Improvements in Network Intrusion Detection through Nonlinear Dimensionality Reduction

MSc in Computing

Ian Garret Dalsin

d026994h@student.staffs.ac.uk

Staffordshire University

A dissertation submitted in partial fulfilment of the requirements of Staffordshire University for the degree of Master of Science

September 2020

Abstract

Anomaly-based Intrusion Detection Systems (IDS) have several advantages over rule-based IDS when encountering novel network attacks on modern datasets. However, the domain of anomaly detection on large scale data is characterized by high computational overhead, memory consumption, and hardware requirements. This project proposes the application of nonlinear dimensionality reduction, or manifold learning, to improve the performance of classification algorithms. Manifold learning, specifically the techniques of locally linear embedding, Isomap, and LTSA, is applied to the benchmark CSE-CIC-IDS2018 dataset to improve the performance of KNN, Decision Tree, Random Forest and Adaboost classification algorithms. Experimental results show that manifold learning can improve some aspects of classification with minimal negative impact on overall classification accuracy.

Table of Figures

ABBREVIATIONS

Abbreviation	Definition
AWS	Amazon Web Services
CSV	Comma Separated Values
CTF	Capture the Flag
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
FTP	File Transfer Protocol
HLLE	Hessian Locally Linear Embedding
IDS	Intrusion Detection System
IMAP	Internet Message Access Protocol
KNN	k-Nearest Neighbour
LAN	Local Area Network
LLE	Locally Linear Embedding
LTSA	Local Tangent Space Alignment
MLLE	Modified Locally Linear Embedding
PCA	Principal Component Analysis
POP3	Post Office Protocol
SaaS	Software as a Service
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SVM	Support Vector Machine
t-SNE	t-Distributed Stochastic Neighbor Embedding
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VOIP	Voice over Internet Protocol
XSS	Cross Site Scripting

Table of Contents

Acknowledgements

The completion of this project has been the capstone of an incredible journey, and would not have been possible without the help and support of many people along the way. I would like to thank Dr. Ali Sadegh-Zadeh for his support, guidance, and calming influence throughout the entire project. Your expertise and insight allowed me to strengthen my abilities as both a researcher and a writer, and is greatly appreciated. I must also thank Mr. Daniel Campbell for his guidance and support during the review process, and Mr. Christopher Howard for his contributions through the viva voce process.

To my father, Chris, who is currently teaching all of us an incredible lesson in perseverance – your constant support in the face of your own difficult journey served as motivation throughout the most difficult parts of this process. I hope that reaching this finish line has made you proud.

I owe a great debt of support to my family, both during this work and far before. To Lynn, Julie, and Sue, thank you for your support from the very beginning, and for teaching me important lessons much before this particular chapter began.

Lastly, I couldn’t imagine tackling this journey without wonderful and supportive friends in my corner. I’d like to thank, in a pseudo-random order, Ian, Paige, Thomas, Scott, Nathan, Tyler, Kristy and Matt for their moral and emotional support, for serving as a sounding board for my half-developed ideas and as my initial defence panel, and for the occasional loaf of sourdough left outside my front door.

CHAPTER 1 – INTRODUCTION

1.1 Introduction

Information security as a discipline has grown in importance alongside the rapid growth of the Internet and connected technology. The discipline contains many distinct areas of focus, each with distinct techniques and tools. As networks have grown in importance, network attacks have increased drastically as well. The evolution of network attacks has forced a constant development of defensive technologies, including the concept of an intrusion detection system. This project is a contribution to the technical development of intrusion detection systems. It aims to improve the results of a conceptual intrusion detection system using the application of manifold learning, a recently developed mathematical concept for the extraction of features from large-scale data.

1.2 Research Background

As network devices have grown in both number and importance, malicious actors have grown in number and sophistication. Several strategies and frameworks exist to prevent, detect and respond to network attacks. Network owners may achieve “defence in depth” through a combination of tools layered throughout an enterprise. Each tool maintains coverage over a specific network niche, such as the use of web application firewalls for specific machines facing the public internet, firewalls and access control lists providing network segmentation, or intrusion detection systems on both the host and network level.

This dissertation focuses on the use of network level intrusion detection systems (IDS). These tools are located at a point in a network or sub-network where it is possible to maintain visibility over all traffic passing between all machines. Intrusion detection systems are commonly divided into rule-based or anomaly-based systems. Rule-based IDS are dependent on known heuristics in the form of rules or signatures. Signatures are maintained by a central authority and periodically updated with individual clients similar in method to a standard antivirus system. If an attacker is able to use a novel attack against a rule-based IDS, they may be able to avoid all known signatures and evade the intrusion detection system entirely. Rule-based systems attempt to mitigate this shortcoming through combinations of distributed sensors, threat intelligence, and active research in order to minimize the time between the discovery of an attack and the deployment of a signature to distributed clients.

Recent developments in network traffic have reduced the ability of rule-based IDS to maintain visibility into a network. Over the past four years, the proportion of network traffic encrypted by

TLS or SSL has increased by from 25 percent to approximately 85 percent [CITATION Let20 \l 1033]. The impending rollout of DNS-over-HTTPS will continue this erosion of visibility by encrypting Domain Name Service traffic within HTTPS traffic [CITATION Clo191 \l 1033] and removing one of the few remaining avenues for the analysis of plaintext network traffic. While visibility in traffic continues to decline, overall traffic volumes continue to rise. [CITATION Cis20 \l 1033] An increase in the use of software-as-a-service (SaaS) applications such as Office 365 and Amazon Web Services (AWS) has increased the amount of traffic crossing the boundaries of a standard corporate network by moving previously internal core business applications to external providers.

These factors have collectively increased interest in anomaly-based intrusion detection systems as an alternative to rule-based systems. An anomaly-based system uses statistical approaches to classify traffic based on historical characteristics, rather than explicitly defined rules and signatures. Anomaly based tools attempt to develop a fingerprint of known traffic and classify outliers against that known background. Fingerprints are developed based on features such as standard application use, user behaviour [CITATION Dar19 \l 1033] and aggregated metadata of traffic flows. [CITATION Gig20 \l 1033] Anomaly-based approaches utilize complex machine learning algorithms to classify traffic based on these statistical approaches. The computational intensity and performance times of these approaches can increase rapidly as network traffic increases, and improvement of that performance is an ongoing area of academic research.

1.3 Research Question

The conversion of network traffic from a network format of packets and frames to objects suitable for analysis by intrusion detection systems provides opportunities for the improvement of the efficiency of the system overall. The use of dimensionality reduction techniques can reduce the size and complexity of network data while maintaining predictive characteristics that allow for effective classification of malicious traffic. Previous literature has predominantly focused on the use of linear dimensionality reduction techniques. This dissertation aims to use nonlinear dimensionality reduction techniques, otherwise known as manifold learning techniques, to improve the performance of a conceptual intrusion detection system.

1.4 Aim

The aim of this dissertation is to research the current state of intrusion detection systems and their performance, to analyze a realistic network traffic dataset, to design a conceptual intrusion detection framework, to evaluate manifold learning approaches, and to test the performance of those approaches alongside understood classification methods.

1.5 Objectives

In order to achieve the aim of the project, the following objectives are identified:

- Review existing literature regarding network attacks, IDS frameworks, linear and nonlinear dimensionality reduction techniques, and available datasets.
- Analyze an existing network dataset and identify possible improvements using nonlinear dimensionality reduction approaches
- To design a conceptual IDS framework suitable for testing manifold learning approaches
- To evaluate available manifold and classification algorithms
- To test and evaluate the performance both manifold and classification approaches

1.6 Deliverables

The objectives listed above each correspond to a deliverable contained within this dissertation.

- The results of a literature review, providing the current state of IDS development, dimensionality reduction techniques, and network datasets
- An analysis of network traffic as understood by intrusion detection systems, including network attacks
- A conceptual design for the implementation of several approaches to intrusion detection
- An evaluation of available algorithms, their suitability, and their expected performance
- A test platform to evaluate the performance of various methods over a network dataset
- An evaluation of the results of the test platform, and a summary of improvements in performance

1.7 Intellectual/Academic Challenge

The Italian semiotician and essayist Umberto Eco framed the development as a dissertation as "…a challenge. You are the challenger. At the beginning, you posed a question which you did not yet known how to answer. The challenge is to find the solution in a finite number of moves…" [CITATION Eco15 \l 1033]

This dissertation poses a multifaceted challenge at the intersection of several research domains. The project exists at the intersection of network security, software architecture, and software development. It requires a depth of knowledge surpassing previous coursework in these areas, alongside an understanding of the application of complex math and statistics in an area where existing literature provides limited guidance. It will require the development of a conceptual artefact using libraries and tools previously unknown to the author, as well as the development a research programme and test architecture suitable to evaluate the artefact.

1.8 Ethical Issues/Framework

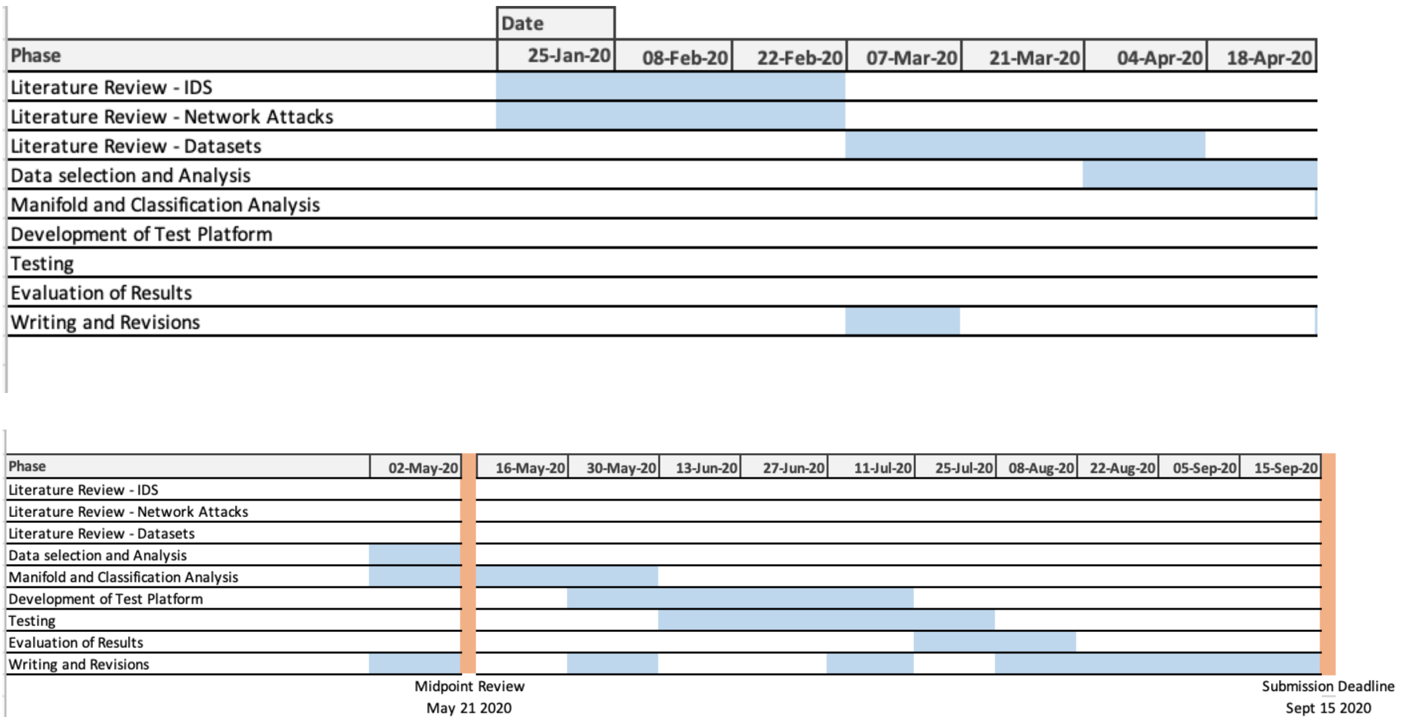
The Staffordshire University research ethical review policy outlines ethical considerations for any research projected conducted under the auspices of the University. The ethical implications of this dissertation are limited by the use of publicly available, machine generated datasets in our analysis. All data used herein is available in the public domain, involves no human participants and contains no personally identifiable information.

In addition to ethical considerations, any research within the area of information security must consider moral and legal considerations surrounding the use of any artefact or proof-of-concept produced. Sanders (2020) summarizes an ongoing debate regarding the release of tools that can be used in an offensive capacity and whether "…the free, unregulated release of offensive security tools [makes] the world inherently more secure or insecure." The artefact produced as a result of this dissertation does not contain any offensive capabilities, and does not provide a pathway for malicious actors to adjust or obfuscate their attacks to circumvent detection.

An ethical disclaimer for this project can be found in appendix A.

1.9 Plan of Work

The plan of work shown in figure 1 below outlines the expected progression of the dissertation objectives over the available period. Each block denotes approximately two weeks of part time work. Each task identified in the plan of work represents an objective identified in section 1.5.



Phase	25-Jan-20	08-Feb-20	22-Feb-20	07-Mar-20	21-Mar-20	04-Apr-20	18-Apr-20
Literature Review - IDS	🟦	🟦	🟦				
Literature Review - Network Attacks	🟦	🟦	🟦				
Literature Review - Datasets				🟦	🟦	🟦	
Data selection and Analysis						🟦	🟦
Manifold and Classification Analysis							
Development of Test Platform							
Testing							
Evaluation of Results							
Writing and Revisions				🟦			

Phase	02-May-20	16-May-20	30-May-20	13-Jun-20	27-Jun-20	11-Jul-20	25-Jul-20	08-Aug-20	22-Aug-20	05-Sep-20	15-Sep-20
Literature Review - IDS											
Literature Review - Network Attacks											
Literature Review - Datasets											
Data selection and Analysis	🟦										
Manifold and Classification Analysis	🟦	🟦	🟦								
Development of Test Platform			🟦	🟦	🟦	🟦					
Testing				🟦	🟦	🟦	🟦				
Evaluation of Results							🟦	🟦			
Writing and Revisions	🟦		🟦			🟦		🟦	🟦	🟦	🟦

Figure 1.1: Plan of Work

1.10 Resources

In order to complete the dissertation, we require access to several resources. The tools and resources listed directly impact the identified deliverables.

- Access to academic journals, articles and literature via Google Scholar and the online libraries of both Staffordshire University and the University of Alberta
- Virtualized platforms to serve as isolated testing environments (VirtualBox virtual machines consisting of installations of Xubuntu version 20.04)
- Statistical and machine learning software libraries: Python 3.8.4, Weka, Pandas, NumPy, MatPlotLib, scikit-learn
- CSE-CIC-IDS2018 intrusion detection dataset, as provided by the University of New Brunswick, Canadian Institute for Cybersecurity and Canadian Communications Security Establishment [CITATION UNB18 \l 1033]

1.11 Research Programme

The research programme underpinning this project is largely informed by the nature of the project and the academic challenge facing the author. The project has been structured approximately according to the structure of the “research onion” devised by Saunders, et al. (2007). Many of the components of the research programme are determined by the nature of the project. Several others are exploratory in nature and will follow the academic development identified in section 1.7.

The project, by nature, follows a positivist approach towards experimentation. The chosen dataset provides a ground truth that will allow the project to determine success or failure, which in turn allows for a deductive approach based on the hypothesis that dimensionality reduction may provide improvement to standard classification algorithms.

The overall strategy of the project consists of an exploratory approach towards the creation of an experimental platform and strategy. Entering the project, the author has limited knowledge of the areas of network traffic analysis and intrusion detection at the level of detail required to properly devise an experimental design. Without the knowledge required for prestructured research, as defined by Punch (2016), the literature review will assist in the development of an experimental framework as it unfolds with the structure of the project “[emerging] in the execution stage.” [CITATION Pun16 \l 1033] It is expected that the development of the test platform will follow an iterative approach with multiple revisions.

The exploratory and iterative nature of this programme fit a specific notetaking strategy to which the author owes a substantial debt. The Zettelkasten (“slip box”) method has existed in various forms since the 18th century[CITATION Chr131 \l 1033] with significant use in social science research. The particular connective nature of this method is well-suited to exploratory research and served as an incredibly useful structure for the development of the project as a whole.

1.12 Structure of Dissertation

Continuing the exploratory framework identified above, the structure of this dissertation leads towards a robust testing and evaluation framework as outlined in chapters 5 and 6. The literature review in chapter 2 explores frameworks for intrusion detection systems, the nature of network attacks, dimensionality reduction techniques, and available datasets that may be suitable for our evaluation. Chapter 3 consists of a preliminary analysis of the chosen dataset and further specificity within our research problem. Chapter 4 details the design of a test platform as a software artefact, and the considerations given to designing a platform for iterative experimentation. Chapter 5 outlines the implementation of chosen manifold learning algorithms and classifiers, while chapter 6 consists of the evaluation of those chosen methods and analysis of the experimental results. Finally, chapter 7 concludes the dissertation with an evaluation of the project as a whole and possibilities for future research.

1.13 Chapter Summary

The previous sections present a preliminary overview of the structure, objectives, and aim of our project. It identifies an exploratory research programme in the areas of network intrusion, dimensionality reduction, and traffic classification. Specifically, our research programme outlines areas where exploratory research will assist in the development of a robust experimental design across areas that present a substantial academic challenge.

CHAPTER 2 – LITERATURE REVIEW

2.1 Introduction

The previous chapter outlined the boundaries of an exploratory literature review in order to inform our experimental design. This chapter serves as an exploration of three distinct domains in order to select a dataset for analysis, design a conceptual IDS, implement that system and test the performance of the system according to known benchmarks. This chapter will provide a clear understanding of the various domains and their contributions to the overall design of the project.

2.2 Domains

2.2.1 IDS, Frameworks and Evaluation Criteria

The most common intrusion detection systems deployed today are often rule-based, such as Snort and Suricata. The shortcomings identified in section 1.2 have led to increased interest in anomaly-based IDS. Several products featuring anomaly-based machine learning approaches have been developed for commercial customers such as DarkTrace's Enterprise Immune System, FireEye's Helix IDS, or Palo Alto Networks' Cortex IDS. These commercially available systems are closed source and protect their specific machine learning approaches as trade secrets. Vendors tout the ability of these products to effectively handle novel attacks [CITATION Dar19 \l 1033] in environments with limited traffic visibility.

Where rule-based systems benefit from clear and explicit definitions of individual attacks, the definition of a network anomaly is vaguer. A commonly accepted definition provided by Hawkins (1980) defines an anomaly as '…an observation which deviates so much from other observations as to arouse suspicious that it was generated by a different mechanism." A focus on generative mechanism is appropriate for the analysis of network traffic. Network traffic represents an outcome from user interaction with a software or hardware interface. Traffic is produced as a by-product of the actions of the user as interpreted through the interface. Unique characteristics of traffic at a network level of abstraction are determined by the interface, not the user. Any two users interacting with a web server through a browser will generate a traffic pattern roughly analogous, regardless of their unique behaviour. However, a traffic pattern generated by a port scanner against the same web server will contain characteristics that can identify it as an outlier when compared to benign traffic. The defining characteristics of a traffic pattern are reflective of the generative mechanism.

Rule-based systems provide an ability to granularly define attack signatures at levels starting from individual packets [CITATION Sno19 \l 1033] and rising to distributed attacks. In order to classify anomalies at similar levels of abstraction, Ahmed, et al. (2016) propose a three-category anomaly classification system that includes point anomalies (a particular instance deviating from the norm), contextual anomalies (an instance that only appears anomalous within a particular context) and collective anomalies (a collection of similar instances that appear anomalous with respect to the dataset.)

The domain of anomaly detection predates the domain of network security. Conceptual frameworks for anomaly detection have been applied to problems across many different fields. All anomaly detection processes consist of similar steps. Detection architectures proposed by Garrett et al. (2003) for classification of EEG signals, by Yang and Zhou (2011) for the analysis of vehicle traffic patterns, and by Subba, Biswas and Karmakar (2016) for network traffic represent a similar process. Unprocessed data is first converted into an object representation suitable for computer processing. Object representations can be described mathematically and are often high dimensional in nature. Each dimension is a feature constructed from the unprocessed data, such as the duration of a signal, the weight of a vehicle, or the network protocol used by traffic. Key features are extracted from these objects through a form of feature extraction or selection in order to reduce dimensionality. A subset of the data is extracted to train a classifier, which is evaluated. The classifier is further trained If the results of that evaluation are unsatisfactory.

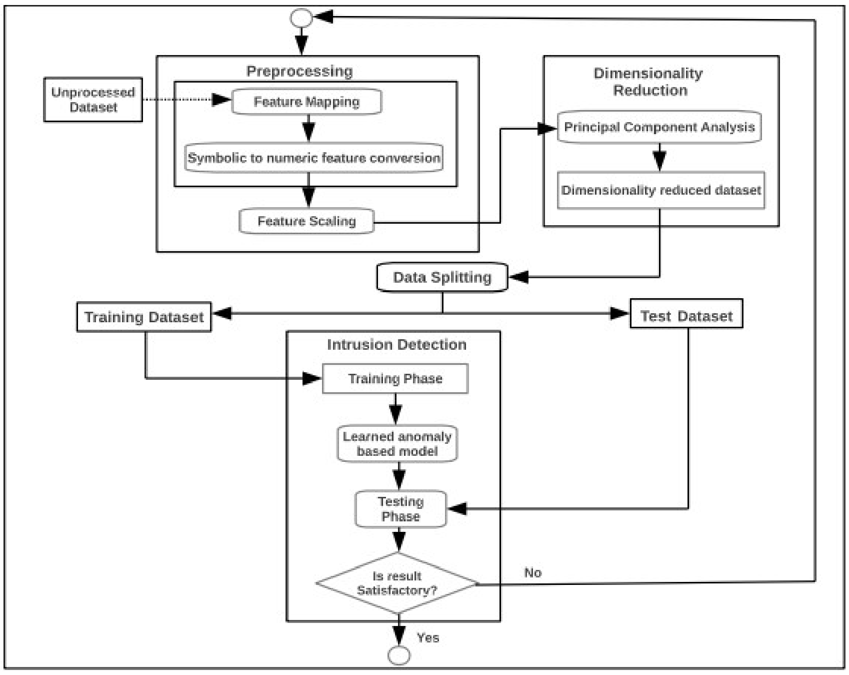


Figure 2.1: Conceptual IDS Framework (Subba, Biswas and Karmakar 2016)

The use of a standardized framework allows us to identify particular areas for improvement. The conversion of unprocessed network traffic captures (pcaps) to network flow objects is achieved through network traffic analysis. A network flow represents a series of packets between two machines as a singular object with defined characteristics, whereas a packet capture displays the flow as a sequence of packets interspersed with packets from other conversations. A flow object provides a level of abstraction and separation that allows for independent examination of specific network conversations between two hosts.

Feature selection and feature extraction can be defined in contrast to each other. Feature selection involves the selection of a subset of available features, while feature extraction derives new, underlying features from the available data. Feature selection can be accomplished through simple statistical tests such as χ2 tests [CITATION Ahm19 \l 1033] or f-tests for explanatory variables. Feature extraction differs through the use of algorithms to extract data representing underlying characteristics of an object from input data that does not explicitly contain those features. Devijver and Kittler (1982) define feature extraction as “extraction from the raw data the information which is most relevant for classification purposes.” Extraction differs from selection in that the most relevant data is not presented as a distinct dimension, but must be extracted through combination, scaling, removal, and transformation of available dimensions to produce a new feature set representing the underlying nature of the object. Both feature extraction and feature selection are forms of dimensionality reduction. Both processes produce an output of a reduced dataset. The proper application of this dimensionality reduction will remove extraneous features and “noise” from the dataset.

Once data has been processed and features selected or extracted, a classification algorithm is trained classify objects according to the needs of the system. Detailed analysis of available classification methods is beyond the scope of this paper. Ahmed, et al.(2016) provide a survey of existing anomaly detection techniques including classification, clustering, rule-based, and neural network techniques. Banerjee et al.(2007) provide a wider survey of unsupervised and supervised classification techniques for anomaly detection, including fault detection in industrial systems and mathematical analysis of fraudulent transactions.

2.2.2 Linear and Non-Linear Dimensionality Reduction

Feature extraction, as defined in the previous section, is a form of dimensionality reduction. It is best suited for classification problems involving high-dimensional data, such as image recognition or biostatistics. Classification of high dimensional data is susceptible to a problem first identified by Bellman (1961) as the “curse of dimensionality.” As the dimensionality of a dataset increases, the computational intensity of some classification algorithms increases exponentially. The k-nearest neighbour algorithm is a common classification algorithm that calculates a Euclidean distance between objects to determine the classification of a test object. As the number of dimensions increases, Euclidean distance approaches a limit of 1 and renders distance measurements meaningless. [CITATION Agg01 \l 1033]

Extensive existing literature demonstrates improvements in performance and classification using linear dimensionality reduction techniques. Zhao, et al.(2017) propose an intrusion detection system using principal component analysis (PCA) and a k-nearest neighbor classifier. The authors identify limitations of standard IDS on Internet of Things devices, often simple devices with minimal specifications, and propose PCA as a dimensionality reduction technique to reduce the complexity and memory usage of a classifier in an environment with hardware limitations. Similar approaches using PCA and KNN classification proposed by Shyu et al. (2003) show some improvement in performance with a cost of slightly reduced classification accuracy. Subba, Biswas and Karmakar (2016) utilize PCA to reduce the dimensionality of data before experimenting with multiple classifiers including support vector machines (SVM), multilayer perceptron (MLP), decision trees, and a Naïve Bayes classifier. They demonstrate that dimensionality reduction can remove features from a dataset without significantly affecting the underlying variance of the data or significantly decreasing the detection rate of the various classifiers. Xiao, et al. (2019) use PCA and a non-linear autoencoder to reduce a network dataset to two dimensions in order to use a convolutional neural network as a classifier. The use of a neural network with many weighted layers allows for dimensionality reduction far beyond other attempts, but relies on a classification process that is much more computationally expensive than simpler classifiers. Gnanaprasanambikai and Munusamy (2018) propose a framework using a similar PCA feature extraction process before classification with a genetic algorithm, which allows a constant evolution of the classification model.

The majority of existing literature in this area focuses on linear dimensionality reduction. Linear dimensionality reduction is effective when the components of an object can be mapped to from the original, high dimension space using linear functions. Linear systems are defined by a linear relationship between a variable and an outcome. Nonlinear systems are defined in opposition to linear systems. Any system that cannot be proven as linear must be considered nonlinear. Mohapatra (1980) expands this definition to include any system with output dependent on state variables. Boyar and Peralta (2013) provide a simpler definition "…that the tools of algebra – and in particular linear algebra –be somehow not applicable to the problem of saying something about *x* given *f(x)*." State variables in a network system include the availability of devices and bandwidth, which influence the nature of network traffic in a nonlinear fashion within a complex system.

Nonlinear dimensionality reduction, also known as manifold learning, is a similar set of techniques for dimensionality reduction and feature extraction in data produced by nonlinear systems. Manifold learning is based on the concept of the "manifold hypothesis." The manifold hypothesis states that high dimensional data is often produced as a representation of a low dimensional manifold [CITATION Fef16 \l 1033], or nonlinear topology. Manifold learning algorithms attempt to preserve both local and global features [CITATION Yan91 \l 1033] while extracting an appropriate feature set to minimize reconstruction error when mapping a high dimensional dataset to a lower dimensional dataset. Further analysis of specific implementations of manifold learning is available in section 4.2.

Olson, Judd and Nichols (2018) propose the use of unsupervised manifold learning for anomaly detection in an image recognition system. Unsupervised manifold algorithms can be suitable for environments with an absence of training data, but suffer a performance penalty that supervised techniques do not. DarkTrace's Enterprise Immune System IDS is a commercial system using multiple manifold models to provide estimates of probability for traffic anomalies [CITATION Dar19 \l 1033], with a focus on clustering data by probability rather than explicit classification. Samani et al. (2019), Wei et al. (2019), and Abdulhammed et al. (2019) propose anomaly detection and classification based on the use of autoencoders, a family of neural networks that attempt to encode features using hidden layers into a low dimensional space.

2.2.3 Available Network Datasets

Several established datasets are publicly available for network traffic analysis research. Network attacks are constantly evolving, and datasets must be kept current with modern tactics, techniques and procedures. Shi, et al. (2019) explore the CTU-13, ISCX2012 and CSE-CIC-IDS2018 datasets as well as the differences between general network traffic datasets and datasets generated for specific subsets of traffic such as industrial control systems. Sharafaldin, et al. (2018) outline several popular datasets, namely DARPA-99, KDD-99, DEFCON (2002), CAIDA (2016), LBNL (2005), CDX (2009) and ISCX2012. Ahmed (2016) identifies several limitations of the KDD and DARPA datasets. Both DARPA and KDD use the Solaris operating system to generate attack profiles, resulting in a unique fingerprint based on Solaris' network traffic characteristics. Additionally, an imbalance exists in the proportion of attack and benign traffic. Further analysis by Mahoney and Chan (2003) raises concerns about the continued use of the KDD and DARPA datasets in intrusion detection research, and suggests possible mitigation efforts for the errors and imbalances discovered. A similar imbalance exists in the DEFCON (2002) dataset, which is generated from the network traffic produced during DEF CON Capture the Flag (CTF) competitions. CTF traffic is predominately malicious in nature and does not reflect an appropriate balance of benign and malicious traffic that would be found in a real-world network capture. The LBNL and CDX datasets suffer from substantial data loss during the anonymization process [CITATION Sha18 \l 1033], and do not contain a wide range of diverse traffic.

The CSE-CIC-IDS2018 dataset [CITATION Uni18 \l 1033] is generated with consideration to the key characteristics of attack diversity, available protocols, completeness, metadata/labelling, and modernity [CITATION Sha18 \l 1033] among others. The dataset is generated using profile agents for benign traffic, which generate traffic to match a statistical distribution and pattern of human interactions over an assortment of protocols. Benign traffic is modelled on observed distributions of packet size, packet flow, payload patterns and request time variance to accurately reflect a wide range of user behaviour. The use of statistical profiles eliminates any ethical or data protection issues that would otherwise arise from sampling traffic created by human users. The generation environment uses modern operating systems (Windows Sever 2016, Ubuntu 16.4, Windows 10, Windows 8.1, Kali Linux) and common, current, open-source attack platforms (Metasploit, Patator, Damn Vulnerable Web App (DVWA), Ares) to generate attack traffic accurately reflecting modern threats and techniques. Attacks are structured on a type-per-day basis, ensuring that there is an appropriate sample size and class balance for malicious and benign traffic for each day of the network capture.

2.3 Findings

This literature review posed several exploratory questions to improve our understanding of the overall problem area. We have outlined the components of an IDS and a specific area of improvement with respect to dimensionality reduction through feature extraction. Section 2.2.2 established previous work in the area of linear dimensionality reduction, and posited that a nonlinear approach to dimensionality reduction would be suitable for network traffic. Section

2.2.3 explored available network datasets, their shortcomings, and the use of the CSE-CIC-IDS2018 dataset for our evaluation.

Anomaly-based IDS are generally less popular than rule-based IDS. Developments in modern network traffic have raised shortcomings of rule-based IDS specific to the detection of novel attacks, increased traffic volume, and decreased effectiveness of techniques such as deep packet inspection, DNS inspection and HTTP payload inspection. Although commercial anomaly-based IDS products are generally closed source, previous literature defines several frameworks for anomaly detection both within and outside the domain of network security. IDS frameworks delineate between the areas of feature mapping, feature selection, and feature extraction. This project will specifically focus on feature extraction.

Feature extraction techniques can be grouped as either linear or nonlinear. Previous literature has demonstrated the effectiveness of linear dimensionality reduction on network datasets. The application of linear techniques such as PCA has been proven to improve the classification abilities of IDS. Nonlinear dimensionality reduction has been successfully applied to image detection, text recognition, and several types of anomaly detection. There is limited literature applying these techniques to network security classification problems.

2.4 Summary

This chapter has provided a foundational knowledge of the domains of intrusion detection, anomaly detection, and dimensionality reduction. It outlined publicly available network traffic datasets and the limitations of popular datasets, and proposed the use of the CSE-CIC-IDS2018 dataset produced by the University of New Brunswick.

Our review of previous literature raises several questions for further analysis. Chapter 3 contains a preliminary analysis of the dataset and network attacks contained within, continuing from the introduction in section 2.2.3. This analysis will inform the selection and design of an appropriate test methodology and platform in further chapters.

CHAPTER 3 – ANALYSIS OF PROBLEM AND IMPROVEMENT

3.1 Introduction

Our exploratory literature review in the previous chapter outlined the nature of IDS frameworks and introduced both dimensionality reduction and the chosen CSE-CIC-IDS2018 dataset. This chapter provides a detailed analysis of the dataset in order to outline the potential improvement of classification using a nonlinear dimensionality reduction approach.

3.2 Selection of Analysis Method and Techniques

The IDS frameworks outlined in section 2.2.1 narrow the scope of our problem area to the specific investigation of the feature space of a high-dimensional network dataset. The use of a pre-existing dataset with network flows extracted from the provided packet captures allows our project to focus on the areas of dimensionality reduction and classification rather than data processing and feature mapping. Sharafaldin et al. (2017) outline the generation of the dataset using statistical profiles and established attack platforms. The combined generation methodology generates a wide variance of network flow objects for analysis.

An exploratory analysis of this data will assist in understanding the dimensionality of network objects within the problem space. Preliminary analysis will determine subtypes within daily network flows, features available in the dataset, and the distribution of key features. For the purposes of this analysis, we develop a Python script to parse the provided network CSV files and generate several sets of descriptive statistics. An analysis of the data cumulatively, within labelled classes, and within identified subtypes is provided in section 3.4.

3.3 Data Collection

The CSE-CIC-IDS2018 data set represents a model corporate network containing five subnets on a common LAN, as well as an external attack platform. Four of the subnets contain 100 Windows desktops, while the fifth contains 20 Ubuntu workstations representing an IT department and 30 servers representing a range of application, web, and email servers running various operating systems. The network captures are positioned as if the main was mirrored to a capture port in order to provide full visibility over the entirety of the network.

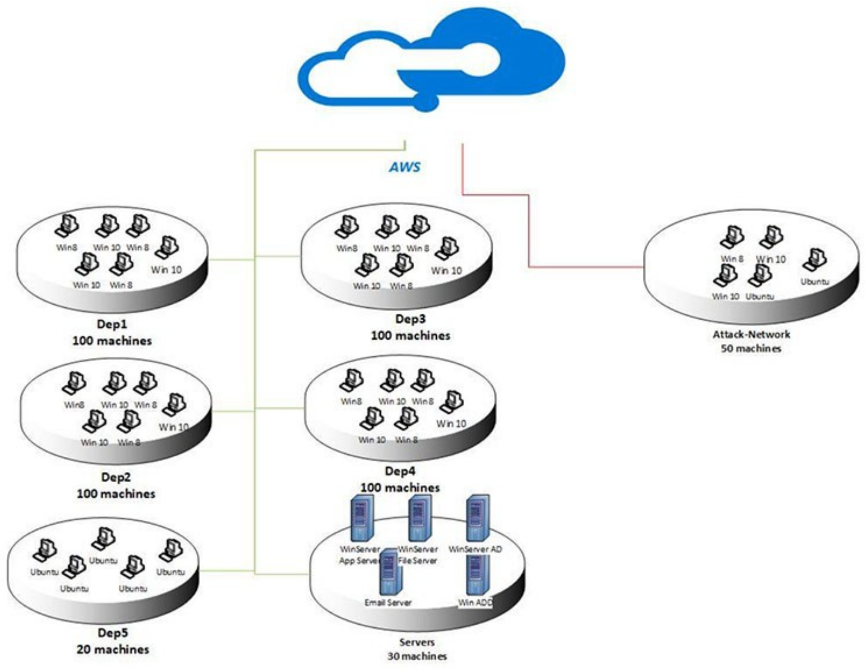


Figure 3.1: Network Topology [CITATION Uni18 \l 1033]

The dataset contains both benign and malicious traffic generated programmatically via user profiles. It uses two types of profiles, B-profiles and M-profiles, to generate benign and malicious traffic respectively. The B-profile system, originally proposed by Sharafaldin et al. (2017) is "designed to extract the abstract behaviour of a group of human users." Profiles are used to simulate HTTPS, HTTP, SMTP, POP3, IMAP, SSH and FTP protocols. [CITATION Uni18 \l 1033]

The dataset contains several filetypes. For the purposes of this project, we will ignore the system logs collected for each host. Packet captures are collected from a network position equivalent to a network tap on the main switch, divided by date of capture, and divided again by machine to produce approximately 450 pcap files per day. The packet captures are processed through a feature extraction tool in order to produce network flow objects, divided into one CSV file per day.

The nature of a network flow object accounts for several of the possible shortfalls in an IDS that were previously identified. In the network capture files provided, HTTP traffic is predominantly unencrypted. As noted previously, the vast majority of modern web traffic is encrypted with TLS1.2 or greater. A network flow object provides a level of abstraction from network traffic captures that removes data that would otherwise be viewed unencrypted, such as the specific GET/POST requests contained in an HTTP request. These objects represent encrypted and unencrypted HTTP traffic equally and allow us to disregard the lack of encryption applied to our dataset.

The network flows in the CSE-CIC-IDS2018 dataset are produced via the traffic analyser CICFlowMeter, developed by Lashkari, et al. (2016). Flowmeter generates bidirectional flows where the directions are determined by the initial packet. The first packet determines forward direction (source to destination) while any response traffic is marked as the backward direction (destination to source.) The nature of TCP and UDP protocols require adjustments to measurement and feature extraction. TCP, as a stateful protocol, ends a connection with a FIN packet during the connection teardown process. UDP, as a stateless protocol, does not have an explicit procedure to end a connection. UDP flows are measured using an adjustable flow timeout to determine when a flow has ended.

Figure 4, below, describes the features included in a network flow object within the data set. In addition to the features noted below, the CSV files provided also contain fields for destination port number, protocol, timestamp, and label. The label field categorizes a flow as either benign or one of the listed attacks and serves as class labels for classifiers. With the exception of 'Label', all fields are numeric. The fields for destination port and protocol are categorical features, while all other features can be considered continuous.

Feature Name	Description
fl_dur	Flow duration, timed in microseconds (1x10^{-6} seconds)
tot_fw_pk	Total packets in the forward direction
tot_bw_pk	Total packets in the backward direction
tot_l_fw_pkt	Total size of packet in forward direction
tot_l_bw_pkt	Total size of packet in backward direction
fw_pkt_l_max	Maximum size of packet in forward direction
fw_pkt_l_min	Minimum size of packet in forward direction
fw_pkt_l_avg	Average size of packet in forward direction
fw_pkt_l_std	Standard deviation size of packet in forward direction
Bw_pkt_l_max	Maximum size of packet in backward direction
Bw_pkt_l_min	Minimum size of packet in backward direction
Bw_pkt_l_avg	Mean size of packet in backward direction
Bw_pkt_l_std	Standard deviation size of packet in backward direction
fl_byt_s	flow byte rate that is number of bytes transferred per second
fl_pkt_s	flow packets rate that is number of packets transferred per second
fl_iat_avg	Average time between two flows
fl_iat_std	Standard deviation time two flows
fl_iat_max	Maximum time between two flows
fl_iat_min	Minimum time between two flows
fw_iat_tot	Total time between two packets sent in the forward direction
fw_iat_avg	Mean time between two packets sent in the forward direction

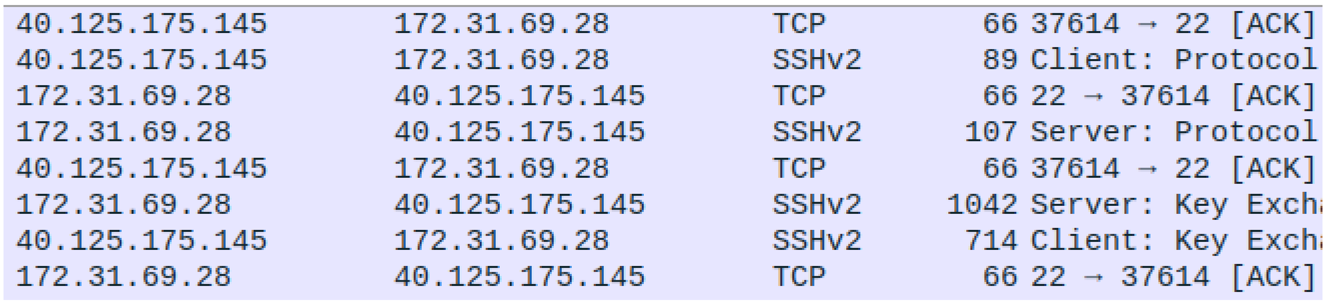
fw_iat_std	Standard deviation time between two packets sent in the forward direction
fw_iat_max	Maximum time between two packets sent in the forward direction
fw_iat_min	Minimum time between two packets sent in the forward direction
bw_iat_tot	Total time between two packets sent in the backward direction
bw_iat_avg	Mean time between two packets sent in the backward direction
bw_iat_std	Standard deviation time between two packets sent in the backward direction
bw_iat_max	Maximum time between two packets sent in the backward direction
bw_iat_min	Minimum time between two packets sent in the backward direction
fw_psh_flag	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
bw_psh_flag	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
fw_urg_flag	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
bw_urg_flag	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
fw_hdr_len	Total bytes used for headers in the forward direction
bw_hdr_len	Total bytes used for headers in the forward direction
fw_pkt_s	Number of forward packets per second
bw_pkt_s	Number of backward packets per second
pkt_len_min	Minimum length of a flow
pkt_len_max	Maximum length of a flow
pkt_len_avg	Mean length of a flow
pkt_len_std	Standard deviation length of a flow
pkt_len_va	Minimum inter-arrival time of packet
fin_cnt	Number of packets with FIN
syn_cnt	Number of packets with SYN
rst_cnt	Number of packets with RST
pst_cnt	Number of packets with PUSH
ack_cnt	Number of packets with ACK
urg_cnt	Number of packets with URG
cwe_cnt	Number of packets with CWE
ece_cnt	Number of packets with ECE
down_up_ratio	Download and upload ratio
pkt_size_avg	Average size of packet
fw_seg_avg	Average size observed in the forward direction
bw_seg_avg	Average size observed in the backward direction
fw_byt_blk_avg	Average number of bytes bulk rate in the forward direction
fw_pkt_blk_avg	Average number of packets bulk rate in the forward direction
fw_blk_rate_avg	Average number of bulk rate in the forward direction
bw_byt_blk_avg	Average number of bytes bulk rate in the backward direction
bw_pkt_blk_avg	Average number of packets bulk rate in the backward direction
bw_blk_rate_avg	Average number of bulk rate in the backward direction
subfl_fw_pk	The average number of packets in a sub flow in the forward direction
subfl_fw_byt	The average number of bytes in a sub flow in the forward direction
subfl_bw_pkt	The average number of packets in a sub flow in the backward direction
subfl_bw_byt	The average number of bytes in a sub flow in the backward direction
fw_win_byt	Number of bytes sent in initial window in the forward direction
bw_win_byt	# of bytes sent in initial window in the backward direction
Fw_act_pkt	# of packets with at least 1 byte of TCP data payload in the forward direction
fw_seg_min	Minimum segment size observed in the forward direction
atv_avg	Mean time a flow was active before becoming idle

atv_std	Standard deviation time a flow was active before becoming idle
atv_max	Maximum time a flow was active before becoming idle
atv_min	Minimum time a flow was active before becoming idle
idl_avg	Mean time a flow was idle before becoming active
idl_std	Standard deviation time a flow was idle before becoming active
idl_max	Maximum time a flow was idle before becoming active
idl_min	Minimum time a flow was idle before becoming active

Table 3.2: List of extracted features

The majority of the features included in the network flow are self-explanatory. It is important to note the unique structure of the destination port feature. The IANA registry of ports is divided into three categories based on port number. *[CITATION Int11 \l 1033]* Ports numbered 0 to 1023 are “system ports” and are uniquely tied to specific protocols. For example, all traffic over port 22 will be Secure Shell traffic, and all traffic over port 80 is HTTP traffic. Ports 1024 to 49151 are “registered ports” and are tied to specific applications. For example, traffic over port 3389 is specific to Microsoft RDP and traffic over port 1433 is specific to SQL Server implementations. Finally, ports 49152 to 65535 cannot be registered and are used for private, customized, or temporary purposes. Temporary ports are used on the server end of client/server connections to distinguish unique connections. A client will request a connection to a server using a system port, after which the server will use a temporary port to reply to the client. In the sample packet capture in figure 5 below, the client has contacted the server over port 22 and receives replies to port 37614. Malicious traffic may attempt to spoof a port number in order to circumvent firewalls or network access control measures. *[CITATION Dur19 \l 1033]*

The destination port numbers contained in the dataset can be used to identify the labelled protocols. Protocol ‘6’ contains ports 22, 23, 80, and 443, among others, implying that it represents TCP traffic. Protocol ‘17’ contains ports 53, 67, 500 and 123, among others, implying that it represents UDP traffic.

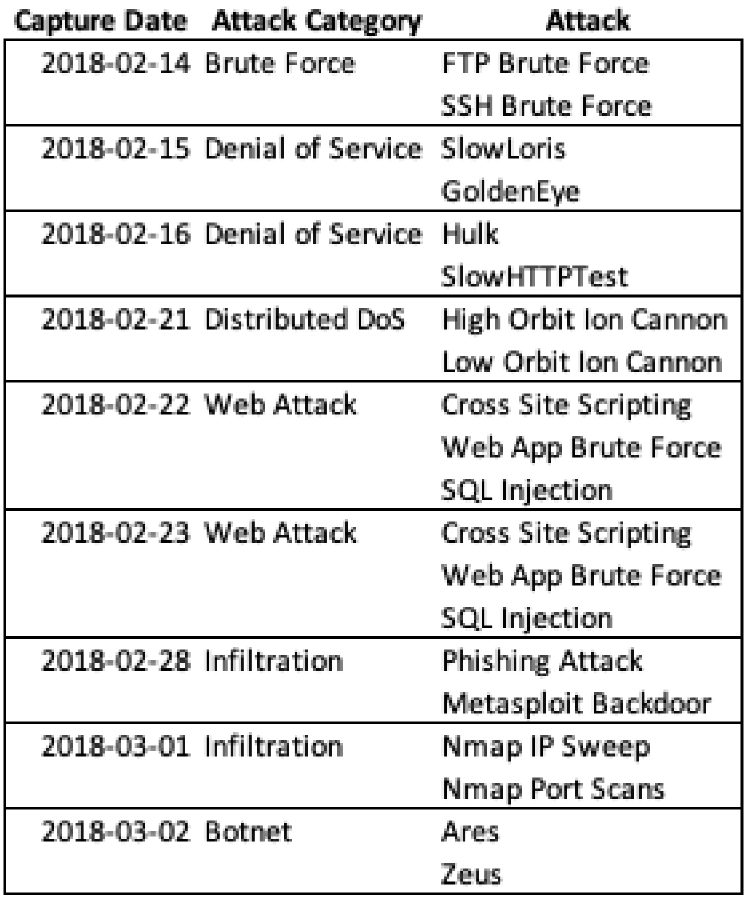


```
40.125.175.145     172.31.69.28       TCP        66 37614 → 22 [ACK]
40.125.175.145     172.31.69.28       SSHv2      89 Client: Protocol
172.31.69.28       40.125.175.145     TCP        66 22 → 37614 [ACK]
172.31.69.28       40.125.175.145     SSHv2     107 Server: Protocol
40.125.175.145     172.31.69.28       TCP        66 37614 → 22 [ACK]
172.31.69.28       40.125.175.145     SSHv2    1042 Server: Key Exch
40.125.175.145     172.31.69.28       SSHv2     714 Client: Key Exch
172.31.69.28       40.125.175.145     TCP        66 22 → 37614 [ACK]
```

Figure 3.3: Partial packet capture of an SSH connection

The features extracted from the network captures contain no information that could be used to identify specific machines on the network. Standard network captures contain IP addresses or device identifiers, which could be used to overfit a classification model on a specific malicious machine if not removed.

The dataset contains a set of fourteen attacks conducted over nine days of traffic captures and labelled in the network flow records, as shown in figure 6 below. It is important to note that the dataset clusters attacks of similar types on the same day. This allows for effective training of classifiers per attack type.



Capture Date	Attack Category	Attack
2018-02-14	Brute Force	FTP Brute Force
		SSH Brute Force
2018-02-15	Denial of Service	SlowLoris
		GoldenEye
2018-02-16	Denial of Service	Hulk
		SlowHTTPTest
2018-02-21	Distributed DoS	High Orbit Ion Cannon
		Low Orbit Ion Cannon
2018-02-22	Web Attack	Cross Site Scripting
		Web App Brute Force
		SQL Injection
2018-02-23	Web Attack	Cross Site Scripting
		Web App Brute Force
		SQL Injection
2018-02-28	Infiltration	Phishing Attack
		Metasploit Backdoor
2018-03-01	Infiltration	Nmap IP Sweep
		Nmap Port Scans
2018-03-02	Botnet	Ares
		Zeus

Table 3.4: List of Attacks

Network attacks generate different traffic flows based on the nature and methodology of the attack. Brute force attacks consist of repeated login attempts over a specific port using a range of possible credentials. The SlowLoris denial of service (DoS) tool attacks a target web server by sending partial HTTP requests and maintaining these connections at regular intervals to consume open sockets on the target server. *[CITATION Kin15 \l 1033]* The GoldenEye DoS tool uses HTTP KeepAlive requests to maintain connections and consume open sockets in a similar fashion. *[CITATION Jan14 \l 1033]* HULK (an acronym for HTTP Unbearable Load King) is a

DoS tool that produces large volumes of obfuscated traffic requests. *[CITATION And201 \l 1033]* The use of constantly shifting HTTP requests bypasses cache servers to directly increase resource use on a web server. SlowHTTPTest is an application layer DoS attack platform that exploits the Slow HTTP Post vulnerability to consume available connections using edited HTTP POST requests. *[CITATION Ser16 \l 1033]* High Orbit Ion Cannon (HOIC) and Low Orbit Ion Cannon (LOIC) are common public domain tools for distributed denial of service attacks. The two tools use similar HTTP and UDP flood methodologies to send malformed requests to a server, which will respond with traffic that is redirected to overwhelm the target device.

The three web attacks launched on February 22 and February 23 use the open source Selenium automation platform as an attack platform. A SQL injection attack consists of HTTP requests that contain SQL queries, which are parsed by an improperly configured web application to return data to an attacker. Cross site scripting (XSS) attacks are similar to SQL injections while using other scripting languages.

The infiltration attack, conducted over a two-day period, consists of a malicious email attachment triggering a Metasploit module to install a backdoor on the victim device. From this victim device, the attacker conducts sweeps of available IP range, port scans, and service probes using Nmap. This is the only attack launched from inside the corporate network. The botnet attack uses both the Zeus and Ares botnets to infect the target machines and request screenshots every 400 seconds.

In the following section, preliminary analysis of the data set will establish the nature of both benign and malicious network flow objects.

3.4 Data Analysis

In order to construct an anomaly-based classifier we must establish the nature of both benign and malicious traffic. The structure of a flow object is determined by a combination of the extracted features. This structure differs substantially across and within different types of traffic. A benign traffic flow may consist of a HTTP browsing session, an SSH connection to a remote device, an FTP file transfer session, a VOIP call, or any other regular user interaction with a network device. Malicious traffic may mimic those objects closely. A SQL injection or XSS attack is simply a set of malicious instructions contained within HTTP requests, while a brute force attack may resemble a series of failed login attempts.

Appendix B contains two Python scripts using the NumPy and MatPlotLib libraries for preliminary analysis of the network flow objects contained in the CSE-CIC-IDS2018 dataset. This script generates basic models of benign and malicious traffic patterns over a low-dimensional analysis of specific key ports. For each subset, it calculates the mean, absolute deviation, standard deviation, variance, and several percentiles for each feature. Subsets are automatically chosen based on attack types and key ports, and statistics calculated per subset.

Several preliminary conclusions can be drawn from this examination. As noted in the previous section, benign traffic is generated through abstract representations of human user profiles. This traffic mimics a wide range of human behaviour and is inherently noisy. Calculating the distribution of values for any specific feature of benign traffic presents high standard deviations and wide distributions, even for a selected subset of traffic over a specific well-known port. Similar distributions for malicious traffic, as shown in figure 7, demonstrate that benign and malicious traffic patterns cannot be consistently separated in a low-dimensional analysis.

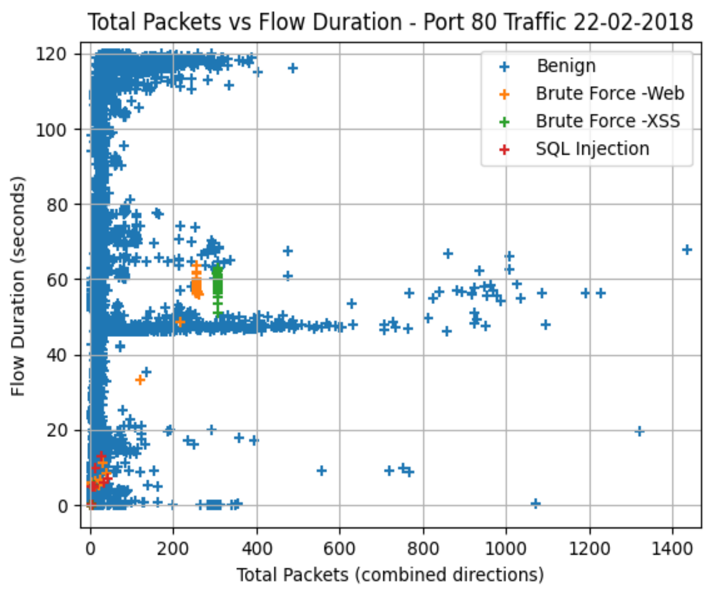


Figure 3.5: Port 80 Web Attack Traffic

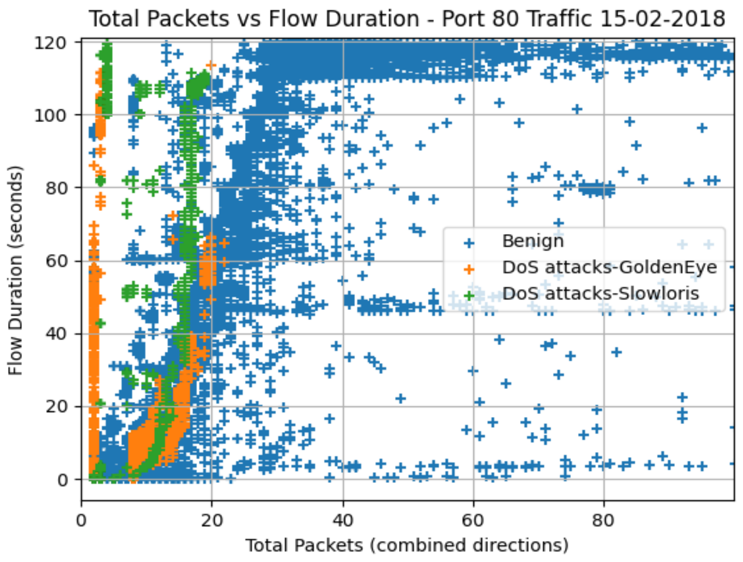


Figure 3.6: Preliminary Analysis of Port 80 Traffic

A preliminary analysis of port 80 traffic in figure 7 shows malicious traffic contained in relative clusters within specific features. Malicious network flows for the network attacks represent the generative mechanism, or attack platform, used to produce them. Network flows generated from a DoS attack are represented as bursts of traffic to a destination port rarely exceeding 20 total packets over the lifetime of the connection. These flows can be reasonably clustered, but share several characteristics with benign traffic. Any attempts at a low-dimensional classification of attack traffic would also classify several types of benign traffic as malicious. Increasing the number of features evaluated increases the classification accuracy rate, but introduces problems of excess dimensionality, computational intensity, and memory usage.

The results of this preliminary analysis of traffic flows provide a basis for dimensionality reduction through feature extraction. The use of manifold learning algorithms to reduce the dimensionality of network flow objects can be framed in several distinct tests, to be implemented through the development of our test platform tool.

To guide the development of the test platform, we must first develop a test plan. The research question outlined in section 1.3 is to determine the impact of nonlinear dimensionality reduction on the performance of classification algorithms. We define performance as a combination of memory usage, computational intensity (time), and classification accuracy.

In order to effectively measure performance, we require a baseline measurement of each classifier without the application of dimensionality reduction. Secondly, we will test each manifold approach to evaluate memory usage, computational intensity, and manifold mapping accuracy. With an understanding of the performance of manifolds and of classifiers, we can then test whether the use of a manifold approach improves upon the baseline classification results.

3.5 Summary

The exploratory analysis provided in this chapter provides an outline of the problem of classification in a high-dimensional space, as well as an analysis of the similarity of malicious traffic flows in a low-dimensional space. Although malicious traffic generated from specific attack platforms and methodologies contains artefacts specific to their generative platform, low dimensional classification does not provide the required granularity to classify these malicious flows as distinct from benign traffic. Chapter 4 details the development of a test platform to evaluate classification in the high-dimensional space, as well as the evaluation of nonlinear dimensionality approaches and their impact on classification. Chapter 5 provides an explanation of chosen algorithms, their implementation, and the limitations of the test platform.

CHAPTER 4 – DESIGN OF IMPROVEMENT

4.1 Introduction

Chapter 3 conducted preliminary analysis of the data, introduced the problem of high-dimensionality, and discussed the impact of these high-dimensional spaces on classification algorithms. Through the preliminary analysis we have developed a specific set of experiments to test the effectiveness of nonlinear dimensionality reduction on classification problems.

In this chapter, we will outline the design of a system to implement this dimensionality reduction, followed by classification. In order to effectively test multiple classifiers and manifold learning techniques, the test platform must be modular, adjustable, and provide the ability to rapidly test and evaluate multiple combinations of dimensionality reduction techniques and classification models.

4.2 Selection of Design Method/Techniques

The framework of the proposed classification tool is based on a modular, object-oriented construction rather than a functional or imperative construction. Given several exploratory combinations of data preprocessing, non-linear dimensionality reduction, and classification functions, the framework aims to allow for rapid iteration between different functions and different tests. The object-oriented design paradigm contains four major elements: abstraction, encapsulation, modularity and hierarchy. [CITATION Bud03 \l 1033] Where functional and imperative programming models are generally procedural and implement actions in a predefined sequence, object-oriented design allows for the partition of a system into individual components with defined interconnection.

It is important to first define the use case for our test platform. The design process is influenced by the experimental questions outlined in chapter 3. We will attempt to evaluate several classification algorithms, several dimensionality reduction approaches, and different preprocessing requirements for each approach. As such, a modular design must consider each step in the experimental process as a distinct module and ensure that the data object passed between each module is consistent and equal. We must also consider the evaluation of each module and implement methods for evaluating the accuracy, computational intensity and memory usage of each step of the experimental process.

In order to effectively and fairly measure the time and memory use of a software artefact, we must control for any other interactions within the operating system that may influence the testing process. For this purpose, our test platform will be built within a virtual machine environment.

This allows for the true isolation of a limited operating system from the underlying hardware, as well as control of hardware specifications. Additionally, a virtual machine platform is scalable during the testing process. Additional copies of a virtual machine can be cloned and created in order to run multiple tests simultaneously while maintaining isolation and independence.

There are several tools available for the implementation of classifiers and manifold learning transformations. Initial exploration of the available software libraries focussed on WEKA, megaman, TensorFlow, MATLAB (specifically the mani and drtoolbox modules) and scikit-learn. Scikit-learn is built as an extension of the SciPy Python library, and is tightly integrated with other scientific Python libraries such as NumPy, pandas, and Matplotlib. Scikit is able to accept and process several common data structures including pandas DataFrame objects, NumPy ndarray objects, and standard Python arrays.

A data flow diagram representing our proposed test platform is shown in figure 8 below. In order to implement a modular design, the data object contained within the platform must remain consistent throughout. The pandas library provides native support for the conversion of CSV files to pandas DataFrames. DataFrames, similar in structure to native Python arrays, provide flexibility in processing through established methods for the conversion of sparse or messy data. Pandas CSV support attempts to interpret data types from the source data, though types can be explicitly declared if required. The user input module will accept input in order to load a file and determine which classification and/or dimensionality reduction algorithms to run. Once data is loaded into a DataFrame, the DataFrame will be passed to a preprocessing module. This module will ensure that data is correctly structured for further analysis. User input will determine if dimensionality reduction is optionally applied to the data before classification. User input will also determine if the performance of the dimensionality reduction or classifier module is measured by the evaluation wrapper, a supervisory function to track memory usage and running time. By placing the evaluation functions in a separate module, we ensure that the experiment does not contain an “observer effect”, and the evaluation functions themselves do not influence the classification or dimensionality reduction functions.

The final output of the platform is twofold. Classification output consists of the predictions made by the classification model for the labelled network flow data. This can be shown on a per-row basis or via other scoring metrics provided by Scikit. If the evaluation wrapper is also triggered, evaluation results are produced to track memory usage and running time of the dimensionality module and/or classification module.

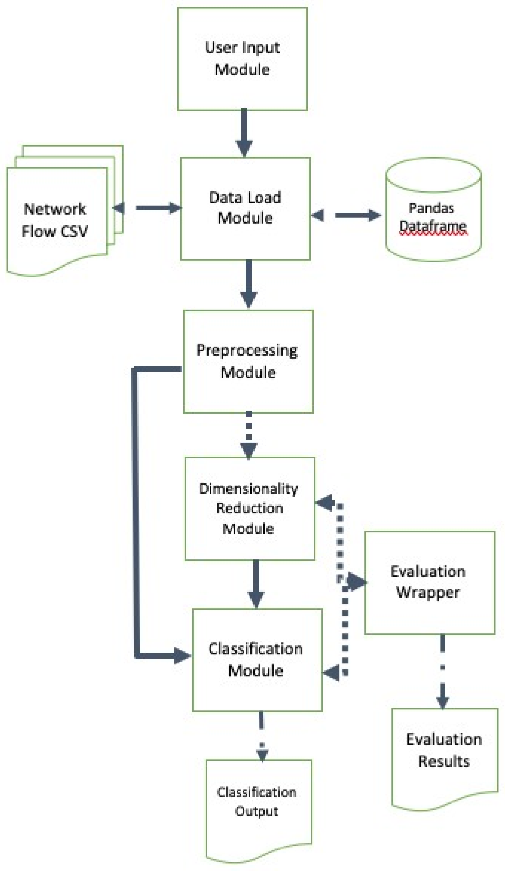


Figure 4.1: Process Flow Diagram

The approximate flow diagram in figure 8 can be compared to the IDS framework suggested by Subba, et al. (2016) in figure 2 above. The use of a network flow dataset reduces the requirements for feature extraction and selection from the unprocessed data, but does not fully eliminate the requirement for preprocessing steps in order to prepare CSV data in a format suitable for input into the dimensionality reduction and classification modules.

4.3 Summary

This chapter has outlined the design considerations for a modular test platform, including an evaluation of available toolsets and libraries. An object-oriented design methodology will enable this approach. The following chapter will detail the implementation of this design using the NumPy, pandas and Scikit Python libraries.

The implementation of this test platform will provide a basis for the tests outlined in chapter 3, while taking an iterative approach to the development of the artefact itself. The following chapter will detail the choice of classification and manifold learning approaches available within the Scikit library.

CHAPTER 5 – IMPLEMENTATION OF DESIGN

5.1 Introduction

The development of the test platform outlined in chapter 4 requires further detail. There are several classification and manifold learning approaches available in the Scikit library, each with their own considerations and requirements. The implementation of these approaches must reflect appropriate constraints for a practical implementation, including memory and performance requirements.

5.2 Selection of Implementation Method

The Scikit library includes 17 supervised learning models, in addition to 8 available implementations of manifold learning algorithms. In order to select appropriate algorithms, we must identify criteria to screen both classifiers and manifold learning approaches.

Classification algorithms available in Scikit can be divided into linear models, support vector machines, nearest neighbor algorithms, Naïve Bayes algorithms, decision trees and ensemble methods. As noted in section 2.2.2, this project posits nonlinearity of the network dataset. Zhang (2004) notes that implementations of Naïve Bayes classification assume that all features are independent given the value of the class variable. This “conditional independence” assumption is rarely true in practical application. Support vector machines (SVM) are powerful tools when applied to high dimensional spaces, but have incredibly high memory requirements as currently implemented in Scikit. The Scikit implementation of SVM uses a quadratic solver of approximate complexity $O(n_{features} * n_{samples}^3)$, which exceeds the capability of nonspecialized hardware when applied to datasets larger than several thousand samples.

Removing SVM and Naïve Bayes from consideration, the remaining classification options are nearest neighbor algorithms, decision trees, and ensemble methods. Scikit provides efficient implementations of random forest and AdaBoost ensemble methods for classification. The expected complexity of these chosen algorithms can be detailed in big O notation according to their specific implementation in Scikit. The complexity of an algorithm depends on several variables – generally consisting of sample size, number of features, and number of chosen nearest neighbors if applicable. Functions can be classified by their order, growing logarithmically, linearly, quadratically, or exponentially as the input variables increase. The complexity of the Scikit implementation of the chosen classifiers are given by Pedregosa et al. (2011) and Zhu et al. (2009) as below:

Classifier	**Complexity Function**	**Notes**
Decision Tree	$O(n_{features} * n_{samples}^2 \backslash log(n_{samples}))$	*n* as sample size
Random Forest	$O(M * n * log(n))$	*M* as number of trees, default 100
KNN	$O(D*n) + O(log(n))$	Assuming K-D tree search algorithm – *D* as input dimensions, *n* as sample size.
Adaboost	$O(dpn * log(n) * M)$	*M* as iterations, *n* as samples, *d* as tree depth, *p* as dimensions

Table 5.1: Complexity of Classification Algorithms

Scikit provides eight implementations of manifold learning algorithms: Isomap, locally linear embedding (LLE), modified LLE, Hessian Eigenmapping, spectral embedding, local tangent space alignment (LTSA), multi-dimensional scaling, and t-distributed stochastic neighbor embedding (t-SNE.)

Isomap is an extension of multidimensional scaling. It implements a lower-dimensional embedding of high-dimensional data that maintains geodesic distance between all points. [CITATION Ten00 \l 1033]

Locally linear embedding (LLE) is similar to the linear principal component analysis (PCA) approach. A locally linear embedding implements linear PCA embeddings locally, which are combined globally to form a non-linear embedding.

A known limitation of LLE is the "regularization problem" [CITATION Zha07 \l 1033], which states that specific implementations of LLE may not find a globally optimal solution if the number of neighbors near a data point is higher than the number of input dimensions. Several methods have been proposed to correct for this problem. Modified locally linear embedding (MLLE) uses weight vectors per neighborhood to appropriately weight features locally and approach a globally optimal embedding. Hessian Eigenmapping (also known as Hessian LLE) replaces the local weight vectors with a complex quadratic form at each neighborhood. [CITATION Don03 \l 1033] LTSA is an algorithmically similar approach, though not technically a variant of LLE.

Spectral embedding (also known as Laplacian Eigenmaps) is a manifold learning algorithm based on the implementation of a cost function to preserve local distance in a low dimensional space. This approach requires a search for neighboring points in the high-dimensional space [CITATION Bel03 \l 1033] in order to construct a weighted graph. The graph construction, as implemented in Scikit, has a complexity of *O[D * log(k) * N * log(N)]* where *D* represents the number of input dimensions, *k* the number of nearest neighbors, and *N* the number of samples. This is comparatively more complex than the other proposed manifold algorithms, and is considered a loglinear ordered function.

t-distributed stochastic embedding (t-SNE) is an approach best suited for the visualization of complex data. [CITATION van08 \l 1033] Although it is often advantageous to apply t-SNE to complex datasets that may contain multiple manifolds, the implementation of t-SNE in Scikit exceeds the capabilities of our test platform hardware. Additionally, this implementation of t-SNE has an approximate upper bound of five features in the lower dimensional space.

Each manifold implementation in Scikit consists of several steps within a single process. All variants of LLE (including LTSA) consist of three steps. LLE consists of a nearest neighbor search, the construction of a weight matrix (differing for each implementation), and a decomposition function that reduces the dimensionality of the dataset to fit a low-dimensional manifold. [CITATION Ped11 \l 1033] Isomap follows a similar process: a nearest neighbor search followed by a graph search, followed by a decomposition. Spectral embedding consists of two graph construction stages followed by a decomposition, differing by a lack of a search stage. The complexity notation functions of each manifold process are given in figure 10 below, with each stage corresponding to a separate function.

Manifold	**Complexity Function**	**Notes**
LLE	$O[D \log(k) * N \log(N)] + O[D N k^3] + O[d N^2]$	Nearest neighbor search, weight matrix construction, decomposition
Hessian LLE	$O[D \log(k) N \log(N)] + O[D N k^3] + O[N d^6] + O[d N^2]$	Nearest neighbor search, weight matrix construction with partial decomposition, decomposition.
Modified LLE	$O[D \log(k) N \log(N)] + O[D N k^3] + O[N (k-D) k^2] + O[d N^2]$	Nearest neighbor search, weight matrix construction, decomposition
Isomap	$O[D \log(k) N \log(N)] + O[N^2(k + \log(N))] +$	Nearest neighbor search,

	O[d N²] → $O[d\ N^2]$	graph search, decomposition
Spectral Embedding	$O[D \log(k) N \log(N)] + O[D N k^3] + O[d N^2]$	Two graph construction stages, decomposition
LTSA	$O[D \log(k) N \log(N)] + O[D N k^3] + O[k^2 d] + O[d N^2]$	Nearest neighbor search, weight matrix construction, decomposition

Table 5.2: Complexity of Manifold Algorithms

An examination of the complexity functions presents a preliminary prediction for possible improvements in computational intensity when a manifold learning algorithm is applied to a dataset before classification. All four classification algorithms include dimensionality (D or $n_{features}$) as a variable. A reduction in features will inherently improve the performance of the classifier, though the influence of feature extraction on accuracy is as yet unknown. The complexity functions of the variations of LLE, Isomap and LTSA contain quadratic ($O(n^2)$) and exponential ($O(D*N*k^3)$) elements. The implementation cost of these algorithms is reduced by a reduction in dimensionality (the difference between input dimensionality *D* and output dimensionality *d*) while still requiring expensive nearest neighbor searches determined by the value of *k* in each implementation.

5.3 Stages of Implementation

To construct our test platform, we will implement the four classification algorithms and six manifold algorithms as discussed in the previous section. This implementation will follow the modular data flow diagram shown in figure 8, with the implementation divided into sections according to the modular construction. Full code for the test platform is provided in appendix C.

5.3.1 – Hardware Considerations

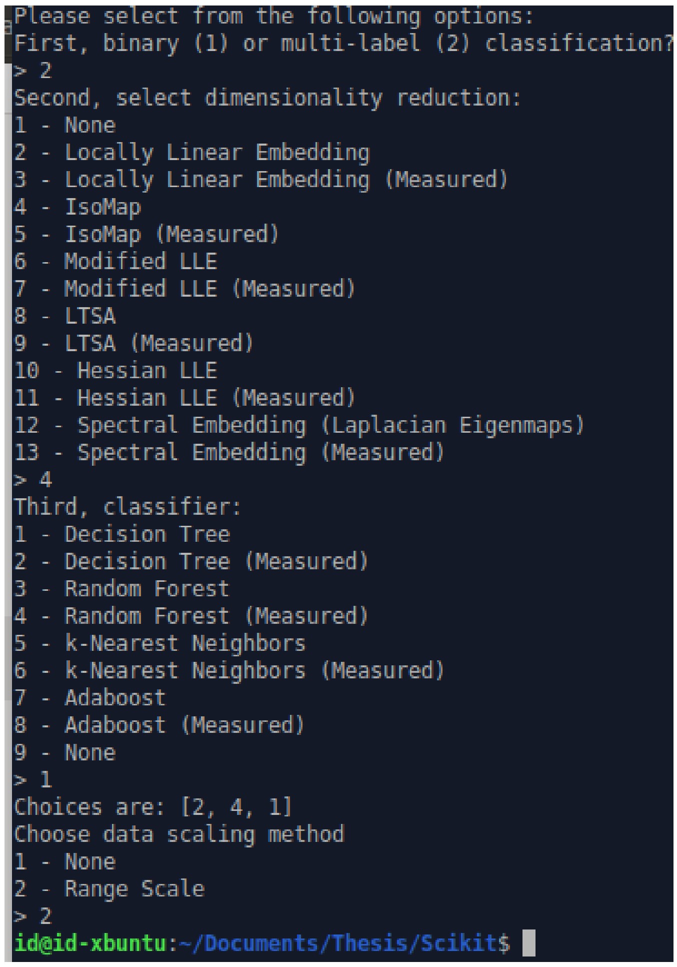
In order to effectively isolate the test platform from any confounding factors, all tests were conducted within a VirtualBox virtual machine environment. The virtual machines were constructed from a clean installation of the Xubuntu operating system, version 20.04. Xubuntu contains Python version 3.8.4 by default. Additional libraries were installed via the Python package manager pip. This libraries included scikit-learn version 0.23.2, pandas version 1.1.1, NumPy version 1.18.9, Matplotlib version 3.3.1, and any required dependencies. Each virtual machine was constructed with 10GB of RAM and access to 1 CPU at 3.60 GHz from an underlying Intel i3-8100 processor.

The use of a virtual machine for a test platform provides several advantages. Foremost, the Xubuntu operating system removes all but essential background processes. This allows the test environment to be completely isolated from the underlying operating system. Secondly, hardware access can be explicitly defined. Thirdly, the virtual environment provides the ability to rapidly scale the test platform during the test process. The combinations of manifold and classification modules required approximately 360 individual tests, ranging from several minutes to an hour each. This was accomplished by cloning several virtual machines to provide identical test platforms for simultaneous tests.

The specifications of the virtual machines were to suit the implementation of scikit-learn as a single threaded library by default. [CITATION Sci18 \l 1033] Several of the manifold objects contain a variable *n_jobs* that allows for multithreaded implementations of the manifold algorithms. This option was not available for all manifolds, and was therefore kept at a default value ensuring that tests were conducted using a single thread to ensure fairness.

5.3.2 – User Input & Selection

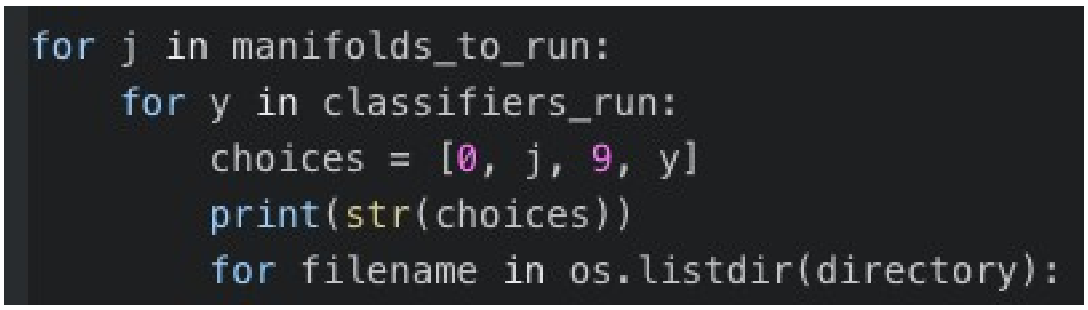
The test platform begins with an initial call for user input. The user is asked to choose a filename for processing, as well as several options corresponding to data preprocessing, data labelling, dimensionality reduction, and classification. Each selection is denoted by an integer and appended to an array. The array is passed to a core function that uses the array of user choices to select the appropriate dimensionality reduction, classification and processing steps using a case/switch statement. All further steps in the test process are contained within the core function.



```
Please select from the following options:
First, binary (1) or multi-label (2) classification?
> 2
Second, select dimensionality reduction:
1 - None
2 - Locally Linear Embedding
3 - Locally Linear Embedding (Measured)
4 - IsoMap
5 - IsoMap (Measured)
6 - Modified LLE
7 - Modified LLE (Measured)
8 - LTSA
9 - LTSA (Measured)
10 - Hessian LLE
11 - Hessian LLE (Measured)
12 - Spectral Embedding (Laplacian Eigenmaps)
13 - Spectral Embedding (Measured)
> 4
Third, classifier:
1 - Decision Tree
2 - Decision Tree (Measured)
3 - Random Forest
4 - Random Forest (Measured)
5 - k-Nearest Neighbors
6 - k-Nearest Neighbors (Measured)
7 - Adaboost
8 - Adaboost (Measured)
9 - None
> 1
Choices are: [2, 4, 1]
Choose data scaling method
1 - None
2 - Range Scale
> 2
id@id-xbuntu:~/Documents/Thesis/Scikit$
```

Figure 5.3: User input selection

The use of an array to determine each combination of test elements provides the ability to pass several tests at once using a simple nested loop that iterates over all possible manifold choices, as shown in figure 12. This loop bypasses the user input function, directly passing the `choices` array to the core function for execution.



```
for j in manifolds_to_run:
    for y in classifiers_run:
        choices = [0, j, 9, y]
        print(str(choices))
        for filename in os.listdir(directory):
```

Figure 5.4: Iteration through multiple tests

5.3.3 – Data Conversion & Preprocessing

The second module of the test platform uses the filename provided by the user to create a pandas DataFrame. The pandas library provides a `read_csv` method that accepts the CSV file as input. This method allows for explicit type declarations for each CSV column. Numeric columns are loaded as 64-bit float objects by default while non-numeric columns are loaded as string objects. Our `first_processing` function accepts a DataFrame as input and removes nonessential columns from the dataset. It then parses the DataFrame for values defined by pandas as infinite or NaN. Infinite values are those integers that exceed the maximum size of a float objects, while NaN values are blank or missing in the dataset. For the purposes of this dataset, all rows containing NaN or infinite values are dropped from the DataFrame with a warning provided to the user that some data is lost.

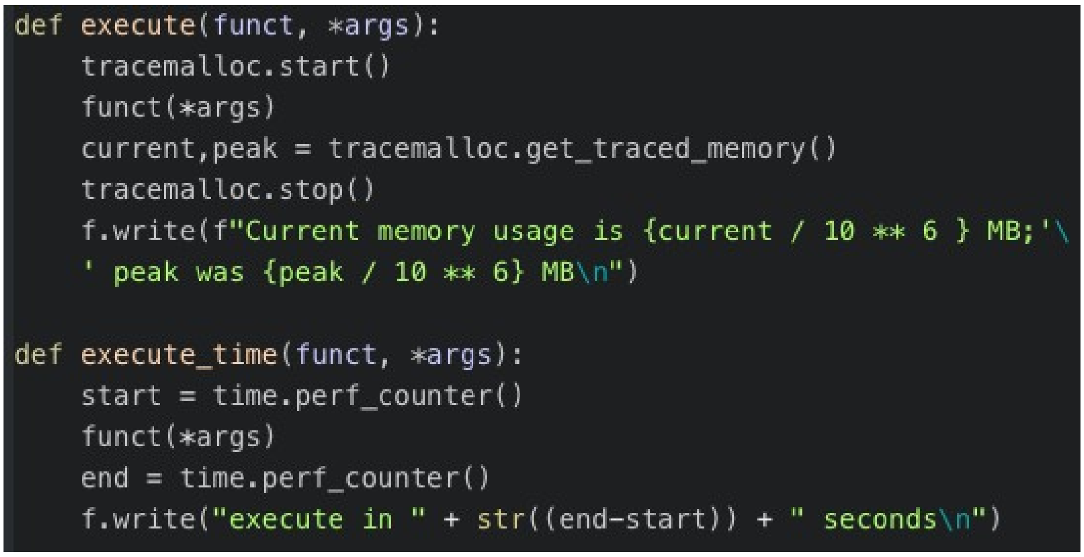
As noted in section 5.2, manifold learning algorithms are generally dependent on nearest-neighbor searches as an initial step in a multipart process. Scikit strongly advises that data is appropriately scaled before manifold learning is applied in order to equalize units of measurement for each feature. The units of measure for the features of a network flow object vary wildly, from flow duration (measured in microseconds) to average packet size (rarely exceeding double digits.) Standardizing these units with an appropriate scaling measure ensures that weight matrices are constructed without undue influence from disparate units of measurement. Although the test platform provides the option to scale or not scale the data, all tests were conducted with data appropriately scaled using the Scikit `MinMixScaler` functionality. Once the data conversion and preprocessing functions are complete, a DataFrame object is returned back to the core function to be passed to later functions.

5.3.4 – Measurement & Evaluation Functions

Measurements of algorithm performance are conducted according to three criteria: time (computational intensity), memory usage, and algorithm specific scoring. Scoring for manifolds is given by a measure of reconstruction error, defined as the difference between the projection of data according to the derived manifold and the actual position of the data in the original high dimensional space. Scoring for classifiers is given by a measure of F1 score, defined as *F1 = 2 * (precision * recall) / (precision + recall),* and confusion matrix. In multi-class classification cases, Scikit calculates the F1 score by averaging the F1 score for each class.

Memory usage is measured using the Python `tracemalloc` function, while time is measured using the Python `perf_counter` function. Python allows the construction of functions that accept other functions as an argument (also known as higher order functions.) The two higher order measurement functions showed in figure 13 accept a lower order function and its required arguments as parameters. The measurement functions run the lower order function within a measurement block in order to calculate memory usage and run time. The use of higher order functions allows a modular approach where any function can be passed to the evaluation function while maintaining equivalent measurements across all tests.

The implementation of classification and manifold learning algorithms requires a specific design consideration in order to effectively measure memory usage. If a DataFrame of size *n* is passed to a function, the minimum memory usage of that function is equal to *n*. If the function returns a new DataFrame of size *m*, the memory usage will account for both DataFrame objects. This is an incorrect estimation if the first DataFrame is no longer used after the conclusion of the function. Memory management in Python is automatic and based on reference counts. If an object in memory is no longer referenced, it will be removed by Python's inbuilt garbage collection. As such, all arrays generated as output of transformation are returned using the same reference variable as the input arrays. This ensures that the input arrays are not held in memory beyond their purpose, limiting the peak memory consumption of the algorithm.



```python
def execute(funct, *args):
    tracemalloc.start()
    funct(*args)
    current,peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    f.write(f"Current memory usage is {current / 10 ** 6 } MB;'\
    ' peak was {peak / 10 ** 6} MB\n")

def execute_time(funct, *args):
    start = time.perf_counter()
    funct(*args)
    end = time.perf_counter()
    f.write("execute in " + str((end-start)) + " seconds\n")
```

Figure 5.5: Measurement functions for time and memory usage

Higher order functions can be used for memory and time evaluation, but are not available for the function specific evaluation of F1 score, confusion matrices and reconstruction error. Each algorithm is placed within a dedicated function, as outlined in sections 5.3.5 and 5.3.6. Each algorithm function accepts three parameters containing the DataFrame, class labels, and a

Boolean variable denoting evaluation. As shown in figure 14, passing a True value to the evaluation variable calculates the performance statistics for the algorithm.

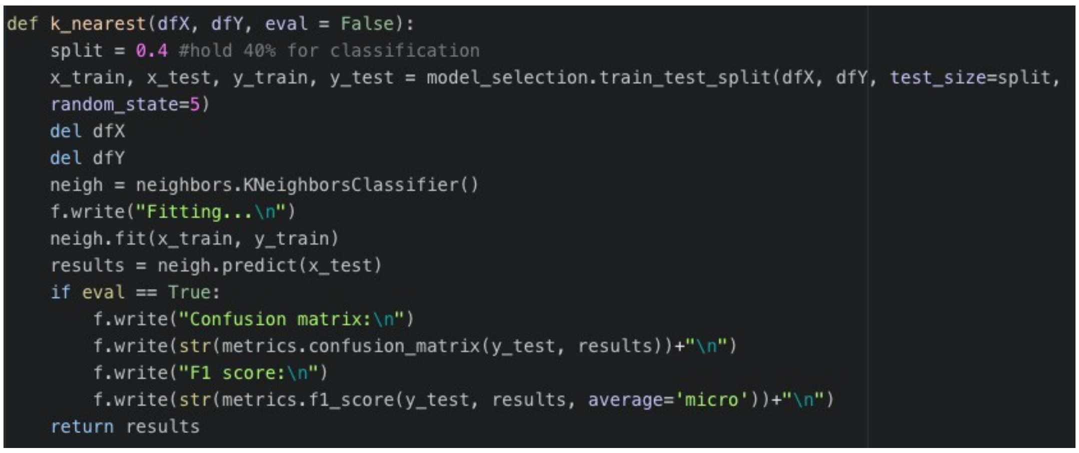

```
def k_nearest(dfX, dfY, eval = False):
    split = 0.4 #hold 40% for classification
    x_train, x_test, y_train, y_test = model_selection.train_test_split(dfX, dfY, test_size=split,
    random_state=5)
    del dfX
    del dfY
    neigh = neighbors.KNeighborsClassifier()
    f.write("Fitting...\n")
    neigh.fit(x_train, y_train)
    results = neigh.predict(x_test)
    if eval == True:
        f.write("Confusion matrix:\n")
        f.write(str(metrics.confusion_matrix(y_test, results))+"\n")
        f.write("F1 score:\n")
        f.write(str(metrics.f1_score(y_test, results, average='micro'))+"\n")
    return results
```

Figure 5.6: Implementation of KNN Classifier

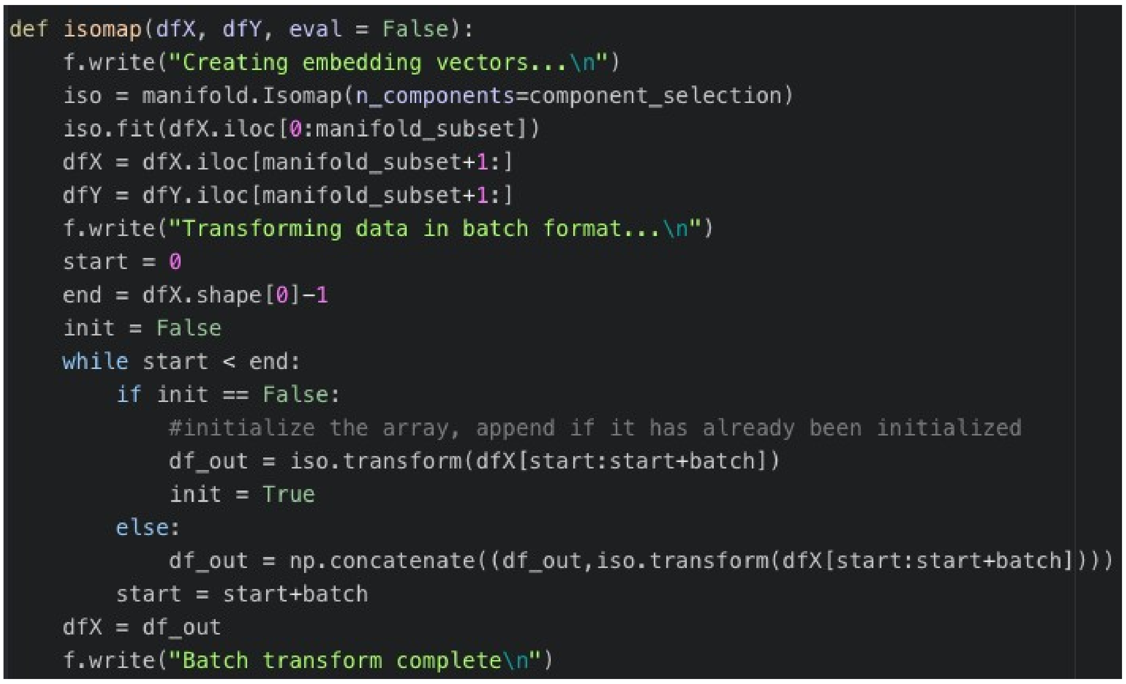
It is important to note the observation effect in our measurement functions. The `tracemalloc` function adds approximately 20% additional computational overhead during the execution of a function. Scoring functions for the algorithms add similar overhead in both computation and memory execution when calculating F1 scores and confusion matrices. As such, it would be improper to measure memory usage and runtime within the same function. It is also expected that there is some variance between experimental iterations based on variability in background processes, random sampling, or randomness in the implementation of the algorithms. To account for this variability, all evaluation metrics are computed three times for a total of nine iterations of an algorithm when a user requests evaluation metrics. Each measurement is calculated separately in order to avoid additional overhead from measurement functions themselves.

5.3.5 – Dimensionality Reduction

Each of the dimensionality reduction approaches outlined in section 5.2 were implemented as separate functions within the test platform. Each function followed a single design pattern in order to maintain equivalency and consistency in testing. All dimensionality reduction approaches follow a pattern of steps as shown by the implementation of Isomap in figure 15.

With the exception of the locally linear embedding function, all dimensionality reduction functions accept three parameters – DataFrame, class labels, and a Boolean variable for evaluation criteria. Scikit implements the three varieties of LLE and the implementation of LTSA through a single object class with an optional variable `method`. This implementation required conditional statements within the LLE function to determine the LLE method, as shown in appendix C. Each implementation of LLE requires a non-default setting for the `n_neighbors` variable. Modified LLE has a constraint of $n_{neighbors} > n_{components}$, while Hessian LLE has a constraint of $n_{neighbors} > n_{components} * (n_{components} + 3)/2$. All function variables, with the except of the $n_{neighbors}$ constraints above, are left at their default values. A separate constraint is applied to the Scikit implementation of LTSA. LTSA must be implemented with a maximum of five components in the output dimension, one quarter of the twenty components implemented for the other manifolds.

The dimensionality reduction module follows a sequence of steps. First, a Scikit object of class `Manifold` is initialized. Secondly, a manifold subset is selected, as defined by a global variable. This subset is used to train the manifold through the fit function. To avoid overfitting, this subset is removed from the implementation of the classifier. Once a manifold is constructed, the high-dimensional data is transformed to a lower dimensional space using the `transform` function.



```python
def isomap(dfX, dfY, eval = False):
    f.write("Creating embedding vectors...\n")
    iso = manifold.Isomap(n_components=component_selection)
    iso.fit(dfX.iloc[0:manifold_subset])
    dfX = dfX.iloc[manifold_subset+1:]
    dfY = dfY.iloc[manifold_subset+1:]
    f.write("Transforming data in batch format...\n")
    start = 0
    end = dfX.shape[0]-1
    init = False
    while start < end:
        if init == False:
            #initialize the array, append if it has already been initialized
            df_out = iso.transform(dfX[start:start+batch])
            init = True
        else:
            df_out = np.concatenate((df_out,iso.transform(dfX[start:start+batch])))
        start = start+batch
    dfX = df_out
    f.write("Batch transform complete\n")
```

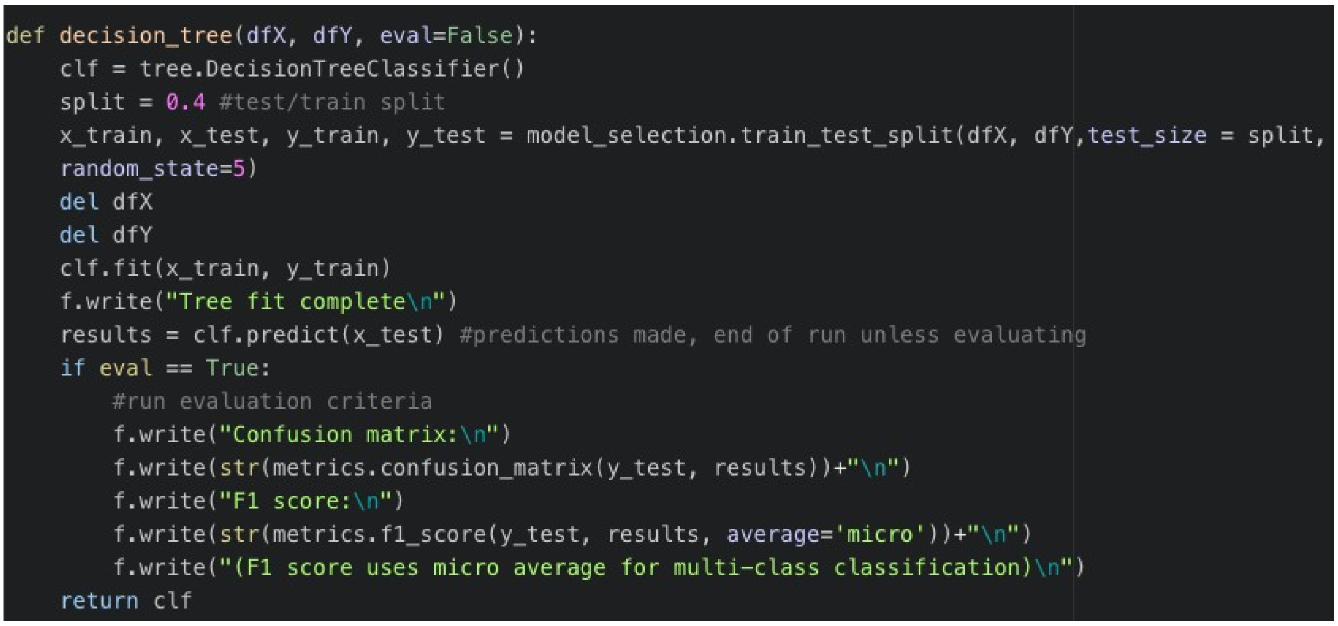
Figure 5.7: Implementation of Isomap dimensionality reduction

The complexity and performance of the implementation of the transform function for each manifold requires the construction of an array of approximately $n * n$ size. Initial implementation of this transformation over full datasets quickly reached memory requirements exceeding 1.5 TiB. Practical implementations of intrusion detection systems include throughput bandwidth limits for this reason. For the purposes of this project, a batch transformation approach was selected to approximate this throughput limit. Batch transformation of the dataset is conducting using one manifold to ensure that all data is transformed according to the same calculated embedding vectors. The outcome of this batch transformation is identical to the outcome of a single transformation over the entire dataset. This implementation limits memory usage while maintaining the same number of operations.

The Scikit implementation of spectral embedding does not provide a transform method suitable for a batch transformation. The spectral embedding object provides a `fit_transform` method that combines the manifold generation and transformation steps. This implementation does not scale effectively, and exceeded the capabilities of our test hardware.

5.3.6 – Classification

The classification stage is implemented similarly to the dimensionality reduction stage in the previous section. As with dimensionality reduction, classifiers accept the same three parameters as input to the functions. Each classification block is designed to the same pattern in order to allow for effective comparison. First, an object of the classifier type is initiated. Secondly, the data for classification is divided into training and testing sets using the Scikit `train_test_split` function. Once the training and testing arrays are created, the original arrays (`dfX` and `dfY`) are removed from memory to ensure proper memory measurement. The training data is used to train the classifier, which is then used to predict the class labels of the test data. Evaluation criteria are produced according to the value of the Boolean argument `eval` as noted previously.



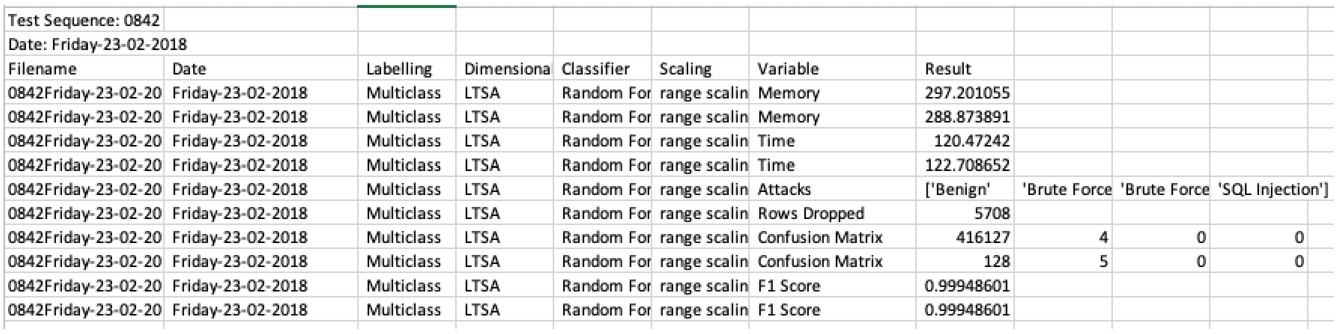
```python
def decision_tree(dfX, dfY, eval=False):
    clf = tree.DecisionTreeClassifier()
    split = 0.4 #test/train split
    x_train, x_test, y_train, y_test = model_selection.train_test_split(dfX, dfY,test_size = split,
    random_state=5)
    del dfX
    del dfY
    clf.fit(x_train, y_train)
    f.write("Tree fit complete\n")
    results = clf.predict(x_test) #predictions made, end of run unless evaluating
    if eval == True:
        #run evaluation criteria
        f.write("Confusion matrix:\n")
        f.write(str(metrics.confusion_matrix(y_test, results))+"\n")
        f.write("F1 score:\n")
        f.write(str(metrics.f1_score(y_test, results, average='micro'))+"\n")
        f.write("(F1 score uses micro average for multi-class classification)\n")
    return clf
```

Figure 5.8: Implementation of decision tree classifier

5.3.7 – Output

The output of the test platform is written to a text file during the execution of each stage, as shown in the implementation of a decision tree classifier in figure 16 above. An example of this output is included in appendix E. The format of the test output was written to be human-readable, in keeping with the iterative adjustments required throughout the implementation process. A separate utility, shown in appendix D, was developed to convert the results to an array format suitable for the analysis of test results in chapter 6.

The utility iterates over each test output file produced by the test platform, and extracts pertinent statistics into a CSV format. Test results, as shown in figure 17, include the processed file name, date of network capture, attacks included in the capture, the options chosen for the evaluation through user input, and the results of the memory, time and accuracy evaluations.



Test Sequence: 0842										
Date: Friday-23-02-2018										
Filename	Date	Labelling	Dimensiona	Classifier	Scaling	Variable	Result			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Memory	297.201055			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Memory	288.873891			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Time	120.47242			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Time	122.708652			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Attacks	['Benign'	'Brute Force	'Brute Force	'SQL Injection']
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Rows Dropped	5708			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Confusion Matrix	416127	4	0	0
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	Confusion Matrix	128	5	0	0
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	F1 Score	0.99948601			
0842Friday-23-02-20	Friday-23-02-2018	Multiclass	LTSA	Random For	range scalin	F1 Score	0.99948601			

Figure 5.9: Sample test output

5.4 Summary

Chapter 3 outlined the nature of our dimensionality reduction problem, while chapter 4 outlined the design considerations of our test platform. This chapter has explored the implementation and practical considerations of that design, including limitations on memory usage and the use of a batch transformation process for manifold conversions.

The test platform developed in this chapter was developed through an iterative process and required several adjustments to suit our design objectives. In the following chapter, this platform is used to examine the specific hypotheses detailed in chapter 3.

CHAPTER 6 – VALIDATION AND TESTING

6.1 Introduction

The test platform artefact developed in chapter 5 provides the ability to run individual tests based on user input, or to run batched tests through nested iterative loops. In this chapter, we will produce a test plan to evaluate the hypotheses outlined in chapter 3. We detail the order and structure of the test plan, run the required experiments, and evaluate the experimental results.

6.2 Selection of Testing/Validation Method

The research questions developed in section 3.4 outline a test plan with which to conduct our experiments. First, a baseline is calculated for classifier performance over the available network captures. Secondly, each manifold approach is evaluated over the same dataset to determine memory usage, time requirements, and reconstruction error. Lastly, each manifold is evaluated alongside each classifier in order to determine the performance impact of nonlinear dimensionality reduction.

Chapter 5 outlined the selection of appropriate manifolds and classifiers. The experimental phase will evaluate the performance of Scikit-learn implementations of decision tree, random forest, Adaboost and k-nearest neighbor classifiers alongside locally linear embedding (standard, modified and Hessian), Isomap, and LTSA manifold algorithms. Section 5.3.5 discusses the removal of spectral embedding from the test suite due to hardware limitations.

6.3 Testing the Implementation

In order to run each test over each of the nine available network capture files the test phase consists of 36 tests in the first phase, 45 tests in the second phase, and 180 tests in the third phase. Each measurement stage requires each algorithm to be evaluated three times (memory, time, and algorithm specific criteria) and runs that evaluation three times in order to account for any variation in the process. The tests were conducted using a series of virtual machines in order to run several tests simultaneously. The test output files were collected and processed to produce a combined CSV output file.

6.4 Results of Testing

In order to establish a baseline for the performance of the four classification algorithms, each classifier was evaluated over each of the nine days of available traffic. Each day contains one type of network attack, as shown in figure 6. F1 scores for each classifier are shown in figure 18 below.

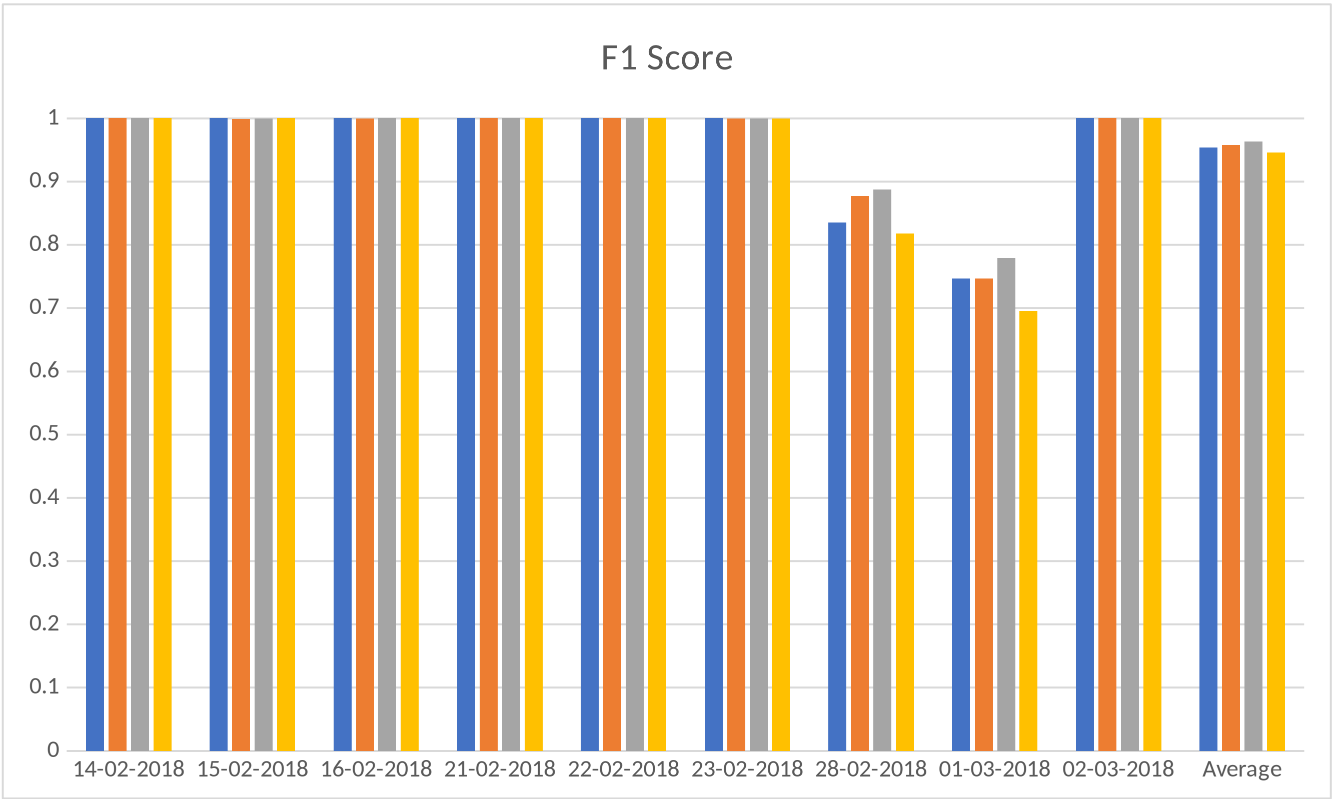


Figure 4.1: Accuracy of each classifier

The accuracy of each classifier before the application of manifold learning is nearly perfect, with two noticeable exceptions. A substantial reduction in accuracy is seen for all classifiers when applied to the network infiltration attack on March 1st and March 2nd. The nature of the infiltration attack is briefly discussed in section 3.3. This attack includes a phishing email, a Metasploit backdoor, and several iterations of port and range sweeps over the internal IP space. This attack is unique in that it contains several distinct steps, each involving distinct generative mechanisms and network artefacts, labelled as one class within the data. Further review of the confusion matrices generated for each classifier shows an equal balance between type I errors (false negative) and type II errors (false positive) for each classifier. The complex nature of the infiltration attack results in the least accurate classification model.

Baseline memory use of each classifier is shown in figure 19, and the time required to run each classifier is shown in figure 20. Memory usage and run time remain similar for each classifier over the range of network captures, with the exception of classification of the infiltration attack.

The k-nearest neighbors classifier shows the highest memory use, while the Adaboost classifier is the slowest average algorithm. Decision tree, as the simplest algorithm, is by far the fastest.

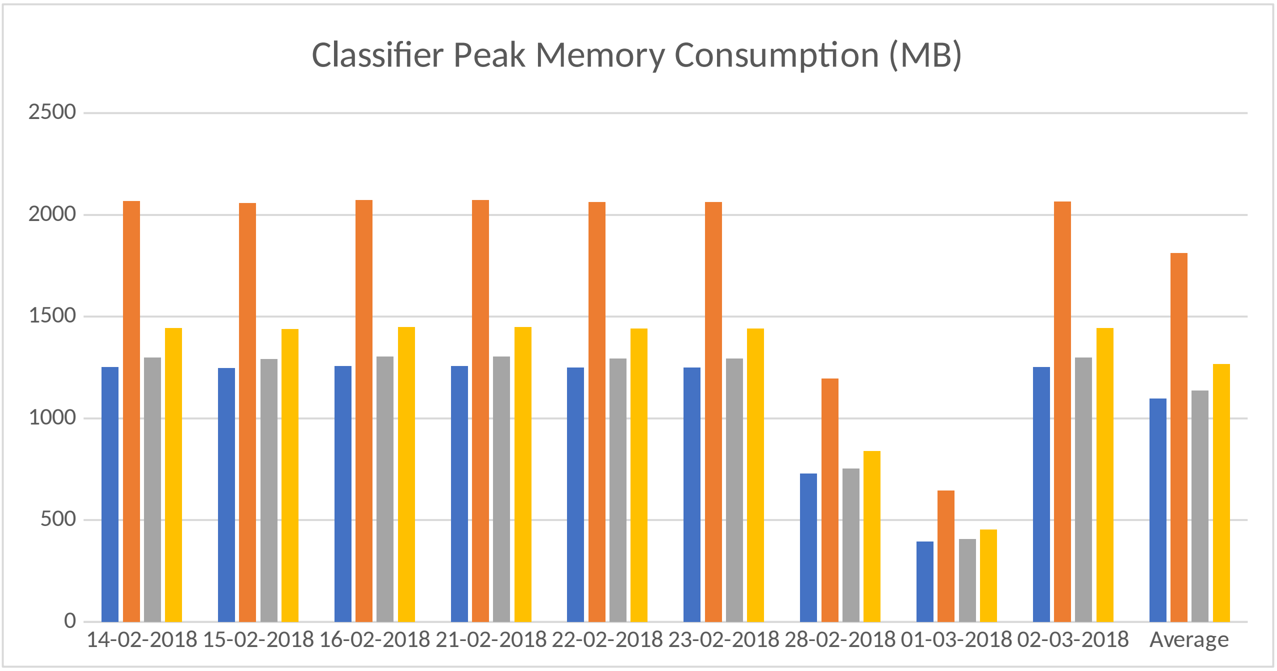


Figure 6.2: Classifier memory consumption

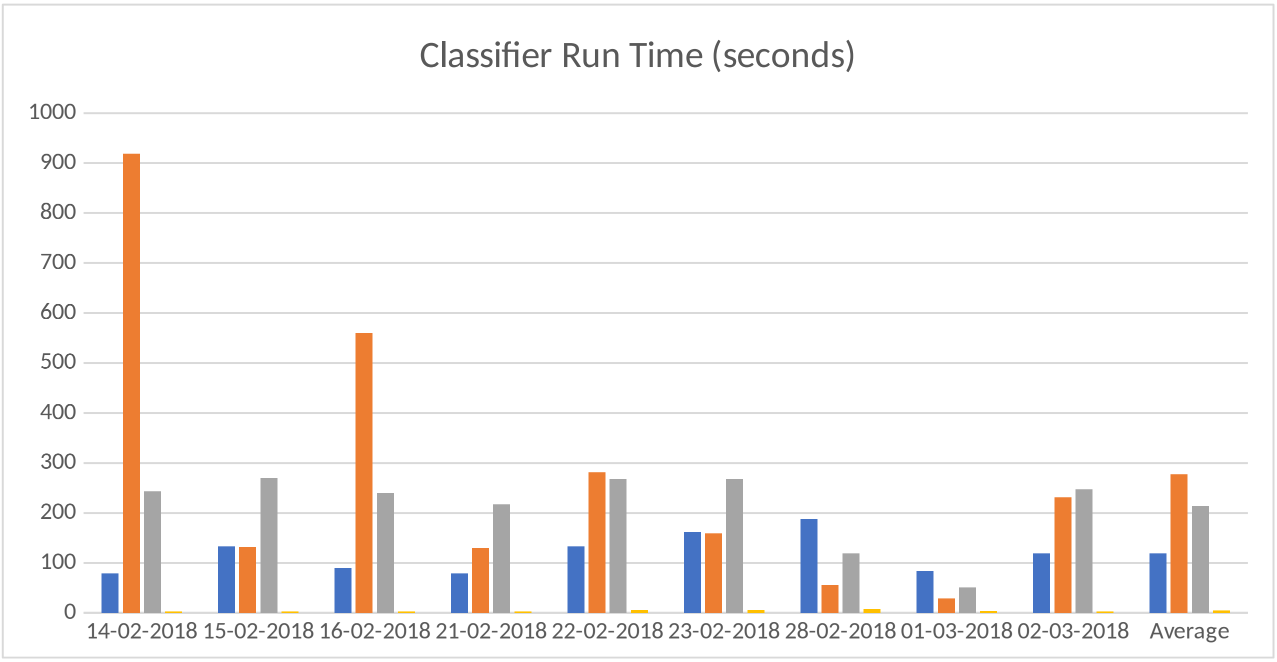


Figure 6.3: Classifier run time

The baseline accuracy of the classifiers exceeded expectations. This left limited room for manifold learning to improve the accuracy of the classifiers, and adjusts the potential outcomes from the second and third stages of the test plan. Where we had originally hypothesized that feature extraction could improve the accuracy of a classifier by reducing the noise inherent in a high-dimensional feature set, we will now evaluate whether any improvements in run time or memory consumption come at the expense of accuracy as calculated in the baseline experiments.

To complete the second phase of testing, each manifold algorithm was evaluated over each network capture. Each manifold reduced the dimensionality of the input data from 78 features to 20 features, with the exception of LTSA. As noted in section 5.3.5, the Scikit implementation of LTSA allows a maximum of 5 features in the output dimension. Experimental results from the LTSA manifold are included, but should be considered as an outlier noncongruent with the remaining manifolds. Memory consumption and run time for each classifier is shown in figure 21 and 22, respectively. Although memory consumption remains relatively equal between the manifolds, Hessian LLE requires an average run time far exceeding all other algorithms. As noted in section 5.3.5, the Scikit implementation of Hessian LLE has a constraint of $n_{neighbors} > n_{components} * (n_{components} + 3)/2$. The function implementing this model in our test platform sets $n_{neighbors}$ accordingly. The notation of the three LLE functions contains the element $O[D\ N\ k^3]$ which scales exponentially as k increases, and the value of the $n_{neighbors}$ variable for each implementation has a substantial impact on performance.

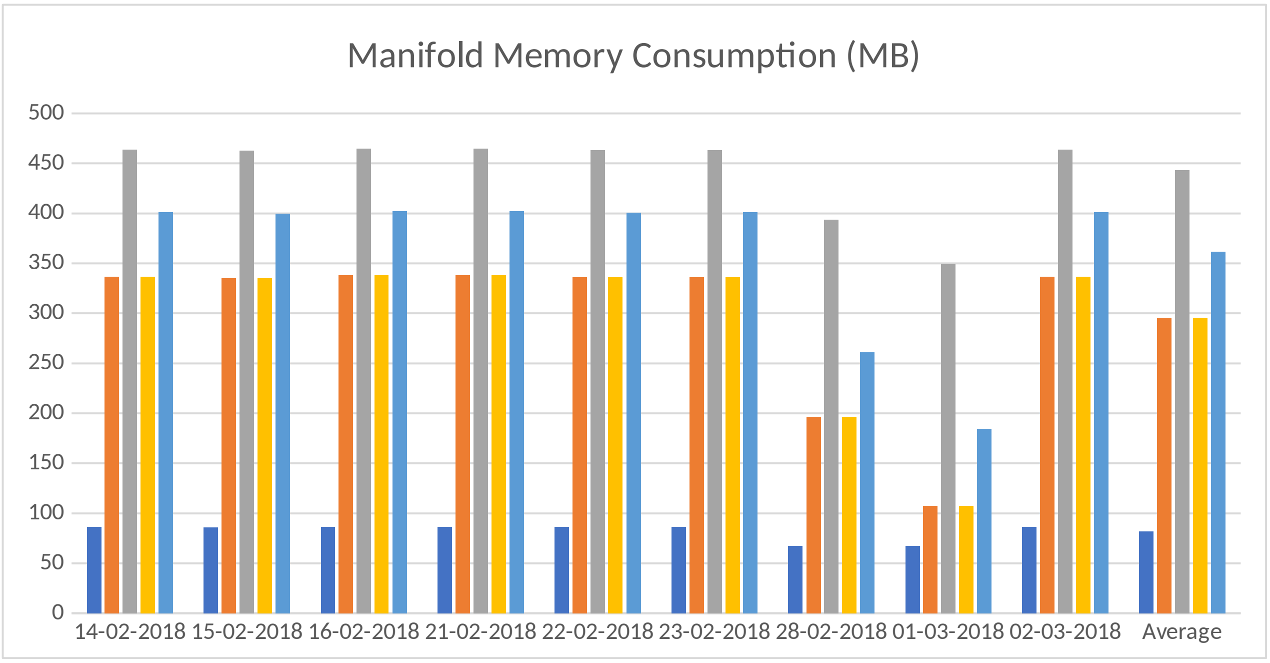


Figure 6.4: Manifold memory consumption

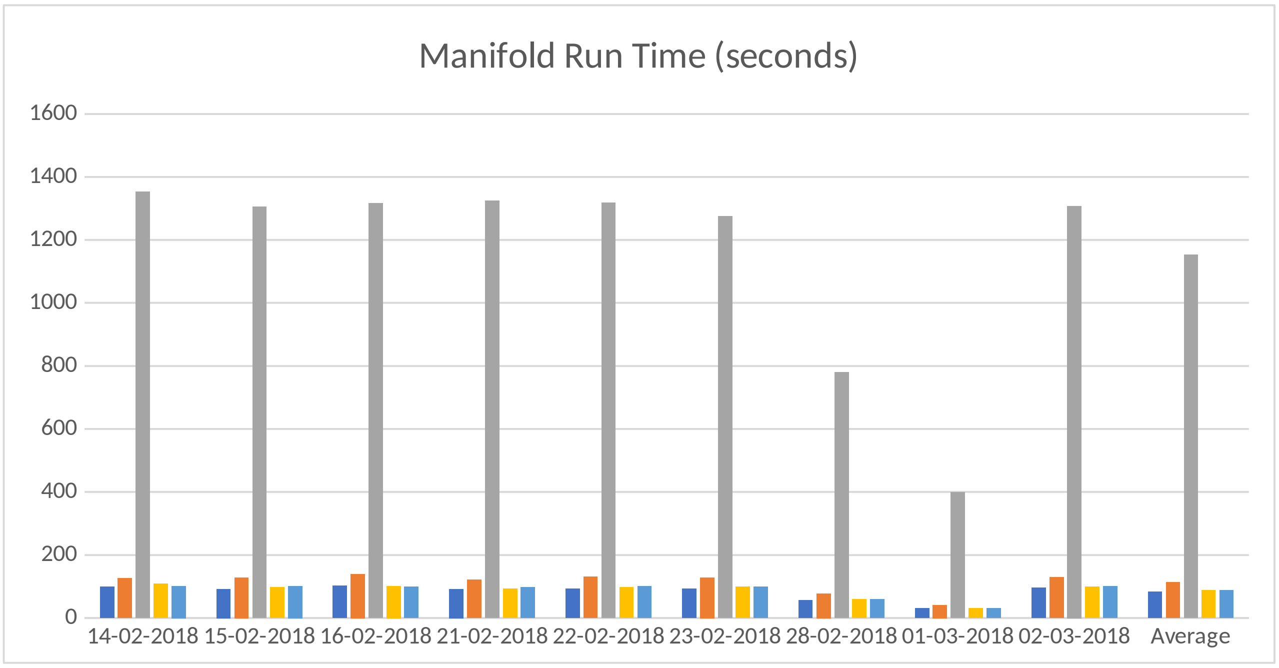


Figure 6.5: Manifold run time

Although the comparison of LTSA to the remaining manifolds is imperfect, it is worth noting that a substantial reduction in output feature space appears to have a limited impact on the run time of the manifold. This implies that the majority of the run time of the manifold is devoted to the transformation of the network data rather than the mapping of embedding vectors from the high-dimensional space to the low-dimensional space. HLLE serves as an outlier in that aspect, demonstrating that a drastic increase in the nearest neighbors search can also negatively impact the run time performance of the manifold process.

Figure 23, below, examines reconstruction error for each manifold approach. The Scikit implementation of Isomap does not provide a reconstruction error metric. The three locally linear variants provide reconstruction error as an evaluation metric. Reconstruction error is a measure of the error inherent in the projection of a data point into a high dimensional space via the embedding vectors. This metric varies substantially within each approach over the dataset. Hessian LLE provides both the best and worst projections, while Modified LLE remains relatively equal excluding a single outlier.

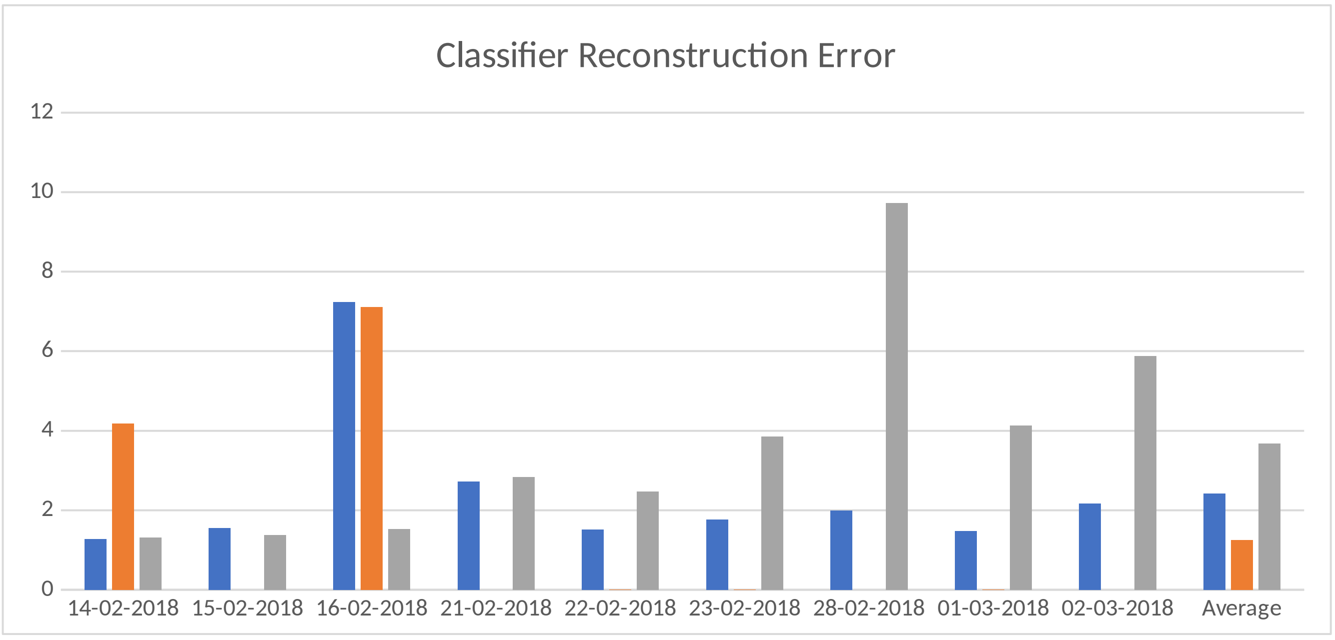


Figure 6.6: Manifold Reconstruction Error

With performance metrics established for the classifiers and the dimensionality reduction functions, we can combine the possible options for evaluation. Detailed output metrics for each combination are shown in appendix F. Memory consumption and run time for the combination experiments are measured for the classifier only, not the combined manifold and classifier. A true approximation of the total cost of the combined manifold and classifier is equal to the measurements for the classifier after dimensionality reduction, plus the cost of the manifold as shown in the second phase.

The performance of the classifiers on a dimensionality reduced dataset, measured by F1 score, shows a slight overall decrease. (Figure 24) Adaboost sees the largest decrease in overall performance with an average F1 score reduction of approximately 7%. No classifier shows a significant difference between the use of an LTSA manifold containing 5 components and the remaining manifolds, containing 20 components.

Combination	**Average**
LTSA, Adaboost	0.92383
MLLE, Adaboost	0.83739
HLLE, Adaboost	0.9377
LLE, Adaboost	0.90508
Isomap, Adaboost	0.88061
None, Adaboost	0.96272
LTSA, Decision Tree	0.95157

MLLE, Decision Tree	0.92313
HLLE, Decision Tree	0.94391
LLE, Decision Tree	0.94654
Isomap, Decision Tree	0.93437
None, Decision Tree	0.94581
LTSA, KNN	0.95576
MLLE, KNN	0.95714
HLLE, KNN	0.95737
LLE, KNN	0.95565
Isomap, KNN	0.95539
None, KNN	0.95789
LTSA, Random Forest	0.95226
MLLE, Random Forest	0.92859
HLLE, Random Forest	0.9556
LLE, Random Forest	0.9501
Isomap, Random Forest	0.94287
None, Random Forest	0.95339

Table 6.7: F1 scores of classifiers for dimensionality reduced datasets

Classification based on the HLLE reduced dataset shows the least average performance reduction. Some combinations of manifold and classifier show a slight performance improvement, though this varies based on the individual experiment and does not represent a significant difference. Performance is not significantly different when examined based on varying attack types, including the infiltration attack that provided the worst baseline performance.

Combination	**Avg. Time**	**Avg. Memory**
LTSA, Adaboost	145.16854	247.85155
MLLE, Adaboost	166.42199	247.8509
HLLE, Adaboost	174.08083	247.85126
LLE, Adaboost	151.10271	247.85508
Isomap, Adaboost	182.22475	247.85083

None, Adaboost	213.49637	1137.7127
LTSA, Decision Tree	3.06083	256.12219
MLLE, Decision Tree	4.72109	256.12206
HLLE, Decision Tree	5.59575	256.12206
LLE, Decision Tree	2.79813	256.12502
Isomap, Decision Tree	5.86232	256.12201
None, Decision Tree	4.11444	1266.4002
LTSA, KNN	73.92909	245.50165
MLLE, KNN	393.32607	245.5035
HLLE, KNN	83.69472	245.50343
LLE, KNN	75.73427	245.50478
Isomap, KNN	83.24476	245.50303
None, KNN	284.13786	1811.373
LTSA, Random Forest	149.61213	257.45786
MLLE, Random Forest	212.61597	257.45886
HLLE, Random Forest	226.42079	257.45867
LLE, Random Forest	146.59039	257.46237
Isomap, Random Forest	268.33577	257.45853
None, Random Forest	118.44459	1098.263

Table 6.8: Average run time and peak memory

The average memory and run time performance of each classifier is shown in figure 25, above. Change in run time performance differs by classifier. Adaboost and KNN classification improve by an average of 23.3% and 50.29%, respectively, while decision tree and random forest classifications become substantially slower.

Memory consumption improves equally for all classifiers regardless of the specific manifold approach chosen. The average peak memory consumption of the KNN classifier on a non-reduced dataset is 1811.373 MB, while the same classification applied to all five dimensionally reduced datasets has an average peak memory consumption of 245.5 MB. This consistency holds for all experimental combinations.

It is worth noting that the average improvement in memory consumption is not proportional to the reduction in dimensionality. Each high-dimensional dataset contains 77 features and each lower dimensional mapping contains either 5 features (LTSA) or 20 features. The average improvement in memory consumption is equal regardless of the size of the lower-dimensional feature space. This implies that memory usage is governed by an approximately logarithmic function within the chosen feature space. The precise nature of that relationship provides an opportunity for further work, as noted in section 7.4.

A practical evaluation of the performance of our conceptual IDS is shown in figure 26 by combining the run time and maximum peak memory usage of both elements. An IDS pipeline constructed with a KNN classifier and LLE dimensionality reduction would require a total runtime of 163.27 seconds, and peak memory usage of 295.67 MB (representing the peak memory usage of the manifold, which exceeds the peak memory usage of the classifier.).

	Classifier (Average)		**Manifold (Average)**		**Components Combined**		
Combination	Memory	Runtime	Memory	Runtime	Peak Memory	Runtime	F1 Score
LTSA, Adaboost	247.85	145.17	82.1	83.77	247.85	228.94	0.924
MLLE, Adaboost	247.85	166.42	295.69	113.64	295.69	280.06	0.837
HLLE, Adaboost	247.85	174.08	443.17	1153.12	443.17	1327.2	0.938
LLE, Adaboost	247.86	151.1	295.67	87.54	295.67	238.64	0.905
Isomap, Adaboost	247.85	182.22	361.47	87.87	361.47	270.09	0.881
None, Adaboost	1137.71	213.5	0	0	1137.71	213.5	0.963
LTSA, Decision Tree	256.12	3.06	82.1	83.77	256.12	86.83	0.952
MLLE, Decision Tree	256.12	4.72	295.69	113.64	295.69	118.36	0.923
HLLE, Decision Tree	256.12	5.6	443.17	1153.12	443.17	1158.72	0.944
LLE, Decision Tree	256.13	2.8	295.67	87.54	295.67	90.34	0.947
Isomap, Decision Tree	256.12	5.86	361.47	87.87	361.47	93.73	0.934
None, Decision Tree	1266.4	4.11	0	0	1266.4	4.11	0.946
LTSA, KNN	245.5	73.93	82.1	83.77	245.5	157.7	0.956
MLLE, KNN	245.5	393.33	295.69	113.64	295.69	506.97	0.957
HLLE, KNN	245.5	83.69	443.17	1153.12	443.17	1236.81	0.957
LLE, KNN	245.5	75.73	295.67	87.54	295.67	163.27	0.956
Isomap, KNN	245.5	83.24	361.47	87.87	361.47	171.11	0.955
None, KNN	1811.37	284.14	0	0	1811.37	284.14	0.958
LTSA, Random Forest	257.46	149.61	82.1	83.77	257.46	233.38	0.952
MLLE, Random Forest	257.46	212.62	295.69	113.64	295.69	326.26	0.929
HLLE, Random Forest	257.46	226.42	443.17	1153.12	443.17	1379.54	0.956
LLE, Random Forest	257.46	146.59	295.67	87.54	295.67	234.13	0.95
Isomap, Random Forest	257.46	268.34	361.47	87.87	361.47	356.21	0.943
None, Random Forest	1098.26	118.44	0	0	1098.26	118.44	0.953

Table 6.9: Combined metrics

From this summary, we can draw several conclusions. When compared to classification without dimensionality reduction, nonlinear dimensionality reduction results in a slight decrease in overall classification performance as shown by F1 score. Although several classifiers (decision tree and random forest) run substantially slower when applied to the dimensionally reduced dataset, KNN and Adaboost show average improved run times of approximately 23.3% and 50.29%, respectively. KNN shows the largest improvement in classification speed, in several cases improving by an amount greater than the run time cost of the manifold stage. Classification without dimensionality reduction requires lower peak memory usage, as peak memory consumption of a dual phase classification is reached during the dimensionality reduction phase.

The combination of these factors has important ramifications for the implementation of our conceptual IDS. An IDS implemented according to our test framework would provide a decreased performance in classification accuracy, peak memory usage, and run time in the majority of cases. The improvement of classification performance over a dimensionally reduced dataset suggests the availability of performance improvements in an implementation where manifolds are trained irregularly and used for multiple classifications. An IDS containing a previously trained manifold transforming data before classification could divide the cost of the manifold learning phase over several instances of classification, providing a net gain to the performance of the IDS. Provisionally, our findings yield promising results. Further work on alternative implementations of manifold learning within an IDS framework is outlined in section 7.4.

6.5 Summary

The research question posed in chapter 1 posited the hypothesis that nonlinear dimensionality reduction could improve the performance of a classification algorithm over a network dataset. We defined performance as a combination of F1 score, run time, and peak memory consumption. The experimental results outlined in the previous section demonstrate marked improvement in the run time and memory consumption of a classifier over a dimensionally reduced dataset, but not at a level exceeding the cost of the manifold learning process. A dual-phase IDS constructed with a manifold learning component does not, on average, deliver a net benefit in performance. However, the identified improvements in classifier performance provide a basis for a future IDS architecture that could distribute the cost of the manifold process over several executions.

The following chapter outlines the limitations of this research, as well as identified further opportunities for additional development.

CHAPTER 7 – CONCLUSIONS AND CRITICAL EVALUATION

7.1 Introduction

The previous chapter provided experimental results and an evaluation of our experimental design. This chapter evaluates the project itself, including an evaluation of the research process and programme as established through the project proposal and first chapter.

7.2 Critical Evaluation

The project aim and objectives as put forward in the first chapter differ from the aim and objectives from the original proposal, developed in January 2020. The initial aim and objectives included determining a modelling approach from unprocessed network traffic and developing a framework for feature mapping before applying manifold learning approaches to the mapped data. The exploratory nature of the project resulted a proposal with a project scope that could be considered naïve with respect to both the scope of work required and the contributions of existing literature.

Although exploratory projects require refinement as they progress by design, improperly narrowing the scope of this project meant both substantial time and effort spent in directions that were ultimately excluded from the final project. With the benefit of hindsight, further literature review before the submission of the proposal would have been beneficial in narrowing the scope of the project before it began.

Similarly, the exploratory nature of the project introduced difficulties in properly setting timeframes for project deliverables. A lack of contingency scheduling in my initial plan of work, coupled with some personal obstacles, resulted in a substantial time crunch near the end of the project. It is often difficult to establish timelines for exploratory tasks, and the project fell victim to both blind spots and true unknowns. An understanding of the problem area similar to the Johari window technique [CITATION Luf55 \l 1033] could assist in framing the areas of exploration and developing a specific focus before the project was underway.

The plan of work identified in chapter 1 was laid out chronologically according to a project management methodology, with the goal of developing knowledge in each area to inform the next task. The nature of the project time crunch lead to simultaneous work on several tasks at once, and the opportunity for tasks to inform each other through lessons learned was not fully realized. For example, the test platform was developed in parallel with my understanding of some of the components of an analysis pipeline. As a result, the development of the artefact included superfluous design elements and several superfluous tests that were not utilized in the final product.

This project has been one of the most difficult undertakings of my academic career, but has led to substantial personal development. My personal growth in both research skills and metacognition has been substantial. The experience of developing a project of this scale, without an established framework, in an area where I held very limited initial knowledge, will serve me well in future academic and professional pursuits. Overall, the personal growth I've experienced through this academic challenge has been a highlight of both this MSc program and my overall career.

7.3 Limitations of Research

Although this project identified performance improvements through the use of nonlinear dimensionality reduction, several limitations were identified during the research process. These limitations align with possible further exploration identified in section 7.4.

The results outlined in section 6.4 are the aggregate of multiple tests, over multiple days, containing multiple attacks. Classification is attempted using data that contains one type of attack per dataset, and which contains an attack in its entirety. Although issues of class balancing are addressed in section 3.3 and 3.4, the responsiveness of our classification mechanisms to network captures containing many classes is unknown. Literature regarding manifold approaches notes that the performance of manifolds should improve in multi-class mapping, as the manifold process adapts to multiple neighborhoods with distinct local characteristics. This hypothesis remains untested during our project.

Similarly, the performance of each combination of dimensionality reduction and classification contains some variance over the aggregate results. This variance is generally unexplored, and is identified as an area of future research in section 7.4.

Dimensionality reduction varies in performance as the number of remaining features changes. The limited scope of this project resulted in an equal evaluation of all manifolds (except LTSA) with 20 components extracted from a subset of 77 components. Further performance gains could be realized if the ideal number of extracted features was identified.

The performance of our classification algorithms is compared to a baseline classification with no dimensionality reduction applied. Although similar literature exists for the application of linear dimensionality reduction, it differs in experimental design, data collection, and implementation.

Our results show improvement through the use of nonlinear methods, but the hypothesis that nonlinear approaches perform better than linear approaches, as outlined in section 2.2.2, remains unexplored.

7.4 Further Work and Recommendations

Several of the limitations identified in section 7.3 provide opportunities for further exploration.

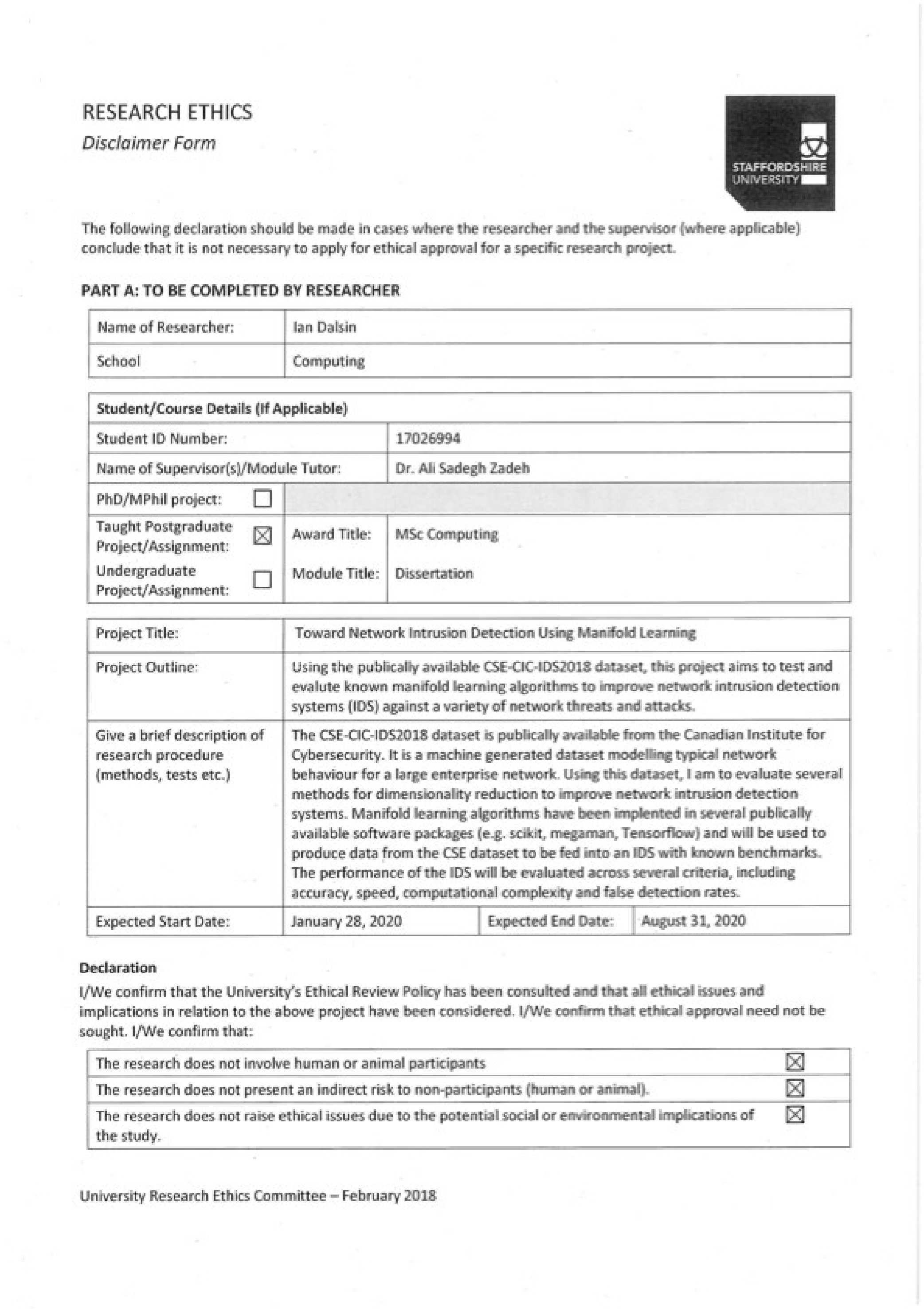
Our experimental results, outlined in section 6.4, are explored as an aggregate of several tests over multiple days containing multiple attacks. The effectiveness of each combination of classification and dimensionality reduction differs for each attack type. Further research to explore the effectiveness of classifiers, dimensionality reduction, and specific attack types may provide support for an IDS containing specific classification pipelines for specific attack types. This exploration could include the evaluation of a classifier for a specific type through novel attacks of a similar type, such as evaluating a DoS classification pipeline with a novel DoS attack. This furthers the implementation of an anomaly-based IDS approach with the use of a true anomaly.

As mentioned in section 7.3, further work including implementations of linear dimensionality reduction and the combination of multiple attack classes within a single evaluation may demonstrate the areas in which nonlinear approaches have specific performance gains when compared to linear approaches. Similarly, an exploration of network attacks where class labelling is binary (malicious vs benign) rather than labelling each attack type may provide an opportunity to demonstrate the use of nonlinear dimensionality reduction on network classification problems with multiple "neighbourhoods." A hypothesis can be drawn from our testing on the infiltration attack that classification performance will be significantly worse in binary labelled data of different shape.

The difference between an intrusion detection system (IDS) and intrusion prevention system (IPS) is briefly explored in chapter 2. The conceptual IDS developed in this paper provides a detection capability after an attack has occurred, but does not explore the potential implementation of a predictive or preventative capability. Further research at the intersection of feature mapping and classification may be able to identify the earliest features of an attack, providing the ability to disrupt attackers at an early step in the "cyber kill chain" identified by Hutchins, et al. (2011). The implementation of a manifold approach in real-time detection introduces several interesting problems regarding the training of both the manifold and classifier on evolving data.

APPENDICES

Appendix A: Ethical Disclaimer



RESEARCH ETHICS

Disclaimer Form

STAFFORDSHIRE UNIVERSITY

The following declaration should be made in cases where the researcher and the supervisor (where applicable) conclude that it is not necessary to apply for ethical approval for a specific research project.

PART A: TO BE COMPLETED BY RESEARCHER

Name of Researcher:	Ian Dalsin
School	Computing

Student/Course Details (If Applicable)			
Student ID Number:		17026994	
Name of Supervisor(s)/Module Tutor:		Dr. Ali Sadegh Zadeh	
PhD/MPhil project: ☐			
Taught Postgraduate Project/Assignment: ☒	Award Title:	MSc Computing	
Undergraduate Project/Assignment: ☐	Module Title:	Dissertation	

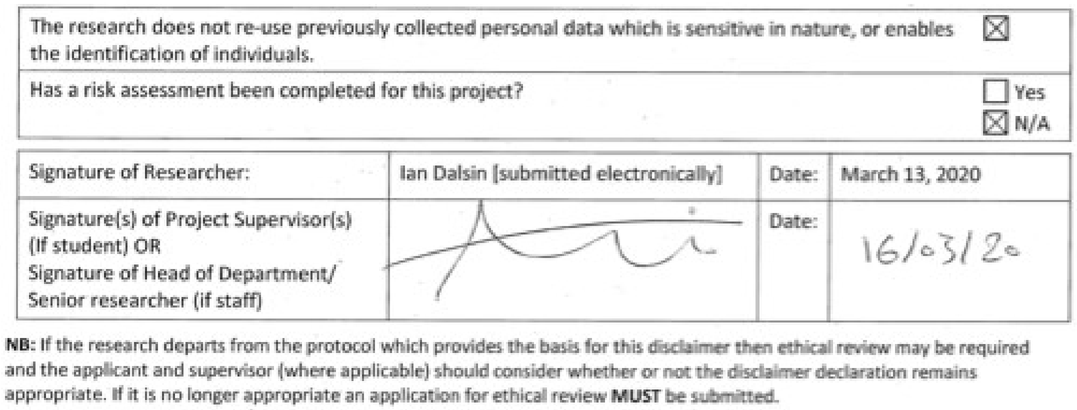
Project Title:	Toward Network Intrusion Detection Using Manifold Learning		
Project Outline:	Using the publically available CSE-CIC-IDS2018 dataset, this project aims to test and evalute known manifold learning algorithms to improve network intrusion detection systems (IDS) against a variety of network threats and attacks.		
Give a brief description of research procedure (methods, tests etc.)	The CSE-CIC-IDS2018 dataset is publically available from the Canadian Institute for Cybersecurity. It is a machine generated dataset modelling typical network behaviour for a large enterprise network. Using this dataset, I am to evaluate several methods for dimensionality reduction to improve network intrusion detection systems. Manifold learning algorithms have been implented in several publically available software packages (e.g. scikit, megaman, Tensorflow) and will be used to produce data from the CSE dataset to be fed into an IDS with known benchmarks. The performance of the IDS will be evaluated across several criteria, including accuracy, speed, computational complexity and false detection rates.		
Expected Start Date:	January 28, 2020	Expected End Date:	August 31, 2020

Declaration

I/We confirm that the University's Ethical Review Policy has been consulted and that all ethical issues and implications in relation to the above project have been considered. I/We confirm that ethical approval need not be sought. I/We confirm that:

The research does not involve human or animal participants	☒
The research does not present an indirect risk to non-participants (human or animal).	☒
The research does not raise ethical issues due to the potential social or environmental implications of the study.	☒

University Research Ethics Committee – February 2018



The research does not re-use previously collected personal data which is sensitive in nature, or enables the identification of individuals.	☒
Has a risk assessment been completed for this project?	☐ Yes ☒ N/A

Signature of Researcher:	Ian Dalsin [submitted electronically]	Date:	March 13, 2020
Signature(s) of Project Supervisor(s) (If student) OR Signature of Head of Department/ Senior researcher (if staff)	[illegible]	Date:	16/03/20

NB: If the research departs from the protocol which provides the basis for this disclaimer then ethical review may be required and the applicant and supervisor (where applicable) should consider whether or not the disclaimer declaration remains appropriate. If it is no longer appropriate an application for ethical review **MUST** be submitted.

Appendix B: Statistical Analysis Code

Python utility to generate descriptive statistics from the provided CSV network flow files of the CSE-CIC-IDS2018 dataset. A downloadable repository of all Python utilities is available at https://github.com/idalsin/dissertation

```
# July 18 - Ian Dalsin
# Python script to analyze a CSV and output descriptive statistics

from sklearn import preprocessing
import pandas as p
import numpy as np
import os

def covar_matrix(data_obj, traffic_type, filename_root):
    print("Generating covariance matrix for: " + traffic_type)
    t = data_obj.cov()
        new_file = traffic_type + "_" + filename_root[:-4]+"_covar.csv"
    t.to_csv(new_file)

def corr_matrix(data_obj, traffic_type, filename_root):
    print("Generating correlation matrix for: " + traffic_type)
    t = data_obj.corr()
        new_file = traffic_type + "_" + filename_root[:-4]+"_corr.csv"
    t.to_csv(new_file)

def column_statistics(data_obj, file_out):
    # writes all stats per column to csv in an array-type format
    cols = list(data_obj)
        list_operations = ['Column Name', 'Mean', 'Mean Abs Deviation', 'Median', 'Min', 'Max', 'Bessel St Dev', 'Unbiased Variance', 'Standard Error of the Mean', '25th Percentile', '50th Percentile', '75th Percentile']
    file_out.write(str(list_operations) + "\n")
    for i in cols:
        #if value is numeric
        #original: if isinstance(data_obj[i][1], str) == False:
        if isinstance(data_obj[i].iloc[0], str) == False:
             temp = str(i) + ", " + str(data_obj[i].mean()) + ", " + str(data_obj[i].mad()) + ", " + str(data_obj[i].median()) + ", " + str(data_obj[i].min()) + ", " + str(data_obj[i].max()) + ", " + str(data_obj[i].std()) + ", " + str(data_obj[i].var()) + ", " + str(data_obj[i].sem()) + ", " + str(np.percentile(data_obj[i],25)) + ", " + str(np.percentile(data_obj[i],50)) + ", "+ str(np.percentile(data_obj[i],75))
```

```
            file_out.write(temp + "\n")
        else:
            continue
    return

def one_file_stats(filename):
    file1 = filename
    print("File to load: " + file1)
     #output file extension set to ODS for Libreoffice, or CSV
for plain text
    out_ext = 'csv'
      outfile = (file1[(file1.rfind("/")+1):-4] + '_outfile.' +
out_ext)
    print("Output to : " + outfile)
    f = open(outfile, 'w')
     f.write("Output for: " + file1[(file1.rfind("/")+1):] + "\
n")

                            set1       =       p.read_csv(file1,
parse_dates=True,dtype={'Label':'string'})
    print("dataset loaded: set1")
       print("This dataset contains the follow traffic object
categories: " + str(set1.Label.unique()))

    x1 = set1
    del x1['Timestamp']
       print("deleting timestamp, can't do any math with it
anyways")

     f.write("Column statistics - all -" + str(x1.shape[0]) + "
rows \n")
    column_statistics(x1, f)
     # if uncommented below, script will create covariance and
correlation matrices
    #covar_matrix(x1, 'All',file1[(file1.rfind("/")+1):-4])
    corr_matrix(x1, 'All',file1[(file1.rfind("/")+1):-4])
    print("Writing column statistics for traffic subtypes...")
    uniques = x1.Label.unique()

    for i in uniques:
        print("Subtype: " + i)
        temp_df = x1.loc[x1['Label'] == i]
        f.write("\n")
             f.write("Traffic subtype: " + str(i) + " - "+
str(temp_df.shape[0]) + "  rows \n")
        column_statistics(temp_df, f)
          # if uncommented below, script will create covariance
and correlation matrices
        #covar_matrix(temp_df,i,file1[(file1.rfind("/")+1):-4])
        corr_matrix(temp_df,i,file1[(file1.rfind("/")+1):-4])
        f.write("\n")

    print("Attack output complete!")

    print("Writing stats for key ports")
      #print for certain port numbers as if they were unique
attacks - ie. statistics for port 80 traffic
    key_ports = [80, 21, 22, 8080]
     #if there is no traffic in these key ports, the dataframe
selected will contain zero rows
     #select only the benign traffic in the key ports, malicious
traffic excluded
    for i in key_ports:
        print("Subtype: port " + str(i))
        temp_df = x1.loc[x1['Label'] == "Benign"]
        temp_df = temp_df.loc[temp_df['Dst Port'] == i]
        if temp_df.shape[0] > 0:
            f.write("\n")
             f.write("Traffic subtype: port " + str(i) + " - " +
str(temp_df.shape[0]) + " rows \n")
            column_statistics(temp_df, f)
            f.write("\n")
        else:
            print("No traffic for port " + str(i))

    print("Key ports output complete")

    f.close()
    return

#the options below represent different 'calls' based on which
lines are commented/uncommented

#call for all files in directory, batch process
# directory = '/home/id/Documents/Thesis/Processed Traffic Data
for ML Algorithms/'
# for filename in os.listdir(directory):
#     if filename[0:5] != ".~loc":
#         one_file_stats(directory + filename)
#         print("File complete!")

# call for one file only
one_file_stats('~/Documents/Thesis/Processed Traffic Data for ML
Algorithms/Thursday-15-02-2018_TrafficForML_CICFlowMeter.csv')
```

Appendix C: Test Platform Code

Test platform python utility and conceptual IDS. A downloadable repository of all Python utilities is available at https://github.com/idalsin/dissertation

```
#Final experimental platform
import pandas as p
import numpy as np
from sklearn import *
import time
import math
import tracemalloc #memory evaluation
import os

# execute - run tracemalloc
def execute(funct, *args):
    tracemalloc.start()
    funct(*args)
    current,peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    f.write(f"Current memory usage is {current / 10 ** 6 } MB;'\
      ' peak was {peak / 10 ** 6} MB\n")

def execute_time(funct, *args):
    start = time.perf_counter()
    funct(*args)
    end = time.perf_counter()
    f.write("execute in " + str((end-start)) + " seconds\n")

def timevar():
    t = time.localtime()
    timestamp = time.strftime('%b-%d-%Y_%H%M', t)
    return str(timestamp)

# data loading
def load_data(filein=None):
    if filein == None:
        print("Is the data to process contained in the same folder as the .py file? (Y/N)")
        choice = input("> ")
        folderpath = None
        if choice.upper() == 'N':
            folderpath = input("Please enter the folder path: ")
        filename = input("Please enter a complete filepath to load: ")
        if folderpath is not None:
            filename = folderpath + filename
    else:
        filename = filein
    set = p.read_csv(filename, parse_dates=True,dtype={'Label':'string'})
    return set #return a dataframe object loaded from a CSV

def first_processing(df):
    #if label_bin exists, drop it
    if 'Label_Bin' in df.columns:
        del df['Label_Bin']
    #if timestamps exists, drop it
    if 'Timestamp' in df.columns:
        del df['Timestamp']
    checkframe(df)
    attacks = df.Label.unique()
    f.write(f"The data contains the following attacks: {attacks}\n")
    f.write("Shuffling data randomly...\n")
    df = df.sample(frac=1).reset_index(drop=True)
    return df

def checkframe(data_obj):
    #check for infinite or NaN values
    found = False
    f.write("checking for NaN values\n")
    if (data_obj.isnull().values.any()) == True:
        #iterate through columns
        cols = list(data_obj)
        for i in cols:
            if (data_obj[i].isnull().values.any()) == True:
                f.write("column " + i + " has " + str(data_obj[i].isnull().sum()) + " NaN values\n")
                found = True
    else:
            f.write("No NaN values
found\n")
    f.write("checking for infinite
values\n")
    cols = list(data_obj)
    for i in cols:
        # if it's a string, ignore
         if isinstance(data_obj[i]
[1], str) == False:
              #not a string, check
for infinite
                                 if
(np.all(np.isfinite(data_obj[i])))
== False:
                 f.write("column "
+ i + " has infinite values\n")
                found = True
         f.write("infinite  check
complete\n")
    if found == True:
                    start_rows   =
data_obj.shape[0]
          f.write("Dropping  rows
with infinite or NaN values\n")
         data_obj.replace([np.inf,
-np.inf], np.nan, inplace=True)
        data_obj.dropna(inplace=Tr
ue)
                      end_rows   =
data_obj.shape[0]
                 f.write(f"Removed
{start_rows-end_rows} rows\n")
    return found

def    decision_tree(dfX,    dfY,
eval=False):
                        clf      =
tree.DecisionTreeClassifier()
    split = 0.4 #test/train split
       x_train,  x_test,  y_train,
y_test                          =
model_selection.train_test_split(d
fX,   dfY,test_size   =   split,
random_state=5)
    del dfX
    del dfY
    clf.fit(x_train, y_train)
    f.write("Tree fit complete\n")
     results = clf.predict(x_test)
#predictions  made,  end  of  run
unless evaluating
    if eval == True:
        #run evaluation criteria
                f.write("Confusion
matrix:\n")
        f.write(str(metrics.confus
ion_matrix(y_test, results))+"\n")
        f.write("F1 score:\n")
        f.write(str(metrics.f1_sco
re(y_test,               results,
average='micro'))+"\n")
          f.write("(F1 score uses
micro  average  for  multi-class
classification)\n")
    return clf

def batch_decision_tree(dfX, dfY):
       f.write("Entering  decision
tree - memory\n")
      execute(decision_tree,  dfX,
dfY)
       f.write("Entering  decision
tree - timed\n")
    execute_time(decision_tree,dfX
, dfY)
     f.write("Evaluating decision
tree metrics\n")
    decision_tree(dfX, dfY, True)

def  LLE(dfX,  dfY,  eval=False,
method='standard'):
    if method == 'modified':
               #required  to  have
n_neighbors > components
             n_neighbors  =  max(5,
component_selection+1) #default is
5, min is as specified
                             lle   =
manifold.LocallyLinearEmbedding(n_
components = component_selection,
n_neighbors     =     n_neighbors,
method=method)
    elif method == 'hessian':
                               v   =
math.ceil(component_selection*(com
ponent_selection+3)/2)+1
          n_neighbors = max(5, v)
#default is 5, min is as specified
                             lle   =
manifold.LocallyLinearEmbedding(n_
components = component_selection,
n_neighbors     =     n_neighbors,
method=method)
    elif method == 'ltsa':
        #maximum components of 5
                             lle   =
manifold.LocallyLinearEmbedding(n_
components = 5, method = method)
    else:
                             lle   =
manifold.LocallyLinearEmbedding(n_
components = component_selection,
method=method)
    print("Fitting manifold")
    lle.fit(dfX.iloc[0:manifold_su
bset])
                        dfX      =
dfX.iloc[manifold_subset+1:]
                        dfY      =
dfY.iloc[manifold_subset+1:]
      print("Transforming data in
batch format...\n")
    start = 0
    end = dfX.shape[0]-1
    init = False
    while start < end:
        if init == False:
            #initialize the array,
append if it has already been
initialized
                         df_out  =
lle.transform(dfX[start:start+batc
h])
            init = True
        else:
                         df_out  =
np.concatenate((df_out,lle.transfo
rm(dfX[start:start+batch])))
        start = start+batch
    dfX = df_out
          print("Batch  transform
complete\n")
    if eval == True:
        #run evaluation criteria
          f.write("LLE evaluation
statistics\n")
          f.write(f"Reconstruction
error   associated   with   this
embedding:
{lle.reconstruction_error_}\n")
             f.write(f"Parameters:
{lle.get_params}\n")
    return [dfX, dfY]

def      batch_LLE(dfX,       dfY,
method='standard'):
    f.write("LLE - evaluation\n")
    LLE(dfX, dfY, True, method)
    f.write("LLE - memory\n")
     execute(LLE, dfX, dfY, False,
method)
    f.write("LLE - timed\n")
      execute_time(LLE, dfX, dfY,
False, method)

def  isomap(dfX,  dfY,  eval  =
False):
       f.write("Creating embedding
vectors...\n")
                        iso      =
manifold.Isomap(n_components=compo
nent_selection)
    iso.fit(dfX.iloc[0:manifold_su
bset])
                        dfX      =
dfX.iloc[manifold_subset+1:]
                        dfY      =
dfY.iloc[manifold_subset+1:]
      print("Transforming data in
batch format...\n")
    start = 0
    end = dfX.shape[0]-1
    init = False
    while start < end:
        if init == False:
            #initialize the array,
append if it has already been
initialized
                         df_out  =
iso.transform(dfX[start:start+batc
h])
            init = True
        else:
                         df_out  =
np.concatenate((df_out,iso.transfo
rm(dfX[start:start+batch])))
        start = start+batch
    dfX = df_out
          print("Batch  transform
complete\n")
    if eval == True:
                     f.write("Isomap
reconstruction   error   is   not
available.\n")
    return [dfX, dfY]

def batch_isomap(dfX, dfY):
    f.write("Isomap - memory\n")
    execute(isomap, dfX, dfY)
    f.write("Isomap - timed\n")
    execute_time(isomap, dfX, dfY)
    f.write("Isomap - stats\n")
    isomap(dfX, dfY, True)

def      spectral(dfX,       dfY,
eval=False):
       f.write("Creating embedding
vectors...\n")
                       spec      =
manifold.SpectralEmbedding(n_compo
nents = component_selection)
                        dfX      =
dfX.iloc[manifold_subset+1:]
```

```
                    dfY      =
dfY.iloc[manifold_subset+1:]
    dfX = spec.fit_transform(dfX)
    dfX = df_out
      f.write("transform complete\
n")
    if eval == True:
                f.write("Spectral
reconstruction    error:    "    +
str(spec.reconstruction_error_)+"\
n")
    return [dfX, dfY]

def batch_spectral(dfX, dfY):
    f.write("Spectral - memory\n")
    execute(spectral, dfX, dfY)
    f.write("Spectral - timed\n")
      execute_time(spectral, dfX,
dfY)
    f.write("Spectral - stats\n")
    spectral(dfX, dfY, True)

def    random_forest(dfX,    dfY,
eval=False):
      split = 0.4 #hold 40% for
classification
       x_train,  x_test,  y_train,
y_test                          =
model_selection.train_test_split(d
fX,     dfY,     test_size=split,
random_state=5)
    del dfX
    del dfY
                    clf      =
ensemble.RandomForestClassifier()
    clf.fit(x_train, y_train)
    f.write("Tree fit complete\n")
    results = clf.predict(x_test)
    if eval == True:
               f.write("Confusion
matrix:\n")
        f.write(str(metrics.confus
ion_matrix(y_test, results))+"\n")
        f.write("f1 score:\n")
        f.write(str(metrics.f1_sco
re(y_test,              results,
average='micro'))+"\n")
    return results

def batch_random_forest(dfX, dfY):
        f.write("Random  forest  -
memor\ny")
      execute(random_forest, dfX,
dfY)
        f.write("Random  forest  -
timed\n")
       execute_time(random_forest,
dfX, dfY)
        f.write("Random  forest  -
evaluated\n")
    random_forest(dfX, dfY, True)
    return

def  k_nearest(dfX,  dfY,  eval  =
False):
      split = 0.4 #hold 40% for
classification
       x_train,  x_test,  y_train,
y_test                          =
model_selection.train_test_split(d
fX,     dfY,     test_size=split,
random_state=5)
    del dfX
    del dfY
                     neigh     =
neighbors.KNeighborsClassifier()
    f.write("Fitting...\n")
    neigh.fit(x_train, y_train)
                   results     =
neigh.predict(x_test)
    if eval == True:
               f.write("Confusion
matrix:\n")
        f.write(str(metrics.confus
ion_matrix(y_test, results))+"\n")
        f.write("F1 score:\n")
        f.write(str(metrics.f1_sco
re(y_test,              results,
average='micro'))+"\n")
    return results

def batch_k_nearest(dfX, dfY):
    print("KNN - memory")
    f.write("KNN - memory\n")
    execute(k_nearest, dfX, dfY)
    print("KNN - time")
    f.write("KNN - time\n")
      execute_time(k_nearest, dfX,
dfY)
    print("KNN - evaluation")
    f.write("KNN - evaluation\n")
    k_nearest(dfX, dfY, True)

def ada(dfX, dfY, eval=False):
      split = 0.4 #hold 40% for
classification
       x_train,  x_test,  y_train,
y_test                          =
model_selection.train_test_split(d
fX,     dfY,     test_size=split,
random_state=5)
    del dfX
    del dfY
                    clf      =
ensemble.AdaBoostClassifier()
    f.write("Fitting...\n")
    clf.fit(x_train, y_train)
    results = clf.predict(x_test)
    if eval == True:
            f.write("Confusion
matrix:\n")
        f.write(str(metrics.confus
ion_matrix(y_test, results))+"\n")
        f.write("F1 score:\n")
        f.write(str(metrics.f1_sco
re(y_test,              results,
average='micro'))+"\n")
    return results

def batch_ada(dfX, dfY):
    f.write("Adaboost - memory\n")
    execute(ada, dfX, dfY)
    f.write("Adaboost - time\n")
    execute_time(ada, dfX, dfY)
          f.write("Adaboost   -
evaluation\n")
    ada(dfX, dfY, True)

def range_scale(dfX):
    #minmax scale to reduce unit
scale and place all values between
0 and 1
                    scaler     =
preprocessing.MinMaxScaler()
    scaler.fit(dfX)
    scaler.transform(dfX)
    return dfX

def chooser():
    choices = []
    print("Please select from the
following options:")
    print("First, binary (1) or
multi-label (2) classification?")
    choices.append(int(input(">
")))
    print("Second,   select
dimensionality reduction:")
    print("1 - None")
    print("2  -  Locally  Linear
Embedding")
    print("3  -  Locally  Linear
Embedding (Measured)")
    print("4 - IsoMap")
    print("5 - IsoMap (Measured)")
    print("6 - Modified LLE")
    print("7  -  Modified  LLE
(Measured)")
    print("8 - LTSA")
    print("9 - LTSA (Measured)")
    print("10 - Hessian LLE")
    print("11  -  Hessian  LLE
(Measured)")
    print("12 - Spectral Embedding
(Laplacian Eigenmaps)")
    print("13 - Spectral Embedding
(Measured)")

    choices.append(int(input(">
")))
    print("Third, classifier:")
    print("1 - Decision Tree")
    print("2  -  Decision  Tree
(Measured)")
    print("3 - Random Forest")
    print("4  -  Random  Forest
(Measured)")
    print("5   -   k-Nearest
Neighbors")
    print("6 - k-Nearest Neighbors
(Measured)")
    print("7 - Adaboost")
    print("8   -   Adaboost
(Measured)")
    print("9 - None")
    choices.append(int(input(">
")))
    print(f"Choices   are:
{choices}")

    print("Choose  data  scaling
method")
    print("1 - None")
    print("2 - Range Scale")
    choices.append(int(input(">
")))
    return choices

def runner(choices, df):
    #preprocessing
    if choices[0] == 1:
            f.write("Adjusting
classification     labels     to
Benign/Malicious only.\n")
                    uniques   =
df.Label.unique()
        for i in uniques:
            if i != 'Benign':
                    df['Label'] =
df['Label'].replace([i],'Malicious
')
    df = first_processing(df)
    dfY = df['Label']
    dfX = df[df.columns[0:77]]
    #scaling, if chosen
    if choices[3] == 2:
        dfX = range_scale(dfX)
                  f.write("Scaling
complete!\n")
        print("Scaling complete")

    #dimensionality reduction
    if choices[1] == 1:
                #no  dimensionality
reduction
        f.write("No dimensionality
reduction chosen.\n")
          print("No dimensionality
reduction chosen")
    elif choices[1] == 2:
          f.write("Running locally
linear embedding\n")
            print("Running  locally
linear embedding\n")
        var = LLE(dfX, dfY)
        dfX = var[0]
        dfY = var[1]
        f.write("LLE complete\n")
        print("LLE complete\n")
    elif choices[1] == 3:
          f.write("Running batched
LLE 5 times\n")
        for i in range(3):
            batch_LLE(dfX, dfY)
        f.write("LLE complete\n")
        print("LLE complete")
    elif choices[1] == 4:
          f.write("Running Isomap\
n")
        var = isomap(dfX, dfY)
        dfX = var[0]
        dfY = var[1]
         f.write("Isomap complete\
n")
        print("Isomap complete")
    elif choices[1] == 5:
           f.write("Running Isomap
(Measured) 5 times\n")
        for i in range(3):
            batch_isomap(dfX, dfY)
         f.write("Isomap complete\
n")
        print("Isomap complete")
    elif choices[1] == 6:
        f.write("Modified LLE\n")
        print("Modified LLE")
        var = LLE(dfX, dfY, False,
'modified')
        dfX = var[0]
        dfY = var[1]
             f.write("Modified  LLE
complete\n")
               print("Modified  LLE
complete")
    elif choices[1] == 7:
             f.write("Modified  LLE
(Measured)\n")
               print("Modified  LLE
(Measured)")
        for i in range(3):
               batch_LLE(dfX, dfY,
'modified')
             f.write("Modified  LLE
complete\n")
               print("Modified  LLE
complete")
    elif choices[1] == 8:
        f.write("LTSA\n")
        print("LTSA")
             var  =  LLE(dfX,  dfY,
'ltsa')
        dfX = var[0]
        dfY = var[1]
        f.write("LTSA complete\n")
        print("LTSA complete")
    elif choices[1] == 9:
         f.write("LTSA (Measured)\
n")
        print("LTSA (Measured)")
        for i in range(3):
               batch_LLE(dfX, dfY,
'ltsa')
        f.write("LTSA complete\n")
        print("LTSA complete")
    elif choices[1] == 10:
        f.write("Hessian LLE\n")
        print("Hessian LLE")
        var = LLE(dfX, dfY, False,
'hessian')
        dfX = var[0]
        dfY = var[1]
              f.write("Hessian  LLE
complete\n")
                print("Hessian  LLE
complete")
    elif choices[1] == 11:
              f.write("Hessian  LLE
(Measured)\n")
                print("Hessian  LLE
(Measured)")
        for i in range(3):
               batch_LLE(dfX, dfY,
'hessian')
              f.write("Hessian  LLE
complete\n")
```

```
            print("Hessian LLE
complete")
    elif choices[1] == 12:
            f.write("Spectral
Embedding (Laplacian Eigenmaps)\
n")
        spectral(dfX, dfY)
            f.write("Spectral
embedding complete\n")
    elif choices[1] == 13:
            f.write("Spectral
Embedding (Laplacian Eigenmaps -
Measured)\n")
        for i in range(3):
            batch_spectral(dfX,
dfY)
            f.write("Spectral
embedding complete\n")

    #classifier
    if choices[2] == 1:
        #run decision tree batch
        f.write("Running decision
tree\n")
        print("Running decision
tree")
            f.write(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
            print(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
        decision_tree(dfX, dfY)
    elif choices[2] == 2:
        #one batch of decision
tree
        f.write("Running decision
tree (measured)\n")
        print("Running decision
tree (measured)")
            f.write(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
            print(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
        for i in range(3):
            batch_decision_tree(df
X, dfY)
    elif choices[2] == 3:
        f.write("Running random
forest\n")
            print("Running random
forest")
            f.write(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
            print(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
        random_forest(dfX, dfY)
        f.write("Complete!\n")
        print("Complete!")
    elif choices[2] == 4:
        f.write("Running random
forest (measured)\n")
            print("Running random
forest (measured)")
            f.write(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
            print(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
        for i in range(2):
            batch_random_forest(df
X, dfY)
        f.write("Complete!\n")
        print("Complete!")
    elif choices[2] == 5:
        f.write("Running KNN\n")
        print("Running KNN")
        k_nearest(dfX, dfY)
        f.write("Complete!\n")
        print("Complete!")
    elif choices[2] == 6:
            f.write("Running KNN
(measured)\n")
            print("Running KNN
(measured)")
            f.write(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
            print(f"X shape is
{dfX.shape} and Y shape is
{dfY.shape}\n")
        for i in range(3):
            batch_k_nearest(dfX,
dfY)
        f.write("Complete!\n")
        print("Complete!")
    elif choices[2] == 7:
        f.write("Running adaboost\
n")
        print("Running adaboost")
        ada(dfX, dfY)
        f.write("Complete!\n")
        print("Complete")
    elif choices[2] == 8:
        f.write("Running adaboost (measured)\n")
        print("Running adaboost (measured)")
        f.write(f"X shape is {dfX.shape} and Y shape is {dfY.shape}\n")
        print(f"X shape is {dfX.shape} and Y shape is {dfY.shape}\n")
        for i in range(3):
            batch_ada(dfX, dfY)
        f.write("Complete!\n")
        print("Complete")
    else:
        f.write("No classifier run.\n")
        print("No classifier chosen")
    f.write("Script complete!\n")
    print("Script complete!")

#global variables
component_selection = 20
manifold_subset = 2000
batch = 2000 #rows to transform at a time in manifolds that are batched

#take user input
choices = chooser()
# #multiclass, LLE with stats, decision tree, no range
df = load_data() #make sure to run this after every "choices" block
f = open('output.txt', 'w')
#run on choices and input dataframe
runner(choices, df)
f.close()
```

Appendix D: Output Conversion Utility

Python utility to convert output text files into a readable CSV for statistical analysis. A downloadable repository of all Python utilities is available at https://github.com/idalsin/dissertation

```
#parse my results txt files into a
CSV
import os #for later directory
parsing

def write_array(arr, extra=None,
name=None):
    for i in arr:
        if extra == None:
              outfile.write(str(i)
+'\n')
        else:
             outfile.write(extra +
", " + name + ", " + str(i) + '\
n')
    return

def num_only(arr):
    new_arr = []
    for x in arr:
         x = ''.join(c for c in x
if c.isdigit() or c == '.')
        new_arr.append(x)
    return new_arr

def parse_csv(filename):
                      filein     =
open(directory+filename, 'r')
    lines = filein.readlines()
    count = len(lines)-1
    temp = filename.find("_")
    temp = max(temp, 20)

     outfile.write("Sequence:  " +
str(sequence)+"\n")
        outfile.write("Date:  "  +
filename[len(sequence):temp]+"\n")
    outfile.write("Filename, Date,
Labelling,          Dimensionality
Reduction,  Classifier,  Scaling,
Variable, Result \n")
    memory = []
    attacks = []
    time = []
    rows_dropped = []
    confusion_matrix = []
    f1_score = []
    recon_error = []
    for x in range(count):
        temp_line = lines[x]
        #memory
          if temp_line.find("peak
was ") > 0:
            memory.append(temp_lin
e[temp_line.find("peak   was   ")
+len("peak was "):])
        #time
                                if
temp_line.find("seconds")>0:
            time.append(temp_line)
        #attacks
                                if
temp_line.find("Benign")>0:
            attacks.append(temp_li
ne)
        #rows dropped, row count
                                if
(temp_line.find("emoved")>0):
            rows_dropped.append(te
mp_line)
        #confusion matrix
                                if
temp_line.find("onfusion
matrix:")>0:
               mat = [lines[x+1],
lines[x+2]]
            confusion_matrix.appen
d(mat)
        #f1 score
           if temp_line.find("1
score:")>0:
            f1_score.append(lines[
x+1])
        #reconstruction error
                                if
temp_line.find("associated    with
this embedding:")>0:
            recon_error.append(tem
p_line[temp_line.find("associated
with       this       embedding:")
+len("associated     with     this
embedding:")+1:])



    #clean all my arrays, removing
units and such
    memory = num_only(memory)
    time = num_only(time)
```

```
                rows_dropped =
num_only(rows_dropped)
    f1_score = num_only(f1_score)
                recon_error =
num_only(recon_error)
    #confusion matrix is a bit of
an edge case
    if len(confusion_matrix)>0:
                    for i in
range(len(confusion_matrix[0])):
                #returns rows of
matrix
            confusion_matrix[0][i]
= ''.join(c for c in
confusion_matrix[0][i] if
c.isdigit() or c == ' ')
                    for i in
range(len(confusion_matrix[0])):
            confusion_matrix[0][i]
= confusion_matrix[0][i].split()
            confusion_matrix =
confusion_matrix[0]
                    for i in
range(len(confusion_matrix)):
                    for x in
range(len(confusion_matrix[i])):
                confusion_matrix[i
][x] = int(confusion_matrix[i][x])

    #write row
     #first, turn sequence into a
string of choices
     #ie. isomap to decision tree,
scaled, multiclass
    if len(sequence) == 5:
        #ex 01042
                    bin_mult =
int(sequence[0])
                    dimred =
int(sequence[1:3])
                    classifier =
int(sequence[3])
        scaling = int(sequence[4])
    else:
                    bin_mult =
int(sequence[0])
                    classifier =
int(sequence[2])
        dimred = int(sequence[1])
        scaling = int(sequence[3])
    if bin_mult == 1:
         options_string = "Binary,
"
    else:
                options_string =
"Multiclass, "
```

```
    if scaling == 1:
            scaling_string = " no
scaling"
    else:
         scaling_string = " range
scaling"
         # print("Debug: " +
str(sequence) + ", len " +
str(len(sequence)))
              # print("Bin:" +
str(bin_mult))
         # print("DimRed: " +
str(dimred))
         # print("class: " +
str(classifier))
        # print("scaling: " +
str(scaling))
     dimred_opt = ['None', 'LLE',
'LLE', 'Isomap', 'Isomap', 'MLLE',
'MLLE', 'LTSA', 'LTSA', 'HLLE',
'HLLE', 'Spectral', 'Spectral']
                dimred_str =
dimred_opt[dimred-1]

      classifier_opt = ['Decision
Tree', 'Decision Tree', 'Random
Forest', 'Random Forest', 'KNN',
'KNN', 'Adaboost', 'Adaboost',
'None']
               classifier_str =
classifier_opt[classifier-1]
    #be sure to substract one for
indexing
               options_string =
options_string + dimred_str + ', '
+ classifier_str + ',' +
scaling_string
    print(options_string)
    print(str(memory))
    print(str(time))
    print(str(attacks))
    print(str(rows_dropped))
    print(str(confusion_matrix))
    print(str(f1_score))
    print(str(recon_error))
     out_str = filename + ', ' +
filename[len(sequence):temp] + ',
'+ options_string
    #write memory
     write_array(memory, out_str,
"Memory")
    #write time
       write_array(time, out_str,
"Time")
    #write attacks
    if len(attacks) > 0:
        outfile.write(out_str + ", 
Attacks, " + attacks[0])
    else:
        outfile.write(out_str + ", 
Attacks, " + "no attacks?")
    #write rows dropped
         write_array(rows_dropped, 
out_str, "Rows Dropped")
    #write confusion matrix
    for i in confusion_matrix:
                           var   = 
str(i).replace("[","").replace("]"
,"")
        outfile.write(out_str + ", 
Confusion Matrix, " + str(var)+'\
n')
    #write f1 score
    write_array(f1_score, out_str, 
"F1 Score")
    #write recon error
          write_array(recon_error, 
out_str, "Reconstruction Error")
    outfile.write('\n')
```

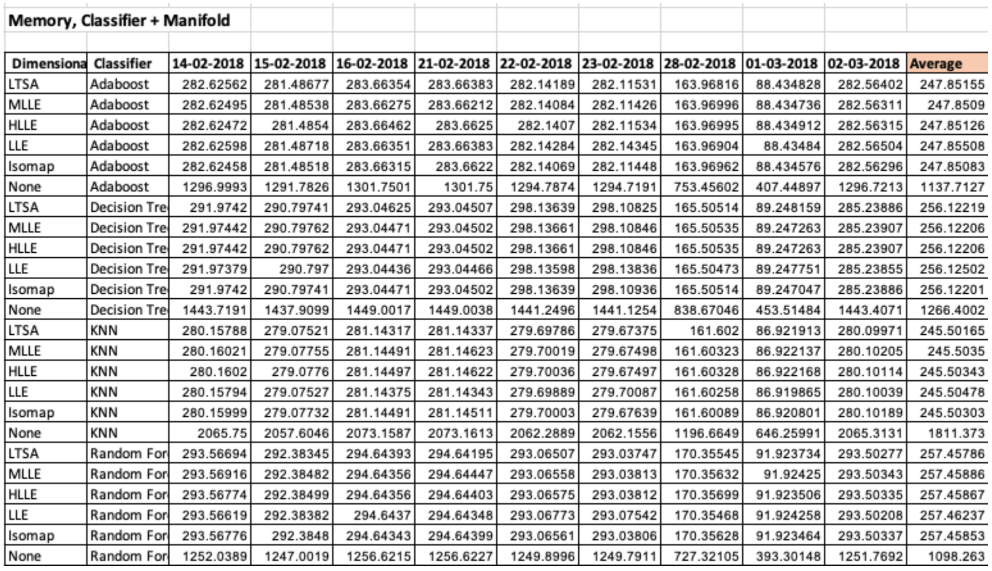
```
    filein.close()
    print('\n')


directory                        = 
'/Users/iandalsin/Nextcloud/scikit
-backup-nc/Test Results/'
outfile                          = 
open('results_parsed.csv', 'w')
for          filename           in 
os.listdir(directory):
    print(filename)
    if filename[0].isdigit():
        if filename[4].isdigit():
                      sequence  = 
filename[0:5]
        else:
                      sequence  = 
filename[0:4]
        parse_csv(filename)


outfile.close()
```

Appendix E: Output Text File Example

Raw output: 2391Thursday-22-02-2018_TrafficForML_CICFlowMeter.csv

```
checking for NaN values
column Flow Byts/s has 3569 NaN values
checking for infinite values
column Flow Byts/s has infinite values
column Flow Pkts/s has infinite values
infinite check complete
Dropping rows with infinite or NaN values
Removed 5610 rows
The data contains the following attacks: <StringArray>
['Benign', 'Brute Force -Web', 'Brute Force -XSS', 'SQL Injection']
Length: 4, dtype: string
Shuffling data randomly...
Running batched LLE 5 times
LLE - evaluation
Manifold fit, transforming...
LLE evaluation statistics
Reconstruction error associated with this embedding: 3.995204755836443e-14
Parameters: <bound method BaseEstimator.get_params of LocallyLinearEmbedding(n_components=20)>
LLE - memory
Manifold fit, transforming...
Current memory usage is 0.010231 MB; peak was 3933.67056 MB
LLE - timed
Manifold fit, transforming...
execute in 90.08341617099995 seconds
LLE - evaluation
Manifold fit, transforming...
LLE evaluation statistics
Reconstruction error associated with this embedding: 3.995204755836443e-14
Parameters: <bound method BaseEstimator.get_params of LocallyLinearEmbedding(n_components=20)>
LLE - memory
Manifold fit, transforming...
Current memory usage is 0.015084 MB; peak was 3933.675413 MB
LLE - timed
Manifold fit, transforming...
execute in 90.50949174800007 seconds
LLE - evaluation
Manifold fit, transforming...
LLE evaluation statistics
Reconstruction error associated with this embedding: 3.995204755836443e-14
Parameters: <bound method BaseEstimator.get_params of LocallyLinearEmbedding(n_components=20)>
LLE - memory
Manifold fit, transforming...
Current memory usage is 0.015406 MB; peak was 3933.675735 MB
LLE - timed
Manifold fit, transforming...
execute in 90.69816065600025 seconds
LLE complete
No classifier run.
Script complete!
```

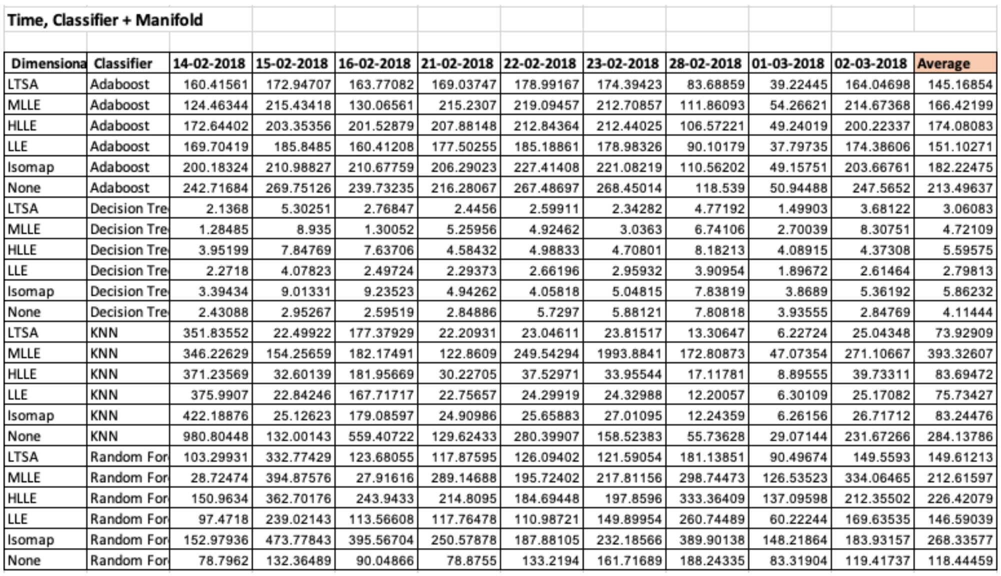
Appendix F: Additional Output Statistics



Memory, Classifier + Manifold

Dimensiona	Classifier	14-02-2018	15-02-2018	16-02-2018	21-02-2018	22-02-2018	23-02-2018	28-02-2018	01-03-2018	02-03-2018	Average
LTSA	Adaboost	282.62562	281.48677	283.66354	283.66383	282.14189	282.11531	163.96816	88.434828	282.56402	247.85155
MLLE	Adaboost	282.62495	281.48538	283.66275	283.66212	282.14084	282.11426	163.96996	88.434736	282.56311	247.8509
HLLE	Adaboost	282.62472	281.4854	283.66462	283.6625	282.1407	282.11534	163.96995	88.434912	282.56315	247.85126
LLE	Adaboost	282.62598	281.48718	283.66351	283.66383	282.14284	282.14345	163.96904	88.43484	282.56504	247.85508
Isomap	Adaboost	282.62458	281.48518	283.66315	283.6622	282.14069	282.11448	163.96962	88.434576	282.56296	247.85083
None	Adaboost	1296.9993	1291.7826	1301.7501	1301.75	1294.7874	1294.7191	753.45602	407.44897	1296.7213	1137.7127
LTSA	Decision Tre	291.9742	290.79741	293.04625	293.04507	298.13639	298.10825	165.50514	89.248159	285.23886	256.12219
MLLE	Decision Tre	291.97442	290.79762	293.04471	293.04502	298.13661	298.10846	165.50535	89.247263	285.23907	256.12206
HLLE	Decision Tre	291.97442	290.79762	293.04471	293.04502	298.13661	298.10846	165.50535	89.247263	285.23907	256.12206
LLE	Decision Tre	291.97379	290.797	293.04436	293.04466	298.13598	298.13836	165.50473	89.247751	285.23855	256.12502
Isomap	Decision Tre	291.9742	290.79741	293.04471	293.04502	298.13639	298.10936	165.50514	89.247047	285.23886	256.12201
None	Decision Tre	1443.7191	1437.9099	1449.0017	1449.0038	1441.2496	1441.1254	838.67046	453.51484	1443.4071	1266.4002
LTSA	KNN	280.15788	279.07521	281.14317	281.14337	279.69786	279.67375	161.602	86.921913	280.09971	245.50165
MLLE	KNN	280.16021	279.07755	281.14491	281.14623	279.70019	279.67498	161.60323	86.922137	280.10205	245.5035
HLLE	KNN	280.1602	279.0776	281.14497	281.14622	279.70036	279.67497	161.60328	86.922168	280.10114	245.50343
LLE	KNN	280.15794	279.07527	281.14375	281.14343	279.69889	279.70087	161.60258	86.919865	280.10039	245.50478
Isomap	KNN	280.15999	279.07732	281.14491	281.14511	279.70003	279.67639	161.60089	86.920801	280.10189	245.50303
None	KNN	2065.75	2057.6046	2073.1587	2073.1613	2062.2889	2062.1556	1196.6649	646.25991	2065.3131	1811.373
LTSA	Random For	293.56694	292.38345	294.64393	294.64195	293.06507	293.03747	170.35545	91.923734	293.50277	257.45786
MLLE	Random For	293.56916	292.38482	294.64356	294.64447	293.06558	293.03813	170.35632	91.92425	293.50343	257.45886
HLLE	Random For	293.56774	292.38499	294.64356	294.64403	293.06575	293.03812	170.35699	91.923506	293.50335	257.45867
LLE	Random For	293.56619	292.38382	294.6437	294.64348	293.06773	293.07542	170.35468	91.924258	293.50208	257.46237
Isomap	Random For	293.56776	292.3848	294.64343	294.64399	293.06561	293.03806	170.35628	91.923464	293.50337	257.45853
None	Random For	1252.0389	1247.0019	1256.6215	1256.6227	1249.8996	1249.7911	727.32105	393.30148	1251.7692	1098.263

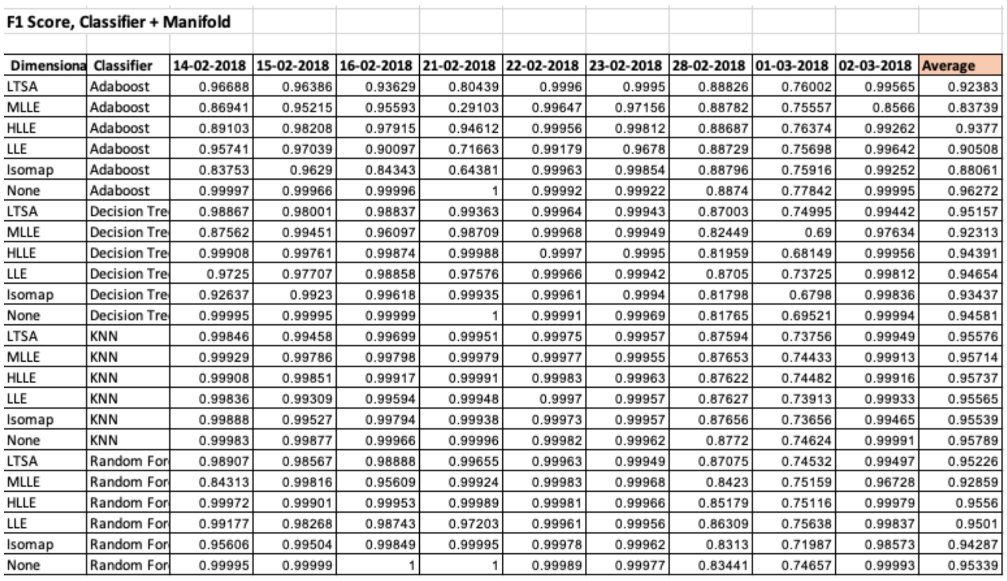
Table 8.1: Memory usage by classifier and manifold



Time, Classifier + Manifold

Dimensiona	Classifier	14-02-2018	15-02-2018	16-02-2018	21-02-2018	22-02-2018	23-02-2018	28-02-2018	01-03-2018	02-03-2018	Average
LTSA	Adaboost	160.41561	172.94707	163.77082	169.03747	178.99167	174.39423	83.68859	39.22445	164.04698	145.16854
MLLE	Adaboost	124.46344	215.43418	130.06561	215.2307	219.09457	212.70857	111.86093	54.26621	214.67368	166.42199
HLLE	Adaboost	172.64402	203.35356	201.52879	207.88148	212.84364	212.44025	106.57221	49.24019	200.22337	174.08083
LLE	Adaboost	169.70419	185.8485	160.41208	177.50255	185.18861	178.98326	90.10179	37.79735	174.38606	151.10271
Isomap	Adaboost	200.18324	210.98827	210.67759	206.29023	227.41408	221.08219	110.56202	49.15751	203.66761	182.22475
None	Adaboost	242.71684	269.75126	239.73235	216.28067	267.48697	268.45014	118.539	50.94488	247.5652	213.49637
LTSA	Decision Tre	2.1368	5.30251	2.76847	2.4456	2.59911	2.34282	4.77192	1.49903	3.68122	3.06083
MLLE	Decision Tre	1.28485	8.935	1.30052	5.25956	4.92462	3.0363	6.74106	2.70039	8.30751	4.72109
HLLE	Decision Tre	3.95199	7.84769	7.63706	4.58432	4.98833	4.70801	8.18213	4.08915	4.37308	5.59575
LLE	Decision Tre	2.2718	4.07823	2.49724	2.29373	2.66196	2.95932	3.90954	1.89672	2.61464	2.79813
Isomap	Decision Tre	3.39434	9.01331	9.23523	4.94262	4.05818	5.04815	7.83819	3.8689	5.36192	5.86232
None	Decision Tre	2.43088	2.95267	2.59519	2.84886	5.7297	5.88121	7.80818	3.93555	2.84769	4.11444
LTSA	KNN	351.83552	22.49922	177.37929	22.20931	23.04611	23.81517	13.30647	6.22724	25.04348	73.92909
MLLE	KNN	346.22629	154.25659	182.17491	122.8609	249.54294	1993.8841	172.80873	47.07354	271.10667	393.32607
HLLE	KNN	371.23569	32.60139	181.95669	30.22705	37.52971	33.95544	17.11781	8.89555	39.73311	83.69472
LLE	KNN	375.9907	22.84246	167.71717	22.75657	24.29919	24.32988	12.20057	6.30109	25.17082	75.73427
Isomap	KNN	422.18876	25.12623	179.08597	24.90986	25.65883	27.01095	12.24359	6.26156	26.71712	83.24476
None	KNN	980.80448	132.00143	559.40722	129.62433	280.39907	158.52383	55.73628	29.07144	231.67266	284.13786
LTSA	Random For	103.29931	332.77429	123.68055	117.87595	126.09402	121.59054	181.13851	90.49674	149.5593	149.61213
MLLE	Random For	28.72474	394.87576	27.91616	289.14688	195.72402	217.81156	298.74473	126.53523	334.06465	212.61597
HLLE	Random For	150.9634	362.70176	243.9433	214.8095	184.69448	197.8596	333.36409	137.09598	212.35502	226.42079
LLE	Random For	97.4718	239.02143	113.56608	117.76478	110.98721	149.89954	260.74489	60.22244	169.63535	146.59039
Isomap	Random For	152.97936	473.77843	395.56704	250.57878	187.88105	232.18566	389.90138	148.21864	183.93157	268.33577
None	Random For	78.7962	132.36489	90.04866	78.8755	133.2194	161.71689	188.24335	83.31904	119.41737	118.44459

Table 8.2: Run time by classifier and manifold



F1 Score, Classifier + Manifold

Dimensiona	Classifier	14-02-2018	15-02-2018	16-02-2018	21-02-2018	22-02-2018	23-02-2018	28-02-2018	01-03-2018	02-03-2018	Average
LTSA	Adaboost	0.96688	0.96386	0.93629	0.80439	0.9996	0.9995	0.88826	0.76002	0.99565	0.92383
MLLE	Adaboost	0.86941	0.95215	0.95593	0.29103	0.99647	0.97156	0.88782	0.75557	0.8566	0.83739
HLLE	Adaboost	0.89103	0.98208	0.97915	0.94612	0.99956	0.99812	0.88687	0.76374	0.99262	0.9377
LLE	Adaboost	0.95741	0.97039	0.90097	0.71663	0.99179	0.9678	0.88729	0.75698	0.99642	0.90508
Isomap	Adaboost	0.83753	0.9629	0.84343	0.64381	0.99963	0.99854	0.88796	0.75916	0.99252	0.88061
None	Adaboost	0.99997	0.99966	0.99996	1	0.99992	0.99922	0.8874	0.77842	0.99995	0.96272
LTSA	Decision Tre	0.98867	0.98001	0.98837	0.99363	0.99964	0.99943	0.87003	0.74995	0.99442	0.95157
MLLE	Decision Tre	0.87562	0.99451	0.96097	0.98709	0.99968	0.99949	0.82449	0.69	0.97634	0.92313
HLLE	Decision Tre	0.99908	0.99761	0.99874	0.99988	0.9997	0.9995	0.81959	0.68149	0.99956	0.94391
LLE	Decision Tre	0.9725	0.97707	0.98858	0.97576	0.99966	0.99942	0.8705	0.73725	0.99812	0.94654
Isomap	Decision Tre	0.92637	0.9923	0.99618	0.99935	0.99961	0.9994	0.81798	0.6798	0.99836	0.93437
None	Decision Tre	0.99995	0.99995	0.99999	1	0.99991	0.99969	0.81765	0.69521	0.99994	0.94581
LTSA	KNN	0.99846	0.99458	0.99699	0.99951	0.99975	0.99957	0.87594	0.73756	0.99949	0.95576
MLLE	KNN	0.99929	0.99786	0.99798	0.99979	0.99977	0.99955	0.87653	0.74433	0.99913	0.95714
HLLE	KNN	0.99908	0.99851	0.99917	0.99991	0.99983	0.99963	0.87622	0.74482	0.99916	0.95737
LLE	KNN	0.99836	0.99309	0.99594	0.99948	0.9997	0.99957	0.87627	0.73913	0.99933	0.95565
Isomap	KNN	0.99888	0.99527	0.99794	0.99938	0.99973	0.99957	0.87656	0.73656	0.99465	0.95539
None	KNN	0.99983	0.99877	0.99966	0.99996	0.99982	0.99962	0.8772	0.74624	0.99991	0.95789
LTSA	Random For	0.98907	0.98567	0.98888	0.99655	0.99963	0.99949	0.87075	0.74532	0.99497	0.95226
MLLE	Random For	0.84313	0.99816	0.95609	0.99924	0.99983	0.99968	0.8423	0.75159	0.96728	0.92859
HLLE	Random For	0.99972	0.99901	0.99953	0.99989	0.99981	0.99966	0.85179	0.75116	0.99979	0.9556
LLE	Random For	0.99177	0.98268	0.98743	0.97203	0.99961	0.99956	0.86309	0.75638	0.99837	0.9501
Isomap	Random For	0.95606	0.99504	0.99849	0.99995	0.99978	0.99962	0.8313	0.71987	0.98573	0.94287
None	Random For	0.99995	0.99999	1	1	0.99989	0.99977	0.83441	0.74657	0.99993	0.95339

Table 1.3: F1 score by classifier and manifold

BIBLIOGRAPHY

1. Abdulhammed, R. et al., 2019. Features Dimensionality Reduction Approaches for Machine Learning Based Network Intrusion Detection. *Electronics,* Volume 8, p. 322.
2. Aggarwal, C. C., Hinneburg, A. & Keim, D. A., 2001. *On the Surprising Behaviour of Distance Metrics in High Dimensional Space.* Berlin, International Conference on Database Theory (Springer).
3. Ahmadi, S. S., Rashad, S. & Elgazzar, H., 2019. *Efficient Feature Selection for Intrusion Detection Systems.* s.l., 2019 IEEE 10th Annual Ubiquitous Computing, Electroncsi & Mobile Communication Conference.
4. Ahmed, M., Mahmood, A. N. & Hu, J., 2016. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications,* Volume 60, pp. 19-31.
5. Anon., n.d. [Online].
6. Babu, A., 2020. *HULK Python Script.* [Online]
   Available at:
   https://gist.github.com/rambabusaravanan/4225ab77d742eb0991c8e30aec4b7317
7. Banerjee, A., Chandola, V. & Kumar, V., 2007. Anomaly Detection: A Survey. *ACM Computing Surveys,* Issue August.
8. Belkin, M. & Niyogi, P., 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation,* 16(6), pp. 1373-1396.
9. Bellman, R., 1961. *Adaptive Control Processes: A Guided Tour.* 2015 e-book edition ed. Princeton: Princeton University Press.
10. Biermann, E., Cloete, E. & Venter, L., 2001. A comparison of intrusion detection systems. *Computers & Security,* Issue 20, pp. 676-683.
11. Boyar, J. & Peralta, R., 2013. *Four measures of nonlinearity.* Berlin, International Conference on Algorithms and Complexity.
12. Budgen, D., 2003. *Software Design.* 2nd Edition ed. Harlow: Pearson.
13. Cisco Networks, 2020. *Cisco Annual Internet Report (2018–2023) White Paper.* [Online]

    Available at: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
    [Accessed 9 June 2020].
14. Cloudflare, 2019. *DNS over TLS.* [Online]
    Available at: https://www.cloudflare.com/learning/dns/dns-over-tls/
    [Accessed 9 6 2020].
15. Darktrace, 2019. *Machine Learning in the Age of Cyber AI.* [Online]
    Available at: https://www.darktrace.com/en/resources/wp-machine-learning.pdf?utm_source=darktrace&utm_medium=technology
    [Accessed 20 4 2020].
16. Devijver, P. A. & Kittler, J., 1982. *Pattern Recongition: A Statistical Approach.* London: Prentice-Hall.
17. Donoho, D. L. & Grimes, C., 2003. *Hessian Eigenmaps: Locally Linear Embedding Techniques for High-Dimensional Data.* s.l., Proceedings of the National Academy of Sciences of the United States of America.
18. Durrani, M., 2019. *Port Spoofing: The Hidden Danger to Your Network.* [Online]
    Available at: https://blog.gigamon.com/2019/06/04/port-spoofing-the-hidden-danger-to-your-network/
    [Accessed 7 8 2020].
19. Eco, U., 2015. *How to Write a Thesis.* s.l.:MIT Press.

20. Fefferman, C., Mitter, S. & Narayanan, H., 2016. Testing the Manifold Hypothesis. *Journal of the American Mathematical Society,* 29(4), pp. 983-1049.
21. Garrett, D., Peterson, D. A., Anderson, C. W. & Thaut, H. M., 2003. Comparison of Linear, Nonlinear and Feature Selection Methods for EEG Signal Classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* 11(2), pp. 141-144.
22. Gigamon, 2020. *Threat Detection Methodologies.* [Online]
Available at: https://www.gigamon.com/content/dam/resource-library/english/solution-overview---technology-overview/so-threat-detection-methodologies.pdf
[Accessed 19 6 2020].
23. Gnanaprasanambikai, L. & Munusamy, N., 2018. Data Pre-Processing and Classification for Traffic Anomaly Intrusion Detection Using NSLKDD Dataset. *Cybernetics and Information Technologies,* 18(3).
24. Hawkins, D., 1980. *Identification of outliers (monographs on statistics and applied probability).* 1st ed. Netherlands: Springer.
25. Hutchins, E. M., Cloppert, M. J. & Amin, R. M., 2011. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare and Security Research,* 1(1), p. 80.
26. IANA, 2020. *Service Name and Transport Protocol Port Number Registry.* [Online]
Available at: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt
[Accessed 6 6 2020].
27. Internet Engineering Task Force, 2011. *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry.* [Online]
Available at: https://tools.ietf.org/html/rfc6335#section-6
[Accessed 8 6 2020].
28. Kinsell, J., RSnake, Gonzalez, H. & Lee, R. E., 2015. *SlowLoris HTTP DoS.* [Online]
Available at: https://web.archive.org/web/20150426090206/http://ha.ckers.org/slowloris
[Accessed 30 8 2020].
29. Lashkari, A. H., Draper-Gil, G., Mamum, M. S. I. & Ghorbani, A. A., 2016. *Characterization of Encrypted and VPN Traffic Using Time-Related Features.* Italy, Proceedings of the 2nd International Conference on Information Systems Security and Privacy.
30. Let's Encrypt, 2020. *Percentage of Web Pages Loaded by Firefox Using HTTPS.* [Online]

Available at: https://letsencrypt.org/stats/
[Accessed 5 8 2020].
31. Luft, J. & Ingham, H., 1955. The Johari window, a graphic model of interpersonal awareness. *Proceedings of the western training labratory in group development,* 246(August).
32. Mahoney, M. V. & Chan, P. K., 2003. *An analysis of the 1999 DARPA/Lincoln Labratory evaluation data for network anomaly detection.* Berlin, International Workshop on Recent Advances in Intrusion Detection.
33. Mohapatra, P. K., 1980. Nonlinearity in system dynamics models. *Dynamica,* 6(1), pp. 36-52.
34. Olson, C., Judd, K. & Nichols, J., 2018. Manifold learning techniques for unsupervised anomaly detection. *Expert Systems with Applications,* Volume 91, pp. 374-385.
35. Pedregosa, F. et al., 2011. *Scikit-learn: Machine Learning in Python.* [Online]
Available at: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html
[Accessed 4 August 2020].

36. Punch, K. F., 2016. *Developing Effective Research Proposals.* 3rd Edition ed. London: Sage.
37. Samani, F. S., Zhang, H. & Stadler, R., 2019. *Efficient Learning on High-Dimensional Operational Data.* s.l., IEEE 15th International Conference on Network and Service Management .
38. Sanders, C., 2020. *The Call for Applied Research on Offensive Security Tool Release.* [Online]
Available at: https://chrissanders.org/2020/07/research-ost-release/?
[Accessed 15 8 2020].
39. Saunders, M., Thornhill, A. & Lewis, P., 2007. Research Onion. In: *Research methods for business students.* Essex: Pearson Education.
40. Scikit-learn, 2018. *Multi-core parallelism using joblib.Parallel.* [Online]
Available at: https://scikit-learn.org/stable/developers/performance.html#multi-core-parallelism-using-joblib-parallel
[Accessed 5 August 2020].
41. Seidl, J., 2014. *GoldenEye DoS Test Tool.* [Online]
Available at: https://github.com/jseidl/GoldenEye
42. Sharafaldin, I., Laskkari, A. H. & Ghorbani, A. A., 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP,* pp. 108-116.
43. Shekyan, S., 2016. *SlowHTTPTest.* [Online]
Available at: https://github.com/shekyan/slowhttptest/wiki
44. Shi, Y., Zeng, H. & Nguyen, T. T., 2019. *Adversarial Machine Learning for Network Security.* s.l., IEEE International Symposium on Technologies for Homeland Security.
45. Shyu, M.-L., Chen, S.-C., Sarinnapakorn, K. & Chang, L., 2003. *A Novel Anomaly Detection Scheme Based on Principal Component Classifier,* s.l.: Miam University Department of Electrical and Computer Engineering.
46. Snort, 2019. *Anatomy of a Snort Rule.* [Online]
Available at: https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/000/116/original/Snort_rule_infographic.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIXACIED2SPMSC7GA%2F20200831%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20200831T0
[Accessed 30 8 2020].
47. Staffordshire University, 2019. *Research Ethical Review Policy.* [Online]
Available at: https://www.staffs.ac.uk/research/pdf/ethical-review-policy.pdf
[Accessed 28 7 2020].
48. Subba, B., Biswas, S. & Karmakar, S., 2016. *Enhancing performance of anomaly based intrusion detection systems through dimensionality reduction using principal component analysis.* s.l., 2016 IEEE International Conference on Advanced Networks and Telecommunication Systems.
49. Tenenbaum, J. B., de Silva, V. & Langford, J. C., 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science,* 290(5500), pp. 2319-2323.
50. Tietze, C., 2013. *Introduction to Zettelkasten.* [Online]
Available at: https://zettelkasten.de/posts/zettelkasten-improves-thinking-writing/
[Accessed 5 3 2020].
51. UNB, 2018. *CSE-CIC-IDS2018 Dataset.* [Online]
Available at: https://www.unb.ca/cic/datasets/ids-2018.html
52. University of New Brunswick, 2018. *CSE-CIC-IDS2018 on AWS.* [Online]
Available at: https://www.unb.ca/cic/datasets/ids-2018.html
[Accessed 3 7 2020].

53. University of Waikato, 2020. *Weka - the workbench for machine learning.* [Online] Available at: https://www.cs.waikato.ac.nz/~ml/weka/
54. van der Maaten, L. & Hinton, G., 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research,* Issue 9, pp. 2579-2605.
55. Xiao, Y., Xing, C., Zhang, T. & Zhao, Z., 2019. An Intrusion Detection Model Based on Feature Reduction and Convolutional Neural Networks. *IEEE Access,* Issue 7, pp. 42210-42219.
56. Yang, H. & Li, H., 2019. Implementation of Manifold Learning Algorithm Isometric Mapping. *Journal of Computer and Communications,* Volume 7, pp. 11-19.
57. Zhang, H., 2004. The Optimality of Naive Bayes. *American Assoication for Artificial Intelligence.*
58. Zhang, Z. & Wang, J., 2007. *MLLE: Modified Locally Linear Embedding Using Multiple Weights.* s.l., Advances in neural information processing systems.
59. Zhao, S., Li, W., Zia, T. & Zomaya, A. Y., 2017. *A dimension reduction model and classifier for anomaly-based intrusion detection in internet of things.* s.l., 2017 IEEE 15th International Conference on Dependable, Automatic and Secure Computing.
60. Zhu, J., Zou, H., Rosset, S. & Hastie, T., 2009. Multi-class Adaboost. *Statistics and Its Interface,* Volume 2, pp. 349-360.
61. Zong, W., Chow, Y.-W. & Susilo, W., 2019. *Dimensionality Reduction and Visualization of Network Intrusion Detection Data.* s.l., Australasian Conference on Information Security and Privacy.