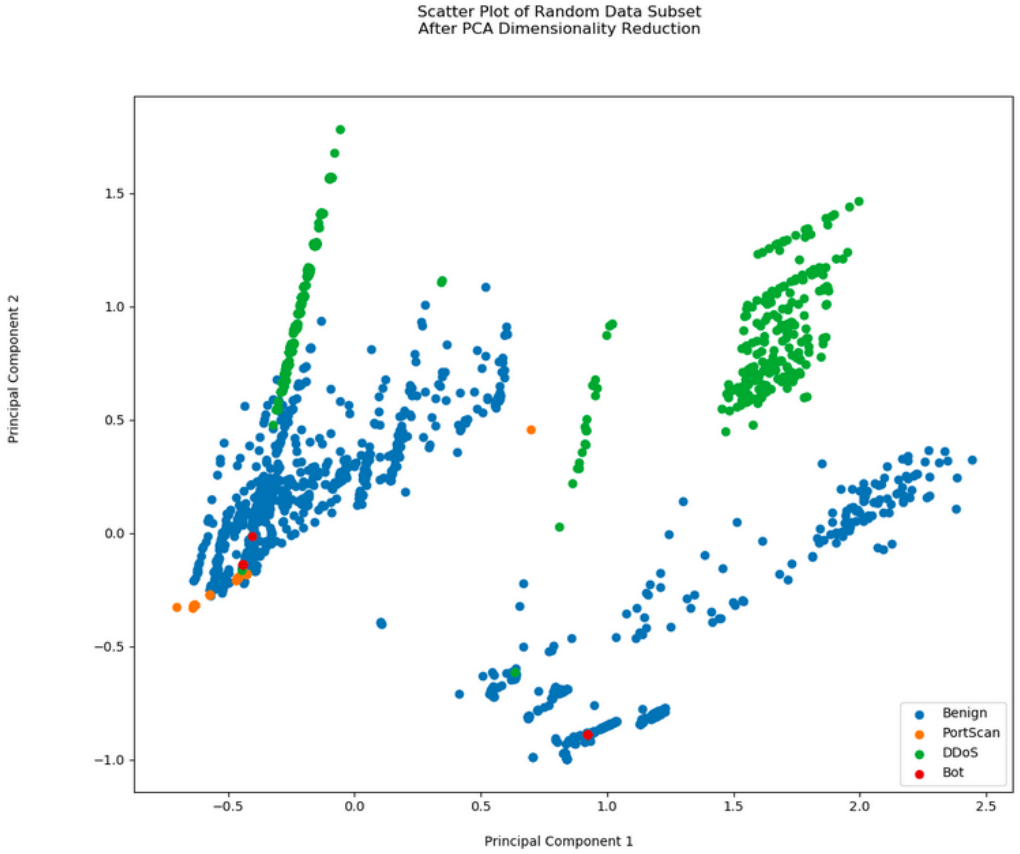


FIGURE 5.4: PCA results after using the covariance matrix.

The diagram shows the possibility of linearly separating most of the data points, with some exceptions, e.g., parts of the green points, red points, etc.

5.5.2 Visualising a Sample using the Pearson Matrix

Replacing the covariance matrix in the PCA algorithm with Pearson's correlation matrix for the same data sample results in the following (Figure 5.5):

FIGURE 5.5: PCA results after using the Pearson matrix.

It looks that the data using PCA with the Pearson matrix are similarly distributed as those with the covariance matrix. The reason is that the Pearson matrix is closely related to the covariance matrix. In fact, it is a normalised version of the covariance matrix.

5.5.3 Visualising a Sample Using the Spearman Matrix

Finally, replacing the covariance matrix in the PCA algorithm with Spearman's correlation matrix in the PCA algorithm produces the following (Figure 5.6):

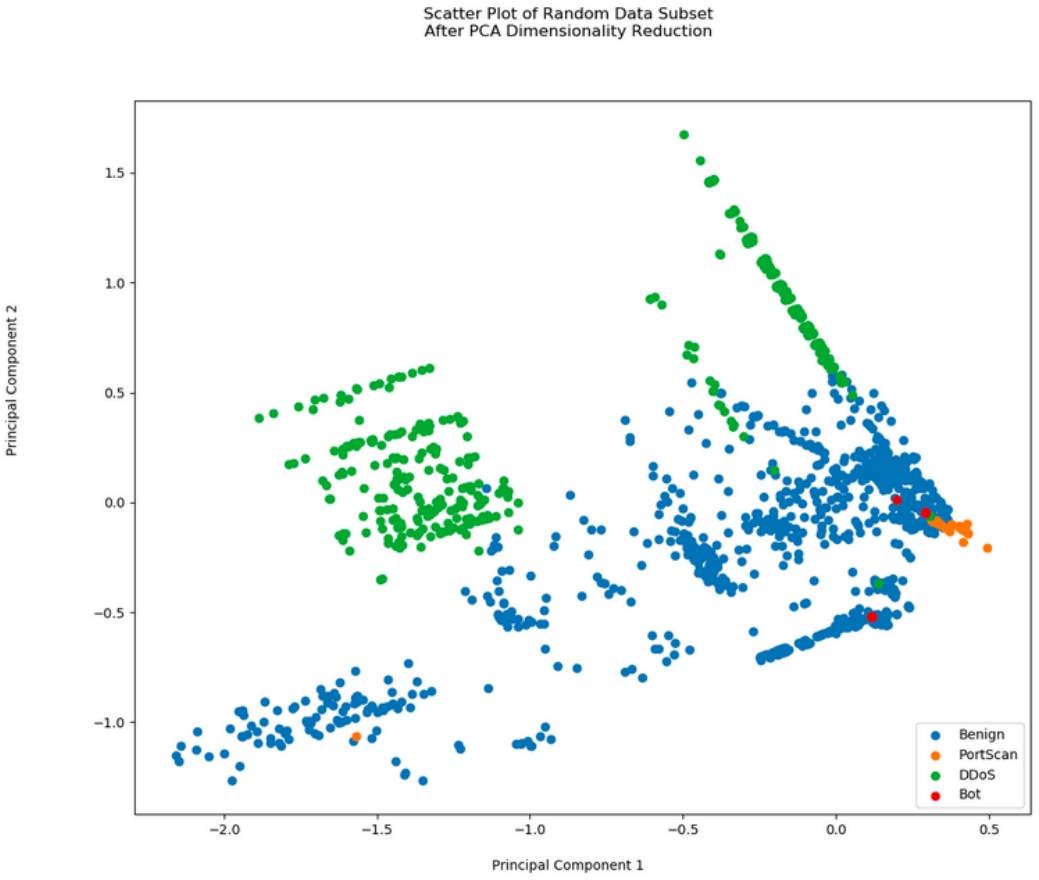


FIGURE 5.6: PCA results after using the Spearman matrix.

The figure above shows that the data are more separable than the case of the covariance and Pearson matrices. This is based on the fact that the data at hand are positively correlated and hence the Spearman matrix can best reflect such correlation. This demonstrates the enhancement that the idea introduced in this thesis makes in this use-case.

5.6 Summary

This chapter has provided an insight into computer network security data visualisation. In particular the principal component analysis has been introduced and discussed, with practical implementation in Python. The improvement to the PCA for the data at hand

has been achieved by using the Spearman matrix over those using the covariance matrix or the Pearson matrix. It is evident that the change of matrix in the PCA results in a better separation of the data items. This has been shown visually throughout the chapter. The next chapter has empirical evaluations of the classification methods introduced as part of this thesis.

Chapter 6

CLASSIFICATION RESULTS AND EVALUATION

This chapter contains the experiments (in order to evaluate the performance of the introduced methods) and results as well as discussion of those results. Here the evaluation of the methods proposed as part of this thesis will be demonstrated. The chapter begins by speaking about the data used in the evaluation process and then moves to a detailed evaluation of the proposed MTL method for classification. After that, the evaluation of the Eigenvector Perturbation (EV) classifier is explained. The experimental results show that the proposed methods are indeed effective.

6.1 Network Traffic Data

Usually network traffic data is captured in a binary format known as the PCAP format (Packet CAPture). It is possible to extract numeric features from this data and transform it into tabular format suitable for machine learning and further analysis. An

existing open-source tool is ISCXFlowMeter (Draper-Gil et al., 2016). After the data is ready for machine learning, various classification techniques can be used to classify traffic and identify its type (i.e. whether it is malicious or benign. And in case of malicious traffic, it is desirable to identify the attack type).

Currently, there are several well known network attacks of which plenty of data is available. On the other hand, there are several attacks that are not well studied because of the scarcity of their captured data. (Figure 3.7 shows an example distribution of instance classes from a dataset captured by an IPS). An example source of such data is available at (Canadian Institute for Cybersecurity, 2017). The available data is of a real-world situation where an Intrusion Prevention System (IPS) can be used to spot network attacks in real-time. As explained in the beginning of this thesis, an IPS is a computer program that is used to monitor computer network traffic, attempt to identify malicious, or suspicious, activities and generate alerts when such activity is found (Di Pietro and Mancini, 2008). Usually, malicious traffic is much less than normal (safe or benign) traffic, therefore, it is often seen as an anomaly.

The data shows a significant difference between the number of available instances that belong to different traffic types (in total it contains 2830742 instances). For example, the number of instances available for benign traffic is as high as 2273096, the number of instances for the DDoS attack is as high as 128027. On the other hand, the number of instances available for an Infiltration attack is as low as 36 and the number of available instances for a Heartbleed attack is as low as 11. The full distribution of instance classes of this dataset is shown in Figure 3.7. This is a typical situation as datasets of this type are usually highly imbalanced.

In a scenario like this (where plenty of data is available for some tasks and very small

amounts of data is available for some other tasks related to these tasks), usually poor predictive models are obtained when learning the tasks with small amounts of data.

As explained in previous chapters, in this thesis, each network traffic category is referred to as a task (i.e. Benign is a task, DDoS attack is a task and so on). Because of the low performance of such models, some improvement is required. This thesis introduces a multi-task learning approach to achieve such improvement. The idea is to learn tasks with large amounts of data together with tasks with small amounts of data in order to obtain better models for the latter tasks. More on multi-task learning was provided in Section 4.1.

In order to run experiments to evaluate the MTL method developed as part of this thesis, four tasks and some instances were randomly selected as shown in Table 6.1. This data was used in the experiments explained in the remainder of this chapter.

TABLE 6.1: Data used for MTL Experimental Evaluation

Network Traffic Type (i.e. Task)	Number of Instances
Benign Traffic	41176
DDoS Traffic	58168
Bot Traffic	45
Portscan Traffic	57

6.2 Evaluation of the Multitask Learning Approach

A detailed evaluation of several classical classifiers was presented in Chapter 3. In this section the experiments are presented and their results are shown and discussed. To

demonstrate the effectiveness of the proposed method, the IPS dataset explained in section 6.1 has been used.

Before starting the actual MTL experiments, an initial evaluation to select the best base-line classifier that can be trained on the dataset resulting after applying Algorithm 2 (which was explained in Section 4.1.1) has been run. The results show that RandomForest is the best performer (and this is consistent with the work in (Alothman et al., 2018)). Hence, RandomForest is going to be used as the base classifier for the MTL method.

An example dataset resulting after applying algorithm 2 is shown in Figure 6.1. Notice the original features are on the left of the *label* column, which is the class column (each class is a task). The similarity values are on the right (this example has four new columns because this dataset contains four unique class labels as shown in the *Network Traffic Type (i.e. Task)* column in Table 6.1).

Fwd Packets/s	Bwd Packets/s	label	Sim2DDoS	Sim2Benign	Sim2PortScan	Sim2Bot
0.639	0.724	BENIGN	0.750253	1.000000	0.999135	0.552243
618.000	1236.090	DDoS	1.000000	0.825122	0.099968	0.489020
216.000	288.000	BENIGN	0.245391	1.000000	0.000055	0.571679
558.000	1116.150	DDoS	1.000000	0.694103	0.000006	0.583908
0.303	0.454	BENIGN	0.562439	1.000000	0.997833	0.564032
...	...	...	...	...	...	...
8.380	10.200	BENIGN	0.649460	1.000000	0.970705	0.598852
9.880	19.800	DDoS	1.000000	0.226170	0.465622	0.193312
8.340	19.500	DDoS	1.000000	0.202440	0.578942	0.370646
15.900	20.200	BENIGN	0.775083	1.000000	0.864287	0.451461
11.600	19.400	DDoS	1.000000	0.289061	0.345324	0.258006

FIGURE 6.1: An Example Dataset resulting after the Proposed MTL Method

Although not all of them are displayed, the *label* column in the dataset shown in Figure 6.1 contains four unique values.

6.2.1 Comparison with RandomForest

In order to demonstrate the effectiveness of the method proposed in this thesis, its performance is going to be compared against the best performing classical classifier (which is RandomForest as stated above). As there are four separate datasets (one for each of the tasks shown in table 6.1), several datasets have been created by using various pairwise combinations of datasets (each two different tasks against each other). After obtaining these dataset combinations a 10-fold cross-validation procedure has been run using RandomForest (cross-validation and its benefits have been explained in Chapter 3). This procedure generates several train/test splits, trains a RandomForest model on the train split and uses it to predict the test split. After this the predictions are grouped by label (i.e. task) and the average accuracy is computed for each label (i.e. task). Figure 6.2 shows the results for each pair of tasks (read row vs column).

	Benign	DDoS	Bot	Portscan
Benign	1	(0.9997, 0.9997)	(0.9999, 0.8000)	(0.9999, 0.7543)
DDoS	(0.9996,0.9996)	1	(1.0, 0.8666)	(0.9999, 0.5263)
Bot	(0.7999, 0.9999)	(0.8888, 1.0)	1	(0.9555, 1.0)
Portscan	(0.7543, 0.9999)	(0.5087, 0.9999)	(1.0, 0.9555)	1

FIGURE 6.2: Results of RandomForest on Pairwise Task Combinations

As for the MTL evaluation procedure, the same datasets that have resulted after the pairwise dataset combination procedure have been used but now the similarity values are added to them as extra features as explained in section 2.6.2. The results are shown in Figure 6.3.

	Benign	DDoS	Bot	Portscan
Benign	1	(1.0, 1.0)	(1.0, 1.0)	(1.0, 0.9824)
DDoS	(1.0, 1.0)	1	(1.0, 0.9777)	(1.0, 1.0)
Bot	(1.0, 1.0)	(0.9777, 1.0)	1	(0.9777, 1.0)
Portscan	(0.9824, 1.0)	(1.0, 1.0)	(1.0, 0.9777)	1

FIGURE 6.3: Results of the MTL Procedure on the same Pairwise Task Combinations

Notice the significant improvement in performance after using out MTL method (compare corresponding cells). Please observe that it does not really matter which base classifier is used (the novel aspect of this work is adding the similarity values to improve the classification accuracy). The usefulness of this method can be observed if one focuses on labels where the dataset size is small (i.e. there is a significant improvement when learning is performed for Bot and Portscan tasks).

6.2.2 Comparison with CB-SBIT

The implementation of the CB-SBIT algorithm (Alothman et al., 2018) is freely available. Therefore, it has been downloaded (it source code is available at (Alothman, 2018*a*)) and used to perform evaluation and comparison with the method proposed in this thesis. In order to run the experiments, the data has been prepared so that it is compatible with how CB-SBIT works. Three attack types have been selected (Bot, Portscan and DDoS) and Benign data has been added to them (making sure the resulting datasets are class balanced and there is no overlap in Benign instances). It is worth mentioning here that accuracy is going to be used as the performance evaluation metric because datasets are class balanced (Santafe et al., 2015). Notice that these datasets do not contain the similarity columns added as part of the MTL procedure. Then the available CB-SBIT code has been as is in the following way:

1. Take the Bot data and randomly split it into two equally sized datasets (one for use as a Target data in the CB-SBIT algorithm and the other for use as test data)
2. The Portscan and DDoS datasets were used as source datasets in CB-SBIT
3. CB-SBIT was run with its default parameters

The original Bot Target dataset size was 45 instances (21 Benign and 24 Bot) and the new Bot dataset size (after instance transfer by the CB-SBIT algorithm) was 118489 instances (59243 Benign and 59246 Bot). As the new Bot dataset is much larger than the original dataset one would expect a better model as plenty of data is now available. However, the results are not as expected. Accuracy before using CB-SBIT (i.e training the RandomForest classifier on the original training Bot dataset) is 94.78% and accuracy after transfer (i.e training the RF classifier on the new dataset that contains the original instances of the small training Bot dataset and the instances transferred to it from the source datasets using the CB-SBIT algorithm) is 93.56%. This is a surprising result as the CB-SBIT is performing what is known as *negative transfer* (i.e. results are worse after using the CB-SBIT algorithm).

The previous steps have been repeated but now with using the Portscan data as Target and Test, and adding the Bot dataset to the source datasets.

1. Take the Portscan data and randomly split it into two equally sized datasets (one for use as target data in the CB-SBIT algorithm and the other for use as test data)
2. The Bot and DDoS datasets were used as source datasets in CB-SBIT
3. CB-SBIT was run with its default parameters

The original Portscan Target dataset size was 57 instances (26 Benign and 31 Portscan) and the new Portscan dataset size (after instance transfer by the CB-SBIT algorithm)

was 203877 instances (101936 Benign and 101941 Portscan). Accuracy before transfer (i.e training the RF classifier on the original small training Portscan dataset) is 82.45% and accuracy after transfer (i.e training the RF classifier on the new dataset that contains the original instances of the small training Portscan dataset and the instances transferred to it from the source datasets using the CB-SBIT algorithm) is 96.25%). This is positive transfer (i.e. results are better after using the CB-SBIT algorithm)

The proposed MTL method has been applied using the same datasets which have been used to evaluate the performance of the CB-SBIT algorithm. Remember in the MTL method the similarity is computed and new columns are added as features (as explained in Section 2.6.2). Therefore, these datasets contain the similarity columns that are computed as part of the proposed MTL approach. The procedure was as follows:

1. Concatenate the DDoS dataset with the training datasets of both Bot and Portscan
2. Train a RandomForest classifier on the resulting dataset
3. Predict the Bot and Portscan test datasets and compute the accuracy for each of them

After MTL, the accuracy was 98.51% on the Bot test dataset and 99.76% on the Portscan test dataset. This shows that the proposed MTL approach outperforms the CB-SBIT algorithms on both datasets. The results of this evaluation are shown in figure 6.4. In summary, for the Portscan data, CB-SBIT shows an improvement in accuracy when compared with using the RF classifier without instance transfer (i.e. the single task learning), but MTL shows a better improvement. On the other hand, for the Bot data, CB-SBIT shows a dis-improvement in accuracy when compared with using the RF classifier without instance transfer (i.e. the single task learning), because accuracy goes down, whereas MTL shows a significant improvement.

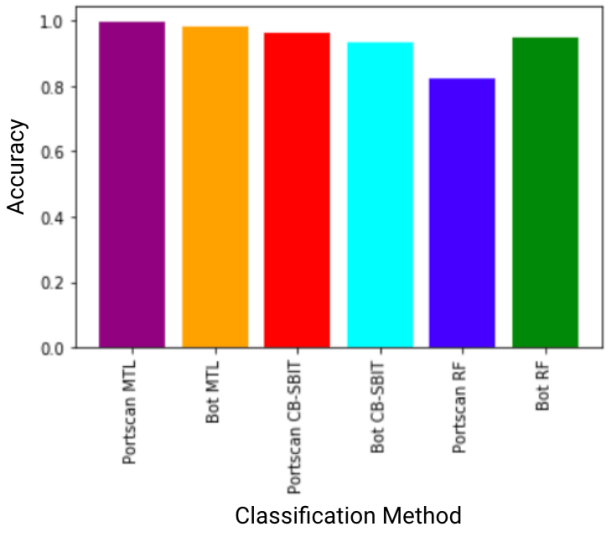


FIGURE 6.4: Results of Comparing Performance of CB-SBIT and the Proposed MTL Approach

6.3 Evaluation of the Eigenvector Perturbation Classifier (EV)

An extensive account of experiments was considered to evaluate the eigenvector classifier. In order to run the experiments, the data were randomly split into two classes (train and test). It is important to highlight that the splits were stratified and that this was a multi-class classification problem. Please remember that the data used here is the random sample mentioned in Table 6.1.

Multiple classifiers were trained on the training split and evaluated using the test split. The classifiers used were:

- Random Forest (RF)
- Multilayer Perceptron (MLP)
- Naive Bayes (NB)
- Decision Tree (DT)

- The developed Eigenvector Perturbation Classifier (EV)

The performance of RF, MLP, NB, and DT were initially compared to decide which of them performed best on the dataset at hand. Then the best classical classifier was compared to the EV classifier.

Among the four classifiers compared, the results confirmed that Random Forest (RF) was the best. Compared to the previous experiments, the best classical classifier was RF as well. This means, if the proposed algorithm (EV) outperformed Random Forest on the same data, then it would be considered better than those classical classifiers. Therefore, the plan was to compare the performance of the EV algorithm with that of RandomForest (EV vs RF). The details are provided in the following sections.

6.3.1 EV vs RF (Multi-Class)

The results of the evaluation experiments are illustrated in the figures below (Notice the various performance metrics). In terms of overall multi-class accuracy, it can be seen that RF and DT perform better than EV, although the difference is very small (Figure 6.5). On the other hand, EV performs better when considering the multi-class AUC metric (remembering that the data were highly imbalanced). This gives EV an advantage over RF.

The fact that the EV classifier works better when data is imbalanced makes it more usable in the real-world because class imbalance is a common problem (Ling and Sheng, 2010). In addition, this could be an indication that the EV classifier picks internal structures and patterns in the data which help it in deciding which classes the instances belong to.

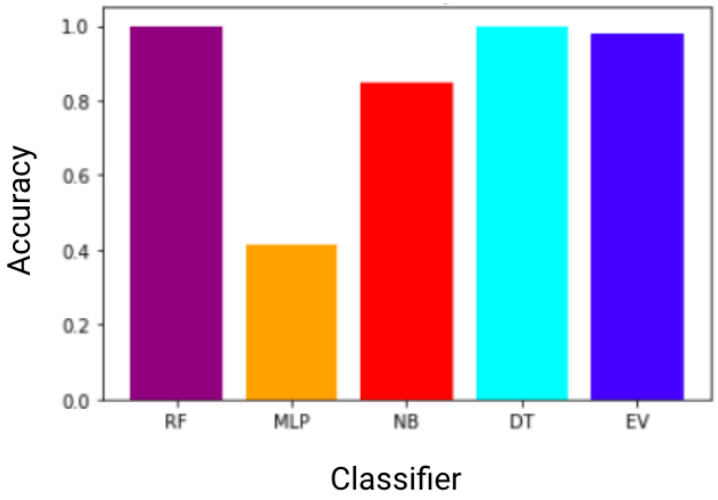


FIGURE 6.5: The accuracy measure of the classifiers. RF, FR and EV achieve very close results

When using multi-class Precision, Recall and F1 the results were interesting. RF achieves a higher multi-class precision (this means it minimises the false positive). On the other hand, EV achieves a higher multi-class recall (this means it minimises the false negative) (Figure 6.6). In the case at hand, minimising the false negative is more important than minimising the false positive (false negative is more costly than false positive). Therefore, EV should be preferred.

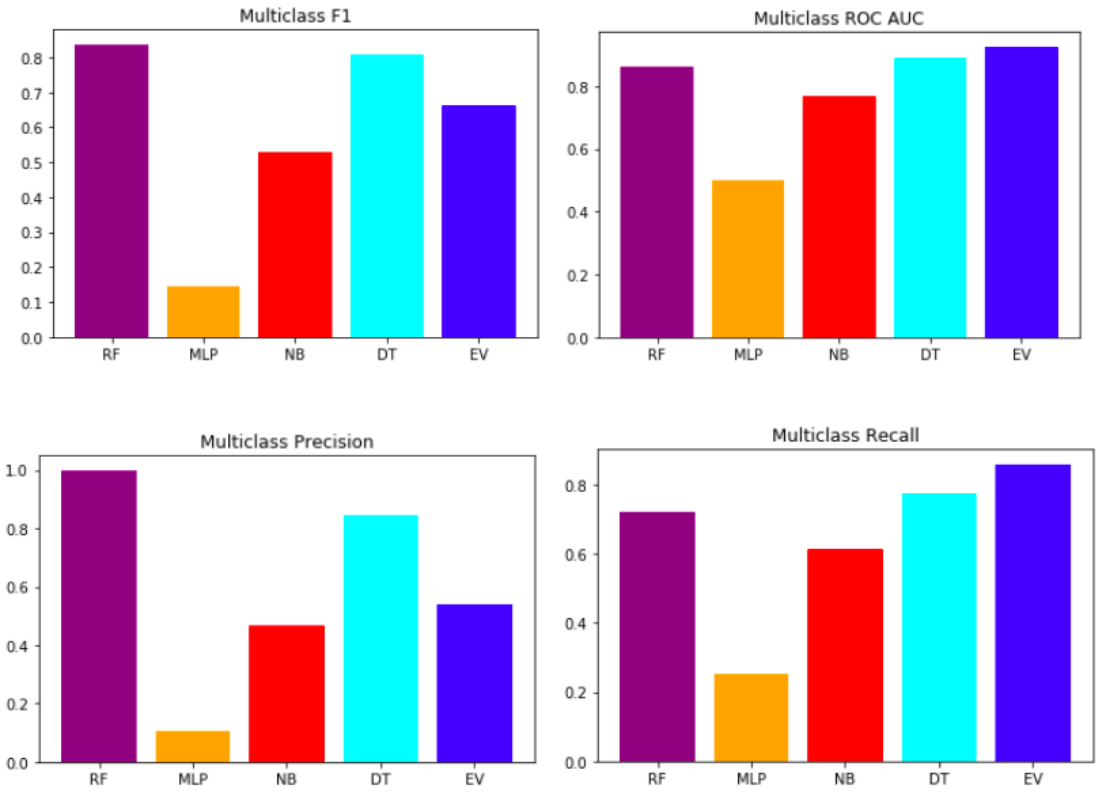


FIGURE 6.6: Classifier Evaluation (X-axis shows the classifier name and Y-axis shows the metric value).

The results are summarised in Table 6.2. Please remember that the values reported are between zero and one. The best classifiers are those with values as close to one as possible.

TABLE 6.2: Comparison Between Multiple Classifiers.

Classifier Name	Multiclass F1	Multiclass ROC AUC	Multiclass Precision	Multiclass Recall
RF	0.84	0.85	0.98	0.71
MLP	0.13	0.44	0.08	0.23
NB	0.52	0.71	0.45	0.61
DT	0.79	0.9	0.82	0.78
EV	0.63	0.98	0.53	0.82

6.3.2 EV vs RF (Individual Classes)

In this experiment, the accuracy of each classifier in each individual class was considered. This means that the predictions were filtered according to the actual class of the test instances and the accuracy was computed. There are four unique classes, namely, 'DDoS', 'BENIGN', 'Bot', and 'PortScan'.

The reason this experiment was conducted was to investigate the performance of each classifier on each type of traffic. This is important especially in cases of small amounts of data (e.g., when there is a new attack and not much data is available such as the case of the Bot and PortScan attacks, where the dataset does not contain many records of these attacks.

When looking at the accuracy of all the used classifiers on 'DDoS' data, the performance of each of RF, NB, DT and EV is relatively good (although RF wins by a small margin). MLP predicts all test data as 'BENIGN' which means all predictions are wrong in this case (hence the 0% accuracy).

When looking at the accuracy of all the used classifiers on 'BENIGN' data, the performance of RF, MLP, DT and EV is again good (although RF wins by a small margin and NB is poor). However, there are a high number of 'DDoS' and 'BENIGN' data in the training and test data.

When evaluating the smaller classes, the story is different! When looking at the accuracy of all the used classifiers on the 'Bot' data, the performance of EV is clearly better than that of RF, NB, DT and MLP.

When looking at the accuracy of all the used classifiers on 'PortScan' data, the performance of EV is clearly much better than that of RF, NB, DT and MLP. Therefore,

the EV classifier should be preferred when data is scarce (only a small amount of data is available) (Figure 6.7).

This explains why EV achieved better recall in the multi-class experiments. Because it is not easy to obtain a highly accurate predictive model when available data is small in size, it is important to use a classifier that learns as much discriminative details as possible. This is what the EV classifier seems to do and this is justified by its superior performance when compared with other classifiers as demonstrated by the experiments.

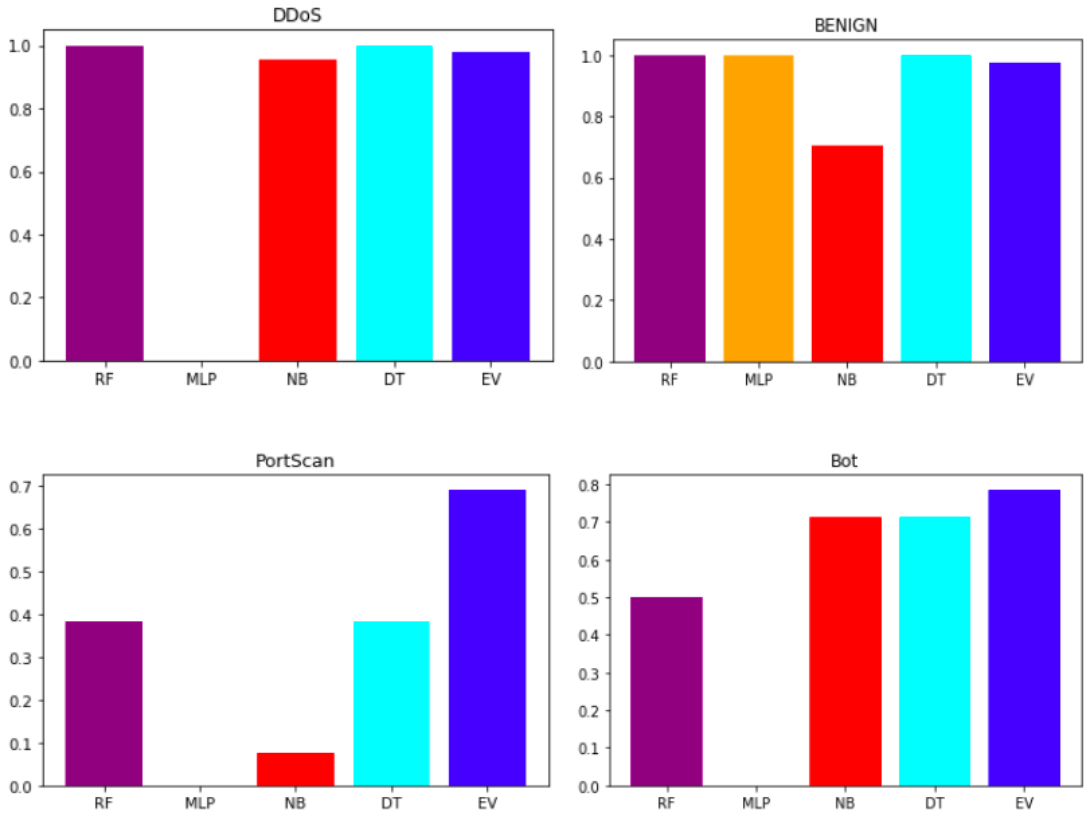


FIGURE 6.7: Classifying individual classes. (X-axis shows the classifier name and Y-axis shows the Recall value).

The results are summarised in Table 6.3. Please remember that the Recall values reported are between zero and one. The closer to one the better the classifier.

TABLE 6.3: Recall Values for Multiple Classifiers on Individual Classes.

Classifier Name	DDoS	BENIGN	PortScan	Bot
RF	0.98	0.99	0.39	0.5
MLP	0	1	0	0
NB	0.96	0.72	0.11	0.71
DT	0.98	0.97	0.38	0.7
EV	0.97	0.95	0.69	0.78

6.4 Comparison of Suggested PCA Variation with Classical PCA Algorithm

This section involves comparison of PCA algorithm implementations using: (1) Pearson's correlation coefficient (this is the variation proposed in this thesis), (2) Spearman correlation coefficient and (3) covariance correlations (which is the classical PCA algorithm). The comparison is done automatically as follows:

1. Apply the three above methods.
2. Use the resulting principal components as input to a classifier (using the same data splits for a fair comparison).
3. Determine the method that leads to better classification results.

After preparing the data and splitting it into train and test splits (and training a RandomForest classifier using the train split and evaluating its performance using the test split), the following results shown in Table 6.4 were obtained:

TABLE 6.4: Comparison with the Classical PCA Algorithm

	PCA using Spearmanr	PCA using Pearsonr	Traditional PCA
Accuracy	99.15%	99.07%	99.11%
Multiclass ROC AUC	0.89	0.86	0.86
Multiclass Precision	0.86	0.81	0.83
Multiclass Recall	0.78	0.72	0.71
Multiclass F1	0.82	0.76	0.76

Table 6.4 shows that, although the application of the three PCA methods leads to similar accuracy values, when considering *precision*, *recall* and *F1 measure*, the results are better when using the variation suggested in this thesis (i.e. to use the Spearman R). Remember that for these three metrics, the closer to one the better the model.

In addition to the above, a visual comparison of results after applying the three PCA methods was carried out. The comparison was done by applying the three PCA methods on the same data, plotting the resulting first two principal components against each other and colouring instances based on their class (BENIGN, DDoS, Portscan, Bot). The results of this comparison, which show that using Spearman's correlation matrix leads to better separation between different classes, were presented and discussed in Section 5.5.

6.5 Summary

This chapter has provided the details of the implementation and evaluation of the multitask learning and eigenvector perturbation (EV) classification algorithms proposed as part of this thesis. The implementation was conducted in Python. The results clearly

show that the proposed algorithms outperform other algorithms, especially when the data provided are scarce. Therefore, it is safe to say that the methods proposed in this thesis are highly suitable especially in situations where not enough information is available for certain network traffic types (i.e. when the amount of available data is small and limited).

Chapter 7

CONCLUSIONS AND FUTURE WORK

This chapter provides a list of multiple lessons gained after the development and evaluation of the algorithms and approaches explained throughout this thesis. It also summarises the importance of this work and why it is useful. The chapter will start with a section about some general points and then it will introduce two sections, one for the conclusions and lessons learnt from this research and the other will discuss possible future work.

7.1 General Points

Several parts of this thesis provided a detailed overview of what this research project entails and the stages it went through. It has provided an overview of Intrusion Prevention Systems (IPS) and explained how they work. It has explained the types of data they capture and how the data can be transformed into a format suitable machine

learning and data mining algorithms. It has also explained security visualisation and one of its most common concepts, namely, the principal component analysis.

In addition, because this research focused on developing accurate methods for the identification of malicious network traffic, there were experiments conducted to evaluate the performance of traditional classifiers such as Naive Bayes, Random Forest and the Artificial Neural Networks on a freely available IPS dataset. The experiments showed that there is a need for more accurate classifiers which highly supports the research.

A detailed explanation of the two classification methods, which were developed as part of this research, has been provided. The two methods are: a multi-task learning method and a method based on matrix eigenvectors. Furthermore, an overview of the PCA algorithm and how it could be enhanced , for the purpose of obtaining better separation between data from different network traffic types, has been provided in this thesis.

There were extensive experiments to evaluate the performance of the above methods. The research evaluated their performance compared with existing methods and studied where the new methods outperformed those methods and where they did not. These experiments were carried out using freely available datasets.

The research also contributed a paper that explains in detail one of the classification techniques developed as part of this research.

7.2 Conclusions Against Objectives

The aims and objectives of this thesis were listed in Section 1.2.1. They will be discussed further in this section to summarised how they were achieved.

1. To investigate data mining techniques and tools available and evaluate their performance for network traffic data analysis (IPS data in particular). This was achieved by running several experiments using classical classifiers and evaluating their performance. The details of these experiments are presented in Sections 3.2.3 and 3.2.4. The main finding of this objective supports the key aim of this thesis which is the need for better classification methods for detecting malicious network traffic in IPS data.

2. To present and implement the two proposed classification techniques. Namely, the eigenvector-based classifier and the multi-task learning (MTL) classification method. To achieve this a detailed explanation of these two classification techniques was presented in Chapter 4. The chapter started by explaining the MTL method then moved to the method based on the eigenvector. The key finding here is that these two methods are easy to implement and can outperform existing methods. This is illustrated in Chapter 6 where an extensive evaluation is performed.

3. To investigate and evaluate existing the classical PCA algorithm and use it to perform classification and visualisation of network traffic data. How the PCA algorithm works and how it can be improved for visualisation and classification of computer network traffic data were explained in Chapter 5. The main finding after carrying out this work is that the PCA algorithm can be enhanced to better separate difference classes in network traffic data captured by an IPS.

4. To test and evaluate the classification techniques using existing open-source machine learning platforms/libraries such as Weka or Scikit-learn. As Python is currently the main computer programming language in the machine learning field, it was selected to achieve this aim. The Scikit-learn package implements a

collection of evaluation metrics that were used in evaluating the methods developed as part of this thesis (evaluation details are in Chapter 6). An important point in the context of this research is selecting the right metric. This is elaborated on more in the next point.

5. To establish criteria for measuring system accuracy as a key step in the evaluation (choosing the correct evaluation metric is highly important) and then to conduct a comparison between the developed methods based on that. To achieve this a detailed explanation of several classification evaluation metrics and their advantages and disadvantages was provided in Section 3.1. A key finding here is that not all metrics all useful under all circumstances. For example, using only accuracy may not be the correct procedure for imbalanced data. Therefore, other metrics such as precision, recall and the F measure should be used. The evaluation explained throughout Chapter 6 is based on this explanation.

7.3 Conclusions and Lessons Learnt

A part of this research thesis introduced a novel classification method that is based on multi-task learning and shows its effectiveness in accurately classifying computer network traffic especially when training data is scarce. The method is inspired by an existing regression method that suffers from drawbacks as was explained in previous sections. This proposed MTL method is useful not only because it results in improved performance, but also because it is easy to understand, implement and extend.

The performance of several classical classifiers was evaluated and it was concluded that RandomForest is the winner (this is consistent with previous existing research) and therefore it was selected for comparison.

In order to run a fair evaluation, and because four separate datasets were obtained after splitting a large dataset into smaller sub-datasets based on the network traffic type (each one representing a task), several datasets were formed by using pairwise combinations (each two different tasks against each other). After that the resulting datasets were used as a basis for binary classification.

The method proposed in this thesis was evaluated against RandomForest (while using the same data) and the results demonstrated significant improvement in performance when using the proposed method as opposed to RandomForest.

In addition, a recent open-source transfer learning approach was used in the evaluation experiments so that it is possible to compare the performance of this MTL method against it (namely this is the successful CB-SBIT approach). Experiments carried out as part of this research reveal that, not only the fact that the MTL method proposed here outperforms CB-SBIT when data is scarce, but also CB-SBIT can result in negative transfer which leads to worse results (when compared with results obtained without applying the transfer learning technique of CB-SBIT).

This thesis showed the details of the development a novel classification algorithm based on eigenvectors of matrices. The algorithm is a classification rather than feature extraction algorithm as it allows classifying new data into a set of established classes (e.g. benign or malicious). The data are arranged in matrices that are used to build classification models (i.e. based on matrix eigenvectors). This model is used to classify new upcoming data.

The suggested classification algorithms resulted after significant research but became particularly clear after inspecting PCA graphs. The change of the PCA covariance matrix was the starting point for the relevance of using matrices to represent these data

and consequently performing matrix operations to understand the data. The connection to the domain has therefore emerged from PCA.

Three classification algorithms (determinant, eigenvalue, eigenvector) can be found in chapters in this thesis. After several evaluation experiments as shown in the thesis, the eigenvector classifier should be preferred when data is scarce (only small amount of data is available).

This work included a simple but effective variation to the well-known PCA algorithm. The project initially aimed to study existing classification methods and come up with a new or enhanced method for classifying IPS data. This aim is particularly concerned with providing network experts an effective way for deciding on the nature of data traffic in their network, and consequently amend existing rules and policies based on these data. This is a highly useful technique as it helps in identifying cyber-attacks early which leads to damage prevention or at least minimisation.

In achieving this aim, extensive evaluation experiments of existing classical classifiers were run to ensure the work stands on a solid foundation.

7.4 Limitations and Future Work

It is not surprising that research of the kind carried out in this thesis has limitations. For example, a limitation of this research is the source of data used in experimentation and evaluation. In other words, it would be useful to experiments with more data from different sources. In general, the future work points discussed below can be considered some limitations of the current research.

In the near future one possible extension is to test the proposed MTL approach on data from other domains and experiment with other similarity measures. In addition, it would also be interesting to explore the possibility of finding a method to learn the similarity between tasks instead of computing it.

Another point is that, the proposed MTL method can be viewed as a simplified kernel-based approach in which only the pairwise inner products/cosine distances are used. A possible future work can look at deepening this approach as a full Kernel-based analysis. For example, it might be interesting to look at radial basis function (RBF) kernels and polynomial kernels in a support-vector machine (SVM) context, and do a comparison for different kernel parameters.

This work is more of an engineering project in the sense that it developed algorithms that can be plugged into existing IPS platforms for better data analysis and visualisation. The process here involves human observation which can add more robustness to the entire process.

As a future extension, this work can be used in a *human in the loop* scenario. In more detail, the human expert can use the algorithm to visualise the data using a suitable graphical user-interface and provide their feedback into the system to improve the overall performance.

The PCA method performs dimensionality reduction and allows visualising large amounts of data in a compact, visual manner, which is crucial for network administrators. It was shown that this method could be enhanced for visualising IPS data. This was achieved by replacing the covariance matrix by Spearman matrix, resulting in the data becoming more separable.

References

Abdulla, S. A., Ramadass, S., Altyeb, A. A. and Al-Nassiri, A. (2014), 'Employing machine learning algorithms to detect unknown scanning and email worms', *Int. Arab J. Inf. Technol.* **11**, 140–148.

Ain, A., Bhuyan, M., Bhattacharyya, D. and Kalita, J. (2016), 'Rank correlation for low-rate ddos attack detection: An empirical evaluation', *Intl J. of Network Security* **18 (3)**, 474–480.

Akashdeep, Manzoor, I. and Kumar, N. (2017), 'A feature reduced intrusion detection system using ann classifier', *Expert Systems with Applications* **88**, 249–257.
URL: *https://www.sciencedirect.com/science/article/pii/S0957417417304748*

Aljoufi, Reem and Lasebae, Aboubaker (2021), 'Multi-task learning for intrusion detection and analysis of computer network traffic', *E3S Web Conf.* **229**, 01057.
URL: *https://doi.org/10.1051/e3sconf/202122901057*

Alothman, B. (2018*a*), 'The cb-sbit source code on github'. Accessed 26/05/2022.
URL: *https://github.com/alothman/CB-SBIT*

Alothman, B. (2018*b*), 'Similarity based instance transfer learning for botnet detection', *International Journal of Intelligent Computing Research (IJICR)* **9**, 880—-889.

Alothman, B. (2019*a*), Raw network traffic data preprocessing and preparation for automatic analysis, *in* '2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)', IEEE, pp. 1–5.

Alothman, B. (2019*b*), Robust Botnet Detection Techniques for Mobile and Network Environments, PhD thesis, Faculty of Technology, De Montfort University, De Montfort University, The Gateway, Leicester, UK, LE1 9BH. `https://dora.dmu.ac.uk/handle/2086/18144`.

Alothman, B., Janicke, H. and Yerima, S. Y. (2018), Class balanced similarity-based instance transfer learning for botnet family classification, *in* L. Soldatova, J. Vanschoren, G. Papadopoulos and M. Ceci, eds, 'Discovery Science', Springer International Publishing, Cham, pp. 99–113.

Baxter, J. (1997), 'A bayesian/information theoretic model of learning to learn viamultiple task sampling', *Mach. Learn.* **28**(1), 7–39.
URL: *https://doi.org/10.1023/A:1007327622663*

Beghdad, R. (2008), 'Critical study of neural networks in detecting intrusions', *Computers & Security* **27**, 168–175.

Ben-David, S. and Borbely, R. S. (2008), 'A notion of task relatedness yielding provable multiple-task learning guarantees', *Mach. Learn.* **73**(3), 273–287.
URL: *https://doi.org/10.1007/s10994-007-5043-5*

Bhattacharya, S., S, S. R. K., Maddikunta, P. K. R., Kaluri, R., Singh, S., Gadekallu, T. R., Alazab, M. and Tariq, U. (2020), 'A novel pca-firefly based xgboost classification model for intrusion detection in networks using gpu', *Electronics* **9**(2), 219.
URL: *http://dx.doi.org/10.3390/electronics9020219*

Bradley, A. P. (1997), 'The use of the area under the roc curve in the evaluation of machine learning algorithms', *Pattern Recogn.* **30**(7), 1145–1159.
URL: *https://doi.org/10.1016/S0031-3203(96)00142-2*

Breiman, L. (2001), 'Random forests', *Machine Learning* **45**(1), 5–32.
URL: *http://dx.doi.org/10.1023/A%3A1010933404324*

Brereton, R. G. (2009), *Exploratory Data Analysis*, John Wiley & Sons, Ltd, chapter 3, pp. 47–106.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470746462.ch3*

Brownlee, J. (2019), *Deep Learning with Python*, Deep Learning Mastery.

Brownlee, J. (2020), *Data Preparation for Machine Learning*, Machine Learning Mastery.

Bruce, P., Bruce, A. and Gedeck, P. (2020), *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python*, O'Reilly Media.
URL: *https://books.google.co.uk/books?id=rycTtAEACAAJ*

Canadian Institute for Cybersecurity (2017), 'Intrusion detection evaluation dataset (cic-ids2017)', `https://www.unb.ca/cic/datasets/ids-2017.html`. Accessed 16/11/2020.

Canbek, G., Sagiroglu, S., Temizel, T. T. and Baykal, N. (2017), Binary classification performance measures/metrics: A comprehensive visualized roadmap to gain new insights, *in* '2017 International Conference on Computer Science and Engineering (UBMK)', pp. 821–826.

Caruana, R. (1997), 'Multitask learning', *Machine Learning* **28**(1), 41–75.
URL: *http://dx.doi.org/10.1023/A%3A1007379606734*

Cherfi, A., Nouira, K. and Ferchichi, A. (2018), 'Very fast c4.5 decision tree algorithm', *Applied Artificial Intelligence* **32**, 119 – 137.

Choi, Y., Farnadi, G., Babaki, B. and Van den Broeck, G. (2020), 'Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns', *Proceedings of the AAAI Conference on Artificial Intelligence* **34**(06), 10077–10084.
URL: *https://ojs.aaai.org/index.php/AAAI/article/view/6565*

CICFlowMeter (2018), 'Cicflowmeter'. Accessed 22/05/2022.
URL: *https://github.com/ahlashkari/CICFlowMeter*

Cisco (2019), 'Catalyst switched port analyzer (span)'. Accessed 22/06/2022.
URL: *https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html*

Costa, A., Rolim, R., Ramos, F., Soares, G., Almeida, H. and Perkusich, A. (2015), A collaborative method to reduce the running time and accelerate the k-nearest neighbors search, *in* 'SEKE', pp. 105–109.

Dai, W., Yang, Q., Xue, G.-R. and Yu, Y. (2007), Boosting for transfer learning, *in* 'Proceedings of the 24th International Conference on Machine Learning', ICML '07, Association for Computing Machinery, New York, NY, USA, p. 193–200.
URL: *https://doi.org/10.1145/1273496.1273521*

Dasgupta, D., Akhtar, Z. and Sen, S. (2020), 'Machine learning in cybersecurity: a comprehensive survey', *The Journal of Defense Modeling and Simulation* **0**(0), 1548512920951275.
URL: *https://doi.org/10.1177/1548512920951275*

Deza, M. M. and Deza, E. (2009), *Encyclopedia of Distances*, Springer Berlin Heidelberg.

Di Pietro, R. and Mancini, L. V. (2008), *Intrusion Detection Systems*, 1 edn, Springer Publishing Company, Incorporated.

Draper-Gil, G., Lashkari, A. H., Mamun, M. S. I. and Ghorbani, A. A. (2016), Characterization of encrypted and vpn traffic using time-related features, *in* 'ICISSP'.

Dray, S., Legendre, P. and Peres-Neto, P. R. (2006), 'Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (pcnm)', *Ecological Modelling* **196**(3), 483–493.
URL: *https://www.sciencedirect.com/science/article/pii/S0304380006000925*

Duong, L., Cohn, T., Bird, S. and Cook, P. (2015), Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser, *in* 'Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)', Association for Computational Linguistics, Beijing, China, pp. 845–850.
URL: *https://www.aclweb.org/anthology/P15-2139*

EtherApe (2020), 'Etherape: A graphical network monitor'. Accessed 22/05/2022.
URL: *https://etherape.sourceforge.io/*

Farnaaz, N. and Jabbar, M. (2016), 'Random forest modeling for network intrusion detection system', *Procedia Computer Science* **89**, 213–217. Twelfth International Conference on Communication Networks, ICCN 2016, August 19– 21, 2016, Bangalore, India Twelfth International Conference on Data Mining and Warehousing, ICDMW 2016, August 19-21, 2016, Bangalore, India Twelfth International Conference on Image and Signal Processing, ICISP 2016, August 19-21, 2016, Bangalore, India.
URL: *https://www.sciencedirect.com/science/article/pii/S1877050916311127*

Fatourechi, M., Ward, R. K., Mason, S. G., Huggins, J., Schlögl, A. and Birch, G. E. (2008), Comparison of evaluation metrics in classification applications with imbalanced datasets, *in* '2008 Seventh International Conference on Machine Learning and Applications', pp. 777–782.

Freedman, D., Pisani, R. and Purves, R. (2007), 'Statistics (international student edition)', *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* .

Freet, D. and Agrawal, R. (2017), A virtual machine platform and methodology for network data analysis with ids and security visualization, *in* 'SoutheastCon 2017', pp. 1–8.

Gadekallu, T. R., Rajput, D. S., Reddy, M. P. K., Lakshmanna, K., Bhattacharya, S., Singh, S., Jolfaei, A. and Alazab, M. (2021), 'A novel pca–whale optimization-based deep neural network model for classification of tomato plant diseases using gpu', *Journal of Real-Time Image Processing* **18**(4), 1383–1396.

Gezerlis, A. (2020), *Numerical Methods in Physics with Python*, Cambridge University Press.

Goodall, J. R. (2009), Visualization is better! a comparative evaluation, *in* '2009 6th International Workshop on Visualization for Cyber Security', pp. 57–68.

Griffith, D. A. and Chun, Y. (2019), 'Implementing moran eigenvector spatial filtering for massively large georeferenced datasets', *International Journal of Geographical Information Science* **33**(9), 1703–1717.
URL: *https://doi.org/10.1080/13658816.2019.1593421*

Guo, C. and Berkhahn, F. (2016), 'Entity embeddings of categorical variables', *CoRR* **abs/1604.06737**.
URL: *http://arxiv.org/abs/1604.06737*

Haddadi, F., Runkel, D., Zincir-Heywood, A. and Heywood, M. (2014), 'On botnet behaviour analysis using gp and c4.5', *GECCO 2014 - Companion Publication of the 2014 Genetic and Evolutionary Computation Conference* .

Han, J., Kamber, M. and Pei, J. (2012), *Data mining concepts and techniques, third edition*, Morgan Kaufmann Publishers, Waltham, Mass.
URL: *http://www.amazon.de/Data-Mining-Concepts-Techniques-Management/dp/0123814790/ref=tmm_hrd_title_0?ie=UTF8&qid=1366039033&sr=1-1*

Hossin, M. and Sulaiman, M. (2019), 'A Review on Evaluation Metrics for Data Classification Evaluations', *International Journal of Data Mining & Knowledge Management Process (IJDKP)* **5**(2), 1–11.
URL: *https://doi.org/10.5281/zenodo.3557376*

Hu, Y., You, J. J., Liu, J. N. and He, T. (2018), 'An eigenvector based center selection for fast training scheme of rbfnn', *Information Sciences* **428**, 62–75.
URL: *https://www.sciencedirect.com/science/article/pii/S0020025517309210*

Huang, H., Deng, H., Chen, J., Han, L. and Wang, W. (2018), 'Automatic multi-task learning system for abnormal network traffic detection', *International Journal of Emerging Technologies in Learning (iJET)* **13**(04), 4–20.
URL: *https://online-journals.org/index.php/i-jet/article/view/8466*

Hunter, J. D. (2007), 'Matplotlib: A 2d graphics environment', *Computing in Science & Engineering* **9**(3), 90–95.

Jacob, N. M. and Wanjala, M. Y. (2018), 'A review of intrusion detection systems', *Global journal of computer science and technology* .

Jadidi, Z., Muthukkumarasamy, V., Sithirasenan, E. and Sheikhan, M. (2013), Flow-based anomaly detection using neural network optimized with gsa algorithm, *in* '2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops', pp. 76–81.

Jakub, B. and Branišová, J. (2015), 'Anomaly detection from log files using data mining techniques', *Lecture Notes in Electrical Engineering* **339**, 449–457.

Japkowicz, N. and Shah, M. (2011), *Evaluating Learning Algorithms: A Classification Perspective*, Cambridge University Press.

Jiang, C., Xie, H. and Bai, Z. (2017), Robust and Efficient Computation of Eigenvectors in a Generalized Spectral Method for Constrained Clustering, *in* A. Singh and J. Zhu, eds, 'Proceedings of the 20th International Conference on Artificial Intelligence and Statistics', Vol. 54 of *Proceedings of Machine Learning Research*, PMLR, pp. 757–766.
URL: *https://proceedings.mlr.press/v54/jiang17b.html*

Jolliffe, I. (2011), *Principal Component Analysis*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1094–1096.
URL: *https://doi.org/10.1007/978-3-642-04898-2_455*

Jolliffe, I. T. and Cadima, J. (2016), 'Principal component analysis: a review and recent developments', *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **374**(2065), 20150202.
URL: *https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2015.0202*

Koklu, M. and Ozkan, I. A. (2020), 'Multiclass classification of dry beans using computer vision and machine learning techniques', *Comput. Electron. Agric.* **174**, 105507.

Kubat, M. (2015), *An Introduction to Machine Learning*, 1st edn, Springer Publishing Company, Incorporated.

Kumar, M., Hanumanthappa, M. and Kumar, T. V. S. (2012), Intrusion detection system using decision tree algorithm, *in* '2012 IEEE 14th International Conference on Communication Technology', pp. 629–634.

Liao, R., Zheng, H., Grzybowski, S. and Yang, L. (2013), 'A multiclass svm-based classifier for transformer fault diagnosis using a particle swarm optimizer with time-varying acceleration coefficients', *International Transactions on Electrical Energy Systems* **23**.

Ling, C. X. and Sheng, V. S. (2010), *Class Imbalance Problem*, Springer US, Boston, MA, pp. 171–171.
URL: *https://doi.org/10.1007/978-0-387-30164-8_110*

Liu, X., Sun, Y., Fang, L., Liu, J. and Yu, L. (2014), A survey of network traffic visualization in detecting network security threats., *in* Y. Lu, X. Wu and X. Zhang, eds, 'ISCTCS', Vol. 520 of *Communications in Computer and Information Science*, Springer, pp. 91–98.

Liu, Y., Zhou, Y., Wen, S. and Tang, C. (2014), 'A strategy on selecting performance metrics for classifier evaluation', *Int. J. Mob. Comput. Multimed. Commun.* **6**(4), 20–35.
URL: *https://doi.org/10.4018/IJMCMC.2014100102*

Mackey, D. S., Mackey, N., Mehl, C. and Mehrmann, V. (2005), Vector spaces of linearizations for matrix polynomials, Technical Report No. 464, The University of Manchester, UK.
URL: *http://www.maths.manchester.ac.uk/~higham/narep/narep464.pdf*

Mane, S. and Rao, D. (2021), 'Explaining network intrusion detection system using explainable AI framework', *CoRR* **abs/2103.07110**.
URL: *https://arxiv.org/abs/2103.07110*

Masci, F. (2013), 'an introduction to principal component analysis (pca)', `http://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/PrincipalComponentAnalysis.pdf`.

McKinney, W. et al. (2010), Data structures for statistical computing in python, *in* 'Proceedings of the 9th Python in Science Conference', Vol. 445, Austin, TX, pp. 51–56.

Mirjalili, S. and Lewis, A. (2016), 'The whale optimization algorithm', *Advances in Engineering Software* **95**, 51–67.
URL: *https://www.sciencedirect.com/science/article/pii/S0965997816300163*

Molnar, C. (2022), *Interpretable Machine Learning*, 2 edn, Independently published (on Github).
URL: *https://christophm.github.io/interpretable-ml-book*

Morgan, S. (2020), 'Cybercrime to cost the world $10.5 trillion annually by 2025'.
URL: *https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/*

Mukherjee, S. and Sharma, N. (2012), 'Intrusion detection using naive bayes classifier with feature reduction', *Procedia Technology* **4**, 119–128. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.
URL: *https://www.sciencedirect.com/science/article/pii/S2212017312002964*

Murugan, N. S. and Devi, G. U. (2019), 'Feature extraction using lr-pca hybridization on twitter data and classification accuracy using machine learning algorithms', *Cluster Computing* **22**(6), 13965–13974.
URL: *https://doi.org/10.1007/s10586-018-2158-3*

Nasution, M. Z. F., Sitompul, O. S. and Ramli, M. (2018), 'PCA based feature reduction to improve the accuracy of decision tree c4.5 classification', **978**, 012058.
URL: *https://doi.org/10.1088/1742-6596/978/1/012058*

NetGrok (2009), 'Netgrok: a tool for visualizing computer networks in real-time'. Accessed 22/05/2022.
URL: *https://www.cs.umd.edu/projects/netgrok/*

Patil, P., Wei, Y., Rinaldo, A. and Tibshirani, R. (2021), Uniform consistency of cross-validation estimators for high-dimensional ridge regression, *in* A. Banerjee and K. Fukumizu, eds, 'Proceedings of The 24th International Conference on Artificial Intelligence and Statistics', Vol. 130 of *Proceedings of Machine Learning Research*,

PMLR, pp. 3178–3186.
URL: *https://proceedings.mlr.press/v130/patil21a.html*

Patkowski, K. (2020), 'Recent developments in symmetry-adapted perturbation theory', *WIREs Computational Molecular Science* **10**(3), e1452.
URL: *https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1452*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011), 'Scikit-learn: Machine learning in Python', *Journal of Machine Learning Research* **12**, 2825–2830.

Ramchoun, H., Idrissi, M. A. J., Ghanou, Y. and Ettaouil, M. (2017), Multilayer perceptron: Architecture optimization and training with mixed activation functions, *in* 'Proceedings of the 2nd International Conference on Big Data, Cloud and Applications', BDCA'17, Association for Computing Machinery, New York, NY, USA.
URL: *https://doi.org/10.1145/3090354.3090427*

Rezaei, S. and Liu, X. (2020*a*), Multitask learning for network traffic classification, *in* '2020 29th International Conference on Computer Communications and Networks (ICCCN)', pp. 1–9.

Rezaei, S. and Liu, X. (2020*b*), A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning, *in* 'International Conference on Learning Representations'.
URL: *https://openreview.net/forum?id=BylVcTNtDS*

Sadawi, N., Olier, I., Vanschoren, J., van Rijn, J. N., Besnard, J., Bickerton, R., Grosan, C., Soldatova, L. and King, R. D. (2019), 'Multi-task learning with a natural metric for quantitative structure activity relationship learning', *Journal of Cheminformatics* **11**(1), 68.
URL: *https://doi.org/10.1186/s13321-019-0392-1*

Saito, T. and Rehmsmeier, M. (2015), 'The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets', *PLOS ONE* **10**(3), 1–21.
URL: *https://doi.org/10.1371/journal.pone.0118432*

Santafe, G., Inza, I. n. and Lozano, J. A. (2015), 'Dealing with the evaluation of supervised classification algorithms', *Artif. Intell. Rev.* **44**(4), 467–508.
URL: *http://dx.doi.org/10.1007/s10462-015-9433-y*

Sarigiannidis, P., Karapistoli, E. and Economides, A. A. (2015), Visiot: A threat visualisation tool for iot systems security, *in* '2015 IEEE International Conference on Communication Workshop (ICCW)', pp. 2633–2638.

Sawant, A. (2018), A comparative study of different intrusion prevention systems, *in* '2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)', pp. 1–5.

Shui, C., Abbasi, M., Robitaille, L.-E., Wang, B. and Gagné, C. (2019), A principled approach for learning task similarity in multitask learning, *in* 'Proceedings of the 28th International Joint Conference on Artificial Intelligence', IJCAI'19, AAAI Press, p. 3446–3452.

Silva, S. S., Silva, R. M., Pinto, R. C. and Salles, R. M. (2013), 'Botnets: A survey', *Computer Networks* **57**(2), 378–403. Botnet Activity: Analysis, Detection and Shutdown.
URL: *https://www.sciencedirect.com/science/article/pii/S1389128612003568*

Smys, D. S., Basar, D. A. and Wang, D. H. (2020), Hybrid intrusion detection system for internet of things (iot), *in* 'Journal of ISMAC'.

Snort Documentation (2022), 'Snort documentation'. Accessed 22/06/2022.
URL: *https://www.snort.org/documents*

Soubestre, J., Shapiro, N. M., Seydoux, L., deÂ Rosny, J., Droznin, D. V., Droznina, S. Y., Senyukov, S. L. and Gordeev, E. I. (2018), 'Network-based detection and classification of seismovolcanic tremors: Example from the klyuchevskoy volcanic group in kamchatka', *Journal of Geophysical Research: Solid Earth* **123**(1), 564–582.
URL: *https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2017JB014726*

Spearman, C. (1904), 'The proof and measurement of association between two things', *The American Journal of Psychology* .

Standley, T., Zamir, A., Chen, D., Guibas, L., Malik, J. and Savarese, S. (2020), Which tasks should be learned together in multi-task learning?, *in* H. D. III and A. Singh, eds, 'Proceedings of the 37th International Conference on Machine Learning', Vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 9120–9132.
URL: *https://proceedings.mlr.press/v119/standley20a.html*

Stevanovic, M. and Pedersen, J. M. (2014), An efficient flow-based botnet detection using supervised machine learning, *in* '2014 International Conference on Computing, Networking and Communications (ICNC)', pp. 797–801.

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C. and Liu, C. (2018), 'A survey on deep transfer learning', *ArXiv* **abs/1808.01974**.

Ting, K. M. (2017), *Confusion Matrix*, Springer US, Boston, MA, pp. 260–260.
URL: *https://doi.org/10.1007/978-1-4899-7687-1_50*

Tipping, M. E. and Bishop, C. M. (1999), 'Probabilistic principal component analysis', *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* **61**(3), 611–622.

Torrey, L. and Shavlik, J. (2009), 'Transfer learning', *Handbook of Research on Machine Learning Applications* .

Vandenhende, S., Georgoulis, S., Gansbeke, W. V., Proesmans, M., Dai, D. and Gool, L. V. (2020), 'Multi-task learning for dense prediction tasks: A survey'.

Venkatesh, G. and Anitha, R. (2016), 'Botnet detection via mining of traffic flow characteristics', *Computers & Electrical Engineering* **50**, 91–101.

Wang, Q. (2012), 'Kernel principal component analysis and its applications in face recognition and active shape models', *CoRR* **abs/1207.3538**.

Watkins, M. W. (2018), 'Exploratory factor analysis: A guide to best practice', *Journal of Black Psychology* **44**(3), 219–246.
URL: *https://doi.org/10.1177/0095798418771807*

Wei, L., Luo, W., Weng, J., Zhong, Y., Zhang, X. and Yan, Z. (2017), 'Machine learning-based malicious application detection of android', *IEEE Access* **5**, 25591–25601.

Weiss, K., Khoshgoftaar, T. M. and Wang, D. (2016), 'A survey of transfer learning', *Journal of Big Data* **3**(1), 9.
URL: *https://doi.org/10.1186/s40537-016-0043-6*

Wes McKinney (2010), Data Structures for Statistical Computing in Python, *in* Stéfan van der Walt and Jarrod Millman, eds, 'Proceedings of the 9th Python in Science Conference', pp. 56 – 61.

Winter, P., Hermann, E. and Zeilinger, M. (2011), Inductive intrusion detection in flow-based network data using one-class support vector machines, *in* '2011 4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011 - Proceedings', 2011 4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011 - Proceedings, IEEE. Copyright: Copyright 2011 Elsevier B.V., All rights reserved.; 4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011 ; Conference date: 07-02-2011 Through 10-02-2011.

XGBoost (2021), 'The xgboost library', `https://xgboost.readthedocs.io/en/latest/`. Accessed: 2021-10-20.

Yang, X.-S. (2008), *Nature-Inspired Metaheuristic Algorithms*, Luniver Press.

Yang, Y. and Hospedales, T. M. (2016), 'Trace norm regularised deep multi-task learning', *CoRR* **abs/1606.04038**.
URL: *http://arxiv.org/abs/1606.04038*

Yuan, R., Li, Z., Guan, X. and Xu, L. (2010), 'An svm-based machine learning method for accurate internet traffic classification', *Information Systems Frontiers* **12**, 149–156.

Zhang, Y. and Yang, Q. (2017), 'A survey on multi-task learning', *CoRR* **abs/1707.08114**.
URL: *http://arxiv.org/abs/1707.08114*

Zhao, J., Shetty, S. and Pan, J. W. (2017), 'Feature-based transfer learning for network security', *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)* pp. 17–22.

Zhou1, J., Chen, J. and Ye, J. (2012), 'Multi-task learning: Theory, algorithms, and applications', `https://archive.siam.org/meetings/sdm12/zhou_chen_ye.pdf`.

Zhu, C., Idemudia, C. U. and Feng, W. (2019), 'Improved logistic regression model for diabetes prediction by integrating pca and k-means techniques', *Informatics in Medicine Unlocked* **17**, 100179.
URL: *https://www.sciencedirect.com/science/article/pii/S2352914819300139*

Zhuo, L., Zhang, J., Zhao, Y. and Zhao, S. (2013), 'Compressed domain based pornographic image recognition using multi-cost sensitive decision trees', *Signal Process.* **93**(8), 2126–2139.
URL: *https://doi.org/10.1016/j.sigpro.2012.07.003*