APIC: A Method for Automated Pattern Identification and Classification

by

Ryan Gavin Goss

Dissertation

submitted in fulfilment
of the requirements
for the degree

Doctor of Philosophy

in

Computer Science

at the

University of Cape Town

Supervisor: Dr. Geoff S. Nitschke

March 2017

APIC: A Method for Automated Pattern Identification and Classification

by

Ryan Gavin Goss

Acknowledgements

My grateful thanks go to the following people and organisations, without whom this dissertation would not have been possible:

First and foremost, I would like to thank the **Lord Jesus Christ**, who gave me the strength to persevere, even through the hard times. I would like to thank my wife, **Kelly**, and our two children, **Rylee** and **Kayley**, for showering me with endless love, support and encouragement. To my parents, **Melvin** and **Julia**, for all the sacrifices they made for me and for demonstrating that we can achieve anything we set our minds to. I would like to thank my brother, **Robert**, for continued encouragement and for leading the way forward, by example, in academia. To my sister, **Kym**, for her support and assistance in rendering many of the figures present in this text.

My sincerest thanks go out to my supervisor, **Geoff Nitschke**, who has helped to expand my writing skills over the past few years. Neither this dissertation, nor my previous publications would have been possible without his constant feedback and tireless efforts. A big thank you also to my copy-editor, **Tanya Wyatt**, who worked through this dissertation with me, recommending changes to improve the overall read. To my friends and colleagues at both my current and previous employers, for providing insights into the real-life case studies described in this dissertation. Thank you to the developers of the **scikit-learn project** (Pedregosa et al., 2011) and **Google TensorFlow** (Abadi et al., 2015), whose libraries were used extensively in this study. Finally, I would like to thank the *National Research Foundation* (**NRF**) for funding many aspects of this dissertation.

Abstract

Machine Learning (ML) is a transformative technology at the forefront of many modern research endeavours. The technology is generating a tremendous amount of attention from researchers and practitioners, providing new approaches to solving complex classification and regression tasks. While concepts such as *Deep Learning* have existed for many years, the computational power for realising the utility of these algorithms in real-world applications has only recently become available. This dissertation investigated the efficacy of a novel, general method for deploying ML in a variety of complex tasks, where best feature selection, data-set labelling, model definition and training processes were determined automatically. Models were developed in an iterative fashion, evaluated using both training and validation data sets. The proposed method was evaluated using three distinct case studies, describing complex classification tasks often requiring significant input from human experts.

The results achieved demonstrate that the proposed method compares with, and often outperforms, less general, comparable methods designed specifically for each task. Feature selection, data-set annotation, model design and training processes were optimised by the method, where less complex, comparatively accurate classifiers with lower dependency on computational power and human expert intervention were produced. In chapter 4, the proposed method demonstrated improved efficacy over comparable systems, automatically identifying and classifying complex application protocols traversing IP networks. In chapter 5, the proposed method was able to discriminate between normal and anomalous traffic, maintaining accuracy in excess of 99%, while reducing false alarms to a mere 0.08%. Finally, in chapter 6, the proposed method discovered more optimal classifiers than those implemented by comparable methods, with

classification scores rivalling those achieved by state-of-the-art systems.

The findings of this research concluded that developing a fully automated, general method, exhibiting efficacy in a wide variety of complex classification tasks with minimal expert intervention, was possible. The method and various artefacts produced in each case study of this dissertation are thus significant contributions to the field of ML.

Contents

Nomenclature

Acronyms

AI Artificial Intelligence
A field of computer science concerned with the development of computer systems capable of mimicking tasks performed by a human expert.

APIC Automated Pattern Identification and Classification
The method presented in this dissertation to address the completeness and accuracy concerns of existing pattern recognition systems.

Classifier A model for deriving an output vector from an input vector.

DDoS Distributed Denial of Service
A network attack with multiple attack vectors toward a single target. The UDP protocol is most often used in conjunction with spoofed source IP addresses.

DNS Domain Name Service
An application protocol used to translate a host name to an IP address and vice-versa.

DoS Denial of Service
A network attack where an attacker attempts to deplete the resources of a remote target, rending the target unavailable on the network.

EA Evolutionary Algorithm
A generic optimisation algorithm inspired by biological evolution. EAs are a component of *Evolutionary Computing* (EC).

EC Evolutionary Computing
A subfield of *Artificial Intelligence* (AI) algorithms for global optimisation inspired by biological evolution

Flow A sequence of packet exchanges on a computer network sharing the same five tuple of source and destination addresses, and source and destination ports and protocol.

Ground Truth A term referring to the accuracy of classification when parsing a training set in supervised learning techniques.

Host A computer connected to a network.

IP Internet Protocol
A set of rules governing the format of data during communications across computer networks.

IP Internet Protocol
The principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries.

ISP Internet Service Provider
A network providing access last-mile Internet services to home and business users.

Latency A synonym for delay, it represents the time taken for a segment of data to arrive at the destination after transmission.

ML Machine Learning
A subfield of *Computer Science* (CS), providing computers the ability to learn without being explicitly programmed.

Packet A formatted unit of data carried by packet-switched networks.

Port An application-specific construct serving as a communication endpoint in a computer's host operating system.

Signature A recipe for identifying a distinct application protocol on a network.

Spoofing The act of imitating another host on a computer network, manipulating traffic so that it appears to be sourced from a host other than itself.

TCP Transmission Control Protocol
A stateful core protocol of the TCP/IP protocol suite.

UDP User Datagram Protocol
A stateless network communication protocol, often used for small, latency-dependent communications such as DNS.

Zero-day Attack An attack which exploits a previously unknown vulnerability in a computer system. Developers have therefore had "zero days" to address the vulnerability.

Symbols

$\boldsymbol{k}$	The number of clusters in a particular data set.
$\boldsymbol{eps}$	Epsilon: A parameter for clustering that specifies how close data points should be to each other to be part of the same cluster.
$\boldsymbol{minPTS}$	Minimum Data Points: Specifies how many neighbours should be in close proximity to be considered a cluster.

Chapter 1

Introduction

Machine Learning (ML), a sub-field of *Artificial Intelligence* (AI) concerned with learning from data or experience, has gained widespread adoption in many data-driven tasks over the past decade. This can be attributed to two main developments, namely, the abundance of data and the availability of adequate computational power to explore it (Rogers and Girolami, 2016). Significantly more data is collected and stored today by a variety of systems than in previous years (Hendrickson, 2010). Owing to the size and complexity of these data sets, useful content needs to be identified and extracted effectively, avoiding *information overload*[1]. ML methods provide the tools to model this high-dimensional, complex data, locating and recommending the most relevant content. Larger data sets require more complex models, inherently demanding significantly more computational resources.

The amount of computational power, per dollar, has increased exponentially over the past decade, at an estimated, average rate of around 55 percent per annum since 1940 (Nordhaus, 2001). These substantial gains have had a profound affect on ML, allowing conventionally simple models, such as *Artificial Neural Networks* (ANN), to expand in complexity and, subsequently, their usefulness. *Deep Learning* (DL) networks are more complex ANNs, consisting of multiple layers. Each of these layers provides its own level of non-linearity, something that cannot be contained in a single layer. Increasing model complexity for complex tasks produces more generalised classifiers, avoiding conventional ANN pitfalls such as

[1]Information overload refers to overwhelming a user with too much data or information

over-fitting. These multiple layers subsequently allow the algorithm to more accurately model high-level abstractions found in large, complex data sets.

The ability of ML to model high-dimensional, complex data sets makes it attractive for a number of classification tasks. The application of this technology in each task is, however, complicated by a number of factors - the most prevalent being dependency on human experts for successful implementation. Overcoming these obstacles will lead to a more pervasive deployment of ML-based classification technologies across a broad range of tasks in multiple domains.

1.1 Motivation

Technologies such as deep learning may allow more complex models to be developed, however the design of these networks is still predominantly performed by human experts, who optimise the design of each classifier through an empirical process. The feature selection, data annotation and classifier development tasks, for the majority of works studied in this dissertation, are all heavily dependent on human expert contributions. Subsequently, the classifiers produced may not be the best for a given solution. Manual definitions often lead to overly complex classifiers, placing unnecessary strain on computational resources or producing models that exhibit high degrees of underfitting. Conversely, experts may develop less complex models, where recall accuracy is high for their test data sets, however perform poorly on previously unseen data (overfitting). In some cases, no expert may be available to describe the problem well enough for an ML algorithm to be useful and thus the results of the algorithm may be inadequate.

Designing and developing models for solving complex classification tasks is extremely difficult to accomplish manually and is often the most intensive part of the ML integration process. Models for each classification task need to be constructed efficiently, yet provide sufficient complexity for fair generalisation (avoid overfitting). The number of options available during classifier development is vast. The task of identifying the best combination of feature vectors, neurons and hidden layers is often too large to discover using a grid (sequential) search. Unsupervised ML algorithms that use

stochastic techniques to imitate natural genetic inheritance and Darwin's theory of survival of the fittest (observed in nature) have demonstrated significant efficiencies for problems like these. These *Evolutionary Algorithms* (EA) provide search functions that generate solutions more efficiently than grid searches. EAs are used by algorithms, such as the *Topology and Weight Evolving Artificial Neural Networks* (TWEANN), to develop the structure of a model, including the neuron connectivity and weight allocations at each neuron. Using *Neuroevolution* (NE) approaches, such as a TWEANN, complex models may be reduced by monitoring the fitness and complexity of models described by each candidate solution.

Research has shown that the efficacy of ML technologies in complex classification tasks is promising, however a significant amount of manual intervention is still required to embed these algorithms in each task. The selection of best features to describe each data set, the annotation of each datum according to it's class, and the development of the best classifier to describe each class is often controlled by human experts who have significant experience in each respective field. Domain knowledge is therefore key when training ML classifiers for each task. The requirement of intervention by a domain expert for each task inhibits the pervasiveness of ML implementation across many general classification tasks.

1.2 Research Problem

Although ML methods have demonstrated impressive results in a variety of classification tasks, a lack of automation in feature selection, pattern identification, and classifier development is evident. In many cases, the structure of classifiers is defined manually, using supervised learning techniques, trained from data sets manually created and annotated by human experts. Each classification problem is described using a distinct feature set, specifically engineered for the task by these experts. This manual intervention is costly, causing delays in classifier production and increased risk due to human error. It is apparent that a general method needs to be devised, incorporating automated, accurate classifier production for a variety of tasks, reducing the dependency on human experts. Tasks such as *Internet Protocol* (IP) traffic classification, anomaly detection, and

handwritten digit identification provide good case studies for evaluating the new method, as they are indicative of many difficult classification tasks. These tasks, characterised by noisy, continuous data sets, are subsequently used to demonstrate the efficacy of the proposed method across a broad range of domains. Both accuracy (achieved by development of optimal classifiers) and completeness (the availability of a classifier for each pattern tested, achieved through automating existing manual processes) are paramount to the success of these systems. The research problem considered by this dissertation can therefore be succinctly stated as follows:

> Current ML-based classification approaches require a significant amount of *complex, manual intervention* from domain experts for embedding ML in each problem. Parameters guiding these deployments are limited to the scope programmed by these experts. Definition of best feature sets, identification of distinct patterns in data (annotation), and the design of optimal, accurate models that generalise well are tasks often undertaken by human experts. This process is pessimal, especially for real-time classification tasks, where the effectiveness of a system is largely dependent on the accurate identification of new patterns in data as soon as they are detected.

1.3 Research Objectives

The primary contribution of this research is the development of a new general method - *Automated Pattern Identification and Classification* (APIC) - which reduces the manual input required from human experts to achieve high degrees of accuracy and completeness for a variety of classification tasks. The primary case study in this work evaluates APIC's ability to improve the state-of-the-art technology for a current, complex classification task, namely IP traffic classification. According to Szabo et al. (2007), the success of IP traffic classification systems can be measured by two key metrics: **completeness** and **accuracy**. Completeness refers to the availability of a signature (classifier) for each application protocol present on the network. Accuracy refers to the number of correct matches a signature makes and the confidence level thereof.

The primary research objective of this dissertation is, therefore, *to ascertain whether APIC can reduce the dependency on manual, expert involvement currently associated with the development of classifiers for complex classification tasks, increasing the overall completeness, accuracy and efficiency of these classifiers compared to existing ML methods.*

In support of this, the following sub-objectives were addressed:

- Determining whether APIC can automatically and accurately identify new patterns in unlabelled, mixed, noisy data sets.
- Determining whether APIC can automatically produce customised classifiers, where accuracy results are comparable with, or exceed, those achieved by comparable ML methods designed by human experts for a specific task.
- Determining whether APIC can serve as a general method, supporting application across a broad range of problems compared to other ML methods.

1.4 Methods

This research falls into the research field of computational learning theory (Kearns and Vazirani, 1994), of which the use of ML to classify data is a sub-field. In this research, the APIC method was investigated as an alternative to human expert intervention for developing classifiers across a broad range of classification tasks. The efficacy of the method was tested using three distinct case studies: IP traffic classification (Chapter 4), anomaly detection (Chapter 5) and handwritten digit recognition (Chapter 6).

The method proposed in this dissertation divided the task of classifying data into three distinct parts. The first, an offline component, used a *Genetic Algorithm* (GA) (Eiben and Smith, 2003) to automatically search for more optimal feature sub-sets. Reducing the original feature set allowed the method to lessen the complexity of the task, while retaining accuracy during the classification process. The second part used *clustering algorithms* (Harrington, 2012) to automatically detect distinct patterns present in a data set described by each feature sub-set. The *hyper-parameter optimisation* (Bergstra et al., 2011) of the clustering

process was also performed using a GA, as the use of random search has shown significant efficiencies when compared with searching for the same by grid search (Bergstra and Bengio, 2012). Part 3 used the clusters discovered in part 2 as labelled training sets for producing *semi-supervised classifiers* (Russell and Norvig, 2009) to identify future instances of each pattern. A GA defines the ANN (Gurney, 2003) topology of each classifier (neuron configuration), with a second GA optimising the weights of each connection, forming a *Topology and Weight Evolving Artificial Neural Network* (TWEANN) (Sher, 2012) classifier. The backpropagation algorithm (Rumelhart et al., 1988) was used to further optimise the weight values of each classifier using a gradient descent function.

The set of newly-defined classifiers (signatures) were tested and, upon reaching a pre-defined criteria, were declared suitable for the given task. If the evaluation criteria were not met, another feature sub-set was produced and tested, repeating the process. The APIC method uses an ensemble of ML algorithms (pipeline) for producing classifiers capable of identifying both known and previously unknown patterns in mixed, noisy data sets automatically. As the method is a pipeline, algorithm substitutions may be made without altering the general operation of the method. For example, a GA may be replaced by a *Bayesian Optimisation* (Mockus, 2012) method for a particular task, or the clustering algorithm altered from agglomerative to divisive in another.

1.5 Contributions

The main contribution of this dissertation is the APIC method, a principled, general method for automated feature set optimisation, pattern discovery, and classifier production for a variety of tasks across multiple domains. Several experiments demonstrated application of the method for solving current problems, including those associated with the completeness and accuracy of IP traffic classification systems (Chapter 4), identifying anomalies in IP traces (Chapter 5) and identification of handwritten digits in a publicly-available data set (Chapter 6). In these case studies, both flexibility of application and protection against innovation were tested to ensure that the method remained robust and future-proof.

The experiments described in Chapter 4 demonstrated that the method was capable of automatically producing TWEANN classifiers for identifying IP traffic flows. Classifiers were automatically produced for each application protocol, with no human involvement, producing a system arguably more complete than comparable methods that rely on constant human intervention. The accuracy achieved by these automatically generated classifiers rivalled, and often exceeded, many comparable systems, designed specifically for this task, for a number of tested application protocols. Research into the application of TWEANN classifiers for classifying IP traffic flows is also a novel contribution provided by this work, as is automating the process of classifier development for previously unseen application protocols. Interestingly, both completeness and accuracy problems are not confined to the IP traffic classification domain. These concerns are ubiquitous in many important areas, including biological (Sakamoto et al., 2013), engineering (Darvishi et al., 2013) and earth science (Foody, 2002) domains. The ability of APIC to act as a general method, improving classification for a variety of tasks across multiple disciplines is another important contribution, as most other ML-based classification systems are designed for one specific purpose.

In Chapter 5, the efficacy of the APIC method was evaluated in the task of *Anomaly Intrusion Detection* (AID), protecting computer networks against malicious attack. The method was tested using a publicly available, pre-labelled data set and private data sets recorded on a live enterprise network. APIC, a general classification method, developed classifiers that resulted in accuracies comparable with those of other ML-based systems, designed by human experts specifically for this purpose.

Finally, in Chapter 6, the APIC method was applied to the task of handwritten digit recognition, using a popular, publicly available data set of handwritten digits. The APIC method was used to develop a more efficient *Convolutional Neural Network* (CNN) model, where the convolutions, kernels, pooling and classification models were determined automatically. Results indicated that the method compared with the best systems designed specifically for this purpose. The APIC classifiers were, however, significantly more efficient (less complex) than these comparable methods. The ability for APIC to form less complex models for tasks like this,

without suffering from overfitting, is another contribution of this research.

All three case studies demonstrated that APIC could be applied in a broad range of classification tasks, exhibiting results that rival systems designed specifically for each task, often with less-complex classifiers. In each case, the APIC method operated independently of human experts, providing significantly more automation for these tasks than had currently been tested. The APIC method is thus the first step toward fully automating the production of classifiers for a variety of tasks, providing exciting avenues for future research.

1.6 Overview of Dissertation

This dissertation is divided into four parts: foundations (Chapter 2), the method (Chapter 3), case studies (Chapters 4, 5 and 6) and conclusions (Chapters 7 and 8).

Chapter 2 reviews the current state of ML-based classification and supporting algorithms. Foundational information for each of the three case studies described in this dissertation is presented throughout as additional motivation for this research.

Chapter 3 presents the APIC method for solving challenges associated with general classification tasks. GA-driven unsupervised learning algorithms provide clear distinction between patterns in mixed, noisy data sets. The method protects against innovation by automatically creating new clusters (groups) of datum for each previously unidentified pattern. These clusters are subsequently used to train classifiers in identifying future instances of each pattern automatically.

Chapter 4 focuses on the application of APIC for identifying the underlying application protocol of IP traffic flows (IP traffic classification). The accuracy achieved by automatically identifying and producing classifiers for six applications is tested against results achieved for the same by other methods. Here, APIC is shown to automatically produce accurate classifiers for each of these protocols, where the accuracy attained by each exceeds those achieved by comparable methods. APIC also exhibits properties that ensure a more complete system is achieved when compared to similar systems.

Chapter 5 evaluates the APIC method in anomaly detection tasks. Specifically, APIC is tested as a *Network Based Anomaly Detection* (NBAD) system, evaluated using a publicly available static data set comprised of completed, annotated IP flow records. A second test evaluates the method for its ability to produce classifiers that profile a live, production enterprise network, detecting *Denial of Service* (DoS) attacks in near real time.

Chapter 6 demonstrates the efficacy of APIC for addressing classification tasks outside of the IP networking domain. In this chapter, a case study is presented that tests the ability of APIC to produce less-complex, comparably accurate classifiers for classifying handwritten digits, compared to existing ML-based approaches.

Chapter 7 provides a discussion on the generality of the APIC method and the efficacy it exhibits in the three case studies presented in this dissertation. The chapter also provides recommendations for avenues of future research.

Chapter 8 concludes the dissertation, providing a summary of the contributions yielded by this research.

1.7 Assumptions and Delineations

In most cases, wherever feasible, results were rounded to four decimal places. Acronyms are defined at least once per chapter and a summarised list of those most popular can be found in the nomenclature section. In *Evolutionary Algorithms* (EA), the terms genotype, genome and chromosome are used interchangeably. This dissertation uses the term "genotype" as the standard way to describe a candidate solution (encoded phenotype).

Chapter 2

Foundations

As a branch of *Artificial Intelligence* (AI), *Machine Learning* (ML) is concerned with the construction of systems that learn from data, adapting to new circumstances and automatically improving with experience (Russell and Norvig, 2009; Mitchell, 1997). These algorithms have been widely used in a number of fields, including search engines, medical diagnoses and load prediction (Hu and Shen, 2012). More recently, applications of ML in *Internet Protocol* (IP) traffic classification, including the identification of underlying applications and detection of anomalous traffic patterns, have been proposed. Another area garnering widespread interest is computer vision tasks, where ML is used to classify objects in two-dimensional images. An ML algorithm learns using statistics, converting data into information (Harrington, 2012). Today, time-critical information is vital to a number of activities, where getting lost in raw data is no longer an option. It is here that ML excels, by abstracting useful information from large data stores (Harrington, 2012). The detection and extrapolation of patterns within data sets is a core function of ML (Russell and Norvig, 2009).

Broadly, ML algorithms can be divided into three main types, according to the feedback they provide. These types, described by Russell and Norvig (2009), are *supervised learning* (classification), *unsupervised learning* (clustering) and *reinforcement learning*. Supervised learning algorithms learn from a set of pre-classified, or pre-labelled (annotated) data sets. From these learnings models are formed describing the observations. These models can be used to classify future, previously unseen, data samples. The most common unsupervised learning task is clustering (Russell and Norvig,

2009), where the algorithm gathers data in groups, or *clusters*, based on similar features without any prior knowledge or guidance. A number of hybrid types have also emerged over the years, combining elements of both supervised and unsupervised learning. These approaches are often referred to as *semi-supervised* learning algorithms (Russell and Norvig, 2009).

The focus of this dissertation is toward automating the development of classifiers, creating more optimal solutions for solving complex classification problems across a broad range of tasks with little assistance from human experts. As such, both supervised and unsupervised algorithms - the ML types used by the proposed solution (Chapter 3) - are discussed in more detail in the following subsections. Past works implementing these algorithms are surveyed, pertinent to the case studies tested by this dissertation in Chapters 4, 5 and 6. This chapter focuses on ML techniques and algorithms used by the proposed method. Additional task information, including background and comparable works, are included in each of the respective case study chapters. The proposed method (Chapter 3) also makes extensive use of *Evolutionary Algorithms* (EA), an optimisation method, to aid discovery of more optimal feature subsets and hyper-parameters for each algorithm. Given the significance of EAs, these algorithms are investigated separate to unsupervised learning in section 2.3. This section describes the composition of EA algorithms and their current applications for evolving *Artificial Neural Network* (ANN) and *Deep Learning* (DL) models, by a process known as *Neuroevolution* (NE).

2.1 Supervised Learning

In ML, supervised learning refers to the task of inferring a function from labelled training data (Mohri et al., 2012). The algorithms are presented with a set of input and output pairs $(x, y), x \in X, y \in Y$, for which a function $f : X \rightarrow Y$ needs to be discovered. This function may be deterministic, or stochastic (Russell and Norvig, 2009). Where the function is deterministic, each output is given by $y = f(x)$. Where a stochastic function is apparent, a conditional probability distribution function $P(Y|x)$ is required (Russell and Norvig, 2009). The function may determine the output as one of a set of finite, discrete values (classification), or alternatively represent the

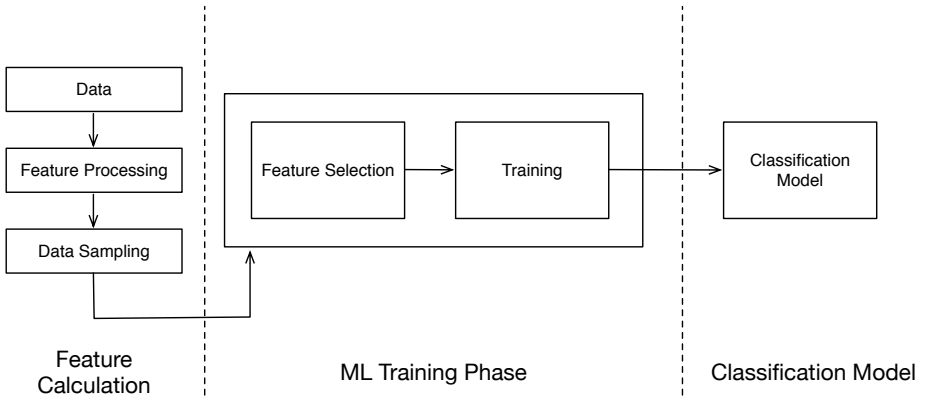


Figure 2.1: A diagram describing the development of most supervised learning classifiers, adapted from Hu and Shen (2012).

output as a continuous value. Where a continuous value is returned, the learning problem is called *regression* (Freedman, 2009). Here, the number represents a conditional expectation (function approximation) rather than an exact classification.

A cost function, which implicitly contains prior knowledge regarding the problem domain, evaluates the inferred function and, subsequently, the mappings produced when it parses a test data set. According to Mitchell (1997), a commonly used cost function and the function used to ascertain the fitness of supervised learning models in this dissertation is the *Mean-Squared Error* (MSE) function (equation 2.1).

$$MSE = \frac{\sum_{i=Y}^{n} (Y_i - A_i)^2}{n} \tag{2.1}$$

Where n is the number of pairs in the data set, Y the desired values (outputs or *targets*) and A the actual value, or output of $f(x)$, produced. The goal of the training process is to minimise the average MSE value of the network for all example pairs (x, y). There are a number of methods to reduce the cost of, or *train* a network, one of which is *gradient descent* (Snyman, 2005). Gradient descent works by incrementally changing a single element of a solution in an attempt to produce a better solution by converging to a local minimum.

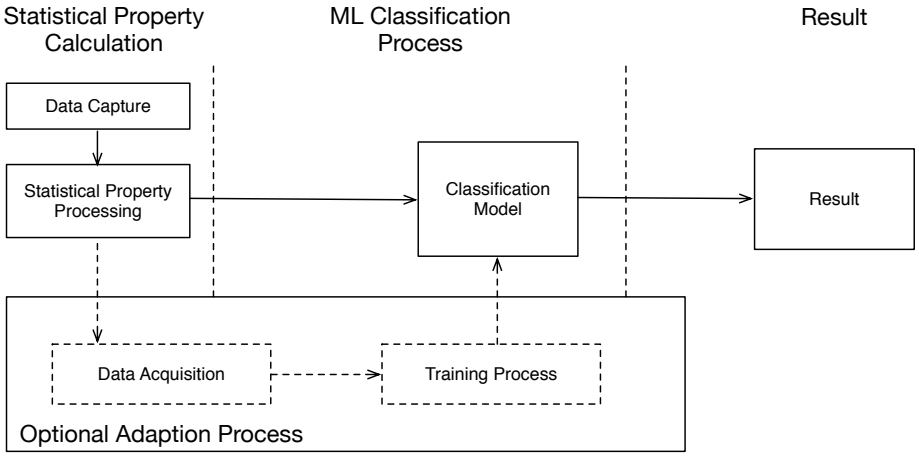


Figure 2.2: A common ML supervised learning test (evaluation) process, adapted from Hu and Shen (2012).

The accuracy of a supervised learning algorithm for a task is dependent on the type of problem, the learning algorithm, the specific *hyper-parameters* (Bergstra and Bengio, 2012) guiding the algorithm, and the supplied training and test data sets. Supervised learning can typically be broken into two phases: *training* and *testing* (Hu and Shen, 2012). The training phase (Figure 2.1) extracts features from a data set, applying labels to the resulting datum. This data is sampled and supplied to a training process, which constructs a model for describing the data. This model can subsequently be used to classify previously unseen datum. The testing phase (Figure 2.2) extracts the same features from previously unseen data before submitting them for evaluation by the classification model. In some implementations, a parallel process is used to adapt the model in real time, much like in the training phase (Hu and Shen, 2012).

The following subsections provide an overview of supervised learning applications related to the three case studies evaluated in this dissertation. These are *Internet Protocol* (IP) traffic classification (Chapter 4), *Network Based Anomaly Detection* (NBAD; Chapter 5) and handwritten digit recognition (Chapter 6). Additional background for each task, and further details regarding each of these related works, is provided in the respective case study chapters.

2.1.1 IP Traffic Classification

A number of supervised learning algorithms, including *Naive Bayesian* (Mitchell, 1997), *Decision Trees* (Quinlan, 1987) and ANNs (Gurney, 2003) have been applied to IP traffic classification tasks over the past few years (Zhang et al., 2009a). Some of these applications include the work of Alshammari and Zincir-Heywood (2009), where *AdaBoost* (Freund and Schapire, 1995), *Support Vector Machines* (SVM) (Cortes and Vapnik, 1995), *Naive Bayesian*, *Repeated Incremental Pruning to Produce Error Reduction* (RIPPER) (William et al., 1995) and *C4.5* (Quinlan, 1993) algorithms were compared for identifying *Secure Shell* (SSH) traffic flows captured on the Dalhousie Campus network[1]. The results showed that the C4.5 algorithm produced a superior model, with an overall accuracy of between 83.7 percent and 97 percent. Their results also indicated that a classifier trained on one network can be deployed on another with reasonable success. It is noteworthy that the signature developed to identify SSH was manually constructed and that the features and rules employed were for detecting the SSH protocol only. According to Alshammari and Zincir-Heywood (2009), additional work would be required to develop a robust signature suitable for classifying other applications, such as *Skype*[2], in a similar manner.

Li et al. (2007) describe a method for classifying applications using SVMs into one of seven classes, a subset of those listed in table 4.2. Using specific statistical features, including destination ports, receive packet size variances and window size information, their optimised method yielded an accuracy of 96.92 percent in unbiased training and testing. For regular training with biased prior probability, the authors achieved an overall accuracy rate of 99.4 percent. While the overall accuracy achieved was very high, the method lacked the ability to individually classify a particular application protocol, instead, relying on grouping these protocols into one of seven categories. Also, the dependency on port information raises doubts in applications where dynamic port selection is in effect.

More recently, Goss and Botha (2012) proposed a fixed set of features, which included *packet directionality*, *packet size* and *Deep Packet Inspection*

[1]Dalhousie University, Halifax, Nova Scotia, Canada. `http://www.dal.ca/`

[2]`http://www.skype.com/`

(DPI) information to discriminate between application protocols on a network in real time. Datum extracted using these feature sets were manually grouped by their application protocol, and annotated by experts. This labelled training set was used to train ANN classifiers in recognising future instances of each protocol. Using the defined features and a fixed ANN structure, or *model*, an overall accuracy in excess of 99 percent was recorded.

In the works discussed, an individual classifier was developed for each application protocol. In the case of Alshammari and Zincir-Heywood (2009), the construction of these classifiers required additional, manual development by experts. For Goss and Botha (2012), expert involvement was required for annotating data sets compiled by extracting specific features from recorded flow samples. Producing signatures in this fashion is not conducive to the completeness of an IP traffic classification system, one of two metrics, namely *accuracy* and *completeness*, used to evaluate these systems (Szabo et al., 2007).

2.1.2 Anomaly Detection on IP networks

It is important to identify and control anomalous traffic flows, or malicious content traversing IP networks. For example, a common, complex classification problem plaguing IP network administrators is the ability to accurately and efficiently classify unsolicited email messages entering and exiting their network. According to Goodman and Heckerman (2004), *spammers* are continually updating their tactics, making it increasingly difficult to sustain a static programmed approach to *spam* email detection. In these cases, learning algorithms work best at keeping spam at bay (Goodman and Heckerman, 2004). Androutsopoulos et al. (2000) compared Naive Bayesian and memory-based approaches for spam filtering, finding both methods achieved very high classification accuracy, outperforming anti-spam keyword patterns supplied by a widely used email reader. The findings in this work showed that it was entirely feasible to construct learning-based filters for spam detection purposes (Androutsopoulos et al., 2000).

Although often only mildly annoying, unsolicited email, or *spam*, can also act more maliciously, wasting resources and preventing legitimate access to

both compute and network resources. These kinds of attacks are known as *Denial of Service* (DoS) attacks - when only a single source host participates using a single type of attack - or a *Distributed Denial of Service* (DDoS) - when multiple hosts attack using many *attack vectors*.

Jalili et al. (2005) trained supervised ANNs for detecting DoS attacks, where an attacker targeted a network with semi-normal packets. Semi-normal packets are normal packets that occur either at an abnormal rate, or contain abnormal payload content. According to (Jalili et al., 2005), the statistical properties that a flow exhibits change under DoS or DDoS attack, deviating from normal distribution. The method was demonstrated capable of differentiating between normal and attack traffic in 94.9 percent of cases.

2.1.3 Handwritten Digit Recognition

Computer vision classification tasks, such as image recognition and, more specifically, handwritten digit recognition, deal with the extraction of high-dimensional data from images to produce numerical or symbolic information, representing each object for ML-based classification models. In most cases, these methods divided the task into two main parts, namely the *feature extractor* and *trainable classifier*. The feature extractor is rather specific to each task and thus requires the most design effort (LeCun et al., 1998). For the general classifier, LeCun et al. (1998) states that a large number of commercial *Optical Character Recognition* (OCR) systems employ some form of multi-layer ANN, trained using *backpropagation* (Rumelhart et al., 1988).

LeCun et al. (1998) tested a variety of supervised, gradient-based methods for classifying the popular *Mixed National Institute of Standards and Technology* (MNIST) data set of handwritten digits. The authors manually configured ANN models using heuristic values for the number of hidden layers and neurons, where the lowest classification error-rate was 1.6 percent. Using *Convolutional Neural Networks* (CNN), the authors managed to reduce the error-rate to 1.1 percent. Finally, the authors tested a *boosting* method that incorporates a fixed number of CNN classifiers and uses the average classification to identify each digit. Here, the error-rate dropped even further, to 0.7 percent.

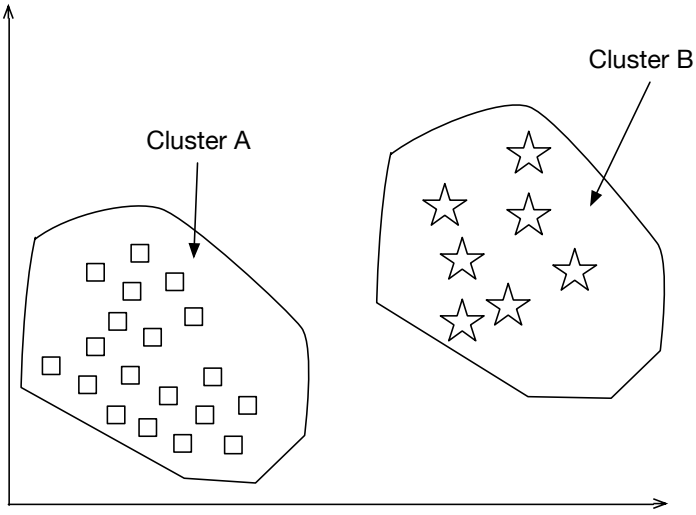


Figure 2.3: Data clustering: one of the primary functions of unsupervised ML.

Ciregan et al. (2012) used CNN classifiers to redefine the state-of-the-art technology for classifying images of the MNIST data set, achieving an error-rate of 0.23 percent. The method, like that of LeCun et al. (1998), uses *multi-column Deep Neural Networks* (MCDNN), with convolution and pooling layers as pre-processors.

All of the supervised learning systems described used models that were designed and developed manually by human experts. This process required experts who were knowledgeable in each specific field. For example, an expert on handwritten image recognition may not produce a model fit for IP traffic classification as easily as an IP expert might. These methods also used pre-labelled data sets, where the class of each datum was known as *a priori*. Supervised learning algorithms were subsequently found to rely on the presence of class labels (predefined outputs) for successful training. Contrary to supervised learning algorithms, unsupervised learning can identify similarities amongst data and produce groups, or clusters, automatically. This clustered data could then be used as labelled training sets for training supervised learning algorithms in cases where no experts were available or no pre-labelled data set existed.

2.2 Unsupervised Learning

In supervised learning the objective of the algorithm is to identify a function that maps inputs to a specific output, where correct values are provided by a human expert (Alpaydin, 2014). In contrast, for unsupervised learning there is no human expert (supervisor) nor input to output mapping. The objective of unsupervised learning is to find regularities and infer a function to describe these hidden structures in unlabelled data sets. Some of these structures, or *patterns*, occur more regularly in the data set and it is the job of unsupervised learning algorithms to group, or cluster, these related datum based on their similarities (Figure 2.3). Clustering processes tag each datum with its associated cluster identifier, automatically providing a labelled data set that can be used to train supervised classifiers.

The ability to automatically identify patterns in data sets is important for real-time classification tasks, such as IP traffic classification and *Network Based Anomaly Detection* (NBAD), where new patterns are introduced to networks daily. One of the greatest contributions of ML to the networking domain is the ability to identify advanced and previously unseen application protocols and attack traffic. Most IP traffic classification systems require a unique signature to classify each application protocol (Goss and Nitschke, 2013a). These signatures are provided by vendors of traffic management systems in one or more signature packs on a regular basis. The process of creating and deploying signatures in this manner is sub-optimal, as the development of new application protocols far exceeds signature production by vendors. Likewise, new attacks are launched daily on public networks, thus NBAD systems cannot wait for vendors to release signature packs before taking action. The lack of a suitable classifier for each application protocol or attack profile, in these applications, results in a significant volume of network traffic remaining unclassified, or inaccurately classified, until a classifier is made available.

A number of works, including those of Hu and Shen (2012), have successfully used unsupervised learning to produce annotated data sets for training supervised, or semi-supervised classifiers automatically. Most of these systems follow the basic framework depicted in Figure 2.4.

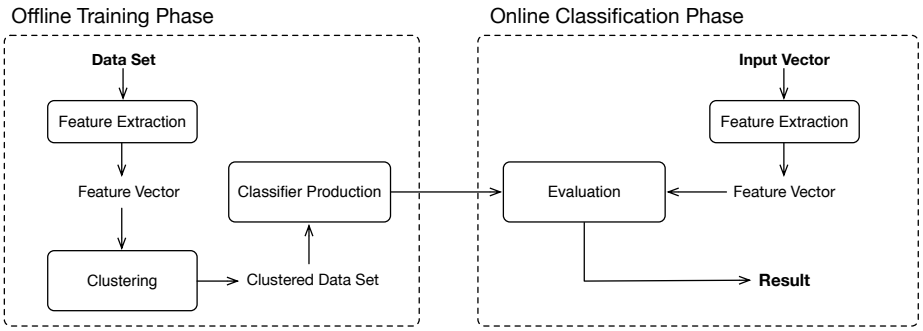


Figure 2.4: ML unsupervised learning framework for automatically creating labelled data sets and training supervised learning algorithms.

The *offline training phase* is responsible for extracting statistical features from a data set. These feature vectors are clustered using an unsupervised learning algorithm. Examples of these algorithms include *Expectation Maximization* (EM) (Dempster et al., 1977), *Autoclass* (Cheeseman et al., 1993), *k-means* (MacQueen et al., 1967b), the *Hierarchical Self-Organising Map* (HSOM) (Kohonen, 1990) and *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) (Ester et al., 1996). Identified clusters are used to form training data sets for developing supervised, or semi-supervised classifiers to recognise future instances of each cluster (pattern) in the *online classification phase*. The online classification phase extracts the same statistical features from an input vector, evaluating the new feature vector using each of the trained models.

Successful clustering of a particular task is heavily dependent on both the feature set describing the problem space and the *hyper-parameters* used to tune, or guide the clustering algorithm. Each classification problem addressed by ML algorithms is very different. Each problem needs to be described to the algorithm, a task often undertaken by human experts. The requirement for optimised feature subsets and hyper-parameters in classification tasks is prevalent for both supervised and unsupervised learning methods. While these parameters are often set heuristically by experts, EAs have been demonstrated as effective optimisation methods for a broad range of tasks.

2.3 Evolutionary Algorithms

Evolutionary algorithms are optimisation methods concerned with incorporating the evolution and adaptation techniques observed in nature into computer systems. These stochastic techniques attempt to imitate natural genetic inheritance and Darwin's theory of survival of the fittest (Fogel, 1997; Schoenauer, 1997). Although they may yield very robust solutions, evolutionary algorithms are, in reality, crude simplifications of biology (Bäck and Schwefel, 1993) and are becoming commonplace in solving complex real-world problems in industry, medicine and defence (Fogel, 1997; Schoenauer, 1997). These algorithms are able to find the global optimum of complex functions (Schoenauer, 1997), where heuristic solutions are either not available or may lead to unsatisfactory results (Fogel, 1997).

Optimised feature sets and hyper-parameters are essential to both the accuracy and completeness of classification systems, especially those operating on real-time data, where previously unseen patterns are regularly observed. These optimisations are often performed manually by experts, using empirical, sequential search processes until suitable results are noted. Evolutionary algorithms are poised to improve this process, replacing manual searches with automated feature set and hyper-parameter optimisation processes (Bergstra et al., 2011). These hyper-parameters, or *meta-parameters*, are the criterion that govern the operation of each ML algorithm.

This section begins with a brief overview of EAs, their biological inspiration, and a survey of their application to date in optimal feature selection and hyper-parameter optimisation tasks.

2.3.1 Biological Inspiration

Although the terms employed in EA are synonymous with those in biology, they are used in the spirit of analogy as the entities referred to in EA are far more simple than their actual biological counterparts. All living organisms consist of cells, which are comprised of one or more *chromosomes*, or strings of *DeoxyriboNucleic Acid* (DNA) (Melanie, 1999). These chromosomes act as a blueprint for the organism, described by a number of features, or *genes*.

Most organisms have many chromosomes present in each cell. A collection of chromosomes is referred to as a *genome*. In biology, there are subtle differences between an organism's genome and its *genotype* - a term used to describe the collective set of all genes within a particular genome. In EA literature, however, the term genotype is synonymous with genome. The genotype of an organism is used to form its *phenotype*, or implementation of physical and mental characteristics. Examples of these characteristics include eye colour, height and so forth.

In biology, the *reproduction* (or *crossover*) process involves both parents contributing to the formation of a new genotype, describing an *offspring*. These offspring are subject to *mutation*, where the values of single genes are forced to deviate from the blueprint. These changes are often results of copying errors (Melanie, 1999). The *fitness* of the resultant offspring is a measure of the probability that the organism will live to reproduce and therefore describes its *viability* amongst the population.

In EA, the terms *genotype*, *genome* and *chromosome* are often used interchangeably, referring to a candidate solution to a problem. These candidate solutions are often stored as bit strings, where genes are either single bits or short blocks of adjacent bits (Melanie, 1999). Other encoding schemes include *Permutation* and *Direct Value* encoding. Permutation encoding is typically used in ordering problems, such as the popular *Travelling Salesman Problem* (TSP), where each genotype is a string of numbers, representing a sequence. Direct value encoding is typically used in problems where real numbers (continuous values) need to be encoded. In direct encoding, each genotype is comprised of a string of numbers connected to the problem. While this type of encoding is good for special problems, it often requires the development of customized mutation and crossover operators specific to the problem. Although the method presented in this dissertation is capable of supporting each of these encoding methods, the bit string method was selected for demonstration purposes to simplify crossover and mutation operations. A customized mutation operator was developed in Chapter 6 to demonstrate the efficacy of the proposed method in situations where special customizations are required.

The mapping from the phenotype space to the genotype space is known as *encoding*, with the inverse mapping referred to as *decoding* (Schoenauer,

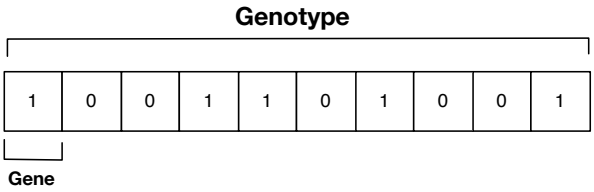


Figure 2.5: An example EA candidate solution encoded as a bit string.

1997). In this dissertation, the term *genotype* is used for describing a candidate solution encoded as a bit string. The term *phenotype* is used to describe the implementation (solution) described by each candidate solution. An example of such encoding is depicted in Figure 2.5. With this terminology in mind, the following subsection provides insight into the operations of a typical EA.

2.3.2 Overview of an Evolutionary Algorithm

Fogel (1997) describes an EA as a search for the extrema of a functional, considering the value of alternatives as solutions to the problem at hand. In section 2.1, supervised learning was defined as the task of inferring a function from labelled training data. This process is usually performed as a sequential search, where the solution will, over time, converge to a local minima. In section 2.2, the parameters provided to clustering algorithms played a significant role in the results these algorithms achieved. These hyper-parameters are often selected by experts heuristically, chosen sequentially and tested by experimentation. Evolutionary algorithms provide the ability to optimise both of these searches, considering alternate solutions in parallel. These algorithms are extremely robust, exhibiting the ability to adapt in dynamic environments. This ability is of critical importance in practical problem solving (Fogel, 1997), such as those considered by the case studies in this dissertation (Chapters 4, 5 and 6). Evolutionary algorithms are also able to address problems for which there are no human experts (Fogel, 1997). An example of a typical EA is presented in algorithm 1.

The initial population of an EA is usually generated at random, performed as uniformly as possible (Eiben and Schoenauer, 2002). The

population comprises a number of individually encoded solutions (genotypes), where the value of genes fall within the bounds of the search space. These candidate solutions do not necessarily have to be unique to the population. Each candidate solution is parsed by a *fitness function*, which uses an objective function to evaluate the solution. The objective function for each task is defined as $D \rightarrow O$, where D is the search space and O the objective. Each candidate solution (genotype) is therefore measured as a direct relation to its objective function value. The resulting value is set as the *fitness* of the genotype, where a higher fitness score delineates a more suitable solution than a candidate with a lower score. The fitness function represents the push toward quality improvements within the population, supporting the selection operator (Eiben and Schoenauer, 2002). The design and implementation of a suitable fitness function for each problem is therefore of critical importance.

```
/* Start with generation 0                                    */
set t = 0;
/* Initialise a random population of individuals              */
initialisePopulation P(t);
/* Determine the fitness of the current population            */
evaluatePopulation P(t);
/* Iterate until termination criterion is met                 */
while not terminate-condition do
    /* Select sub-population for breeding                     */
    P'(t) = selectParents P(t);
    /* Breed parents using recombination (crossover)          */
    applyRecombination P'(t);
    /* Apply genetic mutation stochastically                  */
    applyMutation P'(t);
    /* Evaluate fitness of resultant offspring                */
    evaluatePopulation P'(t);
    /* Replace current generation with best from P'           */
    P(t+1) = selectNextGeneration P'(t);
    /* Increment generation count                             */
    t += 1;
end
```

Algorithm 1: Example pseudo-code for an EA.

An EA executes a number of *generations*, evolving the population until a termination condition is met (algorithm 1). This termination, or *stopping* criteria can be a simple measure - such as the traversal of a fixed number of generations (t) - or fitness of the best scoring solution breaching a particular threshold. These termination conditions could also be complex - such as a measure of the gradient gains over some number of generations - or some measure of diversity amongst the population (Schoenauer, 1997). At the end of each generation, a selection and replacement scheme is used to produce a succeeding population. Here, a selection operator highlights the best individuals within a population for the given problem.

Selection is one of the driving forces behind an EA to ensure increased quality, while reducing genetic diversity (Eiben and Schoenauer, 2002). Identifying the best candidates within a population ensures that resources are concentrated on those that show the most potential. It is important to

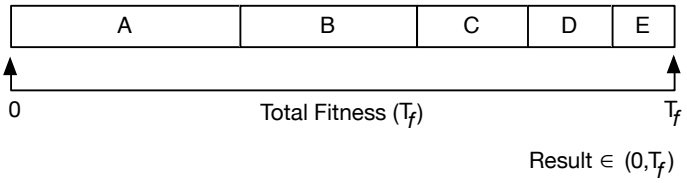


Figure 2.6: An example of roulette wheel selection probabilities for a generation consisting of five genotypes.

note that no new solutions are found by this operator; it simply selects the most suitable candidate solutions, discarding those found to be less suitable. Isolating strong candidates from weaker ones is usually accomplished by filtering them by their fitness scores, as those with a higher fitness have a stronger chance of selection. The genotypes of the succeeding generation's population can be chosen from offspring only or, alternatively, include one or more parents from the current population, through a process called *elitism* (Eiben and Smith, 2003). In either case, this process can be deterministic or stochastic, depending on the operator (Schoenauer, 1997; Eiben and Schoenauer, 2002). Some of the most popular selection operators employed today include: *Fitness Proportionate Selection* (Mitchell, 1997), *Rank Based Selection* (Mitchell, 1997), *Tournament Selection* (Mitchell, 1997) and *Elitist Selection* (Eiben and Smith, 2003).

Fitness proportionate selection, also known as *roulette wheel selection*, uses the assigned fitness scores of a population to calculate and assign a probability of selection to each genotype. The probability of selecting an individual, i, where F_i is the fitness of the individual in the current population and N is the number of genotypes within the population, is given by equation 2.2.

$$P_i = \frac{F_i}{\sum_{j=1}^{N} F_j} \tag{2.2}$$

The result of calculating a probability of selection for each candidate with respect to the total fitness of a population is illustrated in Figure 2.6. Here, candidates with a higher probability occupy more space on the proverbial "roulette wheel" and therefore have a higher chance of being selected.

The *Rank Based Selection* operator differs from fitness proportionate selection by assigning a probability to a candidate according to their rank, rather than their fitness. All candidates of a population are ordered by their fitness in descending order and assigned a probability based on their position with respect to the rest of the population.

Basing selection strategies on global, shared information about the whole population arguably makes both fitness proportionate selection and rank-based selection unsuitable for parallel-based EAs. To overcome this, *Tournament Selection* elicits a set of k candidates from the population with uniform probability and has them engage in a tournament. The victor of the tournament can be chosen either deterministically - as the individual with the highest fitness - or probabilistically, where the probability of the individual winning is proportional to its fitness (Miller and Goldberg, 1995).

Finally, *Elitist Selection*, or *elitism* is a process that links the lifetime of an individual genotype proportionately to its attained fitness. The result is that suitable solutions are retained over a number of generations, rather than being discarded immediately. When elitism is implemented, the best candidate will be copied (unmodified) into the succeeding generation. Depending on the configuration, more than one candidate may be copied in this fashion. Elitism is usually implemented with another selection operator, such as fitness proportionate selection, rank-based selection or tournament selection (Eiben and Smith, 2003).

During the breeding process (algorithm 1) those genotypes chosen by selection methods other than elitism, are cloned. These clones are subjected to genetic operators, which modify the genotype in order to maintain genetic diversity in succeeding populations. This push toward genetic diversity is in contrast to the selection operators and ensures novel solutions are developed and tested by the algorithm (Eiben and Schoenauer, 2002).

Genetic operators are generally classified as one of two types: *Mutation* and *Crossover* (*Recombination*). Mutation operators are stochastic (Eiben and Schoenauer, 2002), mutating or altering specific elements within the genotype to genetically modify the offspring. The purpose of mutation operations is to simulate transcription errors found in nature. While there is no standard method of implementing mutation, general trends are toward

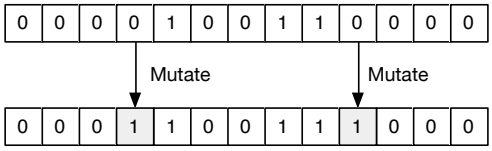


Figure 2.7: An example of bit string mutation (flip bit).

modifying genes of a cloned genotype by a very low probability (Eiben and Schoenauer, 2002). *Bit String Mutation* is one of the more popular mutation operators applied to genotypes encoded as bit strings. These genotypes are described by $\{0, 1\}^n$, where the genotype consists of n genes with values of 0 or 1. One popular example of a bit string mutator is *Flip Bit*. This operator considers each gene (bit) independently, flipping or inverting the value based on a specified mutation rate (probability). Figure 2.7 illustrates the concept of bit string mutation on an example genotype, where two genes were mutated using the flip bit mutation operator.

While there are many other mutation operators available, most of those remaining are applicable to genes represented by continuous values, integer or floating points. This type of encoding is not implemented by the method proposed in this dissertation and are thus not discussed here. Bäck et al. (2000) and Deep and Mebrahtu (2011) provide further insights into these and other genetic operators.

The purpose of crossover operators is to take two or more fit parents and exchange their genetic material to form novel solutions. The new offspring offer solutions that incorporate inheritance information (genetics) from multiple parents, describing new solutions. *K-point crossover* is one of the most popular operators applicable in today's EA implementations (Jansen, 2013).

The k-point crossover technique is applied by selecting k randomly assigned crossover positions in the parent genotype. Offspring are created by taking segments of each parent, alternating between the two genotypes at each crossover point. This process, illustrated in Figure 2.8, allows one or more crossover points to be specified, permitting parents to contribute toward offspring at a gene, rather than segment, level (*Uniform Crossover*).

It is important to note that the use of crossover to form offspring may not be applicable where the ordering of variables is important (for example,

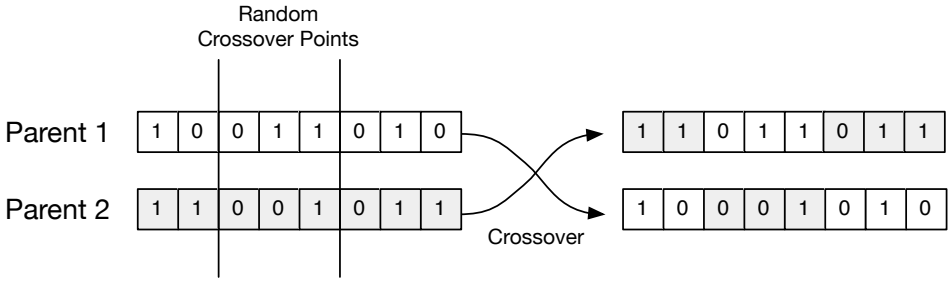


Figure 2.8: An example demonstrating 2-Point Binary Crossover.

the case study described in Chapter 6). For this reason, a number of EA paradigms forgo the use of crossover operators all together, while others consider crossover an integral variation operator (Eiben and Schoenauer, 2002).

2.3.3 Feature Set and Hyper-Parameter Optimisation

Genetic Algorithms (GAs) (Eiben and Smith, 2003) are algorithms belonging to the EA class, which generate solutions to optimisation problems using natural evolutionary processes. These algorithms have been used in classification methods as optimisation techniques for a number of problems, such as feature selection. Park et al. (2006) proposed a feature selection algorithm which produced three kinds of classifiers: *Naive Bayesian with Kernel Estimation* (NBK), *Decision Tree J48* and *Reduced Error Pruning Tree* (REPTree). The results obtained through experimentation showed that classifiers based on decision trees exhibited a higher recall and thus higher overall accuracy than the Naive Bayesian classifiers. This outcome was echoed by Chen et al. (2008) who, using the tabu search algorithm (Glover, 1989, 1990), demonstrated that a decision tree-based classifier outperforms a Bayes-based classifier.

In early works, Vafaie and De Jong (1992) tested GAs against a traditional statistical method of feature selection. Through experimentation the authors showed that a significant reduction in the number of features required for classifying their targets was obtainable using a GA. These reduced feature sets simultaneously yielded higher accuracy in classification when compared to other methods.

Huang and Wang (2006) proposed the use of a GA for selecting feature subsets and parameters for use in SVMs. The *de facto* standard search for hyper-parameter optimisation is an exhaustive one, often referred to as a *grid-search* (Hsu et al., 2003). In this work, Huang and Wang (2006) stressed that the kernel parameters and feature sets supplied to an algorithm had a profound impact on classification accuracy. The objective of this research was to simultaneously optimise both the kernel parameters and the feature subset without decreasing the SVM classification accuracy. Through this optimisation, the task of classification is reduced, while the classification accuracy is retained, or possibly improved. Huang and Wang (2006) demonstrated that, compared to a grid-search, their GA approach was able to discover a more optimal (lower dimensional) feature set, which resulted in improved classification accuracy and task efficiency.

An integral part of an *Intrusion Detection System* (IDS) is the ability to classify IP traffic on a network. Li (2004) presents an interesting GA approach, showcasing the use of GAs in problems within the IDS space. Here, the author stresses that the parameters governing the operation of the GA are crucial as they heavily influence the effectiveness of the algorithm. Garcia-Teodoro et al. (2009) asserts the main advantage of using GAs in IDS systems is their flexible and robust global search methods that converge to a solution from a number of approaches without any prior knowledge of the problem.

Goss and Nitschke (2013b) used GAs for automatically selecting parameters for the k-means clustering algorithm in IP traffic classification problems. In this work, the number of application protocols (or clusters, k) present in a data set of IP flow traces was determined by a GA. Each value of k was encoded in a bit string genotype, whereafter the clustering algorithm was executed. *Silhouette Cluster Analysis* (Rousseeuw, 1987) was used to determine the success of the clustering process and described the fitness of each genotype. Goss and Nitschke (2013b) found that searching for an optimal value of k was more efficient using a GA than searching for the same using traditional, sequential (grid) searches.

The use of EAs for optimising feature sets and hyper-parameters for complex classification tasks is still very much in its infancy. The applications discussed in this section demonstrate the ability of GAs to find

more optimal feature subsets and optimised hyper-parameters for both supervised and unsupervised algorithms. Much of the research within the field of parameter optimisation relates to SVM algorithms, however a number of other algorithms could also benefit. SVMs are typically easier to implement than an ANN and require fewer hyper-parameters be tuned before reaching a global optimum (guaranteed). ANNs, although previously very popular, have - until recently - been the second choice for many systems, owing to complexity of topology design, the high number of tunable parameters, and complexities related to result interpretation. ANNs were used in a number of approaches studied in this chapter, often using static topologies and gradient-based weight adaptation techniques. *Neuroevolution* (NE) uses EAs to search for more optimal topologies and weight configurations for ANN classifiers. While there are many NE algorithms, two distinctions can be used to broadly group them. The first type are those that evolve only the weights of an ANN. These algorithms are often referred to as *Conventional Neuroevolution* (CNE) (Miikkulainen, 2010) algorithms. The second type are those that evolve both the topology and the weights, commonly referred to as *Topology and Weight Evolving Artificial Neural Networks* (TWEANNs) (Sher, 2012).

Common to all EAs, solutions for NE algorithms are encoded as genotypes, representing the design of a particular classifier, or *model*. These models can be encoded using a *direct* or *indirect* encoding scheme. In direct encoding, every neuron and connection is specified in the genotype, while indirect encoding simply specifies how the network should be generated (Kassahun et al., 2007). Each model described by an NE algorithm is evaluated on some task to ascertain its fitness. Unlike supervised ANNs, which require a training set of input and corresponding output vectors, NE algorithms require only the ability to measure a network's performance at some task. ANNs have already shown great promise in classifying IP traffic (Goss and Botha, 2012; Goss and Nitschke, 2013a,b), however the topology of the ANNs tested was fixed for each problem (based on heuristic configurations). Goss and Nitschke (2014) demonstrated that, by adapting the ANN topology (hidden neuron configuration) of an IP traffic classifier, an increase in accuracy was possible. In this work, as in that of Goss and Botha (2012), Goss and Nitschke (2013a) and Goss and Nitschke (2013b),

the tuning of the topology's weights was performed using only a gradient-based search, which has many potential shortfalls and limited guarantees for convergence.

Given the findings of the works discussed in this chapter, the use of GAs for optimising feature sets and selecting parameters governing learning algorithms show marked improvements in classification accuracy and task efficiency. This, in theory, should improve the accuracy, completeness and efficiency of complex classification tasks, such as IP traffic classification, NBAD, and handwritten digit recognition. As far as could be ascertained, the only research where EAs manipulate the structure and weights of IP traffic classifiers was by Goss and Nitschke (2014). Here, the authors used an EA to search for a suitable configuration of hidden layers and their respective nodes. The results showed that an increase in accuracy was achievable by adapting (customising) the topology of an ANN classifier for each application protocol. A full TWEANN implementation in IP traffic classification, where topology and weights of each classifier are determined by an EA has, according to all literature reviewed, yet to be researched. Furthermore, training TWEANNs for identifying anomalous network traffic (Chapter 5) and developing more efficient (optimal), accurate CNN classifiers for identifying handwritten digits have yet to be explored by researchers. A general method capable of addressing each of these types of complex classification problems, and providing increases in completeness, accuracy or efficiency will therefore be a novel contribution to the field of ML.

2.4 Conclusion

ML techniques were identified as the current state-of-the-art technologies in a broad range of classification tasks, however many implementations still have a strong dependency on human expert intervention for feature set extraction and optimisation, data set annotation, and model design processes. Inefficiencies in these areas result in less complete systems, where overly complex classifiers predisposed to human error are commonplace. The ability to optimise feature sets, annotate data sets and produce optimised classifiers is synonymous with most ML classification tasks, including IP traffic classification, anomaly detection and image recognition. The ability to automate these processes, reducing dependency on human experts will lead to more complete, accurate and task-efficient systems compared to current methods.

The following chapter outlines a novel method for addressing these deficiencies, where optimised feature selection, pattern discovery, and optimised classifier production processes are performed automatically. The efficacy of this new method is tested using three distinct case studies: *IP Traffic Classification* (Chapter 4), *Anomaly Detection* (Chapter 5) and *Handwritten Digit Identification* (Chapter 6).

Chapter 3

Automated Pattern Identification and Classification (APIC)

An efficient, flexible method for identifying optimised feature sets, discovering patterns in data sets, and crafting more optimal classifiers is required for a wide variety of classification tasks across multiple domains. This process needs to be automated, adapting to a variety of pattern types within a specific data set. The drive toward automation in *Machine Learning* (ML) is fast becoming more prevalent in modern ML implementations (Feurer et al., 2015), demonstrating a paradigm shift poised to accelerate the task of data analysis in today's modern big data environments. Increased automation provides opportunities for non-experts to implement ML in various applications using off the shelf software (Feurer et al., 2015). To achieve high levels of automation, a candidate solution should be capable of addressing each of the major components currently associated with ML deployment, replacing human expert tasks with methods to perform the same automatically. These major tasks have been identified and are described in this research as *Feature Selection*, *Pattern Discovery* (Data set Annotation) and *Classifier Production* (model development). Various methods for automating each of these components have been investigated in the publications listed in Appendix A. The method proposed in this chapter incorporates the best solutions for each of these works into a single ML pipeline, suitable for application in general

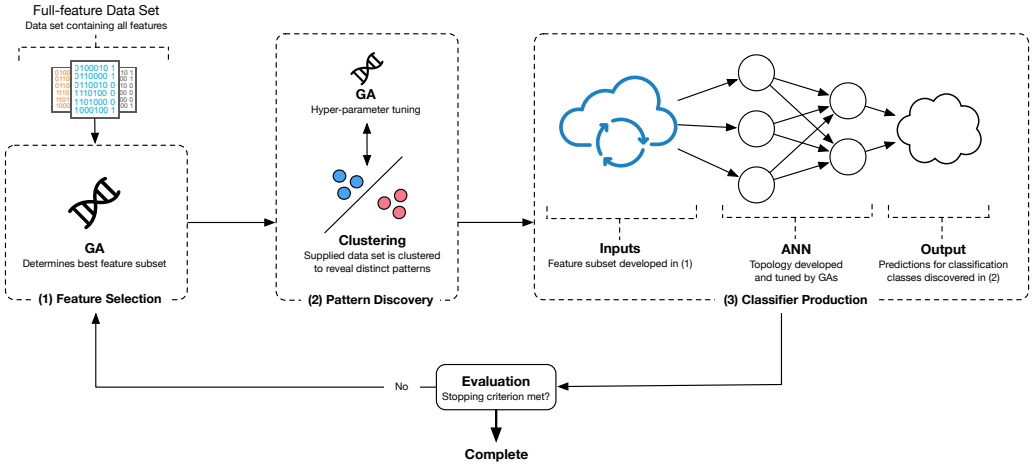


Figure 3.1: A high-level view of the APIC method for automated pattern recognition and classifier production.

classification tasks.

The goal of the *Automated Pattern Identification and Classification* (APIC) general method is to identify distinct patterns in a variety of mixed, noisy data sets, automatically producing efficient (low-complexity), accurate classifiers for any given task. These tasks include, for example, classifying application protocol flows traversing *Internet Protocol* (IP) networks, images processed via visual sensors or voice recognition.

A method addressing these tasks should be complete, offering a classifier for each possible pattern discovered in the data set. The accuracy of these classifiers should be competitive, equal to, or surpassing, the accuracy achieved by classifiers manually created by experts for the same task. A fine balance needs to be reached when creating classifiers, ensuring the overall completeness of the system is maintained, while ensuring a sufficiently high degree of accuracy is achieved across these classifiers.

This chapter presents the APIC method (Figure 3.1), which improves the completeness, accuracy and efficiency of existing classification systems. APIC is an ML pipeline, comprised of feature selection, pattern discovery and classifier production processes. These processes use unsupervised ML algorithms, including *Genetic Algorithms* (GAs), *Topology and Weight Evolving Artificial Neural Networks* (TWEANNs) and *clustering* algorithms. In the sections that follow, the feature selection process

(Section 3.1) searches for a more optimal (reduced) feature set to describe each data set. New patterns are automatically identified and grouped (clustered) in these new data sets (Section 3.2). The clustered data set acts as training sets for the development of classifiers (Section 3.3), such as TWEANNs. These classifiers are trained to identify future instances of each pattern in a given data set. The ability to classify new datum in this manner is important in a number of classification tasks, including IP traffic classification. The APIC method presents a novel approach for the formation of classifiers for a number of complex classification tasks. The first step in this process is to accurately describe data sets for each problem, using optimised feature sets. The following Section describes APIC's approach to identifying the best features for each data set.

3.1 Feature Selection

A feature vector is an n-dimensional vector of numerical features that describe an object. These vectors are used to train ML algorithms in identifying future instances of an object within a mixed data set. Feature vectors with increased dimensionality are more task intensive and incur extra computational overheads when training ML algorithms. For this reason, it is best to discover a more optimised feature subset for each problem. The classification accuracy achieved using this feature subset must equal or surpass that of the original feature set. The APIC method reduces the task of training classifiers by removing redundant or irrelevant features through a process of feature selection. Using a similar approach to that of Vafaie and De Jong (1992) and Huang and Wang (2006), a GA is used to identify near-optimal feature subsets for describing patterns in each data set.

The APIC method encodes each feature subset genotype as a bit string, with the number of bits equal to the dimensionality of the full feature set. Each gene represents a particular dimension, or feature, where "1" indicates the inclusion and "0" the exclusion from the current feature subset. For example, consider the input data set 3.1:

$$\begin{pmatrix} 0.8 & 1.2 & 2.3 & 0.3 & 1.1 & 0.2 & 2.1 & 0.3 \\ 1.2 & 2.3 & 0.3 & 1.1 & 0.1 & 0.0 & 2.1 & 0.4 \\ 2.0 & 1.2 & 1.3 & 0.0 & 2.1 & 0.6 & 0.9 & 1.4 \end{pmatrix} \tag{3.1}$$

This set is comprised of three vectors (rows), described by eight features (columns). The bit string genotype describing this data set will subsequently be eight bits long, where the genotype 00110011 describes a feature subset including the third, fourth, seventh and eighth features; 11001010 the first, second, fifth and seventh features; and 11110000 the first four features.

The APIC method searches for the lowest dimensional feature subset, where classifiers trained using the feature subset achieve performance results that equal or exceed those of classifiers trained using the full feature set. In this method, a low dimensional feature subset is encouraged by applying a penalty to each genotype's fitness, proportional to its dimensionality. This ensures that genotypes with lower dimensionality are more favourable than others, promoting the reduction of unnecessary features within the data.

The initial genotype population describing feature subsets is generated with bits toggled between "1" and "0" randomly. Using these feature subsets, the supplied data set can be partitioned, or divided, into a distinct number of groups. Each of these groups describes a unique pattern (type of object) in the data. A clustering process is used to group (cluster) the data, annotating each datum with its associated cluster identifier. More information on this process is provided in the following section.

3.2 Pattern Discovery

In certain classification tasks, labelled data sets are not readily available. In these cases, human experts are often used to annotate each datum with its associated class label. This process is inefficient as this manual annotation takes a significant amount of time and is prone to error. Worse, a human expert may not be available for annotating data sets for a particular task. In these cases, unsupervised learning (Section 2.2) offers alternatives to manual annotation processes.

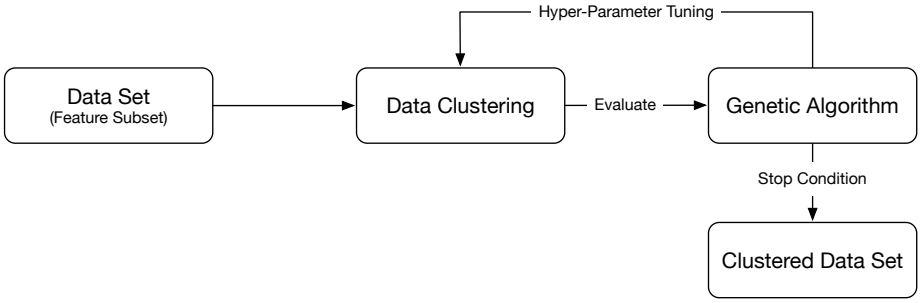
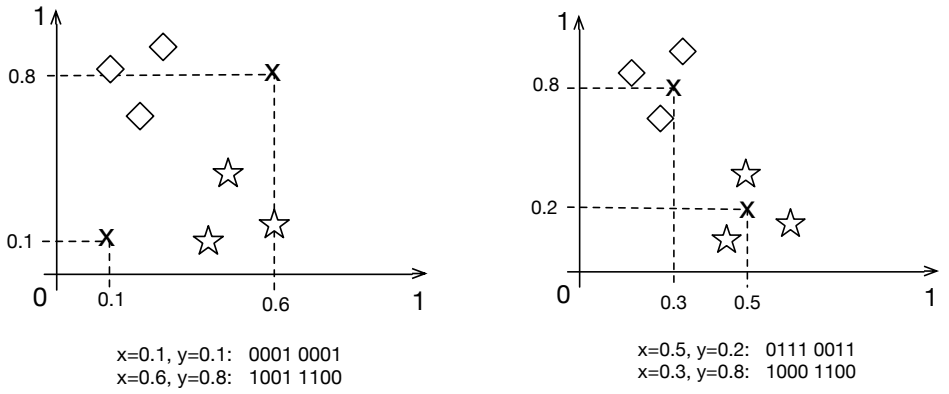


Figure 3.2: A GA-controlled clustering of data extracted using a feature subset.

The APIC method is capable of grouping feature vectors of a particular type (pattern) using unsupervised learning algorithms, illustrated in Figure 3.1 (2). Each group, or *cluster*, is automatically annotated with its cluster id, producing training sets for the production of supervised or semi-supervised classifiers. The method allows the number of groups to be set explicitly, or to be inferred automatically, through a discovery process. Where the number of patterns, or *targets*, are not specified, the method uses algorithms such as *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) (Ester et al., 1996) to discover these automatically. Where the number of targets, k, are known *a priori*, algorithms such as k-means are used to group datum into k clusters.

Using these algorithms, the input data described by the current feature subset is clustered, grouping related datum. The hyper-parameters controlling this clustering process are unique to each problem. Accommodating for this, the APIC method uses a GA to search for optimised values for these parameters, ensuring tighter (more optimal) cluster formation. Figure 3.2 provides an outline of the APIC pattern discovery process, where the hyper-parameters controlling the clustering process are optimised automatically using a GA.

The parameters guiding the clustering process of each problem are encoded as bit string genotypes. Where k-means partitive clustering is used, the location of each centroid is encoded in the bit string. At this point, each value of the feature subset is assumed to have been normalised to a value between 0 and 1. The position of each centroid is represented as four bits within the genotype, which reduces genotype complexity and



(a) Initial Generation

(b) Final Generation

Figure 3.3: A GA encodes the location of two centroids on a two dimensional lattice for a k-means clustering problem. By tuning the locations of each centroid, the GA is able to optimise the search process for centroid positioning. In (a), two centroids (denoted by x) are positioned randomly on the lattice. In (b), the centroids have positioned themselves in the centre of the two distinct patterns present in the data (the diamonds and the stars)

allows for up to 16 distinct positions in the feature space. This granularity is sufficient for the k-means algorithm, as the centroid positions in the feature space are fine-tuned by the k-means algorithm. Figure 3.3 illustrates the encoding of two centroids in an example two dimensional problem. Figure 3.3a illustrates an example of a genotype from the initial population, while Figure 3.3b demonstrates the evolution of centroids in the final generation.

For problems where the number of clusters is unknown, algorithms such as DBSCAN are used to infer these automatically. The DBSCAN algorithm requires values for two hyper-parameters, namely the neighbourhood radius size *Epsilon* (ϵ) and the *minimum number of points* ($minPTS$) per cluster. These two parameters are each encoded as eight bit integers, concatenated to form a complete genotype.

For all clustering algorithms, the initial population of genotypes of hyper-parameters is generated randomly. Each genotype is evaluated by a fitness function, which executes an instance of the clustering algorithm configured with the decoded parameters. Each algorithm is permitted to run until the stopping criteria is met. For the k-means algorithm, a stop

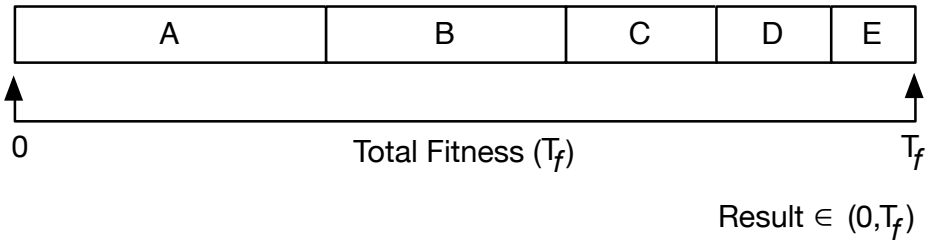


Figure 3.4: An example of roulette wheel selection probabilities for a generation consisting of five genotypes.

will occur when no changes are made to centroid locations during a single pass. Algorithms such as DBSCAN, for example, stop once all datum have been passed and evaluated by the algorithm. After clustering is complete, the resulting clustered data set is evaluated using the *Silhouette Cluster Analysis* (Rousseeuw, 1987) method. The average silhouette for a clustered data set is given by Equation 3.2.

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{3.2}$$

Where $a(i)$ is the average measure of dissimilarity between datum i and another datum within the same cluster, $b(i)$ is the lowest average measure of dissimilarity between datum i and any other cluster. The $s(i)$ for the clustered data set is therefore a measure of how successful the clustering process is. These values are represented as values between -1 and 1, with a value toward the latter signifying more appropriate (tight) clustering. Each genotype is assigned a fitness value equal to the silhouette score achieved post-clustering. Once all genotypes have been scored, a succeeding population is generated for the next generation. Evolution of the successive population is ensured by application of EA operators and processes after each generation. First, *elitism* (Eiben and Smith, 2003) is applied, so that the highest scoring genotypes are transferred directly to the next generation's population. Next, pairs of "parent" genotypes are selected from the current population using *roulette wheel selection* (Al Jadaan et al., 2008). This process ensures a high degree of probability for selecting fitter

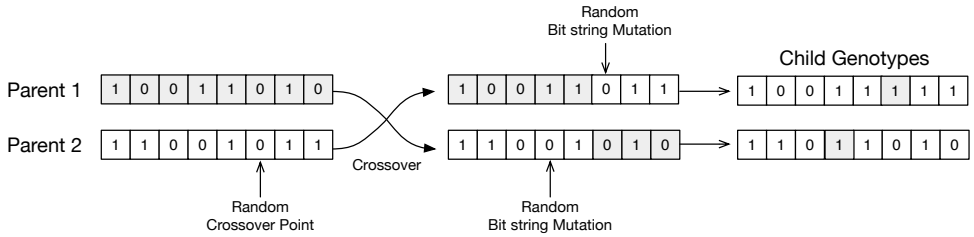


Figure 3.5: Application of EA operators on a pair of parent genotypes

parent genotypes. Where F_i is the fitness of genotype i in the current population, the probability of selection is given by Equation 3.3.

$$P_i = \frac{F_i}{\sum_{j=1}^{N} F_j} \tag{3.3}$$

Where N is the number of genotypes within the current population. The probability of selecting a genotype based on fitness is illustrated in Figure 3.4

The pairs of parents are recombined using single-point crossover to produce two child genotypes (Eiben and Smith, 2003). For each of these genotypes, bit string mutation is applied to flip one bit, resulting in a slightly altered resultant genotype. Both of these new genotypes are subsequently added to the next generation's population. These selection and recombination processes, illustrated in Figures 3.4 and 3.5, respectively, continue until the maximum number of allowed genotypes have been added to the successive generation's population.

Once a sufficient silhouette score has been attained for a clustered data set, the GA is said to have converged and the best scoring solution is forwarded to the classifier production process. A fixed number of generations is specified as a stop condition where, if breached, the best solution at that point is used. The following section describes the third and final process of APIC, the development of customised classifiers.

3.3 Classifier Production

The literature studied in Chapter 2 revealed that a number of works used human experts to develop classifiers through a manual, heuristic process.

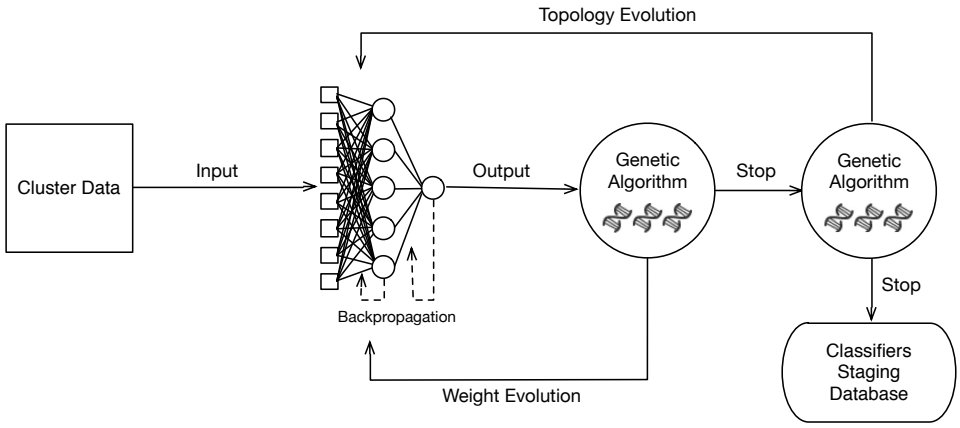
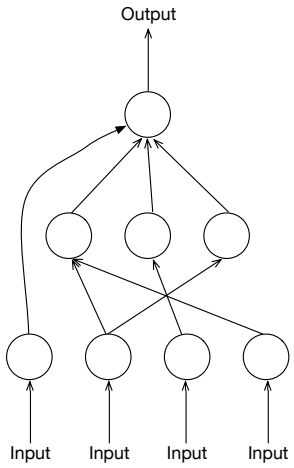


Figure 3.6: APIC's GA-controlled classifier production process.

These classifiers were often not general, requiring a significant amount of customisation for each specific task. In most cases, a manually annotated training set was required for training each classifier. Both the manual labelling of training sets and the manual development of classifiers hinder the scalability of these systems for complex classification tasks. The APIC method addresses these issues, firstly by automatically distinguishing and labelling distinct patterns in unlabelled data sets (Section 3.2) and secondly, by automatically developing customised classifiers to identify future instances of these patterns efficiently and accurately, illustrated in Figure 3.1 (3). Figure 3.6 illustrates the APIC process for developing customised classifiers for any complex classification problem.

Each classifier is customised, providing increased accuracy by adjusting the topology of the classifier to fit each pattern. This customisation is controlled by two GAs - the first evolves the topology and the second determines near-optimal weights for each connection. Topology evolution discovers the best configuration of hidden layers, neurons and node connectivity. While the GA's ability to provide a global search technique often results in near-optimal solutions, it is rather inefficient in a fine-tuned local search (Yao, 1999).

For this reason, a hybrid training approach is adopted, where the weight-evolving GA is used to globally optimise the *Artificial Neural Network* (ANN) weights, while the *backpropagation algorithm* (Rumelhart et al., 1988) adapts these weights over a number of iterations in search of



$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 3.7: Network topology and its associated connectivity matrix.

the best settings. The topology of the network is encoded using *direct encoding* (Kassahun et al., 2007), with each genotype represented using bit string notation. A connectivity matrix can be drawn for each genotype, showing connections between respective nodes of the ANN (Figure 3.7).

In Figure 3.7, each row of the connectivity matrix represents a node and its connectivity to other nodes in the network. The first row describes the input at the bottom left of the ANN topology, which connects directly to the output neuron (indicated by setting the eighth bit to "1"). The next row describes input two, which connects to both node five and seven, indicated by a "1" in columns five and seven. It is important to note that inputs do not connect directly to one another.

Using this encoding scheme, APIC randomly generates genotypes for the initial generation's population, where each genotype specifies a proposed topology for a TWEANN classifier. A second, weight-evolving GA allocates the initial weights to each connection described by the topology evolving GA (Figure 3.6). The weights for the network are encoded using direct encoding, formatted in bit string notation, where each connection weight is represented by eight bits.

Using the example in Figure 3.7, a total of eight connections are present, therefore the genotypes of the weight-evolving GA will consist of 64 bits. The initial population of the weight-evolving GA is generated randomly, where each genotype specifies the weight values of each respective connection.

The newly initialised network enters a training phase, where data from

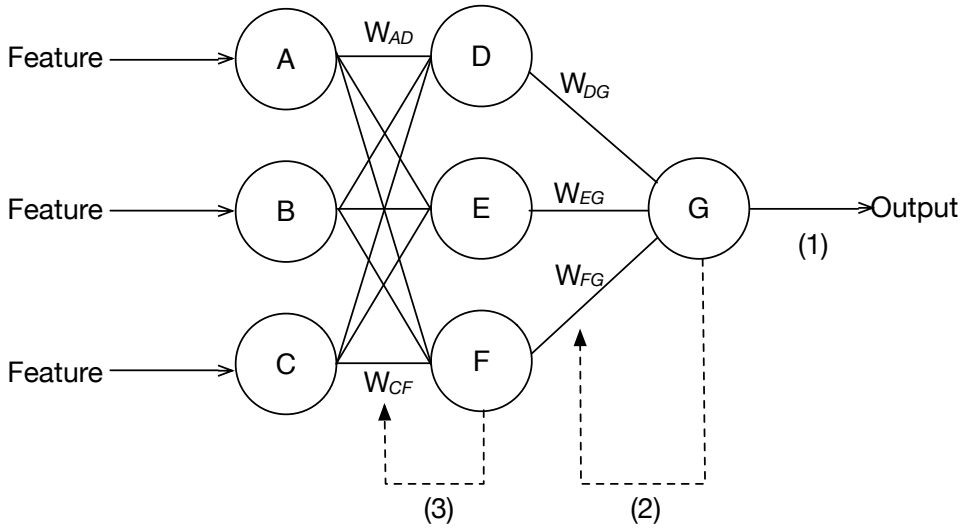


Figure 3.8: An illustration of the backward propagation of error (backpropagation) algorithm, where errors between target and actual outputs are used to fine-tune the weights of an ANN.

the training set is fed through the network in random order over a number of iterations (epochs). This process is repeated until the average *Mean Squared Error* (MSE) of a particular training cycle (epoch) is sufficiently small, or the maximum number of permissible epochs has been reached. Equation 3.4 provides the formula for calculating the average MSE of each epoch.

$$MSE = \frac{\sum_{i=0}^{n}(O_i - A_i)^2}{n} \tag{3.4}$$

Where n is the number of patterns (datum) in the training set, O the desired value (target) and A, the actual value (output) obtained.

The backpropagation algorithm is illustrated in Figure 3.8, where outputs generated for each datum are compared against the input's target to establish the error (Figure 3.8 (1)). The error for each datum is given Equation 3.5.

$$Error_G = Output_G(1 - Output_G)(Target_i - Output_G) \tag{3.5}$$

$Output_G$ is the output of neuron G for an input vector with target value

$Target_i$. Using this value, the algorithm adjusts the weights of each neuron connecting to the output, depicted in Figure 3.8 (2), using Equation 3.6.

$$W_{FG}^{+} = W_{FG} + (Error_G * Output_F) \tag{3.6}$$

Where W_{FG}^{+} is the new weight applied to the connection between F and G. Equation 3.6 is used to adjust all weights connected directly to an output neuron. The weights of neurons indirectly connected (hidden layers) also need to be adjusted, however, as there is no target for these they cannot be calculated directly. The targets for these neurons are "backpropagated" by the algorithm from the output neurons. The first part of this process is to calculate the error of the connecting neuron. This error value, for neuron F (as an example), is given by Equation 3.7.

$$Error_F = Output_F(1 - Output_F)(Error_G W_{FG}) \tag{3.7}$$

Once the error is known, the value of each hidden neuron, W_{CF} shown in Figure 3.8 (3), is calculated using Equation 3.6. By altering weights in this manner, the average MSE of the network is reduced over a number of generations. This process continues until either the average MSE of the network is lower than a minimum set value for a complete epoch, or the maximum number of epochs is reached. If either of these conditions is met, the network is said to have converged.

At this point, the weight-evolving GA will ascertain the fitness of the network (and thus the current genotype) by passing the clustered data set through the network and recording the average MSE attained. A stop condition is triggered if the MSE of the current network is less than a set minimum, or the GA has executed more than a maximum number of epochs.

Upon receiving a stop condition from the weight-evolving GA, control is handed back to the topology-evolving GA to evaluate the best solution provided by the weight-evolving GA. If the MSE of the best scoring network is sufficient, or a maximum number of topology-evolving epochs

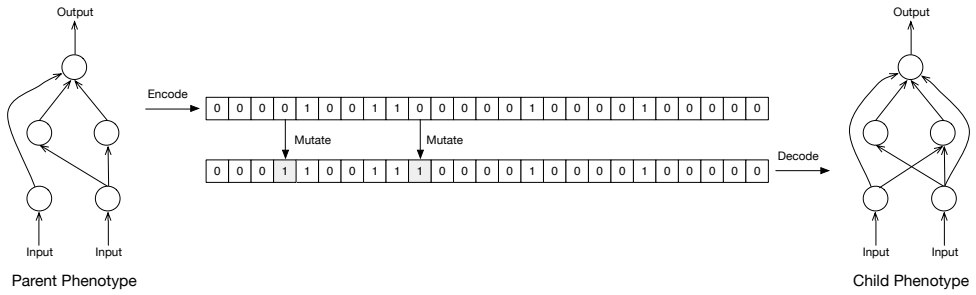


Figure 3.9: An example of a topology-evolving GA, where ANNs are encoded as bit strings. These genotypes are adapted using bit string mutation.

have been exceeded, the GA triggers a stop condition and the best scoring network at that point is retained. This network is subsequently pushed to the classifier staging database, where all new classifiers are collected awaiting final evaluation.

If no stop condition is called, the topology-evolving GA (Figure 3.6) executes variation operators to produce the next generation's population. Elitism is used to ensure the most fit genotypes are transferred directly to the next generation. Unlike the feature selection GA (Section 3.1), no cross-over operators are applied. This is because EAs, which use direct encoding, relying on crossover operators do not perform well in searching for near-optimal ANN architectures (Yao, 1999). Another reason is that, in certain cases, the ANN topology encoded is complex and inverting a single bit may render the entire topology invalid. Parent genotypes are selected from the current generation and mutated at a set rate, adding and removing connections between neurons (Figure 3.9).

Once all classifiers have been produced and deployed to the staging database, the feature-selection GA's fitness function is called to calculate the MSE (Equation 3.4) of the collective set of new classifiers. If the resulting threshold is below the minimum acceptable value, a stop condition is called and the classifiers are deemed suitable for describing patterns in the supplied data set. If no stop condition is called, the feature-selection GA tests the next feature subset and the process repeats until the average MSE (formula 3.4) of the staging classifiers is acceptable, or the maximum number of feature-selection GA generations has been reached.

3.4 Conclusion

The task of identifying and classifying future instances of interesting patterns in complex, mixed data sets is common to a number of scientific domains. In most cases, a significant amount of manual effort is required by domain experts to develop systems for each specific task. This chapter presented APIC, a general method that reduces human involvement in the development of classifiers for a variety of classification tasks. The method is capable of automatically selecting and testing the best features for each task, discovering arrangements in data sets automatically and training customised classifiers to identify future instances of each pattern. The method is an ML pipeline, where principle algorithms can be substituted with alternatives that better fit each particular problem. The APIC method is a novel contribution to automated identification and future classification of distinct patterns within mixed, noisy data sets. The following three chapters present case studies, where the efficacy of APIC is evaluated for accomplishing complex classification tasks in a broad range of applications.

Chapter 4

IP Traffic Classification

Computer networks have grown substantially in recent years due, in part, to the increasing global reach of the Internet, network access speeds and content availability. The Internet - the world's largest public-access network - has emerged as a key enabler for business and personal communication. The proliferation of computer users consuming Internet resources spurs the development of numerous applications, using both *client-server* and *peer-to-peer* (P2P) communication architectures. In most cases, these applications are not well understood and are subsequently difficult to control (Zhang et al., 2009a). Each application communicates using a distinct *application protocol*, generating conversations that compete for limited network resources. This competition needs to be managed to ensure correct priority assignment of resources to each application, based on their designated importance. Organisations, especially *Internet Service Providers* (ISP), need to be aware of traffic flowing across their networks promptly, allowing them to react quickly, thus maintaining their business goals (Nguyen and Armitage, 2008). The classification and management of network traffic is therefore an essential tool for network and system security management (Zhang et al., 2013). *Machine Learning* (ML) (Russell and Norvig, 2009) algorithms have been identified as the current state-of-the-art technology for classifying *Internet Protocol* (IP) traffic, where *supervised* and *unsupervised* learning methods and *Evolutionary Algorithms* (EA) (Eiben and Schoenauer, 2002) have been tested for improving the effectiveness of IP traffic classification systems.

This chapter evaluates the *Automated Pattern Identification and Classification* (APIC) general method for improving the task of IP traffic classification, using *Topology and Weight Evolving Artificial Neural Network* (TWEANN) classifiers. During this evaluation, comparisons against existing systems are drawn, and the results assessed. This case study demonstrates how the task of manual feature selection, application protocol distinction and best classifier (signature) design and production processes, commonly executed manually by human experts, can be automated using the APIC method. This automation expedites the development of classifiers, leading to increased completeness compared to similar methods. Furthermore, APIC's ability to discover more optimal models for each classifier produces results that rival and, in certain cases, exceed those of comparable methods for the majority of application protocols tested.

4.1 IP Traffic Classification

Zhang et al. (2009a) states that the goal of an IP traffic classification system is to understand the type of traffic traversing networks as it evolves in both scope and complexity. These classification systems play an important role in modern network and security architectures, providing an essential component in *Quality of Service* (QoS), *Intrusion Detection System* (IDS) and *Lawful Interception* (LI) services (Zhang et al., 2013; Nguyen and Armitage, 2008). Szabo et al. (2007) asserts that the performance of an IP traffic classification system can be measured by two metrics, namely *accuracy* and *completeness*.

The completeness of the system refers to the availability of a trained classifier for identifying a particular application protocol. Accuracy refers to the ability of such a classifier to precisely identify an application protocol using the trained classifier. According to Kim et al. (2008), two common metrics for measuring the performance accuracy of IP traffic classification systems are *overall accuracy* and *F-Measure*.

Overall accuracy (Equation 4.1) is expressed as a ratio of the sum of correctly classified application protocols to the sum of all samples considered.

$$Accuracy = \frac{correct\ classifications}{number\ of\ samples} \tag{4.1}$$

F-Measure, often used in the field of information retrieval, is given by Equation 4.2. Here, *precision* is the ratio of correctly identified application protocols over all those predicted within a class and *recall* the ratio of correctly identified application protocols over the *ground truth* of the class.

$$F_{Measure} = \frac{2 * precision * recall}{precision + recall} \tag{4.2}$$

It can therefore be stated that the performance accuracy of an IP traffic classification system is directly proportionate to the number of correctly identified application *flows* traversing a network. A flow is defined by a number of successive IP packets sharing a common five-tuple of *source* and *destination* IP addresses, *source* and *destination* ports and *protocol* within a specific interval (Goss and Nitschke, 2013a; Zhang et al., 2013). These packet exchanges can be viewed in two directions: from the *source to the destination*, or from the *destination to the source* (Alshammari et al., 2009b).

A unique *signature* is required to identify the underlying application protocol of a flow. The term "signature" is generally used to describe a recipe for uniquely identifying a particular application protocol or category. These signatures are often provided by vendors of traffic management devices in one or more signature packs; these packs need to be updated regularly, catering to advances in application protocol development (Goss and Nitschke, 2013a). Each vendor is responsible for the creation of their own proprietary signature packs, compatible with their systems. As the recipe for each signature is often a closely-guarded secret, accuracy and performance variances between vendor equipment is not uncommon (Goss and Nitschke, 2013a). Over the years, a number of techniques have been proposed for the development of these signatures. The following subsections describe these approaches, highlighting their benefits and limitations.

4.1.1 Classic Port Matching

Traditionally, signatures were reliant on information inferred from passing IP packet headers to classify a flow. These IP headers, depicted in Figure 4.1, were traditionally 160 bits (20 bytes) long and included routing information,

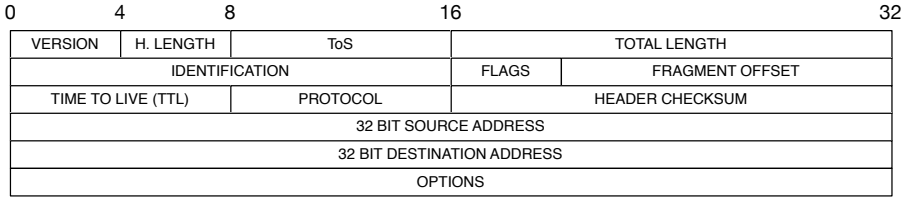


Figure 4.1: Structure of an IPv4 Packet Header.

such as the sender (source) and recipient (destination) host addresses and the IP protocol. Of these IP protocols, *Transport Control Protocol* (TCP) and *User Datagram Protocol* (UDP) are the most popular for transporting data across networks. Each of these protocols has a specific header format, encapsulated within the packet's data part. These headers contain additional information about the flow, including the source and destination *ports* on which the application is communicating.

Using a combination of the host addresses, IP protocol and ports, network administrators build signatures to accurately identify and classify flows as they traverse their networks. This method of traffic classification is both effective and efficient, assuming hosts use consistent, "well-known" TCP or UDP ports (Nguyen and Armitage, 2008). These well-known ports[1] are assigned and managed by the *Internet Assigned Numbers Authority* (IANA). As port and IP protocol information is configured by the two communicating hosts, alternatives can be negotiated by either side. The effects of dynamic port and IP protocol selection impairs the abilities of port-based filters and restrictions (Alshammari and Zincir-Heywood, 2008). This limitation is of increasing importance as both legitimate and devious application developers seek to evade classification using dynamic selection strategies (Goss and Botha, 2011). This problem is exacerbated by P2P application protocols, which operate in a decentralised manner, dynamically selecting the ports and protocols on which they communicate (Auld et al., 2007). Well-known port numbers can therefore no longer be used to reliably identify a flow's underlying application protocol (Moore and Papagiannaki, 2005).

[1] `http://www.iana.org/assignments/port-numbers/`

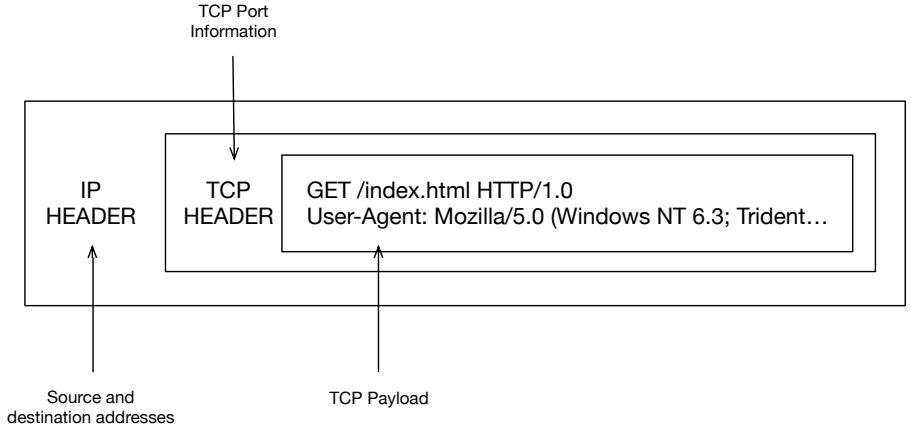


Figure 4.2: HTTP protocol match analysis.

The inefficiencies associated with classic port-based classification dictated that industry and researchers alike consider a number of alternatives. One such alternative gaining widespread appeal is *Deep Packet Inspection* (DPI).

4.1.2 Deep Packet Inspection

Deep packet inspection aims to address problems associated with identifying an application protocol using classic port matching techniques, by searching a packet's payload (content) for string matches. This inspection usually considers only the first few packets of a flow, as it is within these initial packets that the application protocol is negotiated. These searches are often performed using regular expressions in software, or using specialised hardware such as a *Field Programmable Gate Array* (FPGA) (Huang and Zhang, 2008). Each regular expression (signature) is formulated for targeting a specific sequence of contextual characters or numeric values, which are characteristic of a particular application protocol. For example, Figure 4.2 illustrates a match analysis for the *Hypertext Transfer Protocol* (HTTP).

In Figure 4.2, the IP header contains packet routing information used to transport the packet across the network. Some of these fields (Figure 4.1) include the sender and recipient addresses and the IP protocol. The TCP protocol is used by HTTP and thus a TCP *segment* is encapsulated within the data part of the IP packet. The TCP header contains various fields,

including the ports (source and destination) selected by the communicating applications.

Dissimilar to classic port matching (Section 4.1.1), DPI is not concerned with information in either the IP header nor the protocol header, as this information is subject to manipulation by the communicating parties. Instead, DPI concentrates on matching application protocol information contained in the payload of the packet. In the HTTP example, this may be achieved by constructing a simple regular expression, such as:

```
^[GET|POST] * http/[01]\.[019] [\x09-\x0d -~]
```

Note that the regular expression depicted here is overly simplified. In reality, more precise patterns are required to accurately classify application protocols. Examples of these regular expressions are available on the L7-Filter project website[2].

Network administrators perceive DPI as an essential tool since it enables them to search payload information for predefined application protocol signatures (Huang and Zhang, 2008). While P2P applications elude classic port-matching techniques through dynamic port selection, Sen et al. (2004) has shown that DPI solutions are capable of reducing false positives and false negatives to 5 percent of the total bytes for most P2P protocols investigated. Moore and Papagiannaki (2005) combine the use of both classic port-matching techniques and DPI to classify network traffic. In this work, application protocols operating on well known ports are immediately classified. The remaining flows are passed through a DPI engine, which examines the first *kilobyte* (KB) of payload data for a "well known protocol". Failing classification at this stage resulted in the entire flow's payload being inspected. The results of this work showed that up to 69 percent of the applications classified using classic port-matching techniques were correctly identified. By examining the first KB of payload, the accuracy increased to 79 percent. Upon inspecting the full payload, this accuracy approached nearly 100 percent (Moore and Papagiannaki, 2005).

The tools that implement DPI have also been extensively evaluated as of late. Bujlow et al. (2014), for example, offers an extended report on the most popular DPI tools for traffic classification. In this work, Ipoque's *PACE*

[2] `http://l7-filter.sourceforge.net/protocols/`

software was shown to achieve the best results for the test data set, with L7-Filter and *Cisco NBAR* performing poorly. No significant classification differences were noted when using the full test data set, or where each flow was truncated to include a maximum of ten packets (Bujlow et al., 2014).

While DPI has yielded a number of success stories, it is not without its limitations. An increase in privacy concerns has labelled DPI as a controversial topic due to it reading packet payloads (Zhang et al., 2009a; Dorfinger, 2010). Another consideration is the high computational and storage overhead associated with DPI (Alshammari and Zincir-Heywood, 2007). This overhead increases exponentially when interrogating every packet traversing the network, a problem compounded as developments toward higher-speed network connectivity continue.

To ensure completeness of an IP traffic classification system, the number of signatures available should be at least equal to the number of application protocols traversing the network. As the number of applications travelling across networks increase, so the number of signatures required to classify various protocols increases proportionately. The storage resources consumed by these signatures will increase over time, placing increased load on memory requirements for classification systems (Huang and Zhang, 2008). As network speeds increase, scaling problems associated with DPI systems become apparent. For example, the *Deterministic Finite Automata* (DFA) algorithm, used by many DPI systems, becomes overburdened at high data rates (Huang and Zhang, 2008). This strain results in increased latency due to packet queuing, degrading the overall performance of the network (Goss and Botha, 2011). DPI systems will therefore be faced with high-performance challenges as link rates and traffic volumes on networks continue to increase (Huang and Zhang, 2008).

A final consideration, arguably the most significant, is that many application protocols have begun encrypting their payload in transit (Nascimento et al., 2013). In some cases, this encryption is imposed to address privacy concerns associated with DPI, and in others to evade classification by these systems. While DPI may yield excellent results for identifying plain-text flows, encryption renders the content of packets opaque and thus the use of DPI inept (Alshammari and Zincir-Heywood, 2008; Goss and Nitschke, 2013a). Although much early research in traffic

classification did not take encrypted traffic into account, a number of recent works focus primarily on that topic (Alshammari and Zincir-Heywood, 2008). As the number of encrypted flows increase, so the dependency on DPI wanes. The drive of application development toward encryption prompted both industry and research communities to investigate alternatives to DPI. This investigation gave rise to IP traffic classification by *statistical analysis*.

4.1.3 Statistical Analysis

Both classic port matching and DPI techniques are limited due to their dependence on inferred semantics of information gathered from the contents of a packet header or payload parts. By encrypting or otherwise altering the state of the payload, the contents thereof become opaque and are thus no longer manageable by these systems. Application protocols are therefore difficult to detect if the underlying payload is encrypted (Gebski et al., 2006). A number of application protocols, such as *Secure Shell* (SSH) and various *Virtual Private Network* (VPN) technologies make use of encryption to secure data during transit. These opaque network flows are of great concern to network administrators as it impairs their ability to inspect and analyse the traffic. The problem of classifying encrypted flows extends beyond the classification techniques already discussed and has subsequently been the subject of much research as of late in the field of IP traffic classification (Nguyen and Armitage, 2008; Zhang et al., 2009b; Alshammari and Zincir-Heywood, 2008; Alshammari et al., 2009b; Dorfinger, 2010; Goss and Nitschke, 2013a,b, 2014). In each case, the opacity of data in transit presents as possibly the most challenging obstacle to overcome when classifying IP traffic flows. Applications themselves do not need to encrypt their data natively to render their data opaque in transit. Instead, it is possible for applications to make use of VPNs, which encapsulate their flows in an encrypted *tunnel* (Figure 4.3). In the case of encrypted communications (natively or by VPN), the only remaining discernible characteristics of the flow are statistical *characteristics*, or *features*, such as the *direction of packet flow*, *approximate size of packets* and the *timing between packets* (Gebski et al., 2006). These features are descriptive statistics that can be calculated from one or more packet

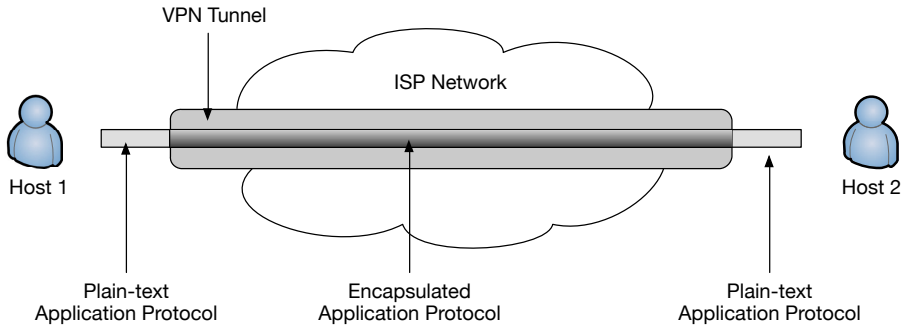


Figure 4.3: Two hosts communicating over an encrypted VPN.

exchanges of a flow (Alshammari et al., 2009b).

An assumption underlying IP traffic classification by statistical analysis is that over a flow's duration, certain statistical properties, such as directionality of packet flows, size of packets and the duration of a flow, will remain intact through encryption. These properties can therefore be used to infer the category or, in some cases, the application protocol of encrypted flows (Nguyen and Armitage, 2008). The accuracy of these classifications is therefore dependent on the features selected to describe each application protocol. A variety of feature combinations have been proposed by researchers over the years, with a unique combination constructed for identifying a specific application protocol. Nguyen and Armitage (2008) provide an extensive analysis of works in IP traffic classification through statistical analysis, with specific reference to the features they incorporate. Some of the more popular features identified by this work are listed in Table 4.1.

The selection of statistical features is also important in order to support *early identification*, a high degree of *accuracy* and *flexibility of application* in IP traffic classification systems. As such, certain features proposed by researchers are not considered suitable for practical operational applications. For example, network administrators require the ability to efficiently identify a flow by its application protocol early in its establishment (Goss and Botha, 2011). The features selected for most operational implementations are thus those which support *real-time classification* (Nguyen and Armitage, 2008).

For this reason, features such as *flow duration* and *flow size* are often removed due to the requirement for observing the entire flow before a

Feature	Description
Packet Direction	The direction of packets within a flow (source to destination and destination to source)
Packet Length	Packet length statistics (min, max, quartiles, standard deviation)
Packet IAT	Packet inter-arrival time (min, max, quartiles, standard deviation)
Flow Duration	The duration of a particular flow on a network, from its start to termination
Number of Packets	The total number of packets observed for a particular flow

Table 4.1: A summary of the most popular statistical features used in IP traffic classification, according to Nguyen and Armitage (2008).

classification is made. Latency-dependent features, such as packet *inter-arrival time* (IAT), are often studied by researchers (Mcgregor et al., 2004; Zander et al., 2005), however these features are difficult to implement in reality. Features which rely on timing are not always feasible in real-world operational environments as these metrics are often skewed by QoS policies and network congestion (Goss and Botha, 2012). The features selected by each system play a significant role in the accuracy achieved by that system. Too few features may allow an application protocol to be broadly classified by a specific group (Table 4.2), while too many may impose inefficient, undue computational cycles on the classification process. The success of a feature set for IP traffic classification is therefore a measure of how well the set delineates a number of application protocols while maintaining operational efficiency.

As many applications move toward encryption, the ability to identify flows "in the dark" is especially helpful and of tremendous value to network administrators (Wright et al., 2006). Many statistical analysis-based IP traffic classification systems allow flows to be classified in a broad context (Table 4.2), with few providing an in-depth, turn-key solution for individual application identification (Wright et al., 2006).

The use of ML has been investigated recently for providing additional granularity and increased accuracy in classifying IP traffic, using features identified through statistical analysis.

Category	Example Applications (protocols)
BULK	FTP
DATABASE	Postgres, MySQL
INTERACTIVE	SSH, Telnet
MAIL	IMAP, POP3, SMTP
SERVICES	DNS, NTP
WWW	HTTP, HTTPS
P2P	BitTorrent, GnuTella
MALICIOUS	Virus Attacks, DDoS
GAMES	Call of Duty, World of Warcraft Online
MULTIMEDIA	Flash Media, Internet TV, Radio Streaming

Table 4.2: Broad categories for application protocol classification, adapted from Moore and Papagiannaki (2005).

4.1.4 Machine Learning

Although ML has offered promising results when classifying IP traffic, certain limitations are still apparent. These limitations hamper both the completeness and accuracy of IP traffic classification systems. For example, most modern classification systems are reliant on the manual definition of signatures for each application protocol (Section 2.1.1). These signatures require substantial customisation by experts to accurately distinguish a particular application protocol in a data set of recorded network traces. Delays caused by this manual intervention directly affect the completeness of the system. Manual definition of signatures (classifiers) and formulation of their training sets also has an impact on the accuracy these systems achieve. Like any classification task, human error and insufficient knowledge of a particular problem may result in less than optimal signatures being produced.

Unsupervised learning, with the ability to find regularities and infer a function to describe these hidden structures in unlabelled data sets has been tested to address these deficiencies. Mcgregor et al. (2004) published one of the earliest works where unsupervised learning was applied in IP traffic classification. In this work, the *Expectation-Maximization* (EM) (Dempster et al., 1977) algorithm was used to group flows with similar behavioural patterns in the same cluster. The results of this work were the grouping of

applications by class (Table 4.2), rather than by application protocol.

Erman et al. (2006) tested the K-means, DBSCAN and AutoClass algorithms for clustering empirical traces from both a publicly available data set and one recorded by the authors at the University of Calgary[3]. Each data set was searched for instances of application protocols from a predefined list. The listed application protocols included HTTP, P2P, *Simple Mail Transfer Protocol* (SMTP) and *Post Office Protocol version 3* (POP3). The results revealed that while AutoClass produced the highest overall accuracy, DBSCAN showed great potential as it placed the majority of connections in a small subset of the clusters, with an overall accuracy of 97.6 percent (Erman et al., 2006). This was likely attributed to DBSCAN's ability to distinguish noise within a data set, a feature lacking in both K-means and AutoClass. Also noteworthy was an evaluation of the time each algorithm took to complete the clustering task, concluding clusters. In the tests conducted by Erman et al. (2006), DBSCAN clustering completed in one minute, K-means in three minutes and finally, AutoClass in four and a half hours. The inefficiencies experienced by the AutoClass algorithm make it unsuitable for classifying IP traffic, as these have a direct affect on the completeness of the classification system.

Goss and Nitschke (2013a) tested the *Hierarchical Self-Organising Map* (HSOM) algorithm for differentiating between application protocols, rather than classes, on a network. In this work, the clusters discovered were used to form training sets for *Artificial Neural Network* (ANN) classifiers, trained to identify future instances of each respective cluster. Each of the ANNs produced scored an overall accuracy in excess of 99.35 percent, with the exception of the HTTP classifier. Further investigation revealed that the HTTP protocol exhibited varying packet directionality properties during initial packet exchanges, which skewed the results. This issue speaks to the features selected and the normalisation of each feature, rather than the clustering algorithm. Goss and Nitschke (2013b) extended the work of Goss and Nitschke (2013a), substituting the K-means algorithm in place of the HSOM. Here, the K-means algorithm was found more accommodating to the directional properties of HTTP, as the classifier achieved 99.88 percent overall accuracy.

[3]The University of Calgary, Calgary, Alberta, Canada. `http://www.ucalgary.ca/`

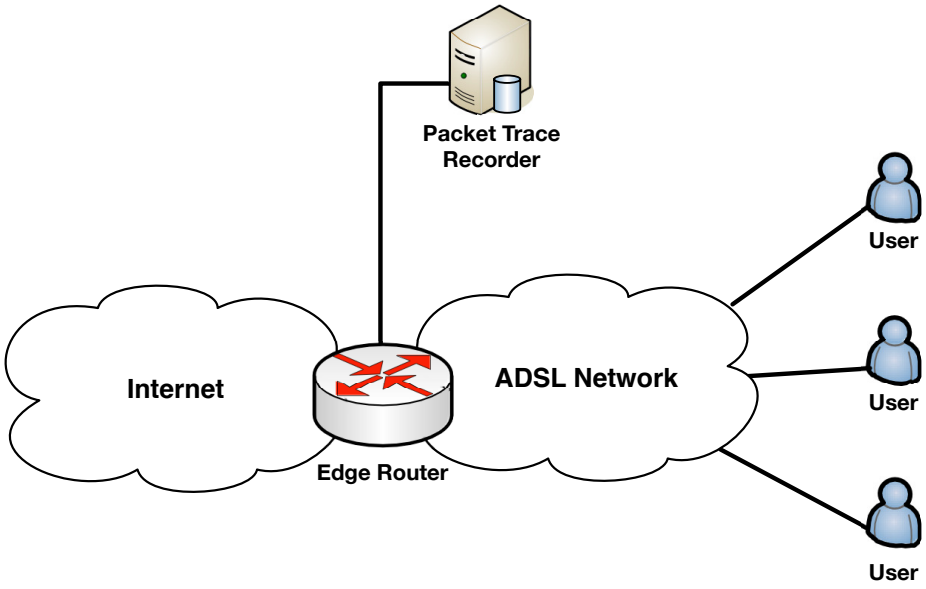


Figure 4.4: A network tap copies flow information to a server, which records specific trace information for each communication observed. In this experiment, the network tap was configured to copy all packets switching through a subscriber network segment at a large South African ISP.

The APIC method, with its ability to automate feature selection, pattern discovery and classifier production processes is poised to address the deficiencies apparent in modern IP traffic classification systems, improving both the completeness and accuracy achieved by current systems. The following section details a case study, where the APIC method is evaluated for improving the task of IP traffic classification.

4.2 Task Description

The task of identifying an underlying application protocol in IP traffic flows is not new. In Section 4.1, a number of works were detailed where solutions for addressing both completeness[4] and accuracy[5] were investigated. In this case study, an experiment is devised where APIC is compared to existing state-of-the-art IP traffic classification systems. The test data set was recorded on an *Asynchronous Digital Subscriber Line* (ADSL) edge router at one of South

[4]Completeness refers to the availability of signatures for each application protocol
[5]Accuracy is a measure of how correctly an underlying application protocol is identified

Protocol	Description
SSH	Secure Shell Protocol
SMTP	Simple Mail Transport Protocol
POP3	Post Office Protocol
HTTP	Hyper Text Transfer Protocol
HTTPS	Secure Hyper Text Transfer Protocol
Bittorrent	Bittorrent Peer-to-Peer Protocol

Table 4.3: A list of commonly tested application protocols in IP traffic classification research. This list also describes the application protocol signatures produced by APIC which are analysed in detail in the discussion part of this case study.

Africa's premier ISP networks (Figure 4.4). All network packets switching through this segment were copied and recorded using custom packet trace sampling software[6], where certain statistical information about each flow was sampled and stored. The statistical properties considered in this task include those listed in Table 4.1. These flow properties were identified in Section 4.1.3 as those considered most popular in modern IP traffic classification systems.

Table 4.3 lists a set of commonly tested application protocols, where a number of IP traffic classification system results have been published. These results indicate how well the classifiers of each system perform when tested on a recorded data set. In this case study, flows are recorded and analysed in real-time using APIC. The results achieved by the classifiers produced in this case study using the recorded data set are comparable to those published by similar systems for the same protocol. This comparison establishes the effectiveness of APIC in terms of accuracy and, at the same time, demonstrates the gains in completeness realised through the method, confirmed by the presence of a classifier for each test protocol.

The recording process was started at midday on a normal week day, where the presence of each protocol listed in Table 4.3 was most likely to occur. The process was set to run for a period of one hour, the maximum time allowed by the ISP, sampling all IP traffic flows. In total, 1,462,038 packets were observed, describing 47,607 TCP and 4070 UDP flows respectively. Each of the application protocols listed in Table 4.3 operate over the TCP protocol, a stateful protocol designed to ensure successful

[6]http://ryan.goss.co.za/download/1500-ptss.tar.gz

Group	Dimensions	Description
0	0-9	Discriminators indicating the direction of flow for the first ten payload-bearing packets
1	10	In vs Out byte ratio
2	11-14	Log IAT (min, mean, max, stddev)
3	15-18	Payload size (min, mean, max, stddev)
4	19-24	Payload bytes

Table 4.4: Feature group breakdown, depicting the dimensions included within each group. The feature group genotype is comprised of five bits, which are toggled to include or exclude certain attributes.

delivery of application data. A stateful (state-based) protocol describes a transport level protocol, where each communicating host retains the state of the connection (a virtual circuit) for the duration of the communication, even when no data packets are being transferred. Either of the communicating hosts may tear down the connection, dropping the connection at both ends. Data transactions over a stateful protocol, in contrast to a stateless protocol, are easily tracked by IP traffic classification systems, allowing greater control over data exchanges between hosts. As each of the application protocols considered in this case study operate over the stateful TCP protocol, only data exchanges operating over this protocol were sampled and recorded for this experiment.

APIC's suitability for the task of IP traffic classification is measured by two distinct objectives - accuracy and completeness. To demonstrate increased completeness, most manual processes should be removed from the task of classifier creation. For APIC to succeed in this, it must demonstrate a process to automate both *feature selection* and *application protocol discovery* tasks, ones commonly performed manually in existing systems by human experts.

The following section details the feature selection process APIC employs to automatically determine the most suitable features for describing application protocols within a recorded flow data set.

```
{
   "Dir": 1,
   "Dir2": 0,
   "Dir3": 1,
   "Dir4": 0,
   "Dir5": 0,
   "Dir6": 0,
   "Dir7": 1,
   "Dir8": 0,
   "Dir9": 1,
   "Dir10": 0,
   "ByteRatio": 0.594988344988,
   "LogIATMin": -3.16787857096,
   "LogIATMean":-0.141905890926,
   "LogIATMax":0.899455358536,
   "LogIATStdDev": 1.21841093522,
   "PayloadMin": 16,
   "PayloadMean": 273.7,
   "PayloadMax": 840,
   "PayloadStdDev":326.250839079,
   "Byte1": 83,
   "Byte2": 83,
   "Byte3": 72,
   "Byte4": 83,
   "Byte5": 83,
   "Byte6": 72
}
```


Figure 4.5: A JavaScript Object Notation (JSON) representation of a vector describing a flow observed during recording.

4.3 Feature Selection

The features incorporated in the experiments that follow were, according to Nguyen and Armitage (2008), those commonly used in modern IP traffic classification research. These features, identified in Section 4.1.3, include the *directionality of flow* for the first ten payload-bearing packets, the *inbound versus outbound payload size (byte) ratio*, log outputs for various *packet inter-arrival times*, *payload size* statistics and, finally, the first three bytes of data inferred from the first payload-bearing packet in each direction. In total, 25 features were recorded for every TCP flow observed during the recording process.

A vector with a length of 25 was subsequently created in memory to store the metrics of each flow trace. Each dimension represents the value of a particular feature recorded for the flow. An example of one such recorded flow described by its statistical features is represented as JSON in Figure 4.5.

A bit string feature genotype with 25 genes was created to represent each feature subset, where 1 denoted the inclusion of a particular feature and 0 an exclusion. To ensure these features were toggled on and off as a group, and not individually, a five-dimension feature group genotype was created. Each bit of this genotype represented one of the five discriminator groups comprising the feature vector. These five groups, the span of their

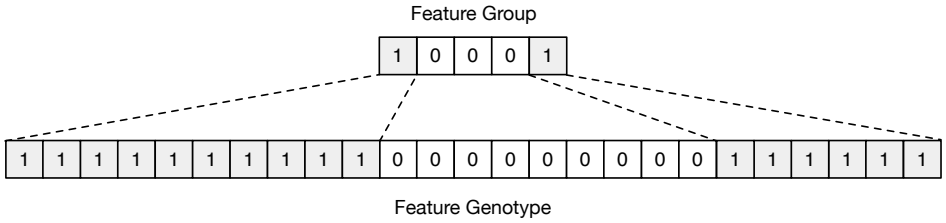


Figure 4.6: The APIC GA toggles bits in a feature group binary string, which in turn expands to form a feature subset genotype. In this example, feature group options 1 and 5 marked as included, whilst options 2,3 and 4 are not. All features are expanded, where those marked for inclusion are set to "1", whilst those excluded are marked with a "0".

dimensions, and a brief description are listed in Table 4.4. Using a feature group allowed APIC to toggle between discriminators for inclusion (as a group) in each feature subset during the feature selection search process.

Group	E_{min}	E_{max}
0	0	1
1	0	1
2	-5	5
3	0	1500
4	0	256

Table 4.5: Heuristic values of E_{min} and E_{max} for each dimension. These values were derived by observing the minimum and maximum values of each dimension within the recorded data set for a particular feature group.

APIC uses a bit string to determine inclusion of each group in a feature subset (Section 3.1). A "1" at position 0 in the feature group expands to include the first ten discriminators in the feature genotype. A "0" at position 0 of the feature group genotype will mark the first ten features as excluded. An example of this expansion is illustrated in Figure 4.6, where the feature subset includes the directions of the first ten payload-bearing packets and the first three bytes of payload in each direction of flow. By simply modifying the bit string of the feature group genotype, the APIC method was able to manipulate discriminators that were included for determining application protocols in the recorded data set.

The objective of APIC in this task was to produce optimised TWEANN classifiers, each trained to identify a specific application protocol. The APIC feature selection process itself does not prepare the recorded data for training these models. The clustering algorithms used by APIC rely on *Euclidean distance* to determine cluster association. To ensure each feature contributed fairly toward this calculation, each was normalised to values between 0 and 1 using standard *min-max normalisation* (Equation 4.3):

$$Normalised(i) = \frac{i - E_{min}}{E_{max} - E_{min}} \tag{4.3}$$

Where E_{min} is the minimum value of the dimension, E_{max} the maximum value and i the value to be normalised. The heuristic values applied to each feature group in respect of E_{min} and E_{max} for the task of IP traffic classification are listed in Table 4.5.

4.4 Distinguishing Application Protocols

Using the normalised data set, the APIC method searched for distinct application protocols, using feature subsets determined by a GA (Figure 4.7). In this case study, five basic feature groups were considered, consisting of the most common features studied in today's IP traffic classification systems. A sequential search for an optimal feature subset within these groups would require a mere 32 tests to consider every possible combination of a five-bit binary feature group ($2^5 = 32$). For the sake of homogeneity, a GA was used in this experiment to identify the most optimal feature set as in a real world context, the number of feature groups may scale far beyond this. For example, VISA's latest fraud detection system, *Visa Advanced Authorization* (VAA)[7], considers more than 500 features per transaction. In these cases, sequential search for best feature sets is not efficient. The APIC method, as a general method, needs to support feature selection across many tasks, where many features may be available for consideration. A GA was identified as a sound choice, using a stochastic search technique instead of sequential one to more efficiently identify suitable feature subsets for a variety of problems.

The APIC feature search GA was configured to traverse a maximum of 64 generations - twice the number of iterations required to find the best feature subset by grid-search. This value was set high for the purposes of monitoring successful convergence during the experiment. According to Ferri et al. (2000), a GA is an ideal choice for locating optimal features in high-dimensional search space, compared to an enumerative search for the same.

A GA-controlled hyper-parameter search was used to select more optimal hyper-parameters for guiding each clustering process. This process (Section 3.2) was required to distinguish between unique application protocols present in the recorded data set. It is paramount to ensure that optimised clustering occurs, as the clustered data sets are used to train TWEANN classifiers to identify future instances of each identified application protocol.

As the number of application protocols (clusters) present within the recorded data set was unknown, the *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) clustering algorithm was chosen to

[7]http://www.visa.com/visariskproducts/

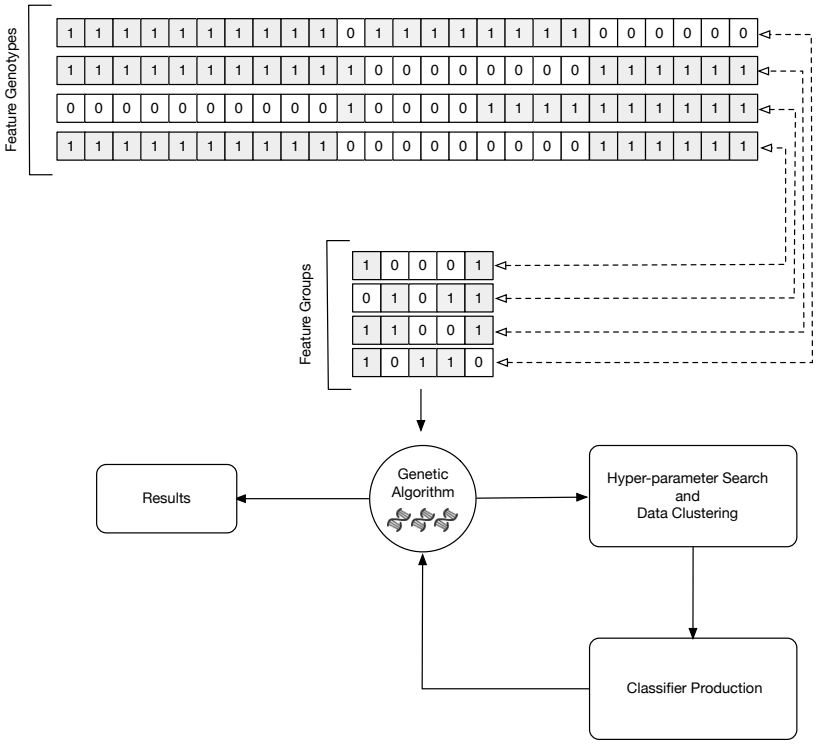


Figure 4.7: The feature selection GA builds a population of feature group genotypes. Each of these genotypes is expanded to include all features represented for each of the groups. These genotypes ultimately determine which features are included in the hyper-parameter search and data clustering process. The resultant clustered data set is used to create TWEANN classifiers, where the average classifier test score is fed back to the feature selection GA as a fitness score. The highest scoring classifier group is returned as a result of this process.

determine this automatically. The algorithm was controlled by two hyper-parameters, namely *eps* and $minPTS$. To ensure most optimal clustering occurs, these hyper-parameters should be carefully chosen.

The selection of *eps* and $minPTS$ parameters is often a manual, heuristic task. An expert will often test a number of combinations until a satisfactory result is achieved. Such manual parameter tuning is not conducive to ensuring a complete classification system, which can only be achieved through full automation. For this reason, a second GA was configured to search for the most optimal set of *eps* and $minPTS$ values for each feature subset tested.

Each candidate solution was encoded as a bit string for consideration by the GA. The number of bits in the bit string (genotype) is dependent on the

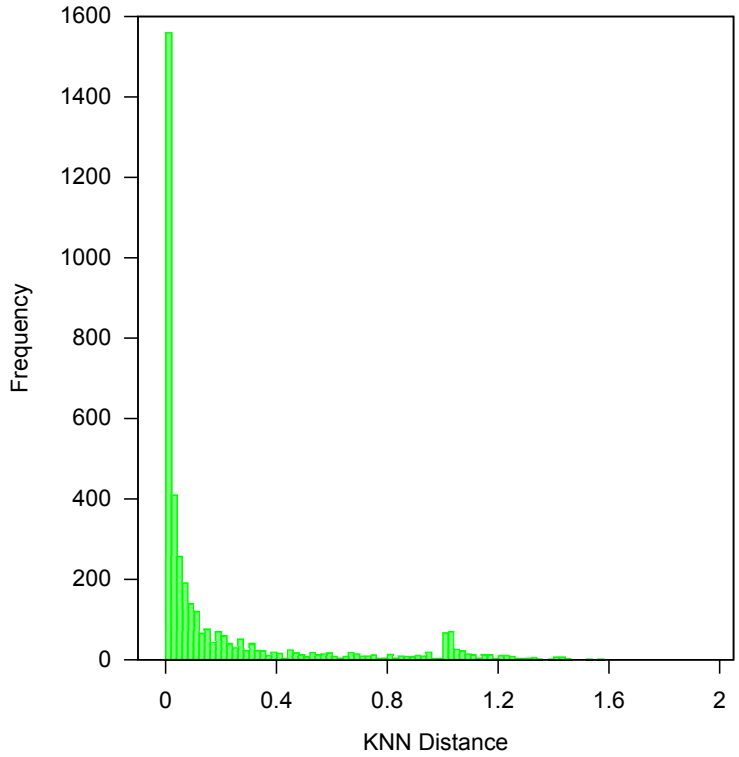


Figure 4.8: A k-nearest neighbour (k-NN or KNN) histogram with the distance between each flow describing vector and its nearest neighbour recorded on the x-axis. The y access tallies the total number of flow vectors whose distance to their nearest neighbour was within the specific range on the x axis. According to Goss and Nitschke (2014), a suitable value for *eps* can be determined by noting the value on the x access at the knee of the graph. In this example, the knee is observed at a x axis value of approximately 0.15.

range of values for each parameter. Goss and Nitschke (2014) implemented the DBSCAN algorithm for the purpose of clustering application protocol flow vectors. In this work, the value of *eps* was determined by plotting the distance from each vector to its nearest neighbour on a *k-nearest neighbour* (k-NN) histogram and setting *eps* to the value at the knee of the graph. Goss and Nitschke (2014) assert that the value at the knee of the graph is the most likely location for optimum *eps*; the point at which the distance between neighbours increases exponentially, due to the presence of outliers (noise).

The k-NN histogram depicting the proximity of nearest neighbours for the normalised, recorded data set of this case study is shown in Figure 4.8. The knee of the graph (Figure 4.8) is visible at approximately 0.15 on the x axis. The maximum k-nn value noted was 1.6, with only a few flow vectors

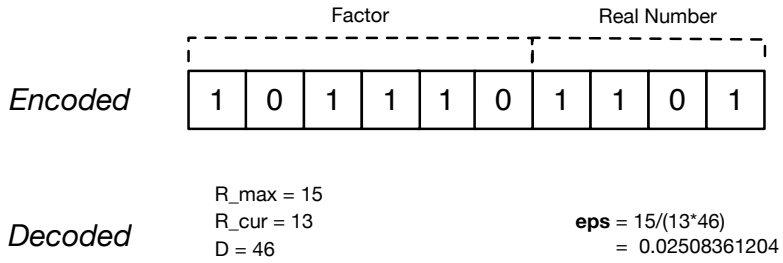


Figure 4.9: An example of *eps* encoded as a bit string. The first six bits represent the division factor (46), with the remaining bits representing the current real number value (13) and the maximum possible real number value (15). By applying Equation 4.4, the decoded value of *eps* is 0.0251.

contributing to this count. The range for consideration in determining *eps* was subsequently set to $0.0158 <= eps <= 15$, covering the full range of k-NN values. This scope was considered to demonstrate that the most optimum *eps* could be found at the knee of a graph. A custom encoding scheme was devised to represent this range within a ten-bit string. The last four bits of the string represent a real number in binary form. This number is multiplied by a factor, determined by the value of the first six bits. The result is divided into the maximum value of the four bits representing the real number ($1111 = 2^4 - 1$, or 15). The value of *eps* was decoded from each genotype using Equation 4.4:

$$eps = \frac{R_{max}}{R_{cur} * D} \tag{4.4}$$

Where R_{max} is the maximum value attainable using the last four bits of the genotype, R_{cur} is the current value for the same bits and D the decimal value of the first six bits. An example of *eps* in both its encoded and decoded form is shown in Figure 4.9. According to Goss and Nitschke (2014), the search space for $minPTS$ can be determined by Equation 4.5:

$$D + 1 \leq minPTS \leq \left(\frac{|db|}{2}\right) \tag{4.5}$$

where the lower range of $minPTS$ is greater than or equal to the number of

dimensions of the flow vector, D, plus one. The upper-most value of $minPTS$ is set to half the magnitude of the recorded flow vector recorded data set, *db*. It is difficult to determine the lower and upper limits of $minPTS$ definitively, without prior insight into the source data set. For this reason, the upper and lower bounds of $minPTS$ in this case study were set using these rule-of-thumb methods. By selecting a range of $minPTS$ values using Equation 4.5, the majority of cases will be adequately covered, avoiding pitfalls such as convergence to a single cluster, or the total number of clusters produced being equal to the number of input vectors supplied.

Although 47,607 TCP flows were recorded, only 31,102 were observed from their start, a requirement for accurately determining the directionality of a flow (Goss and Botha, 2011). A genotype encoding the full range of $minPTS$ for a complete feature set is represented in Figure 4.10. Where the result of $minPTS$ is less than $D+1$, a value of $D+1$ was used as a substitute. This ensures the lower bounds of $minPTS$ is fixed at $D + 1$. An example complete genotype used to find the best value for $minPTS$ in a twenty-five dimensional data set is illustrated in Figure 4.11.

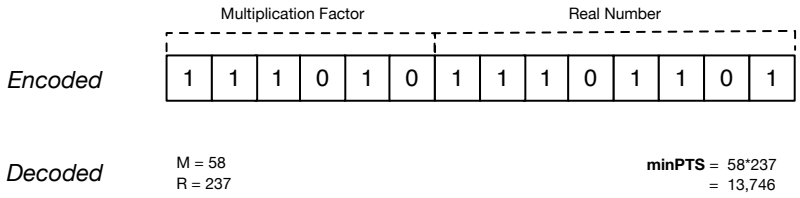


Figure 4.10: An example of $minPTS$ encoded as a bit string for a full feature set. The first six bits represent the multiplication factor (58), with the remaining bits representing the current real number value (237). The value of $minPTS$ encoded on this genotype is the product of the multiplication factor and real number. In this example, $minPTS = 13,746$.

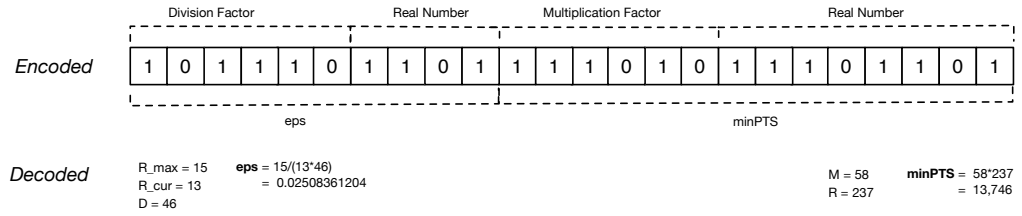


Figure 4.11: A genotype showing an encoded solution for *eps* and $minPTS$ values for a full feature set, comprising twenty-five dimensions and 32,000 records.

For each new feature subset considered by the hyper-parameter GA, the bits allocated for representing both *eps* and $minPTS$ values were derived automatically by APIC, using the encoding process described in Section 4.4, Equations 4.4 and 4.5. The initial population of hyper-parameter genotypes was created with bits set at random. The GA parameters governing both feature subset and hyper-parameter searches are outlined in Table 4.6.

The selection of good parameters to govern a GA improve both computation time and solution accuracy, for example, the population size significantly effects the quality of solutions that a GA evolves (Roeva et al., 2013). Roeva et al. (2013) found that, when identifying model parameters using a GA, smaller populations result in lower accuracy and that by increasing the population size, an increase in solution accuracy was achievable. They also found that, beyond a certain point, a larger population did little to increase the solution accuracy.

Genetic Algorithm	Population Size	Maximum Generations	Mutation Rate	Crossover Rate
Feature Subset Search	64	64	0.1	0.5
Hyper-parameter Search	100	100	0.1	0.5

Table 4.6: Parameter values configured for each GA.

A larger population did, however, increase the computational resource requirements for completing the search. Roeva et al. (2013) found in their research that an optimal population size of 100 achieved the most accurate solutions. The hyper-parameter search method of APIC is very similar to the model parameter identification methods described by Roeva et al. (2013). Noting these observations, the population size for the APIC hyper-parameter search was set to a value of 100.

Using the configured search parameters (Table 4.6), the APIC feature discovery (Section 3.1) and pattern discovery (Section 3.2) processes were started. For each genotype in the feature subset search population, a hyper-parameter search process was executed to find optimal values for guiding the clustering process. These hyper-parameters included both the *eps* and $minPTS$ parameters. Values for each of these parameters were decoded and a DBSCAN clustering process run for each genotype of the current hyper-parameter search population.

The resulting clustered data set was evaluated using the silhouette cluster evaluation method (Equation 3.2), scored within a range $-1 <= score <= 1$, where a score closest to 1 (indicative of most optimal clustering) is more favourable. The scores for each genotype were converted to values between 0 and 1, using a sigmoid logistic function:

$$f(x) = \frac{1}{1 + e^x} \tag{4.6}$$

Where e is the natural logarithm base and x the value being regressed to a value between 0 and 1. A plot illustrating this conversion is illustrated in Figure 4.12.

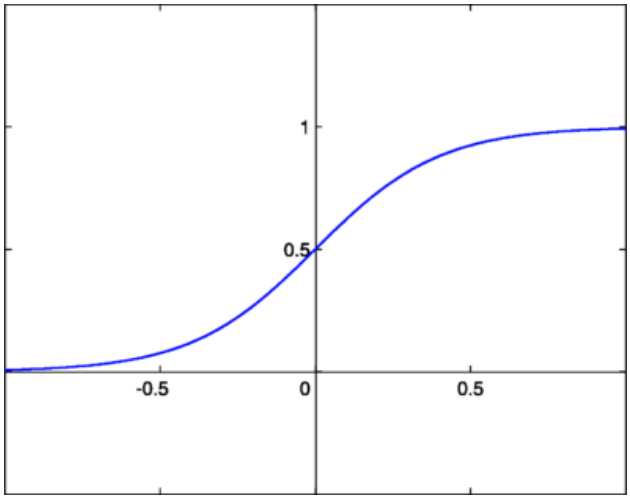


Figure 4.12: Logistic function for calculating hyper-parameter genotype fitness from silhouette clustering score.

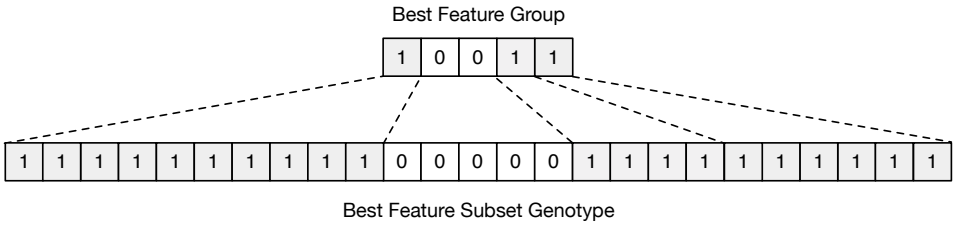


Figure 4.13: The best scoring feature group expanded to highlight the best feature subset for the recorded data set.

All search processes were executed in parallel, where hyper-parameter searches were chained as sub-tasks of feature subset searches. Parallel execution of tasks ensured optimal use of available host resources[8].

The APIC algorithm was executed and allowed to run all generations for each search task. This was to provide evidence that the algorithm did converge at a particular point. In a production deployment, where the APIC method is processing real-world data, additional stopping conditions may be included to conserve computational time and expense when distinguishing application protocols. These stopping conditions may include the observation of simple heuristic values, such as an acceptable fitness level achieved by a particular feature subset.

[8]A single execution was assigned to each *Central Processing Unit* (CPU) core of the host machine running the test. In total, 24 cores were available to the APIC algorithm.

Generation	*eps*	*minPTS*
23	0.1039	152

Table 4.7: Hyper-parameters (from GA search), resulting in optimal clustering

After all generations had concluded, analysis of the results showed that the best configuration was identified at generation four. The best scoring feature subset genotype (Figure 4.13) was used to extract features from the recorded data set for clustering by the DBSCAN algorithm. The highest scoring hyper-parameter search revealed that the best values of *eps* and $minPTS$ were 0.1039 and 152 respectively, found at generation 23 (Table 4.7). The output of the hyper-parameter search process for the winning feature subset genotype was recorded. Each of the hyper-parameter genotypes considered were decoded and their *eps* and $minPTS$ values plotted on a three-dimensional scatter graph along with their associated fitness (Figure 4.14).

Of the clusters formed by the DBSCAN algorithm, those that represent the focus application protocols (Table 4.3) were deemed most important for this experiment. A sample trace of each application protocol was identified by a human expert[9] and assigned cluster membership within the clustered data set, using silhouette cluster comparison. By determining the best $s(i)$ value for each annotated trace, the best-fit cluster was identified and annotated with the name of the application protocol assigned to the new member.

The following section provides additional evidence that the clusters identified are accurate representations of each application protocol.

[9]A qualified computer network expert with experience identifying application protocols from raw traces

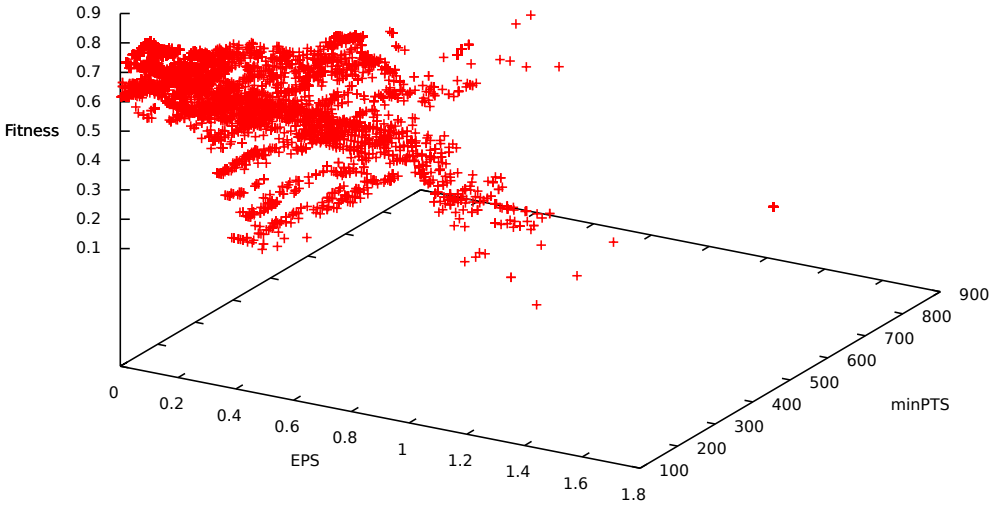


Figure 4.14: A 3D scatter graph depicting the decoded genotypes considered when searching for optimal hyper-parameters for the best scoring feature subset. The best scoring feature subset where hyper-parameter values decoded from the genotype resulted in the most optimal clustered data set. The chart density increased toward *eps* value 0.1 and $minPTS$ value 152, which ultimately produced the best fitness score.

4.5 Verification by Visualising Application Protocols

The potentially high multi-dimensionality of datum clustered by APIC make identified clusters very difficult to express and verify statistically. Radar charts, or star charts (Chambers et al., 1983), simplify the representation of this multi-dimensional data, by expressing each as a spoke on a two-dimensional radial map. These charts make it easier for human experts to identify outliers by visual comparison of similar, clustered datum. The chart plots values of each dimension along a separate axis (spoke) that starts in the centre of the chart (minimum value) and ends on the outer ring (maximum value). As each dimension of the data considered by APIC was normalised to a value between 0 and 1, it follows that the minimum values for each spoke of the radar charts drawn for this data will range between 0 and 1. As APIC found 20 attributes sufficient for describing application protocols in this case study, it follows that each radar chart is comprised of 20 spokes.

Radar charts were plotted for each focus application protocol, assisting experts with the verification of dynamically discovered clusters (Section 4.4). In this experiment, the chart included a value for each of the 20 features of

the feature subset search genotype (Figure 4.13). Multiple colours indicate diversity within each clustered data set. From the radar patterns of the POP3 (Figure 4.15), SMTP (Figure 4.16), HTTP (Figure 4.17), HTTPS (Figure 4.18), SSH (Figure 4.19) and Bittorrent (Figure 4.20) application protocols, diversity between feature patterns and values are visible. Furthermore, the close proximity and pattern conformance displayed by each of the colours on the charts indicates tight (optimal) clustering was achieved. Optimal clustering refers to the best possible grouping of datum, where the least possible average distance between datum in a cluster and the highest average distance between clusters is achieved.

Figures 4.15 through 4.20 represent each of the respective test application protocols. The colours on each chart illustrate the effectiveness of the clustering process which, when plotted on a radar chart, indicate distinct patterns for each application protocol. While some patterns appear similar, there are subtle (at times, significant) differences. These variations were detected by the APIC pattern discovery process (Section 3.2), which separated them into distinct clusters. For example, while the POP3 (Figure 4.15) and SMTP (Figure 4.16) application protocols appear similar, attribute 14 is far more pronounced in SMTP than in POP3. Likewise, although HTTP (Figure 4.17) and HTTPS (Figure 4.18) are similar application protocols, there is a clear distinction between the statistical properties of the two. The HTTP protocol exhibits high values for the 11th, 13th and 14th attributes, while HTTPS only the 14th. This contrast is due to the presence of *Transport Layer Security* (TLS), which encrypts the data exchanges of HTTPS. Although SSH (Figure 4.19) also uses TLS, attributes 16 through 20 differ significantly from those of HTTPS. These attributes act as a differentiator between the protocols.

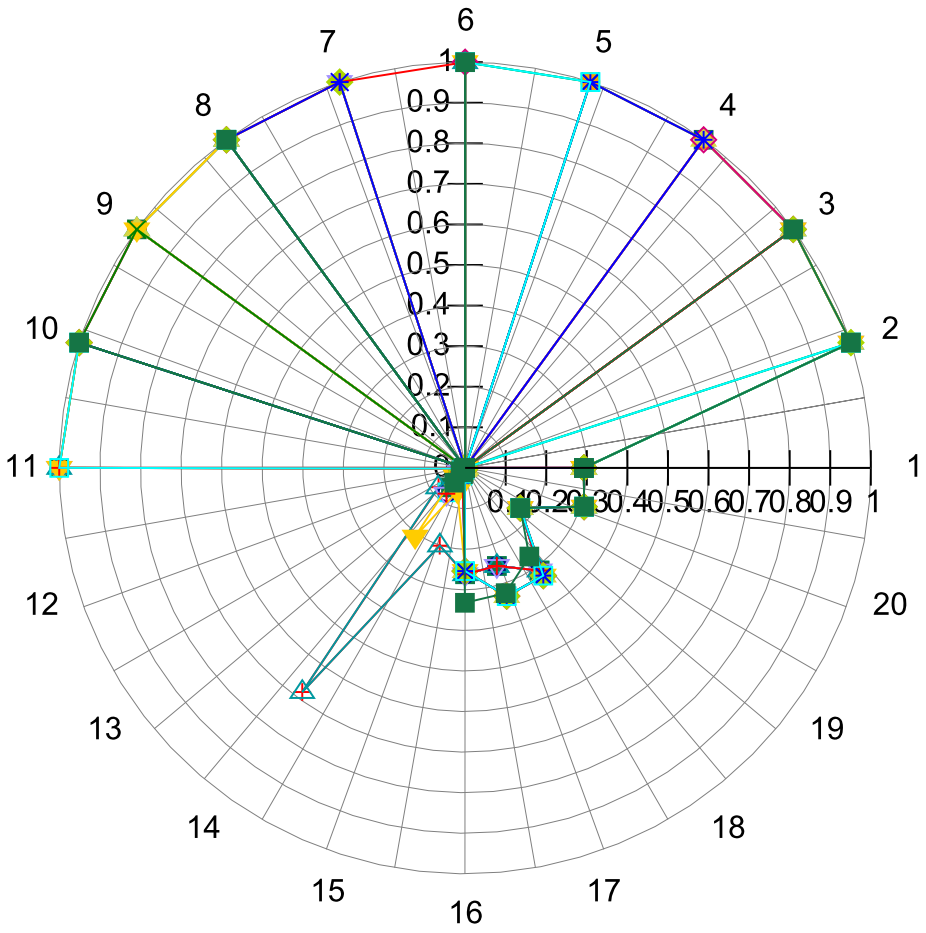


Figure 4.15: A radar chart plotting random datum samples of the cluster known to describe the *Post Office Protocol version 3* (POP3) application protocol. This protocol is used by email clients to retrieve user email from a remote mail server. In this chart, values represented on each spoke show subtle variations in dimensions 12 through 20.

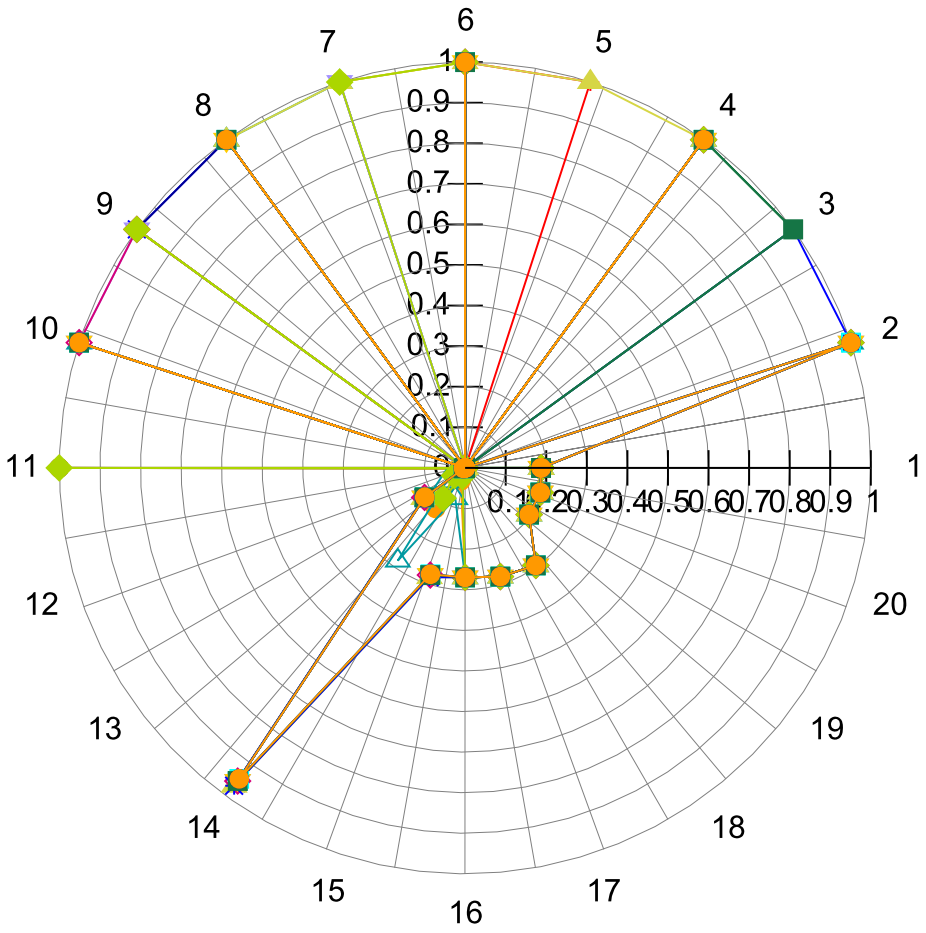


Figure 4.16: A radar chart plotting random datum samples of the cluster known to describe the SMTP application protocol. This protocol is used by email clients to send messages to a remote email server and for inter-mail server message exchange. The features indicating directionality of packet flow, 1 through 10, show consistency amongst the randomly selected flow samples. This is characteristic of SMTP, an application protocol comprising very structured message exchanges.

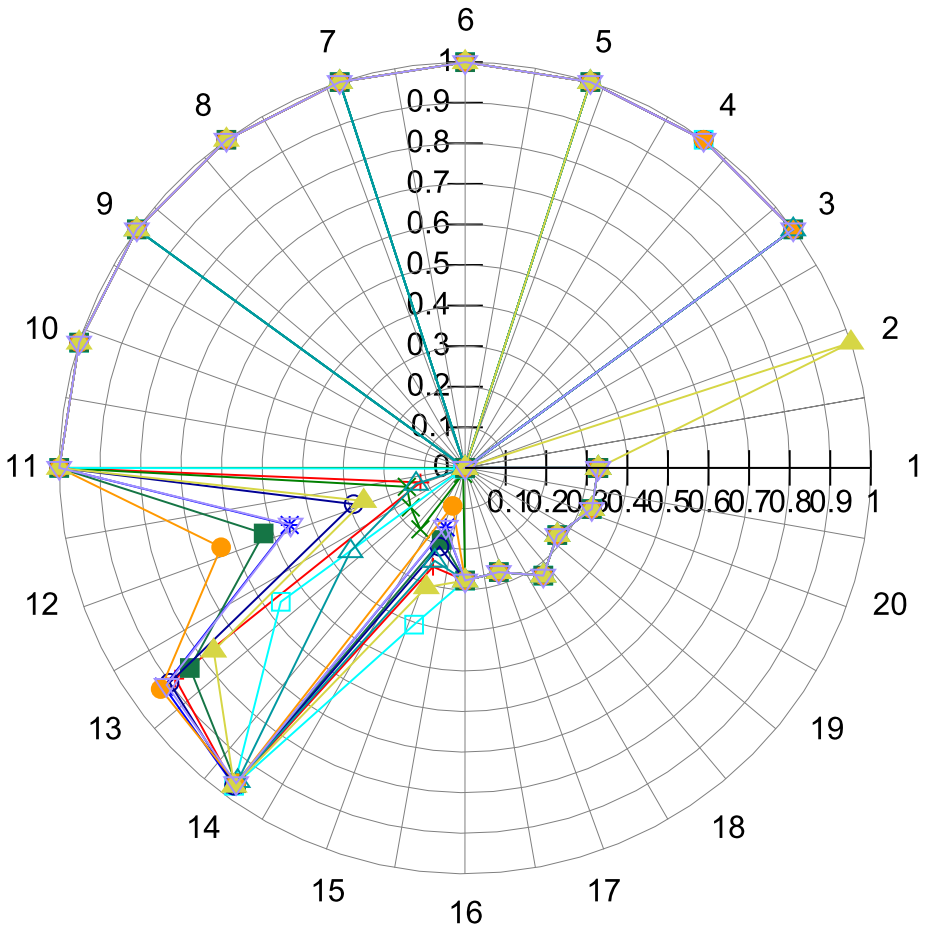


Figure 4.17: A radar chart plotting random datum samples of the cluster known to describe the HTTP application protocol. This protocol is one of the most common application protocols, used to retrieve and publish content on the *World Wide Web* (WWW). Dissimilar to the POP3 (Figure 4.15) and SMTP (Figure 4.16) charts, the diversity amongst random data samples within the HTTP clustered data set is more apparent. This is because the HTTP protocol is less predictable in terms of data exchange than both the POP3 and SMTP protocols.

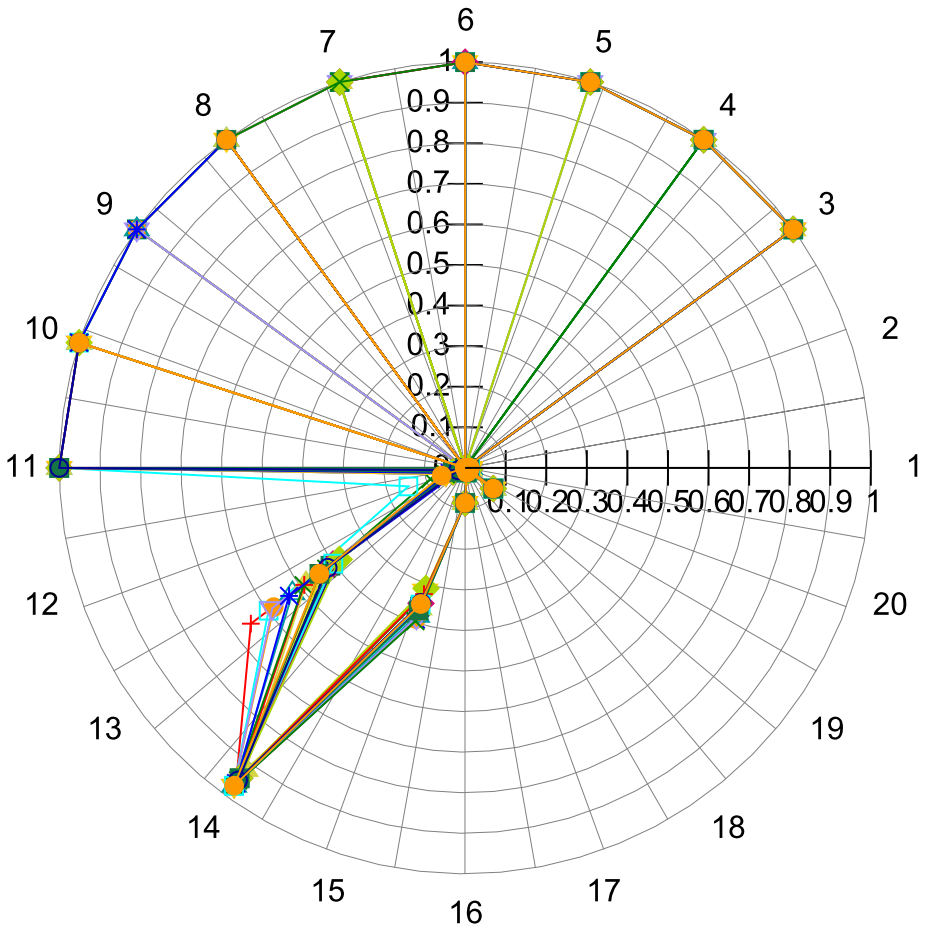


Figure 4.18: A radar chart plotting random datum samples of the cluster known to describe the HTTPS application protocol. This protocol is used to perform the same tasks as HTTP, but over a secure connection. Although the HTTPS application protocol is effectively the HTTP protocol operating over an encrypted, TLS tunnel, there are substantial differences in the values recorded for each feature. This is because the TLS tunnel set-up occurs ahead of application protocol data exchange.

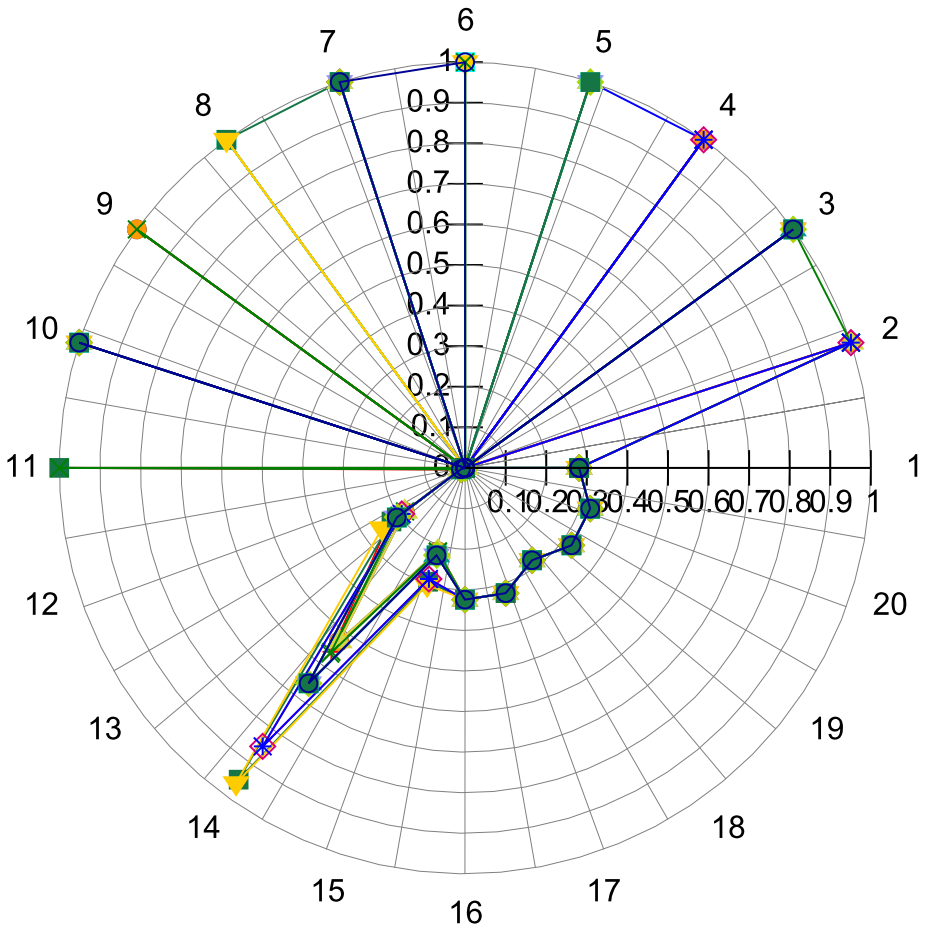


Figure 4.19: A radar chart plotting random datum samples of the cluster known to describe the SSH application protocol. This application protocol is most often used by administrators to interact with remote systems, issuing commands and retrieving files. The SSH application protocol, similar to HTTPS (Figure 4.18), uses TLS to secure its communications. Although both use TLS, distinctions in the feature values extracted for each application flow are apparent.

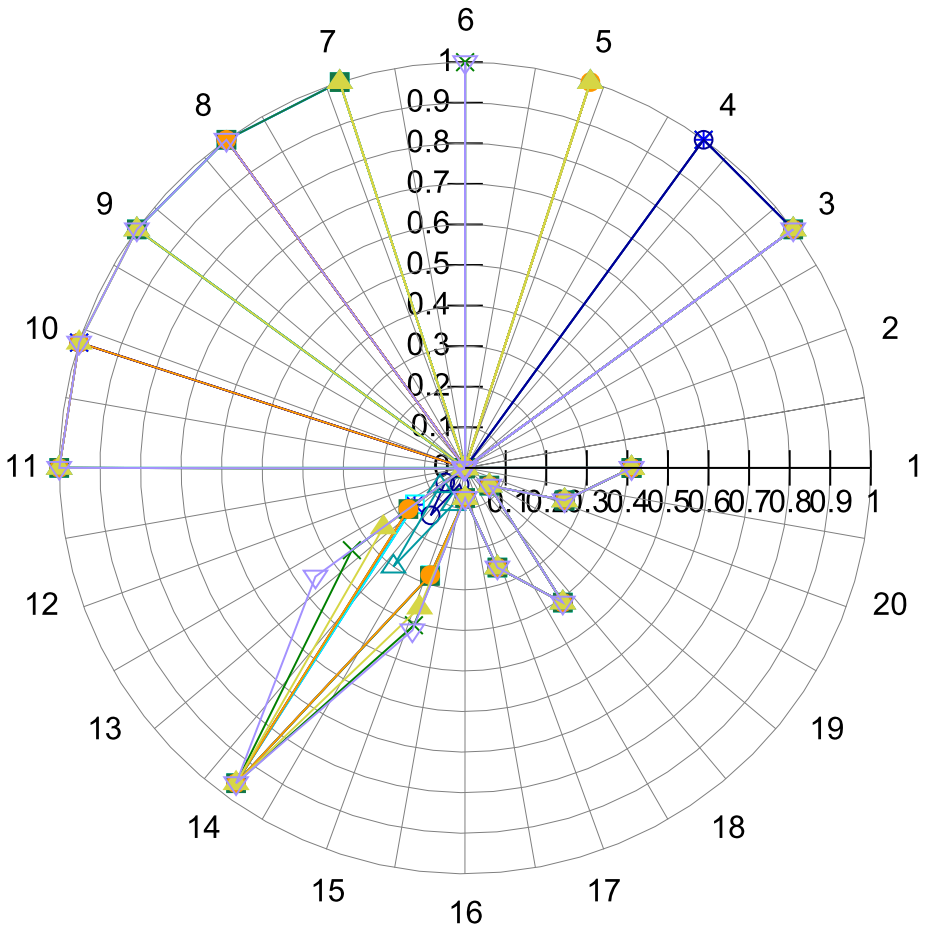


Figure 4.20: A radar chart plotting random datum samples of the cluster known to describe the unencrypted Bittorrent application protocol. This P2P communication protocol, used for efficient file transfer amongst network hosts, does not rely on client-server connectivity, nor static ports for data exchange. These attributes make it difficult to detect and manage. The APIC algorithm was able to identify flow traces of the application protocol, which uses distinct packet exchanges to establish each session.

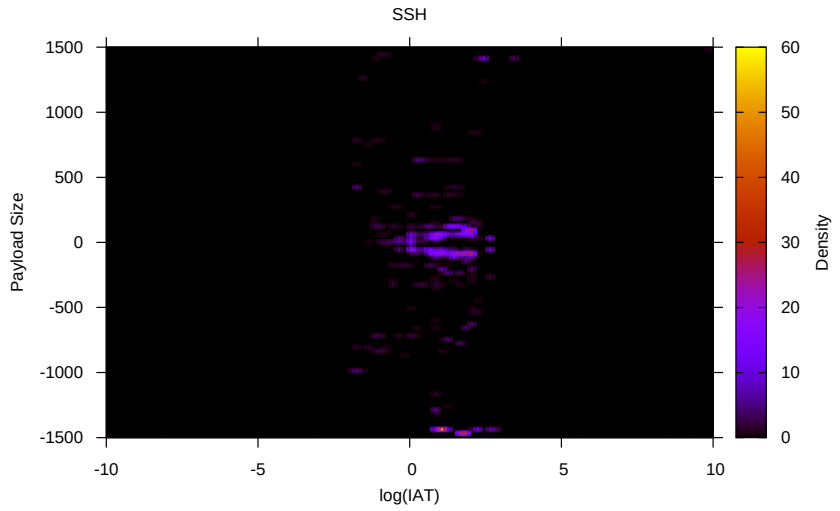
In addition to representing clusters using a radar chart, other methods of confirming successful clustering of application protocol via visual aid have been researched in recent years. Mcgregor et al. (2004) examined plots of packet size against packet IAT for flows of particular application protocols. In this work, the authors observed a number of characteristic shapes, which were indicative of the underlying application protocol. Using a similar methodology, the full trace for each cluster's members were analysed, with the payload size plotted against the value of $log(IAT)$ for each. The advantage of plotting the *log* value of IAT is that it allows one to readily identify features in the data that would not easily be seen if both payload size and IAT had been plotted linearly.

A heat-map was created for each application protocol considered in this chapter. Using the clustered datum associated with each of these application protocols, the recorded payload size and $log(IAT)$ values for each were plotted, illustrating packet size diversity in each direction over time. This chart, according to Mcgregor et al. (2004), provides a visual representation, which can be used to identify or verify the application protocol. The sign bit of the packet size was used to indicate flow direction, where a positive value indicates packet flow in the direction of the initial *synchronise* (SYN) packet, a flag set by the host originating the flow. Negative values were used to indicate packet flow in the reverse direction. For each figure, the first heat-map represents the data clustered by APIC for the protocol. Below each of these, a heat-map plotted using traces manually annotated by experts[10] of the same protocol is shown. The colour of each pixel is indicative of packet density at the given size and time. Brighter areas indicate a high concentration of packets, with darker colours indicating very few packets observed. Heat-maps for SSH (Figure 4.21), HTTP (Figure 4.22), HTTPS (Figure 4.23), POP3 (Figure 4.24), SMTP (Figure 4.25) and Bittorrent (Figure 4.26) protocols are shown below for visual analysis.

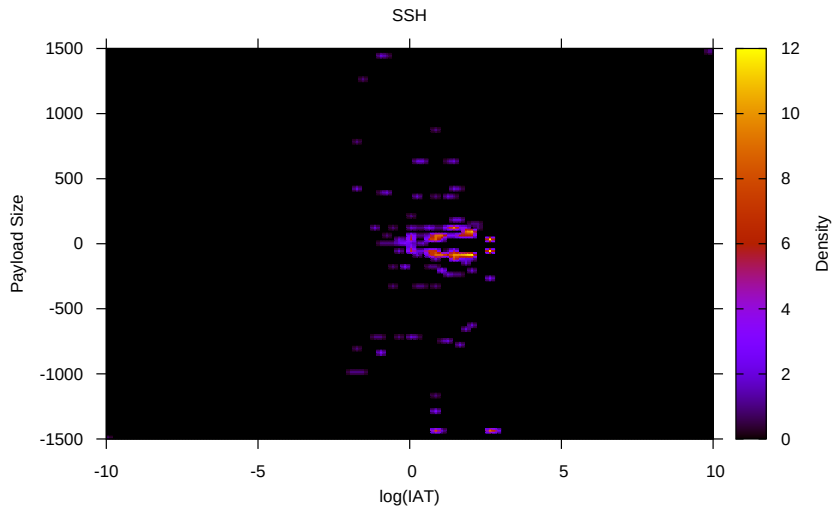
The heat-map describing APIC clustered data for the HTTP protocol (Figure 4.22) shows the highest density of packets is toward the host that initiated the flow. In this case, the majority of packet size observations are

[10]A qualified computer network expert with experience identifying application protocols from raw traces

centred where payload sizes equal the standard ethernet *Maximum Transfer Unit* (MTU) value of 1500. For HTTPs (Figure 4.23), the same observation is apparent, where the majority of packet density is observed toward the host that initiated the flow. Unlike HTTP, there is an apparent increase in packet density spread in both directions. This is attributed to the additional security exchanges HTTPS performs to secure the transit of data. The POP3 protocol (Figure 4.24) exhibits very small packet sizes toward the server, with the majority of packet size density observed toward the host that initiated the flow. This is expected, as POP3 is an inbound mail exchange protocol, designed for consuming email messages from remote mail servers. The SMTP protocol (Figure 4.25), on the other hand, exhibits the majority of packet size density in the direction from the host initiating the flow to the remote host. This is again expected, as SMTP is an outbound mail exchange protocol, used by mail clients to transmit local email messages to a remote server.

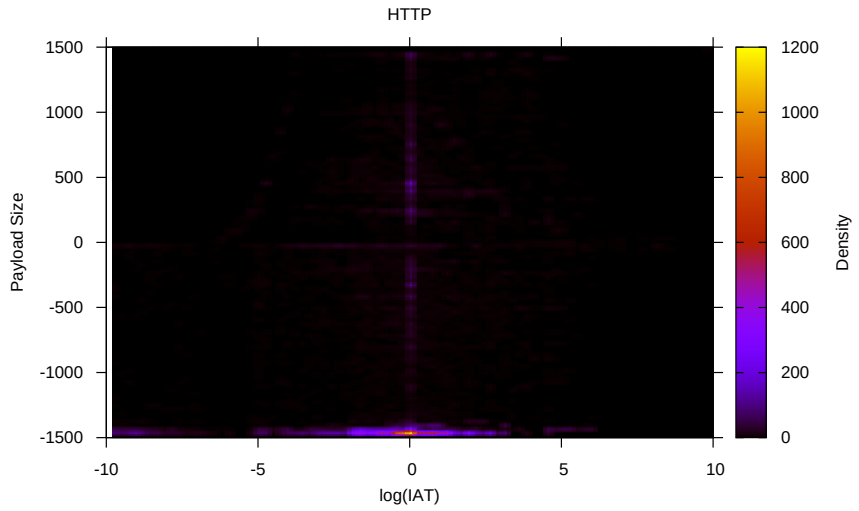


(a) APIC clustered data set

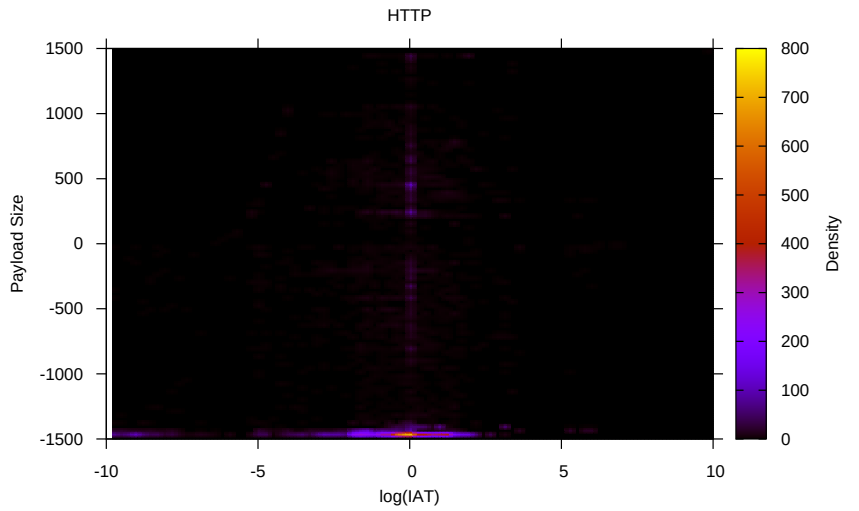


(b) Expert annotated data set

Figure 4.21: Heat-maps showing payload size against $log(IAT)$ values for the SSH application protocol.

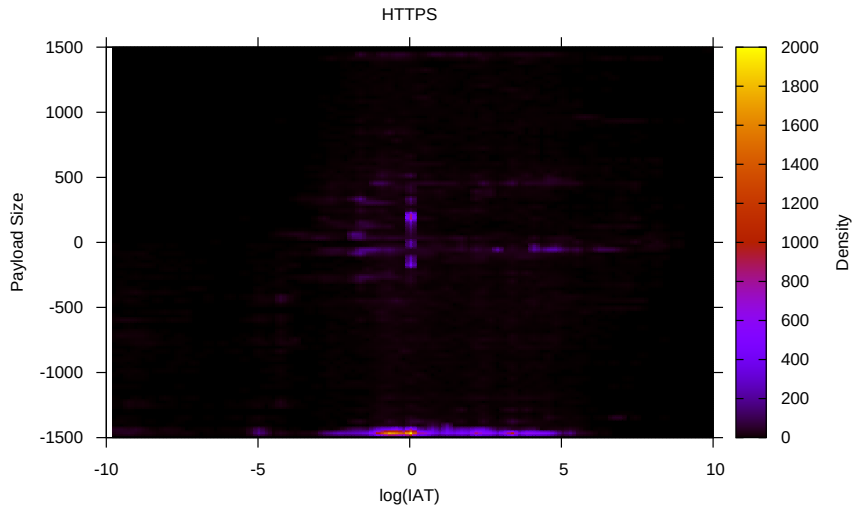


(a) APIC clustered data set

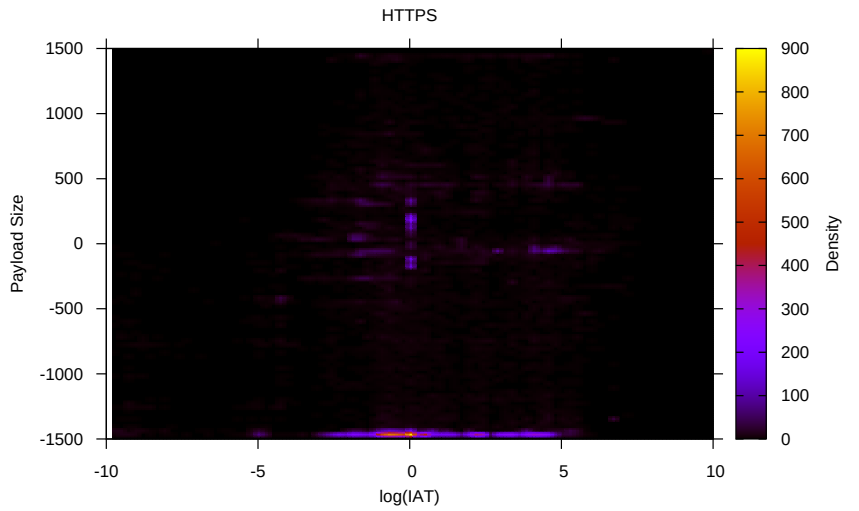


(b) Expert annotated data set

Figure 4.22: Heat-maps showing payload size against $log(IAT)$ values for the HTTP application protocol.

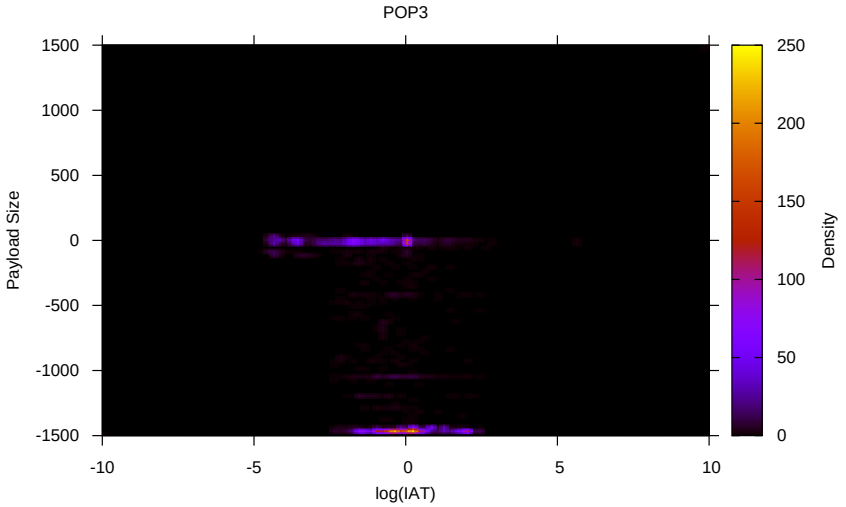


(a) APIC clustered data set

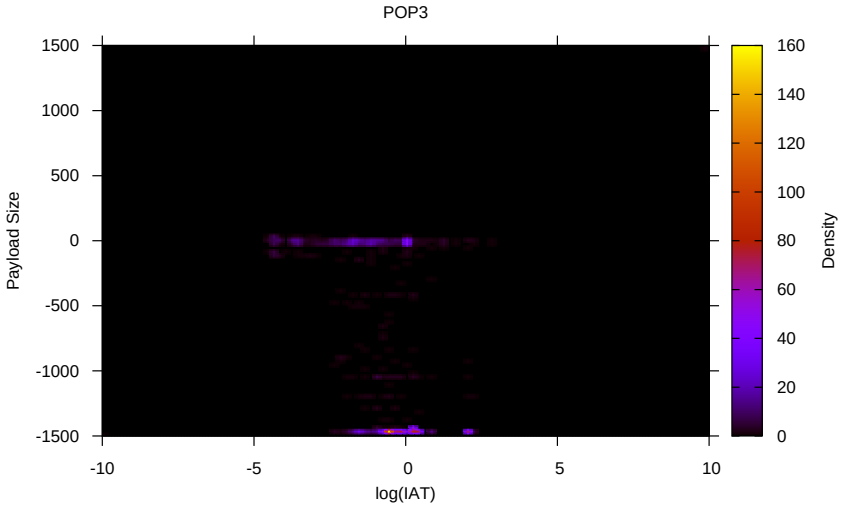


(b) Expert annotated data set

Figure 4.23: Heat-maps showing payload size against $log(IAT)$ values for the HTTPS application protocol.

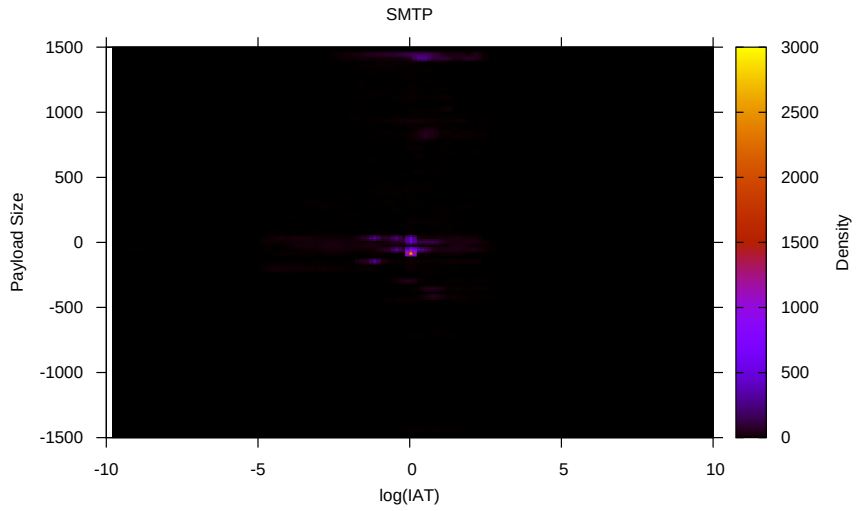


(a) APIC clustered data set

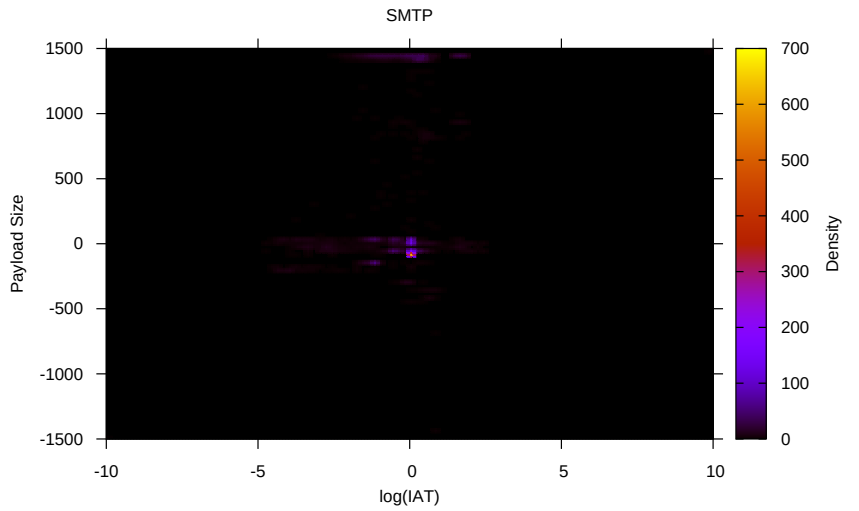


(b) Expert annotated data set

Figure 4.24: Heat-maps showing payload size against $log(IAT)$ values for the POP3 application protocol.

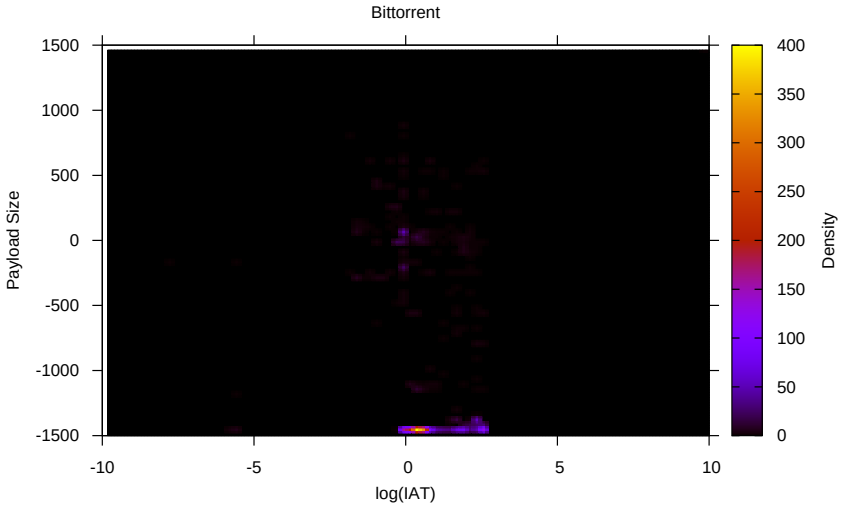


(a) APIC clustered data set

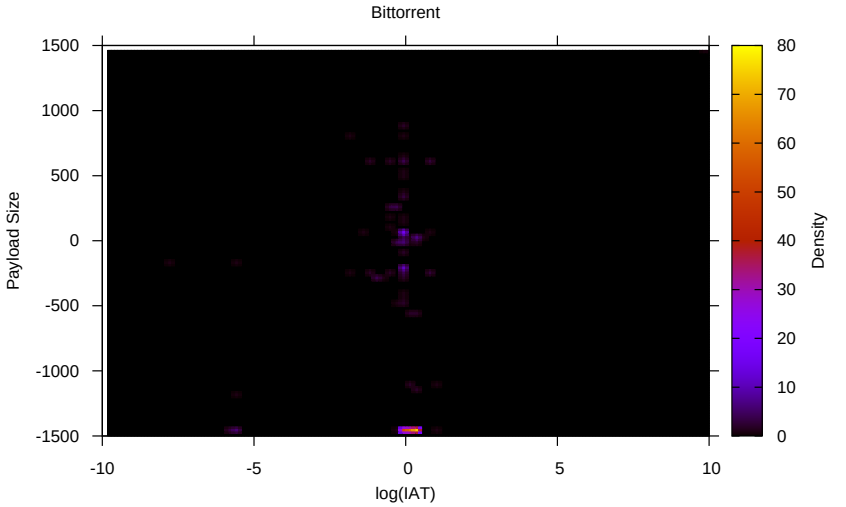


(b) Expert annotated data set

Figure 4.25: Heat-maps showing payload size against $log(IAT)$ values for the SMTP application protocol.



(a) APIC clustered data set



(b) Expert annotated data set

Figure 4.26: Heat-maps showing payload size against $log(IAT)$ values for the unencrypted Bittorrent application protocol.

Distinguishing application protocols automatically is the first step toward a fully automated IP classification system. The ability to remove manual tasks, including feature selection and data set annotation, is important for promoting increased completeness compared to existing, state-of-the-art systems. Another task highlighted in Chapter 2 where a significant amount of manual intervention is required, is the development of classifiers to identify future instances of each application protocol described by the automatically-annotated data sets.

4.6 TWEANN Classifier Development

It has already been established that a significant amount of time is spent by experts in developing accurate, customised classifiers for identifying specific application protocols. The first step is to annotate the data sets (Section 4.4), followed by manually developing a classifier to uniquely identify each protocol. These manual processes reduce the completeness of the system and are prone to mistakes - mostly human error. Section 4.1 reviewed a number of IP traffic classification methods, all of which relied on manual intervention for classifier topology definition. These classifiers were constructed using heuristics identified by experts as most optimal for classifying each application protocol.

Unlike the classification systems studied in Section 4.1, APIC automatically develops custom classifiers for each application protocol it discovers (Section 3.3). For this case study, APIC uses TWEANN models, fine-tuned with backpropagation. TWEANN parameters, including the number of neurons, connections and weights, are determined automatically using the TWEANN's artificial evolution process. The average recall score achieved by the classifiers for each feature subset defines the fitness score for the genotypes tested, where genotypes encode the connection weights and neuronal connectivity of the TWEANNs. If a classifier's results surpass a predetermined score, or if the maximum number of generations is reached for the feature subset selection GA, the last set of classifiers produced is deemed the most suitable for classifying the recorded data set.

In Goss and Botha (2012), where the same focus application protocols were tested, the neuron count for each classifier trained was statically set to twice the number of features considered. These neurons were split between the input layer and a single hidden layer, which converged on a single output neuron. Drawing inspiration from this configuration, the maximum number of neurons permitted for each topology developed in this case study was equal to twice the number of features included in the current feature genotype. The search space for the number of neurons considered by the topology evolving GA was therefore limited to $f_n + 1 \leq neurons \leq f_n * 2$, where f_n denotes the number of features represented in the feature subset genotype. For example, a feature subset genotype with a magnitude of ten would generate topologies with 11 to 15 neurons. Goss and Botha (2012) found that increasing the number of neurons beyond this value proved to increase task load significantly, where more CPU time was required to train the network due to increased computational complexity, while only a marginal, disproportionate increase in accuracy was noted.

Operating within these parameters, a topology-evolving GA determined the number of neurons and neuronal connectivity for each ANN classifier. The weights for each connection were determined using a second, weight-evolving GA. These weights were fine-tuned using the standard error backpropagation algorithm (Rumelhart et al., 1988), which affords APIC additional efficiencies when searching for more optimal weight configurations. Here, the GA prevents the backpropagation algorithm from getting stuck in the local minima, a problem often associated with the algorithm (Sher, 2010). As per Goss and Botha (2012), the total number of training iterations (epochs) for the backpropagation algorithm was set to a heuristic value of 1,000. A static value for the number of epochs allows each topology to be tested fairly, providing a strong comparative base for accurately reflecting and comparing the number of epochs required for convergence.

The best scoring clustered data set from Section 4.4 was found using a data set consisting of 20 features. A new TWEANN classifier was constructed and trained for each identified cluster. The training set for each cluster was automatically annotated, where datum that were members of the cluster being identified were labelled with “1”, while datum external to the cluster

were labelled with a "0". In this case study, TWEANN classifiers were developed to identify each of the six focus application protocols (Table 4.3). The best classifier topology and it's associated score was recorded for each application protocol.

These scores are the average scores achieved when training a classifier using five-fold cross-validation. For each fold, an 80/20 rule was applied (Fan et al., 2008), where 80 percent of the data set was allocated to the training set and 20 percent to the test set.

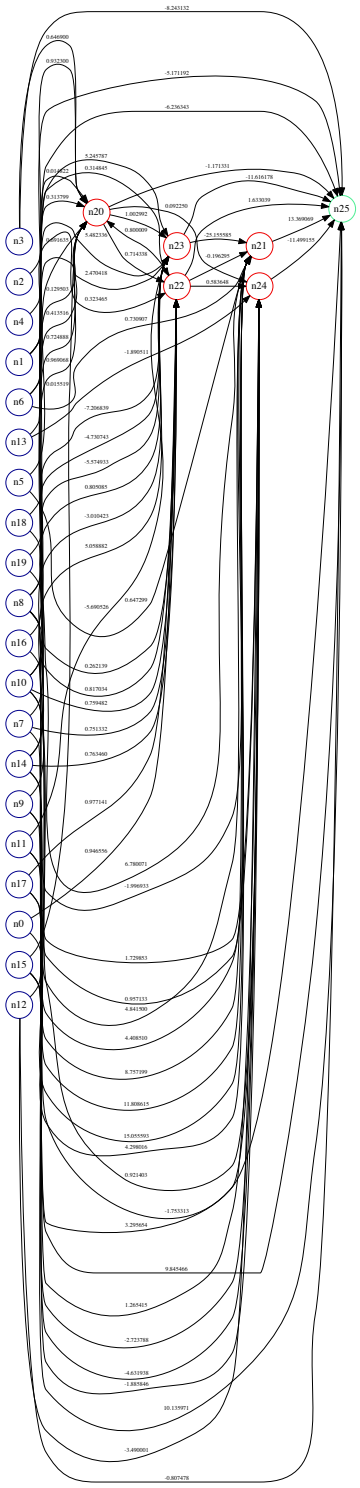


Figure 4.27: The topology of the TWEANN best matching the SSH protocol. Using this topology, an average recall score of 99.7928 percent was achieved. The topology includes 20 input (n_0 to n_{19}), and five hidden (n_{20} to n_{24}), neurons plus one output neuron (n_{25}).

The cluster describing the SSH protocol was best recalled by the topology illustrated in Figure 4.27. This topology scored an average of 99.7928 percent when parsing the training set. The topology consisted of twenty input neurons (one for each feature) and five hidden neurons. These hidden neurons were connected to both each other, the input neurons, and the single output neuron. A number of the input neurons were also mapped directly to the output neuron. The connectivity between each neuron was determined as the best architecture by the topology-evolving GA.

The cluster describing the SMTP protocol was best matched using a topology illustrated in Figure 4.28. Here, all 20 inputs are also connected through five hidden neurons and directly to the output neuron. Through this topology, an average recall score of 99.9413 percent was realised when parsing the test set at each fold.

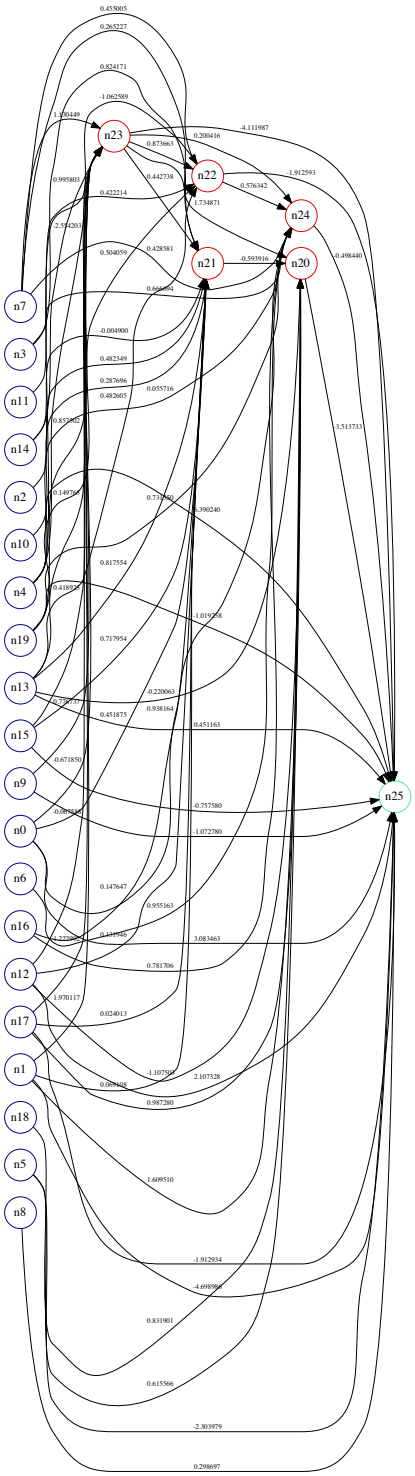


Figure 4.28: The topology of the TWEANN best matching the SMTP protocol. Using this topology, an average recall score of 99.9413 percent was achieved. The topology includes 20 input neurons (n_0 to n_{19}), five hidden neurons (n_{20} to n_{24}) and a single output neuron (n_{25}).

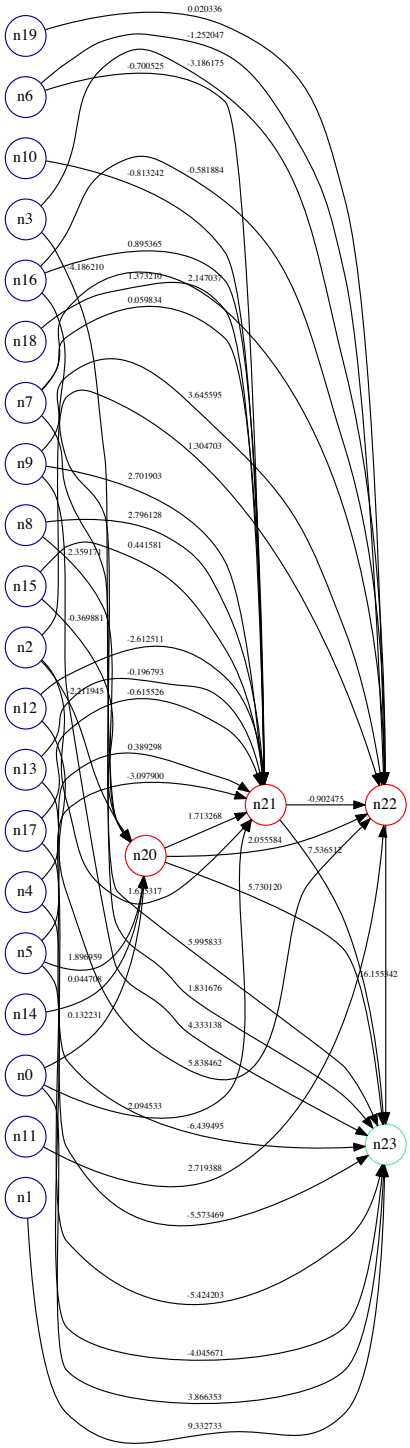


Figure 4.29: The topology of the TWEANN best matching the POP3 protocol. Using this topology, an average recall score of 99.8391 percent was achieved. The topology includes 20 input neurons (n_0 to n_{19}), three hidden neurons (n_{20} to n_{22}) and a single output neuron (n_{23}).

The topology best matching the POP3 protocol is illustrated in Figure 4.29. This classifier, comprised of twenty input, three hidden and one output neuron, achieved an average recall score of 99.8391 percent.

The HTTP protocol, the foundation of communication for the WWW is a distributed, collaborative, hypermedia data transport protocol. The clustered data describing the HTTP application protocol was best described using the topology illustrated in Figure 4.30. The average recall score achieved by this network was 99.8383 percent.

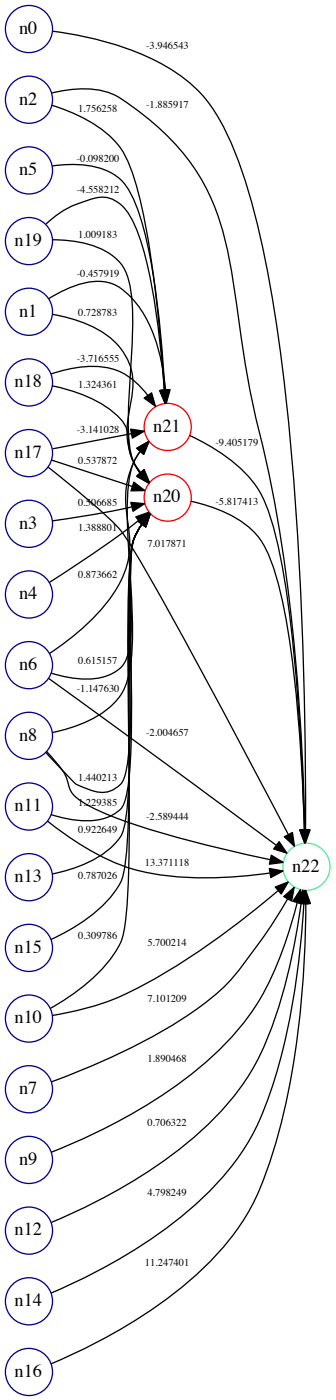


Figure 4.30: The topology of the TWEANN best matching the HTTP protocol. Using this topology, an average recall score of 99.8383 percent was achieved. The topology includes 20 input neurons (n_0 to n_{19}), two hidden neurons (n_{20} and n_{21}) and a single output neuron (n_{22}).

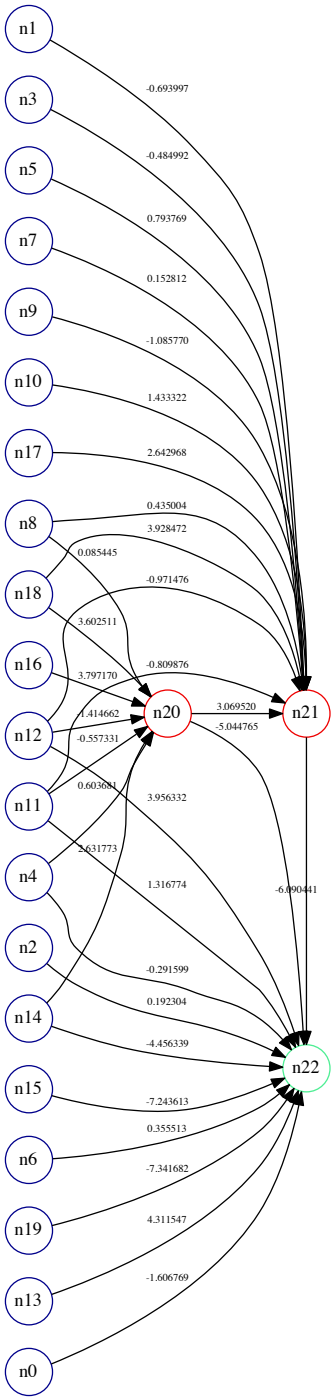


Figure 4.31: The topology of the TWEANN best matching the HTTPS protocol. Using this topology, an average recall score of 99.9159 percent was achieved. The topology includes 20 input neurons (n_0 to n_{19}), two hidden neurons (n_{20} and n_{21}) and a single output neuron (n_{22}).

The HTTPS protocol is not a protocol in its own right, but rather the HTTP protocol operating on top of the *Secure Sockets Layer* (SSL) or TLS protocol. The best matching topology HTTPS is illustrated in 4.31, where the average recall achieved was 99.9159 percent.

Finally, the topology of the TWEANN classifier best matching the unencrypted Bittorrent protocol is illustrated in Figure 4.32. This classifier achieved an average recall score of 99.5700 percent, using a single hidden neuron. Twelve of the 20 input neurons were connected to the output neuron through the hidden neuron, with the remainder connecting to the output neuron directly. Seven of the inputs were connected to the output neuron both directly and indirectly through the hidden neuron.

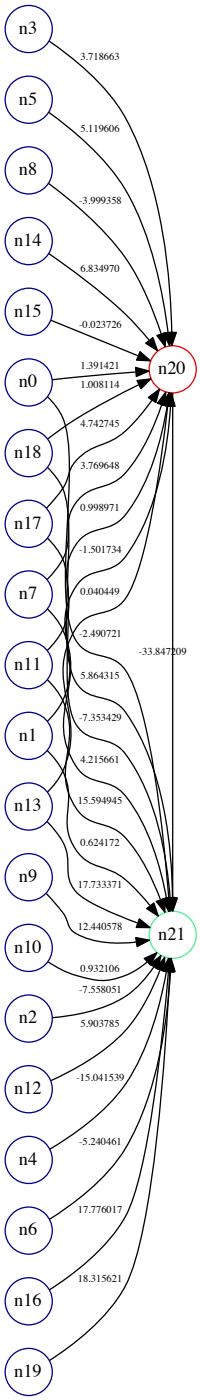


Figure 4.32: The topology of the TWEANN best matching the Bittorrent protocol. Using this topology, an average recall score of 99.5700 percent was achieved. The topology includes 20 inputs (n_0 to n_{19}) plus a single hidden (n_{20}), and a single output neuron (n_{21}).

4.7 Accuracy Comparison

It is important to quantify the results achieved in Section 4.6 by comparing them with successes of similar systems. Table 4.8 outlines results achieved by comparative systems over the past decade, where the same application protocols as those investigated in this case study were tested. The table highlights best scores achieved by each method for the focus application protocols considered in this case study. This section discusses each of these systems, providing insight into the techniques employed and the accuracy achieved for each tested application protocol.

Bernaille et al. (2006) proposes a two-phase model for classifying IP traffic. The first phase, the *training* phase, involves clustering recorded TCP traces of target applications. The authors test three popular clustering algorithms: K-means (MacQueen et al., 1967a), *Gaussian Mixture Model* (GMM) (Lindsay, 1995) and spectral on *Hidden Markov Model* (HMMs)(Baum and Petrie, 1966).

These algorithms are used to identify the number of distinct application protocols present in a supplied data set. For successful clustering, the authors stress that a number of requirements must be met. First, traces for each of the target application protocols must be contained in the data set, with a minimum number of samples present for each. The APIC method, in contrast, does not require any manual intervention for ensuring successful classifier creation (Section 4.6), a process that would otherwise hamper the completeness of the system. The APIC method, instead, retains smaller clustered datum for inclusion in future discovery processes, where these clusters may grow over time to a sufficient size. Bernaille et al. (2006) states that the number of samples for each application protocol should be of similar magnitude, so as not to bias the clustering process. The authors furthermore require manually annotated example traces of each application protocol for calibrating the clustering process. For the purposes of their research, the authors generated these examples using a commercial IP traffic classification system. The APIC method has no such requirement, relying on ML techniques to tune the clustering parameters adequately, accurately segregating each distinct application protocol automatically. The results achieved by Bernaille et al. (2006) for each application protocol

	POP3	SMTP	HTTP	HTTPS	SSH	Bittorrent
Bernaille et al. (2006)	99.70%	98.50%	99.00%	99.10%	93.80%	N/A
Alshammari et al. (2009a)	N/A	N/A	N/A	N/A	99.50%	N/A
Maiolini et al. (2009)	95.70%	N/A	99.50%	N/A	99.30%	N/A
Gargiulo et al. (2009)	99.83%	99.54%	99.29%	N/A	N/A	N/A
De Donato et al. (2014)	100.00%	96.00%	99.60%	99.10%	N/A	98.90%
Bujlow et al. (2015)	100.00%	100.00%	99.80%	N/A	94.19%	99.87%
APIC	99.84%	99.94%	99.84%	99.92%	99.80%	99.57%

Table 4.8: A list of comparable research into IP traffic classification over the past decade. These works are those where classifiers identifying the same application protocols considered by this case study were tested. The table includes the application protocol and best score achieved.

using the aforementioned requirements are presented in Table 4.8. The values represent the best noted classification scores across all tests for each application protocol. In their results, the authors show the highest average, combined score achieved was 98.7 percent, when considering all application protocols. The authors failed to include testing of peer-to-peer application protocols, including Bittorrent. The method proposed by Bernaille et al. (2006) was not easily re-implemented for testing the Bittorrent application protocol, as both the data sets and commercial classification software used were not readily available. Alshammari et al. (2009a) used two data sets to train an SSH classifier. The first data set was labelled using a commercial classification tool, where a DPI system scanned for known signatures. The SSH protocol is easily identified using DPI, as the first few bytes of the application protocol are exchanged in plain-text. The second data set was labelled using a port-based classification method. Alshammari et al. (2009a) used balanced samples of SSH and non-SSH flows to represent a subset of the original data sets. The authors argue that a balanced subset of data improves the performance of the resulting models when compared to those trained using the full data set. In total, 500,000 packets were considered, including TCP control traffic. As the APIC method considers only payload-bearing packets, it requires less packets for

processing the same flow traces, making it arguably more efficient. Alshammari et al. (2009a) trained three classifiers, using C4.5, AdaBoost and Team-based Genetic Programming algorithms, using the pre-labelled, sampled data sets. These classifiers were trained to identify only the SSH protocol using packet header statistics. Basing IP traffic classification on header information was found in Section 4.1 to be unreliable as header information is easily manipulated by the two communicating hosts (Moore and Papagiannaki, 2005; Auld et al., 2007). For this reason, the APIC method makes determinations without this data, focusing rather on statistical information that describes application protocols independent of the environment it traverses. Alshammari et al. (2009a) also states that manual intervention and further research is required to evaluate the proposed method for classifying application protocols, other than SSH. The method proposed by Alshammari et al. (2009a) is therefore only suitable for classifying the SSH application protocol, which greatly impairs its effectiveness in the general field of IP traffic classification. Due to the specific focus on SSH, the method proposed by Alshammari et al. (2009a) was not re-implemented to determine its effectiveness in classifying the other test application protocols listed in Table 4.8. Although the method focused solely on the SSH protocol, the results achieved show that the method has merit and provides noteworthy contributions. For example, the authors found C4.5 as the best performer for classifying both in-class (SSH) and out-of-class (non-SSH) samples. Using this method, a best score of 99.5 percent was observed when recalling training set data. In contrast, a 91.3 percent score was achieved when the classifier was tested against previously unseen samples from the original data set. In this case study, APIC was found to exhibit significantly higher success classifying unseen samples of the SSH protocol, realising scores in excess of 99 percent accuracy, using a method developed to identify a broad range of application protocols.

Maiolini et al. (2009) recorded network packet traces using a mirrored port on their LAN switch. Unlike Alshammari et al. (2009a), TCP control messages were removed through pre-processing methods which, according to the authors, results in a higher degree of accuracy. Not dissimilar to APIC, the method proposed by Maiolini et al. (2009) used min-max normalisation to standardise each dimension of the recorded data set. The

authors used heuristic values to calibrate each dimension of the input vector, which included the directionality of packet flow, packet size values and absolute times. The authors used a cross-validation method to determine the number of clusters, k, within the recorded data set. Using the described method, a best average score of 95.43 percent was realised across all classifiers. Lacking access to the recorded data set, which the authors state was artificially created on a university network and at a private home, the method was unable to be re-implemented. As the authors focus was on the identification of the SSH protocol from other protocols, they failed to record the scores achieved by the classifier describing SMTP, HTTPS and Bittorrent. It is unknown if these protocols were not discovered during the testing, or if the artificial data set production process simply did not cater to them. Artificial data sets remove many of the variables introduced in real-world IP network traffic, such as latency variances caused by congestion and QoS implementations on ISP networks. It is therefore assumed that the results published are higher than the results that would be achieved using data sets captured in a real-world environment, such as those recorded for testing APIC.

Gargiulo et al. (2009) use attributes inferred from the first four payload-bearing packet exchanges to classify each flow. The authors use the destination port number of the flow, combined with the directionality and size of these first four packets. The classification method described by Gargiulo et al. (2009) consists of three stages. Stage one guesses the application protocol using the destination port read from flow packet headers. Stage two determines the pattern of bits showing directionality for each flow over the first four packets, where a "0" indicates the packet was sent from server to the client and a "1", the reverse. The authors assert the distribution of flow directionality patterns across the 16 available patterns (2^4) provides sufficient evidence as to whether the application protocol is, in fact, that which was determined using the port number. Finally, a third stage uses a decision tree to parse packet size statistics from the first four payload-bearing packets. Based on the scores generated by each of the classifiers, a probability is provided as to which application protocol the flow presents. While the average, combined score for all classifiers was not provided, the individual scores presented in Table 4.8 provide sufficient

evidence that high degrees of accuracy are achievable using the proposed method. As part of the method is dependent on packet header information (port), it is possible that the reliability of the method may be called into question in real-world operation, where packet header information is manipulated by end hosts. The APIC method does not suffer this risk, as the attributes considered explicitly exclude those that may be manipulated by the communicating hosts. De Donato et al. (2014) present an open-source tool for traffic classification, the *Traffic Identification Engine* (TIE). This engine uses a number of techniques to classify traffic, including: J48 Decision Trees, K-nearest Neighbour, Random Tree, Ripper, Multilayer Perceptron, Naive Bayes (Russell and Norvig, 2009), Portload (a custom DPI engine) and, finally, classic port-based classification.

Table 4.8 highlights the best results achieved by this method, when each technique (algorithm) was tested individually using network trace data collected at the University of Napoli, Italy. Each classification technique is loaded as a separate plug-in to the system, which supports a number of methods for training classifiers. In each case, the training is provided through a supervised learning approach, using pre-labelled training sets. The authors cite results for all but one of the test application protocols considered by this case study, namely SSH. The TIE engine was not implemented for generating a result for SSH in this case study, owing to tight time constraints.

Bujlow et al. (2015) reviews seven of the most popular DPI-based IP traffic classification systems. These systems include PACE [11], OpenDPI [12], nDPI [13], L7-filter [14], Libprotoident [15] and Cisco NBAR [16]. The results in Table 4.8 are the highest observed using any of these systems. The authors emphasise that the signatures used in each of these systems must be well defined (manually) and then kept up-to-date. This is contrary to operating the APIC method, where classifiers are updated and replaced automatically. Continuous evolution of classifiers is required to maintain

[11] http://www.ipoque.com/products/pace

[12] http://www.opendpi.org/

[13] http://www.ntop.org/products/deep-packet-inspection/ndpi/

[14] http://l7-filter.sourceforge.net/

[15] http://research.wand.net.nz/software/libprotoident.php

[16] http://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html

accuracy as the application protocol itself also evolves. To test each of these systems concurrently, Bujlow et al. (2015) designed their own traffic-replay system. This system is said capable of reading and replaying packet captures (network trace files) through each of the seven IP traffic classification systems. Here the timing associated with each packet is also maintained. The classification results of each replay event was logged, along with the name of the application. The name of each application protocol was inferred from the pre-labelled test data set. The authors assert, as is apparent from the results in Table 4.8, that most traditional application protocols are generally well detected. Due to the lack of access to the test data set and the traffic replay agent, the experiment could not be re-implemented in this case study to ascertain the effectiveness of DPI in classifying the HTTPS application protocol.

Although the accuracy achieved by the classifiers listed in Table 4.8, for the most part, exceed 95 percent, it is important to consider the effect portability has on their accuracy. A classifier should be portable, retaining a high degree of accuracy when deployed on networks other than where it was trained. The scores presented in Table 4.8 are those achieved when classifiers were tested against data sets recorded on the same network from which their training sets were derived.

The following section argues for portability as a new metric for measuring the success of IP traffic classification systems. The following section discusses each of the methods listed in Table 4.8 based on this new metric.

4.8 Portability Comparison

It has already been explained that the completeness of an IP traffic classification system relies heavily on the portability of classifiers. While it is often possible to manually train models for each network, classifier portability ensures consistency in recognising each application protocol across multiple networks. Commercial system IP classification vendors, for example, need to ensure that the classifiers they produce are effective on all customer networks. It is impractical for these vendors to develop custom application protocol classifiers for each of these networks. In Section 4.1, it was argued that specific features considered by a number of systems do not

	PKT IAT	PKT Size	PKT Header	PKT Direction	Payload
Bernaille et al. (2006)		M		M	
Alshammari et al. (2009a)	M	M	M		
Maiolini et al. (2009)			M	M	
Gargiulo et al. (2009)		M	M	M	
De Donato et al. (2014)	O	O	O	O	O
Bujlow et al. (2015)					M
APIC	O	O	O	O	O

Table 4.9: A high-level view of features used by each of the comparative systems considered in this case study. Based on the investigations concluded in Section 4.1, many of the feature groups listed in this table are not conducive to the production of portable classifiers. In this table, an "M" denotes a mandatory feature while an "O", one that is optional. Finally, no character for a particular feature group indicates that the feature is not considered or supported by the method.

promote portability, due to their dependence on network-specific metrics, such as latency. This section compares these findings with the features and methods selected by the focus systems listed in Table 4.8.

Table 4.9 lists each of these systems, broadly grouping the features each considers into a high-level class. Here, an "M" denotes at least one feature of the class that is mandatory (required by the system), while an "O" dictates the features that are optional. Each of these feature classes, or categories, are discussed below with specific reference to their support of portability.

Packet Inter-arrival Time (PKT IAT) is a metric inferred by classification systems by calculating the *time difference of arrival* (TDoA) between packet exchanges of a flow. Although expressing these metrics as a ratio may provide similar results on a single network, the same application protocol may exhibit a completely different pattern on another. Features that rely on timing are not always feasible in real-world operational environments as this information is often skewed by QoS policies and network congestion (Goss and Botha, 2012). Alshammari et al. (2009a) required the PKT IAT attribute for their method, acknowledging that their proposed method was not portable.

The size of the first few packets of a flow (PKT Size) is used by a

number of systems, including Bernaille et al. (2006), Alshammari et al. (2009a), Gargiulo et al. (2009) and APIC. Section 4.1 affirms that the size of the first few payload-bearing packets remains constant for each application, as these packets are responsible for application protocol establishment. After the application protocol has been established, data exchange begins, where packet sizes will vary. For this reason, most systems consider size metrics for only the first few packets of a flow. Features describing the size of the first payload-bearing packets are therefore considered portable and support the portability of classifiers that include them.

Packet header (PKT Header) information, including TCP port and protocol information, is easily manipulated by the communicating hosts. While the IANA assign well-known ports to most applications, devious application protocols are capable of communicating on any port. Classic port-based application protocol identification has subsequently become increasingly redundant in modern networks, as discussed in Section 4.1. Alshammari et al. (2009a) and Gargiulo et al. (2009) both use packet header information to infer application protocol type from flows. As this information may have been manipulated by the communicating hosts, there is no guarantee that the results they achieved in Table 4.8 would perform as well in a less controlled environment. This, according to the published results, remains largely untested (Alshammari et al., 2009a; Gargiulo et al., 2009).

The directionality of packet flow (PKT Direction) is a feature considered mandatory by Gargiulo et al. (2009). Not dissimilar to the PKT Size attribute, the direction of the first few payload-bearing packets of a flow remains constant during application protocol initialisation. This attribute is therefore considered portable as it is fixed per protocol and is not influenced by network variables such as latency and congestion.

Finally, payload inspection or DPI is an extremely accurate, portable method of classifying plain-text application protocols. The classifier uses predefined regular expressions to match specific byte strings exchanged between communicating hosts during the initial application protocol establishment. These methods have recently, however, been subjected to a number of privacy concerns as the classifier requires access to sensitive user

data. Furthermore, these classifiers are only capable of classifying application protocols where the byte exchanges are exposed as plain text. An encrypted, or otherwise opaque, application protocol string renders the classifier inept.

Based on these findings, the only systems whose methods support portability are those proposed by Bernaille et al. (2006), De Donato et al. (2014), Bujlow et al. (2015) and APIC.

Both accuracy and completeness were identified as important metrics for evaluating IP traffic classification systems. This chapter also introduced a third metric - portability. Portability provides reuse of classifiers across networks, ensuring consistent classification accuracy for each application protocol. This increases the completeness of a system, as application protocol classifiers developed on one system can be shared with other, disparate systems. Achieving a high level of accuracy and completeness has already been argued as heavily dependent on the level of automation the method exhibits. The following section provides a comparison of the level of automation offered by each method in Table 4.8.

4.9 Automation Comparison

The level of automation achieved by an IP traffic classification system has a profound effect on the completeness of the system. Systems exhibiting higher degrees of automation can efficiently produce classifiers for identifying new application protocols as they present on a network. Table 4.10 presents a number of categories in which automation could be achieved in this task, providing a measure for each of the comparative systems discussed in this case study.

Each of the comparative systems listed in Table 4.10 requires annotated data sets for training, or otherwise tuning, their classifier creation processes. This dependency on the supply of manually annotated data sets severely reduces the completeness of these systems, as new classifiers may only be defined once an expert has supplied the training data. The APIC method is the only system in this list that did not require a manually annotated data set for producing accurate classifiers.

The *Protocol discovery* category describes the ability of the IP traffic

	Data Set Annotation	Protocol Discovery	Classifier Production
Bernaille et al. (2006)		X	
Alshammari et al. (2009a)			
Maiolini et al. (2009)		X	
Gargiulo et al. (2009)			
De Donato et al. (2014)			
Bujlow et al. (2015)			
APIC	X	X	X

Table 4.10: A list of comparative systems and the degree to which each demonstrates automation. An "X" indicates that the system achieves automation for the respective category. In this table, only APIC achieves automation for each of the specified categories.

classification system to automatically aggregate and segregate application protocol traces from mixed, noisy data sets of network flow traces. Bernaille et al. (2006), like Maiolini et al. (2009), use clustering techniques to identify and group related flow trace datum within a supplied data set. While Bernaille et al. (2006) used clustering to group application protocol traces, full automation was not achieved as annotated data sets were required for tuning the clustering process. The method proposed by Maiolini et al. (2009) also depended on the supply of manually labelled data sets as input to their clustering process. The APIC method was the only system that was able to discriminate between application protocol traces automatically, without depending on manually annotated data sets.

Finally, the only method that concentrated on automatically creating a distinct, custom classifier for identifying each application protocol identified, was APIC. Bujlow et al. (2015) required that a unique classifier be manually defined by experts using regular expressions for each plain-text application protocol on the network. De Donato et al. (2014) provided an engine for defining and training classifiers, however this process was executed manually. The method provides a mechanism for supervised training, where the supply

of manually annotated data sets for each application protocol was required.

4.10 Discussion

The case study described in this chapter highlighted the contribution of the APIC method in that, although a general method, it automatically created classifiers to identify previously unseen application protocols on an IP network with a high degree of accuracy when compared to similar systems designed manually, specifically for this task. While accuracy is an important metric for IP traffic classification systems, completeness - or the availability of classifiers - is equally important. The ability to automatically create new, portable and accurate classifiers for identifying each application protocol traversing a network is important for supporting both the completeness and accuracy of an IP traffic classification system. In this case study, APIC was demonstrated to excel in both of these areas. The following subsections provide highlights of this success, supporting the first two objectives of this dissertation (Section 1.3).

4.10.1 Completeness

The APIC method was shown capable of automatically selecting the best feature subset (verified by a manual, exhaustive search of the feature space), identifying distinct patterns in this data set and defining customised, highly accurate classifiers to identify future instances of each pattern. Optimisation of the feature set, by pruning irrelevant and redundant features, reduces the complexity of classification tasks, drastically improving task efficiency (Dash and Liu, 1997). In addition to feature pruning, APIC used GAs to find the best hyper-parameters for clustering these feature subsets, grouping network flows by their respective application protocol. Providing optimisation here supported the clustering process by increasing diversity between clusters, while ensuring that datum within the same cluster were more closely related. This is unlike the clustering process described by Bernaille et al. (2006) and Maiolini et al. (2009), where manually labelled data sets were required for tuning the clustering process. In Section 4.9, APIC was found to be the only method capable of producing its own labelled training data. After the clustering process was complete, the APIC method labelled each of the

clustered datum with their respective cluster identifier. This automatically-labelled clustered data formed the training sets for the TWEANN semi-supervised learning, hybrid development process. These TWEANN classifiers were trained to identify future instances of each cluster (application protocol).

A unique TWEANN classifier was automatically produced by APIC for each of the identified application protocols, using the labelled data set. Unlike conventional ANNs, a TWEANN classifier allowed the topology of each classifier to be modelled in a way that best described the application protocol. This customisation produced highly accurate classifiers, capable of identifying all tested application protocols with a confidence score[17] exceeding 99 percent. In Section 4.8, the features selected by the APIC method were found to support portability, therefore TWEANNs produced using these features were also inherently portable.

Any unidentified flows were recorded and appended to a new data set, which was periodically parsed by APIC in search of new application protocols (Figure 4.33). TWEANN classifiers trained for each newly-discovered application protocol were automatically fed back to the classification engine (Figure 4.33) for identifying future instances of the application protocol. The automatic creation of signatures in this way ensured that the classification system was always up-to-date, with classifiers readily available at the first sign of a new application protocol. As application protocols evolved and evaded existing classifiers, new classifiers were produced and fed back into the classification system.

The automated production of accurate, portable classifiers for identifying application protocols across disparate network deployments is one of the main contributions of the APIC method in the field of IP traffic classification. The ML processes adopted by APIC for the production of these classifiers required no human intervention, improving on the state-of-the-art technique, where it was found - in Section 4.9 - that the majority of classifier training and definition processes still require manual interaction by human experts. The manual definition of classifiers limits completeness, a problem APIC overcame by applying ML techniques. The APIC method, with its automated classifier development and replacement

[17]A confidence score is the level of certainty, or probability, that the subject datum is correctly matched by a specific ANN

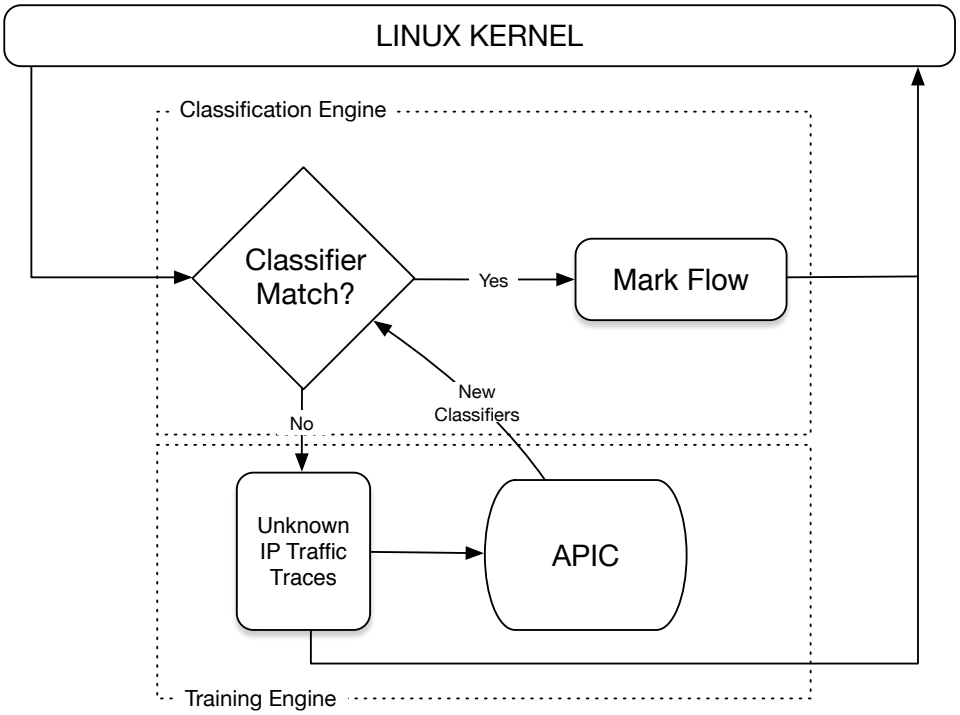


Figure 4.33: Any flow not identified by APIC classifiers was recorded. These recordings were parsed by APIC periodically, while searching for new application protocols. Any new TWEANN classifiers produced by APIC were fed back into the classification engine to identify future instances of the same application protocol.

mechanisms, demonstrated a higher degree of completeness compared to conventional IP traffic classification systems which depend on human experts employed by the vendor to release new classifiers periodically through software updates. Instead, APIC fostered a self-sustaining IP traffic classification ecosystem, where new application protocols were automatically discovered and new classifiers introduced on the network in near real-time.

This supports the first objective of this dissertation (Section 1.3), in which the hypothesis regarding whether the APIC method could automatically and accurately identify new patterns in unlabelled, mixed, noisy data sets (such as those describing IP traffic flows) was found to be true. The following subsection focuses on the formation of, and the accuracy achieved by, the TWEANN classifiers produced in this case study.

4.10.2 Accuracy

The second metric most often associated with measuring the success of an IP traffic classification is *accuracy*. While an IP traffic classification system should be complete, the available classifiers should be capable of identifying previously observed (learned) application protocols with a high degree of accuracy. This subsection describes the formation of classifiers and the accuracy each achieved for the focus application protocols considered in this case study. To challenge the state-of-the-art method of IP traffic classification, the classifiers produced by APIC should produce task performance results that surpass, or are comparable to, existing systems.

The best scoring feature subset discovered by APIC in this case study consisted of 20 features, where the remaining five were discarded as insignificant or irrelevant. Of the 20 features, ten were payload-bearing packet directionality indicators, six were initial payload bytes and the remaining four were payload size statistics. These features are comparable to those manually selected by Goss and Botha (2012) and all supported portability, as described in Section 4.8. While Goss and Botha (2012) used payload size information of the first four payload-bearing packets, APIC used the mean, min, max and standard deviation values recorded for the first ten payload-bearing packets. The APIC method ruled out the inclusion of the inbound versus outbound byte ratio and packet IAT. The packet IAT metric was also excluded by Goss and Botha (2012), due to external skewing by QoS policies and network congestion. Removing the dependency on network conditions affords classifiers portability to networks other than where they were produced.

The application protocols tested in this case study included those that transport their payload data in plain text, and those obfuscating their payload data through encryption. In addition to payload obfuscation, the application protocols tested included those that operate using client-server connectivity architectures and those adopting the emergent peer-to-peer connectivity model. This case study, therefore, focused specifically on six popular application protocols, namely SMTP, POP3, HTTP, HTTPS, SSH and Bittorrent. These application protocols were chosen due to their diversity and the number of comparable works available where the same application protocols were tested. Table 4.11 provides information about

Protocol	Payload	Connectivity Model
SSH	Encrypted	Client-server
SMTP	Plain-text	Client-server
POP3	Plain-text	Client-server
HTTP	Plain-text	Client-server
HTTPS	Encrypted	Client-server
Bittorrent	Plain-text	Peer-to-peer

Table 4.11: A list of test protocols considered in this case study, along with their payload obfuscation and connectivity model information.

each of these protocols, with specific reference to payload obfuscation and connectivity models. It is important to note that certain protocols, such as Bittorrent, operate using both plain text and encrypted communications. In this case, each would be described by a distinct TWEANN classifier. The application protocols chosen for accuracy comparison in this case study were those of similar works with already-published results. These application protocols, listed in Table 4.11, are common to many public networks and are thus the focus of many IP classification system tests. The specific works chosen for comparison in this case study (Table 4.8) were the most commonly cited of these works, published over the last decade.

The APIC method created a TWEANN classifier for each of the six considered application protocols listed in Table 4.11. The topology of the network, including the number of neurons and their neuronal connectivity was evolved to an optimal structure using a GA. A second GA was used to evolve the weights of each network, providing increased efficiencies compared to a local search for optimal weights from random weights adjusted by the backpropagation algorithm. Instead, the GA calculated near optimal initial weights, which the backpropagation algorithm fine-tuned, achieving convergence on a satisfactory local maximum. While 1000 training iterations were afforded to the backpropagation algorithm, in each case the algorithm achieved convergence almost immediately, as illustrated in Figure 4.34. Here, a sharp decline in the output deltas during training is observed almost immediately. This sharp decline is a testament to the efficiencies gained through initial weight selection by a GA, prior to being parsed by the backpropagation algorithm.

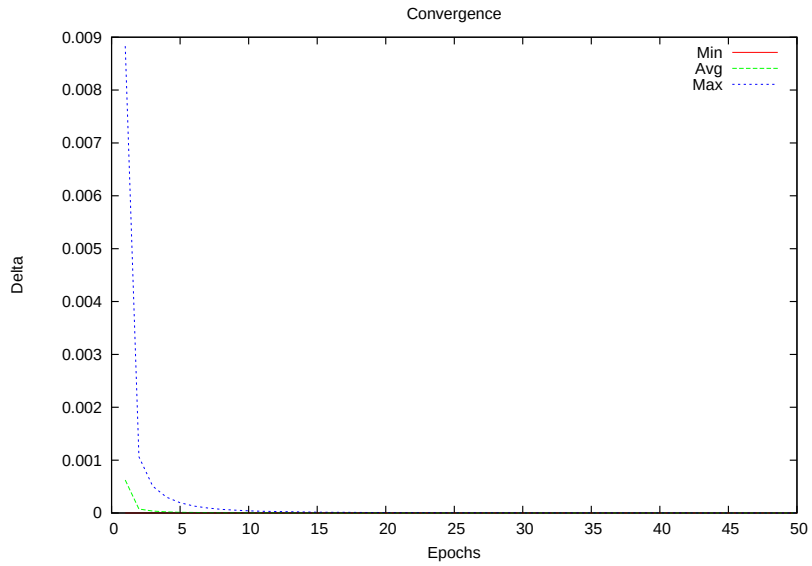


Figure 4.34: The convergence graph for the backpropagation algorithm executed on the topology illustrated in Figure 4.28. The graph has been zoomed to include data for the first 50 epochs only. The steep drop in delta over a short number of training epochs (less than ten) is testament to the success of APICs weight-adjusting GA.

Section 4.7 compared six similar systems to the accuracy results achieved by the APIC method. While many reported remarkably high accuracy, there were a number of caveats accompanying the published results. For example, the results published for the classifiers tested by Bernaille et al. (2006) were only achievable using their cluster and port-labelling heuristic, where an application's "standard port" was assigned to each cluster. The authors argued that this attribute greatly improved the accuracy achieved by their system, which was evident in their results. This finding is contrary to findings in Section 4.1.1, where the use of standard ports to identify application protocols was found redundant and prone to error. While Bernaille et al. (2006) stated that port numbers were only used "when meaningful", they agreed that the applicability of port-based classification was becoming increasingly limited due to dynamic port selection strategies.

Alshammari et al. (2009a) assert that an assumption is made, whereby all application protocols are adhering to IANA-assigned application ports. Section 4.1.1 concluded that current trends show that an increasing number of application protocols select port numbers at random, or elect to operate on well-known ports to confuse and evade traffic shapers and firewalls. This, along with latency-sensitive metrics considered by the method, brings into question the ability of the classifiers to produce high degrees of accuracy when ported to networks other than where the initial training set was recorded. Alshammari et al. (2009a) also recognise this in their publication, stating that tests showed that the classifiers performed poorly when deployed on other networks.

The DPI-based classifiers produced by Bujlow et al. (2015) produced exceptionally high accuracy scores for many application protocols, however all of these classifiers were created manually using regular expressions to match plain-text application protocols. This was demonstrated by Bujlow et al. (2015) attempting to classify encrypted bittorrent, where only 78.68 percent accuracy was realised. This is in accordance with the findings of Section 4.1.2, where DPI-based approaches were deemed extremely successful for identifying plain-text flows, however are rendered inept when packet payload is rendered opaque. This observation is of increasing importance, especially as the trend for application protocol developers to use encryption, prevails.

The best scoring topology for each of the six application protocols produced by APIC tracked in this experiment was illustrated in Section 4.6. Here, the best scoring classifiers all scored in excess of 99 percent. While these results are comparable to those in Section 4.7 and Section 4.1, it is important to remember that these results were achieved autonomously, without the aid of a human expert. Where the comparable works used manual processes to annotate data sets and create classifiers, APIC was able to perform these tasks automatically using ML techniques. Furthermore, the classifiers produced by APIC were found to be portable. Both the automation of classifier creation and portability of these classifiers increases the completeness of IP traffic classification systems. A more complete system results in less "unknown" IP traffic flows, increasing the control administrators have over their network. In addition to being

complete, an IP traffic classification system should also be accurate. In this case study, APIC was found capable of producing classifiers that rivalled those of comparative systems, designed manually by experts over the past decade. These results are summarised in Table 4.8.

4.11 Conclusion

This chapter presented a case study describing the efficacy of the APIC general method for the task of IP traffic classification. In this case study, APIC demonstrated the ability to automate feature selection, application protocol identification and classifier production processes, increasing the overall completeness of IP traffic classification systems. The ability to select an optimal feature subset, identify patterns within a mixed, noisy data set and automatically produce portable, accurate classifiers for IP traffic classification tasks are each contributions to the field of IP traffic classification. Furthermore, the use of TWEANN models to classify IP traffic is also a novel approach and a further contribution of this dissertation. The ability of APIC to automatically and accurately identify new patterns in unlabelled, mixed, noisy data sets, producing classifiers for identifying future instances of each pattern, supports the research objectives defined in Section 1.3.

The following chapters explore the application of the APIC method in a more general sense, across a variety of classification problems in a broad range of tasks. Successfully demonstrating this property satisfies the third and final objective of this dissertation - to demonstrate APIC's value as a general, broadly-applicable method for most classification tasks.

Chapter 5

Network Anomaly Detection

Protecting a computer network against unlawful intrusion, or against otherwise malicious attacks, is important. This is becoming more relevant, especially as these networks expand their reach by connecting to public networks, such as the Internet. The Internet was designed with minimal processing and best-effort packet switching in mind, forwarding both safe and malicious packets (Bhuyan et al., 2013). This seemingly unregulated network provides a multitude of opportunities for cyber attackers, motivated by a number of factors, including revenge[1], prestige[2], politics[3] and money[4].

Intrusion Detection Systems (IDS) are tools designed to thwart these attacks, both pro-actively and reactively. These systems have become fundamental components of computer security architecture (Juma et al., 2014), identifying vulnerabilities and mitigating attacks against the networks they protect. IDSs can be divided into two main categories: *Misuse Intrusion Detection* (MID) and *Anomaly Intrusion Detection* (AID). Misuse detection is based on prior knowledge of an attack signature and the application of classifiers to identify instances of these attacks using pattern recognition *Machine Learning* (ML) algorithms. This category of IDS is akin to *Internet Protocol* (IP) traffic classification (Zhang et al., 2009a) and anti-virus (Szor, 2005) systems. In contrast, anomaly detection identifies any network activity deviating from normal trends as a possible

[1] `https://goo.gl/rCJqJq`
[2] `https://goo.gl/0wEwDf`
[3] `https://goo.gl/kwc7nH`
[4] `https://goo.gl/x59waE`

intrusion. While MID systems have shown favourable results, regrettably, AID systems are often plagued by high rates of false alarms (Tsai and Lin, 2010).

Chapter 4 demonstrated the capabilities of the APIC method as an IP traffic classification system, where it was found to exhibit properties for increased completeness, accuracy and portability when compared to similar systems. The method has, therefore, already been demonstrated as suitable for application as a MID system. This chapter presents a case study that evaluates the efficacy of the APIC general method as an AID system, first using the pre-labelled *Information Security Centre of Excellence* (ISCX) IDS 2012 public data set[5] (Section 5.4), followed by an evaluation using a private data set comprised of Cisco netflow traces, reported by the edge router of an enterprise network (Section 5.5). The first test demonstrates the efficacy of the APIC method compared to similar systems, while the second test evaluates application of the method in a real-world setting. A successful demonstration in both of these areas provides evidence that the APIC method can be applied in a broad range of network classification tasks, including MID and AID tasks, supporting the third and final objective of this dissertation (Section 1.3).

The following sections provide a brief background on information security and anomaly detection, with specific reference to the type of anomalous traffic tested in the case study.

5.1 Information Security and Anomaly Detection

It has already been established that a computer network should be protected to ensure the *Confidentiality, Integrity and Availability* (CIA) of an organisation's data is maintained. Often referred to as the CIA triad (Greene, 2006), these elements serve as a guideline for security policy development within an organisation. An attack against any of these elements, illustrated in Figure 5.1, is an attack on the information security of the organisation. The protection of an organisation's data is critical as it

[5] `http://www.unb.ca/research/iscx/dataset/`

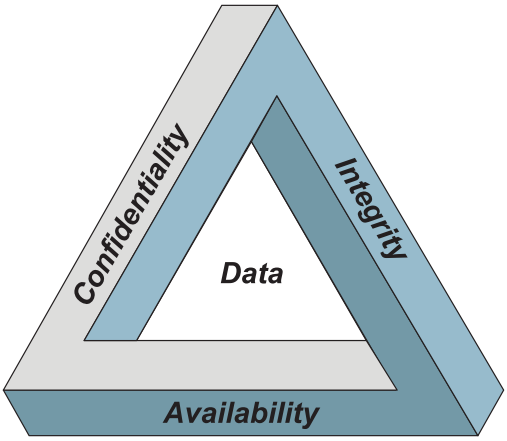


Figure 5.1: The confidentiality, integrity and availability elements comprising the CIA triad. A breach of any element may compromise the security of the data of an information security system. Adapted from Gregory (2014).

is often viewed as the organisation's greatest asset (Redman, 2008). Safeguarding the data by protecting the CIA elements is therefore akin to protecting the tangible assets of the company.

Within a computer network context, an AID system is more commonly referred to as a *Network Behaviour Anomaly Detection* (NBAD) system. These systems are one approach to detecting attacks that threaten the information security of an organisation. This approach is often viewed as complimentary to systems that detect network threats using a signature-based approach, such as those reviewed by Liao et al. (2013). The NBAD system monitors the network for any event or trend that could indicate the presence of a threat. For these systems to be effective, a baseline of normal network or user behaviour must first be observed over a sustained period. Once the initial network profile has been defined, any deviation from normal operating parameters will result in an anomalous traffic flag being raised. Anomaly detection is not limited to computer networks, having already been applied in a number of domains including intrusion detection (Garcia-Teodoro et al., 2009), fraud detection (Phua et al., 2010), fault detection (Idé and Kashima, 2004), system health monitoring (Niu et al., 2011) and event detection in sensor networks (Zhang et al., 2010). Anomalies may be introduced in data sets for a variety of reasons, including malicious activity, such as credit card fraud,

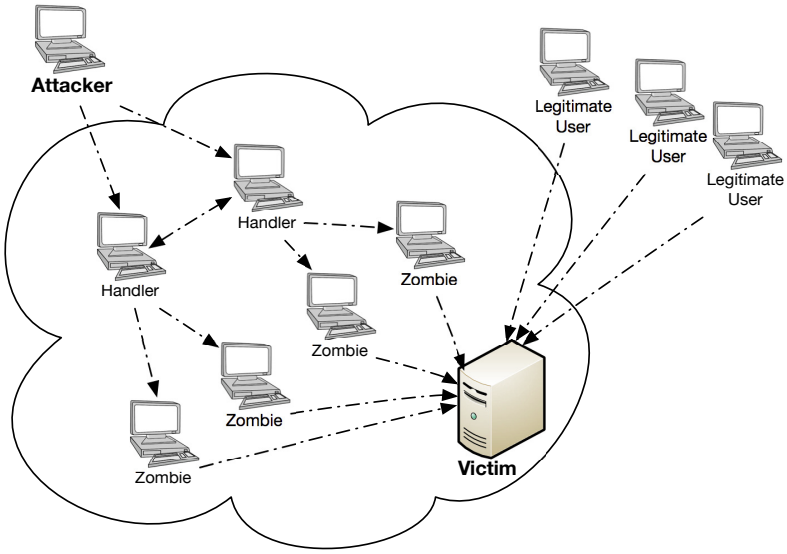


Figure 5.2: An illustration, adapted from Bhaya and Manaa (2014), depicting the typical *Distributed Denial of Service* (DDoS) attack model. An attacker sends an attack command to a number of handlers, who forward the command to zombie attackers. These in turn send malicious packets to the victim, quickly overwhelming their resources, thus preventing access for legitimate users.

cyber-intrusion, terrorist activity or the breakdown of a system (Chandola et al., 2009). While this chapter demonstrates application of the APIC method as an NBAD system, the methods discussed support implementation in many of these other domains as well.

An attack commonly considered by NBAD systems, due to their abounding presence on today's public networks, is the *Distributed Denial of Service* (DDoS) attack. The following section details this type of attack, describing it's many forms and effects on computer network infrastructure.

5.2 Distributed Denial of Service Attack (DDoS)

In a DDoS attack (illustrated in Figure 5.2), an *attacker* commands a number of *handlers* to orchestrate an attack directed at a specific *victim*. These handlers are often compromised host machines, where the operator (user) is often unaware that their machine is participating in these attacks.

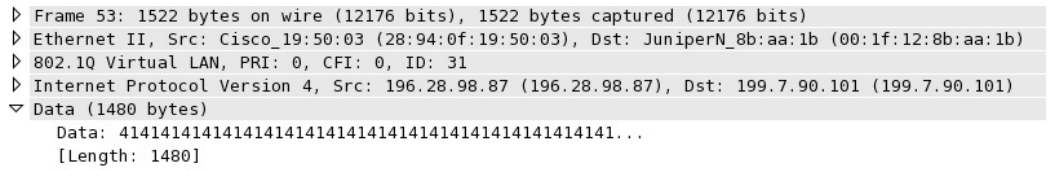


Figure 5.3: An example DDoS packet, showing a stateless packet (IP protocol) with a payload consisting of repeated hexadecimal characters, representing the ASCII "A" letter (0x41). The payload size is 1480 which, when added to the IP header, makes the packet - at 1500 bytes in size - the *Maximum Transfer Unit* (MTU) for standard ethernet networks.

The handler machines forward the attack instructions, including the victim's information, to a number of *zombies*. These zombies are compromised host machines, responsible for crafting and forwarding attack messages. By using multiple attack sources (zombies), the power of the DDoS attack is amplified, increasing solution complexity needed for defence (Peng et al., 2007). A typical DDoS attack involves flooding a victim with a large volume of stateless protocol messages, such as *User Datagram Protocol* (UDP), IP or *Internet Control Message Protocol* (ICMP) packets. These packets are often spoofed[6], masking the true identity of the zombie. The attack packet (Figure 5.3) contains meaningless, often repetitive payload data, where the size of the packet is often the maximum allowed by an ethernet frame. Sending large payloads reduces the number of packets required to overwhelm the victim's network and system resources in a short amount of time. This type of attack prevents legitimate network communications from reaching the victim, reducing the availability of data, one of the critical elements described by the CIA triad. The attack operates on the premise that it is easier for hosts to generate requests than it is for a server to check the validity of these requests. This makes it difficult to protect the server from malicious requests, designed to waste resources (Peng et al., 2007).

The impact arising from DDoS attacks range from mild user annoyances to substantial financial losses for companies who rely on network access to operate (Peng et al., 2007). These companies include online retailers (e-commerce), banking institutions and postal services. It is therefore

[6]IP spoofing is the creation of an IP packet using a forged source address in an effort to conceal the true identity of the sending party.

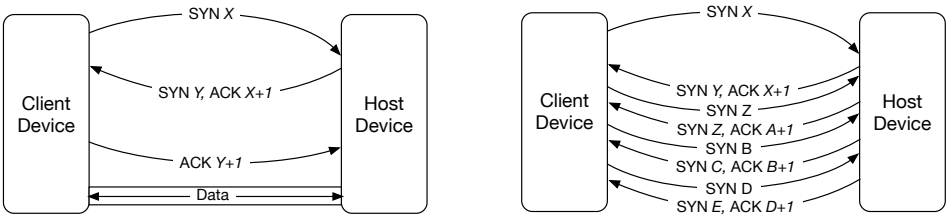
crucial that these attacks be deterred, or otherwise mitigated to avoid associated damages. According to Belson (2015), the second quarter of 2015 saw a 132.43 percent increase in the number of DDoS attacks compared to the same time in 2014, with the average lifespan of an attack lasting 18.99 percent longer. More than half of the DDoS attacks noted were multi-vector attacks, where two or more assaults were launched at the victim simultaneously (Belson, 2015). These multi-vector attacks were used to generate large-scale assaults, reaching peak traffic rates in excess of 100 gigabits per second. With the wide variety of systems now connected to the Internet, such DDoS attacks may be extremely harmful, even life-threatening. It is therefore crucial to discourage or mitigate the effects these attacks have on computer networks (Peng et al., 2007).

5.2.1 Classes of DDoS Attack

DDoS attacks are realised in a number of forms, or types. These attack types are commonly divided into four categories: *TCP Connection Attacks*, *Volumetric Attacks*, *Fragmentation Attacks* and *Application Attacks*. This section describes each of these attacks, their intent, and their possible effects if the network is not adequately protected.

5.2.1.1 TCP Connection Attacks

A TCP connection-based attack focuses on occupying a significant number of connections at the victim, consuming all available connection resources on infrastructure equipment, such as traffic load-balancers (Bourke, 2001), firewalls (Stewart, 2013) and application servers (Leander, 2000). Although many of these devices are designed to handle scale, they are often no match for a sizeable DDoS attack. A common attack in this category is the TCP SYN flood attack, which exploits a weakness in the TCP protocol's three-way handshake (Figure 5.4). Normally, a client device will send a *synchronise* (SYN) packet to a host device, initiating a TCP flow. The host will respond with an *acknowledge* (ACK) for the original SYN packet and a new SYN message. The host then waits until the client sends an ACK message, after which the data channel is open and the hosts exchange data packets. In the case of a SYN flood attack, the client (a zombie) sends a



(a) Normal TCP three-way handshake

(b) SYN Flood

Figure 5.4: A normal TCP connection setup (three-way handshake) is depicted in (a), where the three-way handshake is successfully completed, resulting in an open data channel between the communicating hosts. Figure (b) shows a SYN flood, where SYN+ACK responses from the host are ignored by the client device. The victim (host device) leaves the connection open until an ACK is received from the client, an unlikely event as the SYN packet is often spoofed.

number of SYN packets to a host (the victim), often using a spoofed source address. The host will respond to these requests with a SYN+ACK response and will wait for the client to respond. The zombie never responds to the message from the host, but rather sends additional SYN messages, resulting in further connection resources being consumed at the host. The host will often continue to accept new connections, binding resources until no new connections are possible, ultimately resulting in a loss of service.

5.2.1.2 Volumetric Attacks

Volumetric attacks focus on consuming all available bandwidth, causing congestion either within the target or service, or between the target and the rest of the network. Volumetric-based attacks are often easier to detect than other DDoS attacks, due to their size and collateral damage (downtime, lost contracts, data loss). In these attacks, the zombie hosts craft packets using a number of protocols, loading the payload with random data until the maximum packet size is reached. These packets do not need to contain valid payload data, other than the target host's IP address information. The damage inflicted by this attack is caused by the transfer of the large packets, which often overwhelm the limited network resources of the target.

Examples of this attack include UDP Floods, where unsolicited packets

are sent to a target using the UDP, session-less transport protocol. Unlike the TCP protocol, UDP does not require a connection be established prior to data transfer. The attack involves sending a number of UDP packets to random ports on the victim, causing the host to repeatedly look up an application binding to that port. When no matching application is found, the host generates an ICMP "Destination Unreachable" message to the spoofed IP. This attack consumes host resources and often leads to inaccessibility of the victim. Another similar attack is the ICMP flood. This attack sends ICMP echo requests (ping) to a host, as fast as possible without waiting for a response. As hosts are often configured to respond to ping requests, this attack often consumes both ingress and egress bandwidth on a victim's network, resulting in a significant slowdown of legitimate communications.

5.2.1.3 Fragmentation Attacks

The largest IP packet allowed on a network is 65,535 bytes (including the IP header), however layer 2 protocols, such as ethernet, often impose a MTU size of 1500 bytes. To send a single packet that exceeds this size, it is necessary for the sending party to fragment the original message, breaking it up into multiple messages. The recipient needs to receive each of these fragmented messages to reassemble and process the complete message. In a fragmentation attack, the zombie hosts send so many fragmented packets that the host is quickly overwhelmed and is unable to reassemble the streams. Other examples include the *Ping of Death* (PoD), where the zombie constructs fragmented packets maliciously, such that the recipient assembles a packet that exceeds 65,535 bytes. This often leads to buffer overflow issues, causing a loss of service for legitimate traffic.

5.2.1.4 Application Attacks

Certain DDoS attacks operate more conspicuously than others. Application attacks often involve very few packets, where the message contained in the payload is specifically designed to overwhelm a target application. In this type of attack, only the targeted application is affected, while the other ports and network traffic remain available. Slowloris[7] is an example of an

[7] `https://www.incapsula.com/ddos/attack-glossary/slowloris.html`

application attack, enabling a single host to bring down a web server with minimal bandwidth and nominal impact (if any) to other applications. The attacking host opens a number of connections to a remote web server. A partial request is sent to each of these connections periodically, ensuring that the web server keeps them open. A complete request is never sent, thus the web server continues to wait, holding these connections open. These open connections fill the maximum concurrent connections pool, resulting in service unavailability.

5.2.2 Identifying DDoS Attacks

A number of attacks, such as volumetric-based attacks, are fairly easy to identify due to the high data or packet volumes they generate. Others that operate more discreetly, such as application attacks, are more difficult to detect. To identify these attacks, NBAD systems commonly use time series data to learn normal operating values for a range of data points (attributes or features) in specific windows (time frames). Data points that have a low probability of being generated from the normal distribution are flagged as potential anomalies (Chandola et al., 2009). An example of anomalies (outliers) in a two dimensional data set is illustrated in Figure 5.5. In this illustration, the data points A_1 and A_2, are sufficiently far from the normal data sets (N_1 and N_2) to be flagged as anomalies.

5.2.3 Generic Architecture of DDoS Defence Systems

NBAD systems are deployed in a number of network locations, based on their architecture. These include *victim-end*, *source-end* and *intermediate* network locations (Mirković et al., 2003). Victim-end NBAD systems are deployed on a network to protect and warn the operator of imminent, or current, inbound attacks. The architecture of source-end NBAD systems is similar to victim-end detection system architecture. In a source-end configuration, the network operator deploys an NBAD system to detect hosts on their network who participate in zombie-like activity, flooding the network with packets in an outbound (egress) direction. Bhuyan et al. (2013) emphasises that stopping a DDoS attack at the source is the best possible defence against a DDoS attack. This, however, is not feasible in free-for-all public network architectures, such

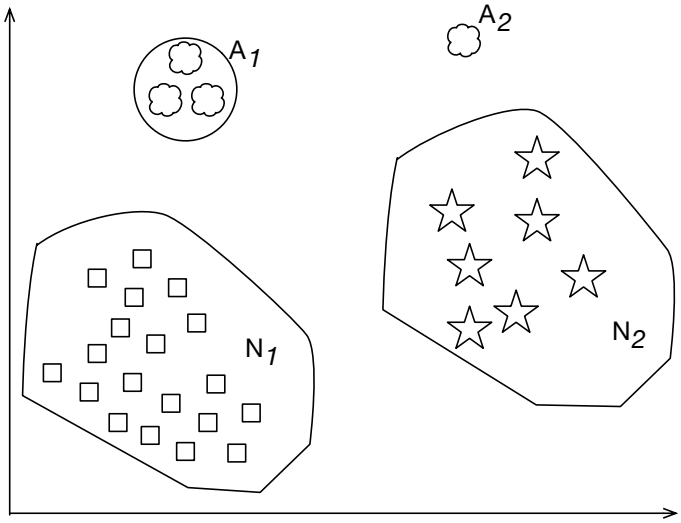


Figure 5.5: A two-dimensional representation of a data set where anomalies (A_x) and normal data points (N_x) are clearly differentiable. Data sets considered by NBAD systems are often multi-dimensional, extending beyond two dimensions.

as the Internet. Source-end detection would not only prevent the occurrence of flooding at the victim side, but also across the intermediate networks between the source and the victim. Intermediate defence systems balance the trade-offs between detection accuracy and bandwidth consumption. These systems monitor traffic profiles as packets transit the network, and compare them to previously observed patterns. They consider information shared by a number of routers on the network before deciding whether a particular traffic flow is an attack. Similar to a source-end detection system, if an attack is identified, a rate-limit is imposed to reduce the load imposed on the victim. Due to the collaboration of neighbouring routers, tracing back the source of an attack is simplified (Bhuyan et al., 2013).

ML paradigms, including supervised and unsupervised learning algorithms, are increasingly used in DDoS attack detection due to their ability to distinguish normal and attack traffic accurately and automatically (Bhuyan et al., 2013). These learning algorithms are tolerant to imprecision and uncertainty, factors for consideration when observing real-world network traffic. Using features extracted from temporal data sets describing network traffic, ML-based systems can learn to detect attacks and, in certain cases, prevent them.

The following section details a number of recent advances in statistical and ML-based approaches to DDoS detection, the evaluation methodologies employed to test these systems and the results achieved.

5.3 Statistical and Machine Learning-based DDoS Detection

ML is poised as an effective DDoS detection and mitigation tool, due to its ability to learn and make predictions on previously unseen data. Many ML-based approaches have been tested, using both supervised and unsupervised approaches. Supervised ML learning approaches are the most prevalent in recent DDoS detection systems, leveraging statistical features of flows to identify anomalous traffic on the network.

Chen (2009) investigated a statistical model, where the inbound *Synchronise Arrival Rate* (SAR) was compared to the normal distribution of traffic flows originating on the Internet toward their campus network. The method identified an attack using a two-step process. First, the current SAR was compared with the normal, expected SAR distribution. If no significant difference was noted, the ratio of *Synchronize* (SYN) and ACK packets was checked against the normal distribution. Chen (2009) stated that the method incurred low computational overhead as it simply counted the SYN and ACK packets received, without storing or tracking the entire three-way handshake. The efficacy of the method was tested and validated through experimental simulations, where results against a private data set indicated that a low false positive and false negative rate was achieved with short detection times. According to Chen (2009), the method was capable of detecting even subtle attacks as the SAR and SYN-to-ACK ratio deviate from the norm. While Chen (2009) failed to provide a summary of accuracy achieved for their method, the results published showed reduced computational complexity compared to two similar methods. These methods are the SYN arrival method, where a heuristic threshold is configured for SYN packet counts over set intervals and the SYN-FIN method, where TCP flags for the entire flow are monitored. The ability for the method to detect both high and low bandwidth DDoS

Feature	Description
N_{ICMP}	Percentage of $ICMP$ packets
N_{UDP}	Percentage of UDP packets
N_{TCP}	Percentage of TCP packets
N_{TCPSYN}	Percentage of SYN flags in TCP packets
$N_{TCPSYNACK}$	Percentage of $SYN + ACK$ flags in TCP packets
N_{TCPACK}	Percentage of ACK flags in TCP packets
$A_{AVGHEADER}$	The average size of packet headers
$A_{AVGDATASIZE}$	The average size of data packets

Table 5.1: Features extracted from flows between time intervals, used to construct the neural network input data in the method proposed by Jalili et al. (2005).

attacks make it suitable for identifying TCP Connection Attacks (Section 5.2.1.1), which is critical as TCP is the most important traffic on the Internet, targeted by 90 percent of DDoS attacks (Moore et al., 2006).

Jalili et al. (2005) trained supervised *Artificial Neural Networks* (ANN) for detecting flooding attacks, where an attacker targets a network with semi-normal packets. Semi-normal packets are correctly formed data packets exchanged within a flow, occurring at an abnormal rate or containing abnormal payload. The statistical features a flow exhibits change under DDoS attack, deviating from the normal distribution of features (Jalili et al., 2005). Jalili et al. (2005) also stated that these features are time-based and may exhibit divergent profiles at different times, such as day and night. For this reason, Jalili et al. (2005) divided the provided normal and attack traffic samples into minor time intervals, forming temporal data sets. These data sets included all packets and their associated timestamps. The statistical features extracted from these traces form the training data for a pre-processor neural network. The authors implemented this ANN for DDoS detection in conjunction with their *Unsupervised Neural Network based Intrusion Detector* (UNNID) method (Amini and Jalili, 2004), a method that implements *Adaptive Resonance Theory* (ART)(Grossberg, 2013). Combined, these methods form the the *Statistical Pre-Processor and Unsupervised Neural Network-based Intrusion Detector* (SPUNNID) method. The features used by Jalili et al. (2005) to describe flows at each interval are listed in Table 5.1. To evaluate the

Feature	Description
Time	Time the connection was first observed (start time)
SrcIP	Source IP address of the flow
DstIP	Destination IP address of the flow
SrcPort	Source port of the flow
DstPort	Destination port of the flow
Protocol	IP protocol of the flow

Table 5.2: Features extracted from the public, anomymous data sets used by the DDoS detection method proposed by Bhaya and Manaa (2014).

method, the authors recorded real network traffic, which they replayed in a simulated environment while DDoS traffic was introduced by a traffic generator. Evaluation results revealed that with a time interval not exceeding two minutes, the method was able to differentiate between normal and attack traffic in 94.9 percent of the cases.

Bhaya and Manaa (2014) asserted that data mining techniques (Witten and Frank, 2005) were capable of successfully distinguishing normal traffic from attack traffic with good accuracy. The authors proposed a hybrid approach for detecting and analysing DDoS attacks in real world traffic, tested using the *CAIDA UCSD "DDoS Attack 2007"* data set[8] for attack traffic and traces from the *CAIDA Anonymized Internet Traces 2008* data set[9] for normal traffic. The data sets contained anonymised flow traces, where the payload of each packet had been removed. The method proposed by Bhaya and Manaa (2014) used attributes inferred from IP packet headers only. These attributes are described in Table 5.2. The authors extracted 2,000,000 packets from each data set and, using Shannon's Entropy (Shannon, 2001) and min-max normalisation, transformed the data to suitable input vectors for their k-means clustering algorithm.

[8] `http://www.caida.org/data/passive/ddos-20070804_dataset.xml`

[9] `http://www.caida.org/data/passive/passive_2008_dataset.xml`

		Prediction outcome		
		p	**n**	**total**
actual value	**p′**	True Positive	False Negative	P′
	n′	False Positive	True Negative	N′
	total	P	N	

Table 5.3: A confusion matrix for DDoS detection, where *True Positive* denotes the number of packets correctly identified as malicious, *False Positive* where normal traffic is incorrectly identified as malicious (attack), *False Negative* when attack traffic is incorrectly identified as normal traffic and *True Negative* when normal traffic is correctly identified.

Bhaya and Manaa (2014) evaluated the success of their method using a confusion matrix (Stehman, 1997) (Table 5.3), where success was determined by Equations 5.1, 5.2 and 5.3.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$detectionrate = \frac{TP}{TP + FP} \tag{5.2}$$

$$falsealarm = \frac{FP}{FP + TN} \tag{5.3}$$

The results published by Bhaya and Manaa (2014) show their *centroid-based rules* method achieved better results than a simple centroid-based method (Oh and Lee, 2003). Testing showed that an accuracy of 99.77 percent was possible using the centroid-based rules method on the CAIDA data set, whereas the comparable centroid-based method achieved 97.67 percent.

Yassin et al. (2013) proposed a hybrid machine-learning approach to anomaly detection on networks using *K-means Clustering* (KMC) and *Naïve Bayes Classifier* (NBC), called "KMC+NBC". The authors argued that NBC alone produced a high percentage false alarm rate, whereas when combined with KMC, the same accuracy and detection rates are noted,

with a significantly reduced false alarm rate. The authors evaluated their method using the ISCX 2012 benchmark data set, containing over 1,512,000 packets, describing network traffic for a seven-day period by 20 distinct attributes (Table 5.4). More information about this data set is available in Shiravi et al. (2012). Yassin et al. (2013) showed that, with the KMC+NBC method, 99 percent accuracy and 98.8 percent detection rate was achievable, compared to NBC alone, which scored 88.2 percent and 85.0 percent respectively.

Juma et al. (2014) extended the work of Yassin et al. (2013), replacing the K-means and NBC algorithms with *X-means clustering* (Pelleg et al., 2000) and *Random Forest Classifier* (Ho, 1995) respectively. Juma et al. (2014) stated that the configuration was better suited to reducing false alarms, compared to similar methods. The method also used the ISCX 2012 IDS data set during evaluation, where average accuracy scores reached 99.96 percent, with detection rates of 99.99 percent, whilst maintaining false alarm rates of 0.02 percent. Yassin et al. (2013) and Juma et al. (2014) demonstrated that instituting clustering algorithms such as K-means and X-means, as first-phase classifiers, greatly enhanced accuracy and detection rates of anomalous traffic in IP flow trace data sets, while reducing the number of false positives.

The APIC method adopts a similar architecture, where an EA guides a clustering algorithm using descriptive feature sets, prior to producing unique classifiers to identify future instances of each cluster. The following section describes experiments where the tests devised by Yassin et al. (2013) and Juma et al. (2014) were implemented using the APIC method.

Feature	Description	Data Type
appName	The name of the application protocol	String
totalSourceBytes	Total bytes from the source	Integer
totalDestinationBytes	Total bytes from the destination	Integer
totalSourcePackets	Total packets from the source	Integer
totalDestinationPackets	Total packets from the destination	Integer
*sourcePayloadAsBase*64	First source packet payload (Base64 Encoded)	String
sourcePayloadAsUTF	First source packet payload (UTF Encoded)	String
*destinationPayloadAsBase*64	First destination packet payload (Base64 Encoded)	String
destinationPayloadAsUTF	First destination packet payload (UTF Encoded)	String
direction	Direction of flow	String
sourceTCPFlags	TCP flags observed in source packets	String
destinationTCPFlags	TCP flags observed in destination packets	String
source	Source IP address	String
protocolName	Transport layer protocol	String
sourcePort	Source port of the flow	Integer
destination	Destination IP address	String
destinationPort	Destination port of the flow	Integer
startDateTime	Start time of the flow	Integer
endDateTime	End time of the flow	Integer
tag	Indicator of “normal” or “attack” traffic	String

Table 5.4: Features included in the ISCX 2012 IDS labelled data set.

5.4 ISCX 2012 IDS Experiment

This section details experiments where the APIC method was evaluated as a victim-end NBAD system using the pre-labelled ISCX 2012 IDS data set. The experiments follow the design set forth by Yassin et al. (2013) and Juma et al. (2014), where a subset of the ISCX 2012 IDS data set was used to evaluate their methods. These authors extracted IP flow information for a specific target host, where normal and attack samples were evaluated using their methods. Yassin et al. (2013) and Juma et al. (2014) divided samples into training and testing data sets, where the former were used to train ML algorithms to identify future instances of "normal" and "attack" flows. These included combinations of K-means and X-means clustering, Naïve Bayes and *Random Forest* (RF) classifiers. The trained systems were evaluated using previously unseen IP flow samples from the testing data set, extracted from the ISCX 2012 IDS data set.

Using the same target host as Yassin et al. (2013) and Juma et al. (2014) provides a good basis for comparing the efficacy of the APIC method against these methods. As many of the implementation details were omitted by Yassin et al. (2013) and Juma et al. (2014), a number of variations to the original experiments may be observed in those described in this section. The following subsections detail the implementation of each experiment using the APIC method. Each subsection highlights any deviations from the experiments conducted by Yassin et al. (2013) and Juma et al. (2014). The section ends with a discussion of the results, which indicate that the APIC method is suitable for application as a victim-end NBAD system, accurately identifying anomalous activity in data sets containing complete flow information.

5.4.1 Experimental Data Sets

The experiments conducted by Yassin et al. (2013) and Juma et al. (2014) used the ISCX 2012 IDS data set (Shiravi et al., 2012) as a benchmark for testing their respective KMC+NBC and XM-RF methods. The data set contains almost 1,512,000 packets, describing complete IP flows recorded over a period of seven days. The data set contains labelled records, where each flow is described by 20 features (Table 5.4), tagged as "normal" or "attack".

	Training Data		Testing Data	
	Normal	Attack	Normal	Attack
Day 1	0	0	147	0
Day 2	22,612	0	0	0
Day 3	16,260	1,973	0	0
Day 4	0	0	19,115	37,159
Day 5	22,879	0	0	0
Day 6	13,621	181	0	0
Total	**77,526**		**56,421**	

Table 5.5: The partitioning of complete IP flow traces into training and testing data sets for the selected host. The six days encompass the entire ISCX data set. Flows for day two, three, five and six are assigned to the training data set, while those recorded on the first and fourth days are assigned to the testing data set. The days chosen and data set split were performed in accordance with Yassin et al. (2013) and Juma et al. (2014)

Yassin et al. (2013) and Juma et al. (2014) reduced the complexity of the task by testing flows destined for a single target host in the data set. The daily traffic flows for the selected host were divided into training and testing data sets. Table 5.5 outlines the composition of each of these data sets, where the training data set comprised 77,526 flow samples and the testing data set 56,421 samples. These samples contain summary information describing web, *Electronic Mail* (E-Mail), *Domain Name Services* (DNS) and *Secure Shell* (SSH) flows - services running on the target host.

5.4.2 Feature Selection

Many details of the experiments conducted by Yassin et al. (2013) and Juma et al. (2014) are absent in their publications. These details include the features that were included as inputs to their clustering algorithms, how these features were encoded, and the normalisation technique applied to each feature. After an initial manual pruning of the ISCX 2012 IDS features, the remaining, compatible features included in this experiment were automatically determined by the APIC method, using the feature selection process described in Section 3.1.

Feature	Encoding	Example
appName	Application name translated to well-known application port	$HTTP = 80$
totalSourceBytes	Unchanged	432
totalDestinationBytes	Unchanged	121
totalSourcePackets	Unchanged	20
totalDestinationPackets	Unchanged	3
direction	$L2L = 0, R2L = 1, L2R = 2, R2R = 3$	1
sourceTCPFlags	Boolean indicating inclusion of each flag: S,F,R,P,A	$1, 0, 0, 0, 1$ for SYN/ACK
destinationTCPFlags	Boolean indicating inclusion of each flag: S,F,R,P,A	$1, 0, 0, 0, 0$ for SYN
protocolName	Translate name to known protocol number	tcp=6
sourcePort	Unchanged	1025
destinationPort	Unchanged	80

Table 5.6: Feature set considered by the APIC experiments. These features were chosen from the original 20 features of the ISCX 2012 IDS data set.

The ISCX 2012 IDS data set describes complete flows using features listed in Table 5.4. Of these features, only those offering statistical significance to the problem were included for consideration by the APIC method. These are features that enable the method to classify each flow as either normal, or an attack. The final list of features, their encoding scheme and example translations provided to the APIC method are described in Table 5.6.

A number of the original 20 features were excluded from the APIC data sets. These included the *sourcePayloadAsBase64*, *sourcePayloadAsUTF*, *destinationPayloadAsBase64* and *destinationPayloadAsUTF* features. These features were excluded as APIC relies on normalised input datum, where a feature value range between zero and one is expected. It was deemed unfeasible to encode each of these variable length strings as features in the required format. The *source* and *destination* features were also excluded, as directionality information was already provided through the *direction* feature and IP address encoding presents a similar challenge to variable length string encoding. The *startDateTime* and *endDateTime* attributes were also stripped, as the data set provided to APIC was without temporal knowledge.

The remaining 11 features were those suitable for encoding in the required format. The *appName* feature was encoded by substituting the well-known port for the application. The *totalSourceBytes*, *totalDestinationBytes*, *totalSourcePackets*, *totalDestinationPackets*, *sourcePort* and *destinationPort* remained unchanged as integer values can be reduced to a value between zero and one using a normalisation function. The string value describing the *direction* feature was encoded by substituting an integer value for each of the four known directions, using values from zero to three. Both the *sourceTCPFlags* and *destinationTCPFlags* features were represented by five bits, where observation of a particular flag would set the value of the bit at that location to "1", whilst an absence of the flag was represented by "0". Finally, the *protocolName* feature was encoded by translating the string name of the protocol to its protocol number. In total, 19 features were provided to the APIC method for describing the training and testing data sets.

Each of the 19 features considered by APIC were normalised using the min-max normalisation function (Equation 4.3). This function requires a bounded value range for calculating the normalised value. The low-end and high-end values for each of the features were chosen heuristically, by noting the lowest and highest values for each feature in the sample data sets. The inputs to the min-max normalisation function for each feature are listed in Table 5.7.

The normalised training data set comprising all 19 features was passed to the APIC method, where feature selection genotypes were generated, each 19 bits long (Table 5.8). Each bit represented the inclusion of a specific feature during the clustering process. The feature selection process implemented by the APIC method is described in Section 3.1. The values used to configure the feature selection GA are listed in Table 5.8.

Feature	E_{min}	E_{max}
appName	0	200
totalSourceBytes	64	7,755
totalDestinationBytes	0	313,248
totalSourcePackets	0	100
totalDestinationPackets	0	256
direction	0	3
sourceTCPFlag-S	0	1
sourceTCPFlag-F	0	1
sourceTCPFlag-R	0	1
sourceTCPFlag-P	0	1
sourceTCPFlag-A	0	1
destinationTCPFlag-S	0	1
destinationTCPFlag-F	0	1
destinationTCPFlag-R	0	1
destinationTCPFlag-P	0	1
destinationTCPFlag-A	0	1
protocolName	0	20
sourcePort	1000	60,610
destinationPort	0	200

Table 5.7: The bounded number ranges serving as inputs to the min-max normalisation function for each feature.

Genetic Algorithm	Population Size	Maximum Generations	Mutation Rate	Crossover Rate	Genotype Length
Feature	32	100	0.1	0.5	19
Hyper-param	100	100	0.1	0.5	16

Table 5.8: A table describing the algorithmic parameters configured for each of the two GAs used by APIC: the first GA searches for an optimal feature subset (Section 3.1), the second optimal hyper-parameters to control the pattern discovery process (Section 3.2).

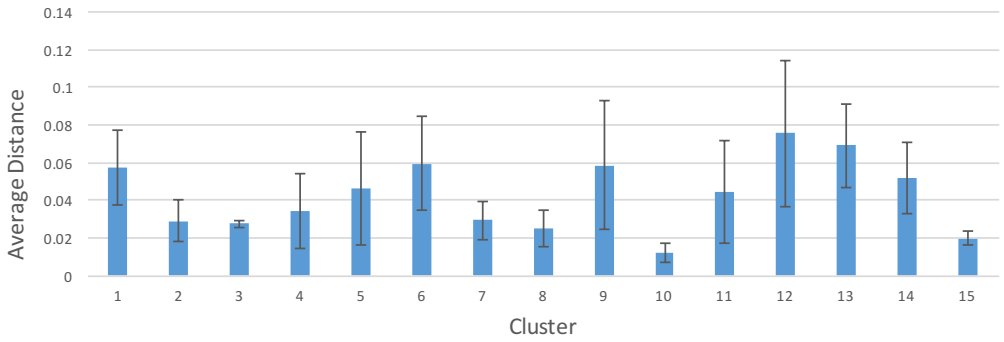


Figure 5.6: A histogram comparing the average distance between datum in each cluster, calculated using the silhouette clustering method (Equation 3.2). Low average distances provide evidence of successful, tight clustering by the GA-controlled DBSCAN algorithm. The maximum value for the Y axis is determined by $n*1$, where n is the number of features considered during the clustering process and 1 the highest value possible for the normalised feature. The chart includes the standard deviation of average distances between clusters.

5.4.3 Behavioural Profiling of IP Flow Summary Data

Each feature set genotype was tested by clustering the training data set, described by the 19 normalised features (Table 5.7). The $minPTS$ and eps parameters for each *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) execution were tuned using a hyper-parameter search GA (Section 3.2). These hyper-parameters are the tunable parameters provided to learning algorithms to guide their execution (Bergstra and Bengio, 2012). The parameters guiding this process are listed in Table 5.8. The automated generation of optimal hyper-parameters for guiding the APIC method's clustering algorithms is described in Section 3.2.

After executing the APIC method, the best feature set was found to include 18 features, where only the $appName$ feature was excluded. The tag label for each datum of the training set was excluded, preventing the APIC clustering process (Section 3.2) from forming an unfair bias using pre-labelled datum. The best scoring feature set genotype identified 15 clusters, with an average distance of 0.043 observed between datum of the same cluster. Figure 5.6 depicts the average distance between datum in each cluster. After an extensive search process - over 100 generations - the hyper-parameter optimisation concluded with optimal parameters of $eps = 0.09$ and $minPTS = 205$. An initial verification of the clustered data

Cluster	Average Distance	Attack Tag	Normal Tag
1	0.057	0%	100%
2	0.029	0%	100%
3	0.028	0%	100%
4	0.034	0%	100%
5	0.047	0%	100%
6	0.059	0%	100%
7	0.030	0%	100%
8	0.025	0%	100%
9	0.059	0%	100%
10	0.012	0%	100%
11	0.044	0%	100%
12	0.076	100%	0%
13	0.069	100%	0%
14	0.052	100%	0%
15	0.020	100%	0%
Average	0.043		

Table 5.9: A list showing the average distance between datum and the distribution of the ISCX "tag", or label, for datum in the cluster. The tag distribution indicates how many datum were tagged as "normal" and how many were "attack" in each cluster.

set was performed, comparing the distribution of tags of each datum within the cluster. The average distance and tag distributions for each of the 15 clusters are listed in Table 5.9.

Although the "tag" flow label feature was excluded from the clustering process, post-cluster verification of datum tags showed each cluster distinctly described either an attack, or normal traffic. The APIC clustering methods were therefore able to correctly distinguish normal from attack flow behaviours using an unlabelled data set. The APIC clustering method did, however, fail to associate 2.7 percent (2,093 flow samples) of the training data set with a cluster. Manual verification of these unclassified samples revealed that all were tagged as normal traffic flows in the ISCX data set.

Each of the identified flow behaviour classes (clusters) were used to form training sets for developing *Topology and Weight Evolving Artificial Neural Network* (TWEANN) classifiers to identify future instances of each class.

Parameter	Value
maxNeurons	30
learningRate	1.0
momentum	0.5
populationSize	10
maxGenerations	100
trainingIterations	1,000

Table 5.10: A list of parameters and their associated values supplied to the APIC TWEANN creation process (Section 3.3).

The following subsection details the creation and composition of each of these TWEANN classifiers.

5.4.4 IP Traffic Profile Classification

The best scoring clustered data set was used to create TWEANN classifiers to identify future instances of each IP flow class. Using the method described in Section 3.3, a custom TWEANN classifier was produced for each identified cluster. The parameters guiding the TWEANN creation process are summarised in Table 5.10.

These parameters were chosen heuristically, based on previous experience working with similar data sets. The maximum number of neurons for the TWEANN topology, including the inputs and output, was limited to 30. The learning rate and momentum values were set to 1.0 and 0.5 respectively. The GA's population size was set to 10, where a maximum number of 100 generations was allowed. This rather small number of generations is reasonable as, in previous tests, the best scoring TWEANN topology was discovered at generations earlier than 25. Finally, the backpropagation algorithm (Rumelhart et al., 1988) was limited to a maximum of 1,000 training iterations for fine-tuning the weights after the GA had chosen the best neuron structure and weight values. The backpropagation algorithm was used to fine-tune the weights found by the GA since, although the GA's ability to provide a global search technique results in near-optimal solutions, it is rather inefficient at fine-tuned local search (Yao, 1999). A hybrid training approach was therefore adopted,

CID	Topology	TP	FP	TN	FN	Normal	Attack	Score
1	18 - 2 - 1	6,999	0	70,527	0	6,999	0	99.63%
2	18 - 6 - 1	7,054	0	70,469	3	7,054	0	99.28%
3	18 - 3 - 1	9,690	0	67,836	0	9,690	0	99.83%
4	18 - 4 - 1	3,153	0	74,373	0	3,153	0	99.26%
5	18 - 0 - 1	3,876	0	73,650	0	3,876	0	99.34%
6	18 - 1 - 1	3,643	0	73,800	83	3,643	0	99.01%
7	18 - 2 - 1	4,275	0	73,251	0	4,275	0	99.61%
8	18 - 2 - 1	3,078	0	74,448	0	3,078	0	99.24%
9	18 - 9 - 1	2,447	0	75,045	34	2,447	0	99.56%
10	18 - 0 - 1	3,599	0	73,967	0	3,599	0	99.26%
11	18 - 1 - 1	3,599	0	73,820	147	3,599	0	99.45%
12	18 - 6 - 1	1,173	0	76,353	0	0	1,173	99.93%
13	18 - 3 - 1	281	39	77,206	0	0	281	98.96%
14	18 - 2 - 1	322	0	77,204	0	0	322	99.73%
15	18 - 4 - 1	378	0	77,148	0	0	378	99.66%
Result		**53,567**	**39**	**1,109,097**	**267**	**51,413**	**2,154**	**99.45%**

Table 5.11: A list of the identified clusters along with the best scoring TWEANN topology, the *True Positive* (TP), *False Positive* (FP), *True Negative* (TN) and *False Negative* (FN) values achieved during recall of the training data set. The "Normal" and "Attack" columns tally the number of normal and attack tags of the datum the TWEANN was trained to identify. Finally, the average score achieved by recalling the training set for each cluster is also listed.

using a weight-evolving GA to globally optimise the ANN weights, whose weights were fine-tuned using the backpropagation algorithm.

Information regarding each of the best scoring TWEANN classifiers for each cluster is listed in Table 5.11. The "normal" and "attack" columns indicate that clusters one through 11 were trained to detect normal flows, while clusters 12 through 15 detect attacks. The combined confusion matrix for the recall tests, post training, are depicted in Table 5.12. The accuracy, detection and false positive rates for the APIC method and comparable works are outlined in Table 5.13.

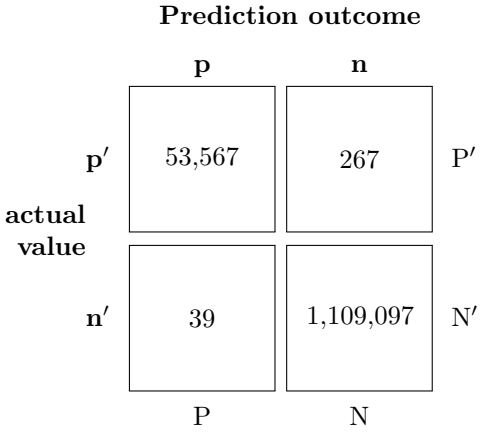


Table 5.12: A confusion matrix for the APIC training set recall tests. The True Negative count is substantially high due to the way the TWEANN training set is created, where any datum external to the cluster being trained is marked with a "0". The True Positive count includes all datum that were part of a cluster (cluster membership is mutually exclusive).

Method	Accuracy	Detection	False Alarms
NBC (Yassin et al., 2013)	82.800%	13.800%	17.600%
KMC+NBC (Yassin et al., 2013)	99.800%	95.400%	0.130%
RF (Juma et al., 2014)	99.990%	99.810%	0.010%
XM-RF (Juma et al., 2014)	99.990%	99.950%	0.000%
APIC	99.970%	99.920%	0.004%

Table 5.13: The results of training set recall tests by comparable NBAD systems. The accuracy, detection and false positive rates are calculated using Equations 5.1, 5.2 and 5.3.

After the TWEANN development process was concluded, an average recall score of 99.45 percent was recorded for the classifiers (Table 5.11). While the training set recall test demonstrated good results, it was important to evaluate each of the classifiers using previously unseen data. The following section tests the newly-trained TWEANN classifiers against the sample testing data set, comprised of 56,421 IP flow traces (Table 5.5).

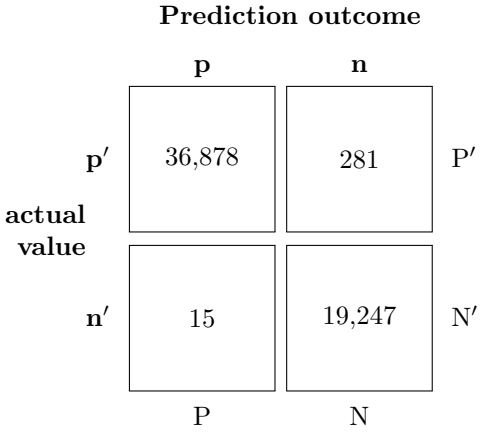


Table 5.14: A confusion matrix for the testing set parsed through the trained TWEANN classifiers. A total of 36,878 samples were correctly identified as attacks (TP), with 19,247 samples correctly identified as normal traffic (TN). A total of 281 attacks were incorrectly marked as normal traffic (FN), with 15 normal samples erroneously marked as attacks.

5.4.5 Classifier Evaluation

A test data set was created using the 19,262 normal and 37,159 attack traces extracted from the filtered data set (Table 5.5). The data set was normalised using the same limits applied to the training data set (Table 5.7). Each datum of the testing set was passed through each of the 15 classifiers, where it was marked as either a normal or attack flow, based on the best scoring classifier. A probability score in excess of 95 percent describes a positive match, with anything below that value describing a mismatch. Datum generating a positive match by TWEANNs trained to identify clusters one through 11 were marked as normal, while those positively matched by classifiers 12 through 15, an attack. As the purpose of NBAD systems is to identify anomalous traffic, any datum not explicitly marked as an attack by clusters 12 through 15, or normal flows by classifiers one through eleven, were implicitly marked as normal.

The results of parsing each of the testing datum through the 15 classifiers are described using the confusion matrix in Table 5.14. The *True Positive* (TP) tally denotes all flows marked by clusters 12 through 15 as a positive attack, corroborated by the tags read from the ISCX 2012 data set.

Method	Data set	Accuracy	Detection	False
NBC (Yassin et al., 2013)	ISCX2012	88.20%	85.00%	33.70%
KMC+NBC (Yassin et al., 2013)	ISCX2012	99.00%	98.80%	2.2%
XM-1R (Juma et al., 2014)	ISCX2012	93.68%	95.20%	9.26%
RF (Juma et al., 2014)	ISCX2012	99.91%	99.94%	0.11%
XM-RF (Juma et al., 2014)	ISCX2012	99.96%	99.99%	0.02%
APIC	ISCX2012	99.48%	99.99%	0.08%

Table 5.15: The results of comparable NBAD systems, reporting their accuracy, detection and false positive rates tested using the pre-labelled ISCX data set.

The *True Negative* (TN) count tallies all occurrences where an IP flow, marked "normal" in the ISCX 2012 data set, was not positively identified as an attack by the APIC method. The *False Positive* (FP) reading counts all instances where ISCX 2012 "normal" tagged traffic was incorrectly marked as an attack by the APIC method. Finally, the *False Negative* (FN) count shows all ISCX 2012 "attack" flows that were marked "normal" by the APIC method. Using Equations 5.1, 5.2 and 5.3, the accuracy, detection rate and false alarm rate for the testing set was calculated. The results of these calculations are shown alongside comparable NBAD systems in Table 5.15.

5.4.6 Discussion

The ISCX 2012 data set is comprised of complete flow traces, annotated as "attack" or "normal" traffic by a commercial classification system. The data set serves as a reference for many works, whose results are compared against those achieved by the experiments of this chapter, in Table 5.15. In these results, the APIC method is shown to exceed those of many works, and remain comparable with the rest. The experiments described in this section highlight the success APIC achieved as a victim-end NBAD system. While accuracy and detection rates are important aspects of these systems, many score well yet suffer a high number of false alarms (Tsai and Lin, 2010; Muda et al., 2011). For APIC to succeed as a victim-end NBAD system, it should exhibit high accuracy and detection rates, while maintaining a low false alarm rate. The experiments in this section compare the results APIC achieved using IP flow samples extracted from the ISCX 2012 IDS pre-labelled data set with comparable research using the same data set. The

results achieved by each of these methods are listed in Table 5.15. The APIC experiments were modelled around those conducted by Yassin et al. (2013) and Juma et al. (2014), using the ISCX 2012 IDS data set (Section 5.4). The discussion that follows subsequently, focuses on the experimental results achieved by these methods, using the ISCX 2012 IDS data set.

5.4.6.1 Data Sets

The ISCX 2012 IDS data set does not offer ready-made training and testing data sets. To reduce the complexity of the classification task, Yassin et al. (2013) and Juma et al. (2014) proposed extracting flows destined for a particular target host to build their data sets. The extracted flow information was then partitioned into training and testing data sets (Table 5.5). The experiment described in this section selected 75,372 normal and 2,154 attack flow vectors from day one, two, three, five and six of the filtered data set. The testing data set was comprised of 19,262 normal traces and 37,159 attack traces from day four of the same data set. The purpose of reducing the size of the original ISCX data set was to lower the complexity of the task and, subsequently, the computational resources required for execution.

5.4.6.2 Feature Set

Little guidance was offered by Yassin et al. (2013) and Juma et al. (2014) regarding which ISCX 2012 IDS features were included in their methods. Furthermore, no information regarding the normalisation process or value range for each dimension of these features were discussed. The features considered and the encoding methods implemented to convert them to values suitable for clustering are therefore expected to differ significantly between APIC and these comparable methods.

The original 20 features from the ISCX 2012 IDS data set were pruned, where the sourcePayloadAsBase64, sourcePayloadAsUTF, destinationPayloadAsBase64, destinationPayloadAsUTF, source and destination features were removed. This was due to complexities realised when encoding variable length string features as inputs for unsupervised learning algorithms, such as K-means and DBSCAN. The source and destination flag features were converted from variable length string values

to five separate features, each one bit in length. A "1" value at each location signals the inclusion of a particular flag with a "0", the exclusion thereof. A summary list of the features considered by APIC and the ranges supplied to the min-max normalisation function are shown in Table 5.7. An interesting observation is that, of the 19 features presented to APIC, only the *appName* feature was excluded from the best scoring feature subset. It is surmised that the method found this attribute redundant, due the presence of the *dstPort* feature, which contained values equal to *appName*.

5.4.6.3 Training Set Clustering

Yassin et al. (2013) and Juma et al. (2014) implemented a hybrid learning approach, not dissimilar to APIC, to identify anomalous traffic in the ISCX 2012 IDS data set. The pre-classification module, described by Yassin et al. (2013), used K-means clustering to partition the data set into three distinct clusters. Juma et al. (2014) substituted an enhanced version of K-means - X-means - as the pre-classification clustering algorithm. The X-means algorithm initially started with three clusters, however the algorithm has the ability to split clusters until more favourable results are achieved. The APIC method uses the DBSCAN algorithm, where a GA evolves the hyper-parameters governing the search. Using DBSCAN, the number of clusters, K, is not required prior to execution of the algorithm.

The K-means clustering algorithm is known to form convex cluster shapes and does not perform well where non-convex cluster shapes are required. This clustering method also requires that all datum be joined to a cluster. The clustered datum gather around a central point (centroid), which is prone to being drawn toward outlier datum. This shifting of the centroid causes the average distance between datum in the cluster to increase, resulting in loose, less accurate clustering. The only input required for this algorithm is a value for K - the number of clusters to form. Both Yassin et al. (2013) and Juma et al. (2014) used a value of three ($K = 3$) as input to the clustering algorithm.

In contrast, the DBSCAN algorithm is capable of finding arbitrary-shaped, non-convex clusters. The algorithm does not align cluster datum around a centroid and is capable of discarding outliers. The APIC method uses a GA to evolve hyper-parameters for the DBSCAN algorithm, allowing it to automatically discern the number of clusters in the data set. The algorithm discovered 15 clusters, where an average distance of 0.043 between datum (Table 5.9) was observed. Unfortunately, the average distance within clusters produced by Yassin et al. (2013) and Juma et al. (2014) were not reported. While the APIC DBSCAN implementation failed to associate 2.7 percent (2,093 flows) with a cluster, each of these flows were tagged as "normal" flows and thus did not affect the accuracy or detection rates attained by APIC.

The resultant, best scoring clustered data set from APIC was evaluated by restoring the original tags from the ISCX 2012 IDS data set on each cluster member. It was observed that all members of each of the 15 clusters shared the same tag, either "normal" or "attack". This provides evidence that the APIC clustering process correctly profiled and grouped (clustered) the datum according to their behaviour. Not only did APIC correctly group normal and attack datum, but unlike Yassin et al. (2013) and Juma et al. (2014), it also provided enhanced visibility into the different types of attacks destined for the target host. According to the clustered data set, a total of four distinct attack profiles (clusters 12 through 15) were identified in the training set. Each of the clusters discovered by APIC were used to train TWEANN classifiers to identify future instances of each attack.

5.4.6.4 TWEANN Classifier Production

TWEANN classifiers were trained to identify each of the 15 clusters, reconfiguring their topology to achieve best recall scores over each successive epoch. The results of this training are depicted by the confusion matrix in Table 5.12. As a result, 99.97 percent accuracy, 99.85 percent detection and 0.01 percent false positive rates were achieved using the training set. By comparison, the NBC method and enhanced KMC+NBC methods, tested by Yassin et al. (2013) scored accuracy 82.8 percent, detection 13.8 percent, false alarm 17.6 percent and accuracy 99.8 percent, detection 95.4 percent, false alarm 0.13 percent rates respectively. Juma

et al. (2014) reported that the training set scores of the XM-RF method were accuracy rate 99.99 percent, detection rate 99.95 percent and false positive rate 0.00 percent. Overall, the APIC method beat all but the results reported by the XM-RF method in the training set recall tests. The scores achieved by each of these methods are summarised in Table 5.13.

5.4.6.5 TWEANN Classifier Testing

Testing the classifiers produced by APIC using the testing data set, yielded results represented in Table 5.15. Here, 99.48 percent accuracy, 99.99 percent detection and 0.08 percent false positive rates were noted. By comparison, the KMC+NBC testing data set recall by Yassin et al. (2013) resulted in an accuracy score of 99.00 percent a detection rate of 98.80 percent and a false alarm rate of 2.2 percent. When parsing the testing data set, the XM-RF method, described by Juma et al. (2014), resulted in an accuracy score of 99.96 percent a detection rate of 99.99 percent, with a false alarm rate of 0.02 percent. These results are summarised in Table 5.15.

Once more, APIC scored higher than comparable systems, except the XM-RF method described by Juma et al. (2014). The XM-RF method scored 0.48 percent higher accuracy than APIC, with equal detection rates and 0.06 percent lower false positives. These variances are not substantial, especially when accounting for differences in experiment implementation for each system. Furthermore, it is important to note that the XM-RF method was designed specifically as an NBAD method, while APIC is a generic, versatile method for pattern identification and classification for a broad range of tasks.

In summary, the APIC method was tested against comparable methods, using the same static, pre-recorded data set. The results showed that the APIC method performed favourably, achieving outcomes similar to those described by Yassin et al. (2013) and Juma et al. (2014). While this experiment yielded truths regarding the performance of APIC classifiers compared to similar works on static data sets, in reality, NBAD systems are required to detect anomalous traffic in near real-time on live computer networks. The following section describes an experiment that demonstrates the efficacy of the method for identifying anomalous traffic on a live network.

5.5 Live Network Evaluation

This section describes experiments where the APIC method was evaluated as both a victim-end and source-end NBAD system on an enterprise network. The experiments conducted in Section 5.4 compared APIC against methods using pre-labelled, public data sets. The APIC method yielded comparable task performance to related methods in this area, however this kind of analysis is not a true reflection of how well systems perform as real-world NBAD systems. While network administrators may require retrospective analysis of recorded data sets containing complete, summarised flow records, more often their requirements are to identify and mitigate the impact of attacks as they occur on live networks.

A number of methods exist for sampling and extracting live flow records from a network router. Some of these include *Simple Network Management Protocol* (SNMP), network taps (traffic mirroring) and Cisco's Netflow protocol. The former two methods incur significant computational overhead on both the router and the collector and thus do not scale well on today's high-speed computer networks. The Netflow protocol (Hofstede et al., 2014) offers a viable alternative to these methods, exporting summarised flow statistics from the router to a Netflow data aggregation point. Due to its low computational overhead, the protocol has gained wide-spread adoption and is included by a number of high-end, carrier-grade router manufacturers, such as Cisco[10] and Juniper[11] as well as less expensive, small- to medium-size enterprise router manufacturers, like Mikrotik[12].

The objective of this case study is to demonstrate that, using only sampled Netflow records, the APIC method can profile normal host communications, identifying anomalous traffic almost immediately as flow information is reported by the edge router. A successful demonstration provides further evidence that the APIC method is capable of addressing many complex classification problems over a broad range of domains.

[10] `http://www.cisco.com`
[11] `http://www.juniper.com/`
[12] `http://www.mikrotik.com/`

5.5.1 Task Description

Using an ML method to detect network anomalies using real-time data is not new. A number of methods have been proposed specifically for this task, some of which are described in Section 5.3. The purpose of the experiments conducted in this section was to ascertain whether the efficacy of the APIC general method extends to the identification and classification of abnormal host communications on a live network. The results obtained through these experiments were compared to those achieved by methods designed specifically for this task. In this case study, the APIC method was deployed as both a victim-end and source-end system, with the intention of identifying attacks from the Internet toward internal hosts and also attacks derived from internal hosts toward external (Internet) hosts.

An enterprise network with approximately 80 active user devices was selected to produce a live, private data set of summarised network host communications. According to Bhuyan et al. (2013), testing against a private data set is the best approach for evaluating an NBAD system. The private data set was used by APIC to profile "normal" host communications on the network. According to Jalili et al. (2005), a single network may exhibit various profiles over different periods of each day. For this reason, it is important to factor the time, or time-slot, of the day when profiling host communications. The APIC method used the heuristic value of five minutes, dividing each day's flow records into 720 distinct, five minute time-slots.

The Netflow protocol was used to construct a private data set of live flow data. The protocol exported all active flows traversing the router at a particular time to one or more Netflow data collector services, or *targets*. A custom Netflow collector[13] was used to receive and store flow information. As the Netflow protocol summarised flows at the IP layer, it did not include any layer 2 (Ethernet) headers, nor payload (application layer) information. It therefore follows that fragmentation attacks (Section 5.2.1.3) and application attacks (Section 5.2.1.4) were excluded from this case study.

Using the recorded data, the APIC method was able to accurately detect simulated instances of both TCP connection (Section 5.2.1.1) and volumetric

[13]Developed by Ryan Goss, available at: `https://goo.gl/Ck0ZGk`

Parameter	Description	Value
Interfaces	Ingress network interfaces with flow tracking enabled	ether-wan; ether-lan
Cache Entries	Maximum number of flows tracked simultaneously by the router	4,000
Active T/O	Maximum lifetime of a flow	30 minutes
Inactive T/O	Maximum idle time for an active flow	15 seconds
Target	IP Address of the Netflow collector	192.168.10.5
Port	UDP Port on which the Netflow collector is bound	9992
Version	Netflow protocol version to export	9
v9 Template Refresh	Number of packets before the v9 template is resent to the Netflow collector	20
v9 Template Timeout	Maximum wait time before sending template	1800

Table 5.16: Netflow parameters configured on the edge router of the chosen enterprise network. The ether-wan interface records ingress flow information from the Internet, while ether-lan tracks ingress flows from the LAN toward Internet-based hosts. Pairing these flow records provides bi-directional flow information, from which features for the APIC method are derived.

(Section 5.2.1.2) attacks on the network. The following sections provide insight into the design and execution of the experiments used to determine the efficacy of the APIC method as both a victim-end and source-end, NBAD system for live IP networks.

5.5.2 Data Sets

The edge router of an enterprise network was configured to report IP flow statistics, using version 9 of the Netflow protocol, to a server running the Netflow collector software. The values assigned to each configuration parameter on the router are outlined in Table 5.16.

The flow samples reported by the router included all attributes listed in Table 5.17. The Netflow collector recorded flow samples for a period of seven days during a normal work week, from Monday morning 00h00 to the same time the following week. The flow records provided by the router were unidirectional, where separate entries for requests (source host to destination host) and responses (destination host to source host) are recorded.

Feature	Description
$Timestamp$	An integer time value (epoch) reported by the router
$Protocol$	An integer value representing the transport layer protocol of the flow
ToS	The *Type of Service* (ToS) bits set in the flow
SYN	An indication if a SYN flag was observed for the flow
ACK	An indication if an ACK flag was set for the flow
$Exporter$	The router's IP address, encoded as a 32-bit integer
SRC	The source IP address of the flow, encoded as a 32-bit integer
DST	The destination IP address of the flow, encoded as a 32-bit integer
$sPort$	The source port of the flow
$dPort$	The destination port of the flow
$Packets$	Total packets switched during the current reporting period
$Octets$	Total bytes switched during the current reporting period
$IfIn$	Inbound interface of the flow
$IfOut$	Outbound interface of the flow

Table 5.17: Features included in each flow record reported by the router. Netflow version 9 is template based, meaning the features used to described each flow may be customised on the router. The features listed in this table describe the default flow template defined on the test router.

Related flow samples were combined and aggregated into a single entry per host for each time-slot. Both internal and external hosts were aggregated, as the objective of this case study was to evaluate the APIC method as both a victim-end and source-end NBAD system. The aggregate features, calculated for each active host per time-slot, are outlined in Table 5.18.

Feature	Description
timeslot	The five-minute time-slot associated with this record Example: 2016-01-01 00:05:00
host	The IP address of the summarised host Example: 192.168.10.245
$outTotalFlows$	The total number of flows from this host toward others
$inTotalFlows$	The total number of flows toward this host from others
$outICMPFlows$	The total number of ICMP flows from this host toward others
$inICMPFlows$	The total number of ICMP flows toward this host from others
$outTCPFlows$	The total number of TCP flows from this host toward others
$inTCPFlows$	The total number of TCP flows toward this host from others
$outUDPFlows$	The total number of UDP flows from this host toward others
$inUDPFlows$	The total number of UDP flows toward this host from others
$outSYN$	The total number of flows with TCP SYN flag set from this host toward others
$inSYN$	The total number of flows with TCP SYN flag set from others toward this host
$outACK$	The total number of flows with TCP ACK flag set from this host toward others
$inACK$	The total number of flows with TCP ACK flag set from others toward this host
$outTotalPackets$	The total number of packets sent by this host to others
$inTotalPackets$	The total number of packets sent by others toward this host
$outICMPPackets$	The total number of ICMP packets sent by this host to others
$inICMPPackets$	The total number of ICMP packets sent by others toward this host
$outTCPPackets$	The total number of TCP packets sent by this host to others
$inTCPPackets$	The total number of TCP packets sent by others toward this host
$outUDPPackets$	The total number of UDP packets sent by this host to others
$inUDPPackets$	The total number of UDP packets sent by others toward this host
$outBytes$	The total number of bytes (octets) sent by this host to others
$inBytes$	The total number of bytes (octets) received by this host from others
$dstHosts$	The number of unique hosts this host sent packets toward
$srcHosts$	The number of unique hosts sending packets toward this host

Table 5.18: Features used to describe the communications of each host on the network. A unique record was created for each host at every time-slot it was observed.

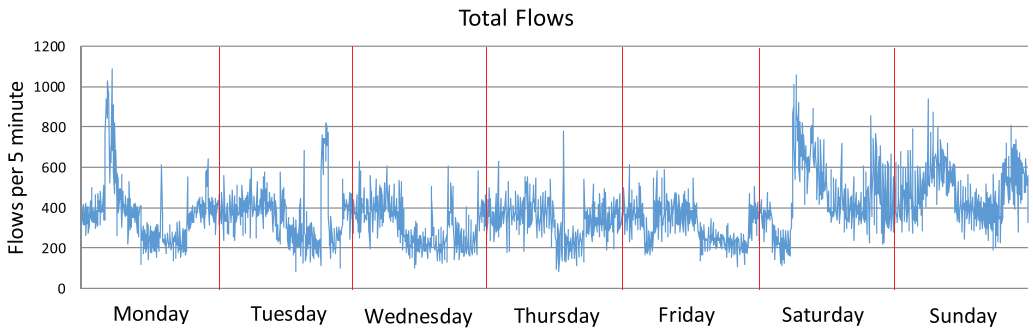


Figure 5.7: Total number of flows recorded per five-minute time-slot during the seven-day observation period.

Feature	Description
rOutTotalPkts	The ratio of total outbound packets to total inbound packets
rInTotalPkts	The ratio of total inbound packets to total outbound packets
rOutICMPPkts	The ratio of ICMP outbound packets to ICMP inbound packets
rInICMPPkts	The ratio of ICMP inbound packets to ICMP outbound packets
icmpOutPerc	Outbound ICMP packets as a percentage of total outbound packets
icmpInPerc	Inbound ICMP packets as a percentage of total inbound packets
rOutTCPPkts	The ratio of TCP outbound packets to TCP inbound packets
rInTCPPkts	The ratio of TCP inbound packets to TCP outbound packets
tcpOutPerc	Outbound TCP packets as a percentage of total outbound packets
tcpInPerc	Inbound TCP packets as a percentage of total inbound packets
rOutUDPPkts	The ratio of UDP outbound packets to UDP inbound packets
rInUDPPkts	The ratio of UDP inbound packets to UDP outbound packets
udpOutPerc	Outbound UDP packets as a percentage of total outbound packets
udpInPerc	Inbound UDP packets as a percentage of total inbound packets
rOutBytes	The ratio of total outbound bytes to inbound bytes
rInBytes	The ratio of total inbound bytes to outbound bytes
rOutSYN	The ratio of outbound flows with SYN TCP flag set to inbound flows with ACK TCP flag set
rInSYN	The ratio of inbound flows toward this host with SYN TCP flag set to outbound flows with ACK TCP flag set
rDstHosts	The ratio of unique hosts outbound flows were directed to against the number of flows inbound from unique hosts
rSrcHosts	The ratio of unique hosts inbound flows directed to the host against the number of flows outbound from the host

Table 5.19: Statistical features derived from the aggregated flow records, used to describe and profile host communications on the network. The features listed were included in the full feature set provided to the APIC method.

Day	Training Data	Testing Data
Monday	116,938	29,234
Tuesday	80,068	20,017
Wednesday	80,989	20,247
Thursday	71,085	17,771
Friday	75,930	18,983
Saturday	81,282	20,321
Sunday	110,227	27,557
Total	**616,519**	**154,130**

Table 5.20: Netflow experiment training and test data sets. Each count represents normal, individual host communication profiles summarised per timeslot.

A plot of the total number of aggregate flows recorded per time-slot, for the seven-day observation period, is shown in Figure 5.7. These aggregated records were used to produce statistical features, describing the communication profile of each host to the APIC method. These features are catalogued in Table 5.19.

In total, 96 internal and 62,889 external hosts were observed communicating through the router over the seven-day period. These communications generated 4,578,500 flow entries, all of which were aggregated, transformed and summarised per time-slot using the features described in Table 5.19. A plot showing the average value for each of these statistical features per time-slot is provided in Figure 5.8. A total of 770,649 host communication profiles were generated using the recorded flow data set. These temporal host profiles were divided into both training and testing data sets. For each time-slot, 80 percent of the recorded flows were randomly selected and moved to the training set, with the remaining 20 percent assigned to the testing set. The distribution of training and testing datum, per day, is listed in Table 5.20.

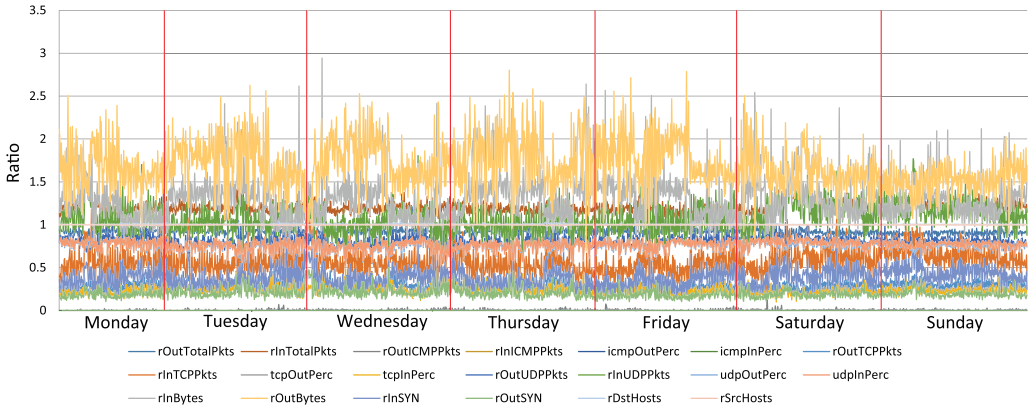


Figure 5.8: A plot describing the distribution of the average value for each statistical feature (Table 5.19) for each timeslot recorded over the seven-day period.

5.5.3 Feature Set Selection and Normalisation

The features chosen to represent each host communication profile per time-slot (Table 5.19) were based on known attack profiles for both TCP connection and Volumetric attacks (Section 5.2.1.2). Expressing data exchanges between source and destination hosts as a ratio allowed a distribution profile to be realised - one that is retained through varying network load. A significant ratio skew in either direction (higher or lower) was indicative of anomalous traffic, potentially an attack.

For example, the average $rOutSYN$ ratio recorded for a single source host communicating with a single destination host using the TCP protocol was maintained, even as the source initiated similar, valid TCP connections to other remote hosts concurrently. A higher than normal $rOutSYN$ ratio indicated that the source host was initiating a significant volume of unanswered TCP connections, potentially acting as a DDoS zombie. A high $rInSYN$ ratio coupled with a high $rSrcHosts$ ratio may indicate that the host is the target of a TCP connection attack. These features are adapted from those proposed by Chen (2009) to discern TCP connection attacks using their statistical method.

A higher than normal value for $rSrcHosts$ coupled with a higher than normal $rInBytes$ ratio may indicate that the host is the target of a volumetric attack. It follows that a host with a low $rDstHosts$ ratio, accompanied by a high $rOutBytes$ ratio may indicate that the host is participating in a volumetric attack. Flooding attacks may be identified toward a host when a

high $rSrcHosts$ ratio is accompanied by a high $udpInPerc$ or $icmpInPerc$ ratio. Similarly, a low $rDstHosts$ ratio coupled with a high $udpOutPerc$ or $icmpOutPerc$ ratio may indicate that the host is participating as a zombie in a flooding attack.

The variety of combinations used to describe normal traffic and, subsequently, to identify anomalous traffic is vast. An assumption of this experiment was that no attacks were active during the seven-day recording period. As attack traffic was not present on the network, it could not be included in the initial training set. The APIC method's clustering algorithm was therefore used to distinguish and group a variety of normal traffic profiles. Any future flow samples that fall outside of these groups would be deemed anomalous. The full complement of features was considered during this experiment, ensuring that a full profile for each host communication was constructed. All features listed in Table 5.19 were subsequently used to describe each flow for the training and testing processes.

The mean average for each statistical feature (Figure 5.8) remained constant throughout the seven-day recording process, with the highest recorded *standard score* reaching a value of 1.7. A standard score, more commonly referred to as a *z-score* (Rubin, 2012), is the number of standard deviations a value is from the mean average for that distribution. It is argued that any z-score value in excess of three is indicative of an anomalous traffic flow. It is further argued that beyond a certain point, specifically a z-score of five, the value may be transformed to a value corresponding to a z-score of five without impacting the anomalous traffic assessment. These arguments form the basis for the upper bounds of the min-max normalization method (Equation 4.3), used to normalise each of the feature dimensions in this case study. The minimum bound of each feature was set to the minimum value observed during the recording period, with the maximum bound set to the value corresponding to a z-score of five for each dimension. Each of the features, along with their derived bounds for the min-max normalisation method, are listed in Table 5.21.

Feature	E_{min}	E_{max}
$rOutTotalPkts$	0	364
$rInTotalPkts$	0	341
$rOutICMPPkts$	0	216
$rInICMPPkts$	0	4
$icmpOutPerc$	0	4
$icmpInPerc$	0	4
$rOutTCPPkts$	0	380
$rInTCPPkts$	0	313
$tcpOutPerc$	0	4
$tcpInPerc$	0	4
$rOutUDPPkts$	0	392
$rInUDPPkts$	0	352
$udpOutPerc$	0	4
$udpInPerc$	0	4
$rOutBytes$	0	399
$rInBytes$	0	383
$rOutSYN$	0	99
$rInSYN$	0	292
$rDstHosts$	0	40
$rSrcHosts$	0	16

Table 5.21: Each feature used to describe a host communication record was normalised using the min-max normalisation function. The parameters guiding the E_{min} and E_{max} bounds were determined by observing the lowest and highest values for each of the features over the seven-day period. These bounds were recorded and are listed in this table.

5.5.4 Normal Communication Profile Determination

All 616,519 normal records from the training set were clustered using the DBSCAN method, using the APIC pattern discovery process (Section 3.2). The *minPTS* and *eps* parameters used to guide the clustering process were controlled by a hyper-parameter search GA, which tuned the values until the most optimal clustered data set was found. For the purposes of these experiments, a more optimal clustered data set is one where the average silhouette evaluation (Equation 3.2) approaches a value of "1". The GA parameters used to control this search process are listed in Table 5.22.

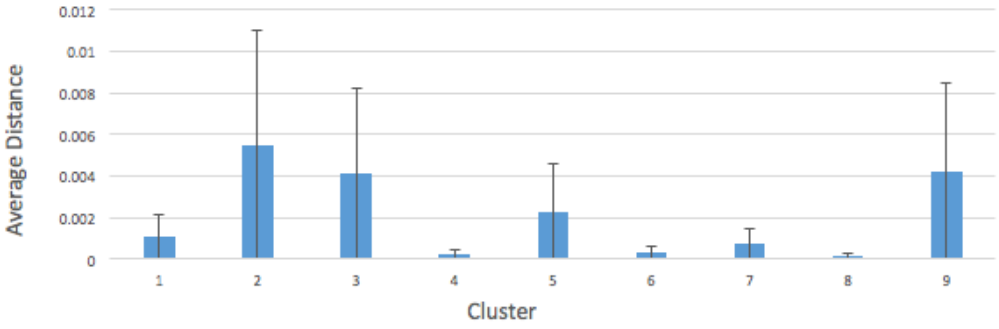


Figure 5.9: A histogram comparing the average distances between datum of each of the nine clusters discovered by the APIC clustering method. A low average distance between datum in each cluster provides evidence of successful, tight clustering by the GA-controlled DBSCAN algorithm. The chart includes the standard deviation of average distances between clusters.

Genetic Algorithm	Population Size	Maximum Generations	Mutation Rate	Crossover Rate
Hyper-parameter Search	100	100	0.2	0.4

Table 5.22: Hyper-parameter search GA parameters. This GA was used to find the best combination of $minPTS$ and eps parameters, resulting in the most optimal clustering process. The search process for determining the best hyper-parameters for guiding the APIC method's clustering algorithm is detailed in Section 3.2.

As the feature set was fixed, only a single GA was required for this experiment. The GA found the best clustered data set at generation 14, where silhouette evaluation revealed a best score of 0.89. The resultant data set was comprised of nine clusters, where the distance between cluster members ranged from 0.0001 to 0.006, with an average distance of 0.002. The average distance between member datum of each cluster is represented as a histogram in Figure 5.9. The histogram indicates that small average distances are exhibited by datum of the same cluster, an indication that the clustering process concluded successfully. The best clustered data set was achieved using hyper-parameters $minPTS = 1310$ and $eps = 0.04$, discovered by the GA. The DBSCAN clustering process did, however, fail to associate 0.12 percent (740) host profile records with a cluster.

Each of the nine clusters, describing nine distinct normal communication profiles, was used to train TWEANN classifiers to identify future instances

Parameter	Value
maxNeurons	40
learningRate	1.0
momentum	0.5
populationSize	10
maxGenerations	100
trainingIterations	1,000

Table 5.23: The parameter values supplied to the APIC TWEANN creation process (Section 3.3) for the Netflow live data experiment. The *learningRate* and *momentum* parameters are used in the backpropagation algorithm.

of each normal traffic profile. The following subsection details the creation, training and testing of these classifiers using the APIC classifier production process (Section 3.3).

5.5.5 Developing Normal Traffic Profile Classifiers

The APIC method is capable of producing optimised models for classifying various patterns in mixed, noisy data sets. In this case study, the APIC classifier production process (Section 3.3) was used to train TWEANN classifiers for identifying future instances of each cluster, each describing a normal host communication profile. Contrary to anomalous or abnormal communications, normal host communications are flows traversing the network that exhibit expected patterns. The APIC method uses GAs to optimise both the topology of the model and the weights for each neuron connection, prior to using the backpropagation algorithm (Rumelhart et al., 1988) for fine-tuning the model.

The parameters guiding the TWEANN creation process in this experiment were chosen heuristically, based on previous experience. A maximum of 40 neurons were allowed per TWEANN, including the inputs and output. This value is double the number of input features. A learning rate of 1.0 was provided, along with a momentum value configured at 0.5. A GA population size of ten was chosen, with a maximum of 100 generations permitted. A low population size is suitable in this semi-supervised training process, as the weight values determined by the GA are further fine-tuned by the backpropagation algorithm. Only the

Cluster	Topology	TP	FP	TN	FN	Score
1	20 - 2 - 1	17,508	5	615,774	1	98.93%
2	20 - 4 - 1	124,709	2	615,777	1	99.62%
3	20 - 0 - 1	123,021	3	615,776	0	97.77%
4	20 - 3 - 1	115,241	1	615,778	0	99.84%
5	20 - 2 - 1	76,229	1	615,778	0	99.91%
6	20 - 4 - 1	69,127	6	615,771	0	98.71%
7	20 - 2 - 1	54,273	2	615,777	0	99.94%
8	20 - 1 - 1	19,226	3	615,776	6	98.49%
9	20 - 1 - 1	16,437	4	615,775	0	99.86%
Result		**615,771**	**27**	**5,541,982**	**8**	**99.34%**

Table 5.24: A list of the identified clusters along with the best scoring TWEANN topology, the TP, FP, TN and FN values achieved during recall of the training data set. The score field indicates the average probability achieved by each classifier during the recall test.

topology and initial weight vectors are determined by each genotype. The backpropagation algorithm was set to run for 1,000 iterations, fine-tuning the weights of each network after initial GA weight selection concluded. These parameters are summarised in Table 5.23.

The training data set for each cluster was comprised of the cluster's member datum, marked as "1", and datum from all other clusters, marked as "0". With this annotation, the classifiers learnt to distinguish between normal traffic profiles discovered during the clustering process. It is important for the TWEANN classifier training process to include samples of both matching (normal) and non-matching (anomalous) traffic flow samples. Simulated anomalous flow samples were generated by selecting random feature values, where values achieving z-scores in excess of 3 were chosen. These flow samples undoubtedly describe anomalous traffic for the profiled network, as the magnitude of their features exceed the mean average value of normal traffic flows observed over the period of one week by more than three standard deviations. These records provide the training process with samples indicative of anomalous (attack) traffic profiles. The total number of anomalous flow samples generated for each cluster was equal to the number of normal member datum for each cluster.

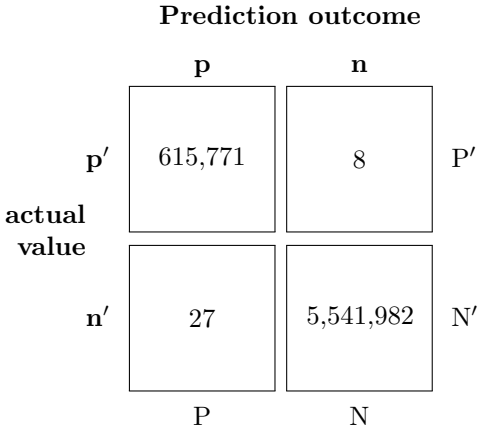


Table 5.25: A confusion matrix for the training set parsed through the trained TWEANN classifiers. A total of 615,771 samples were correctly identified as normal traffic profiles (TP), with 5,541,982 samples correctly identified as belonging to another cluster (anomalous according to the classifier being tested) (TN). A total of 27 were incorrectly marked as belonging to this cluster (FP), with eight normal traffic profile samples erroneously marked as external to this classifier (FN).

A recall test was executed on each TWEANN classifier post its creation, confirming its ability to correctly identify previously seen data. The topology of each classifier, along with its recall scores for the previously seen training set, are listed in Table 5.24. These values are represented as a confusion matrix in Table 5.25. The accuracy, detection rate and false positive rates for the training set recall tests were calculated using Equations 5.1, 5.2 and 5.3 respectively. The accuracy of the recall test was 99.99 percent, with a detection rate of 99.99 percent and a false alarm rate of 4.87E-06 percent.

5.5.6 Classifier Evaluation

The training set recall test results in subsection 5.5.5 demonstrated that the TWEANN classifiers were able to accurately identify instances of previously seen, normal network traffic. In this subsection, the test data set (Section 5.5.2), containing previously unseen flow records derived from the seven-day recordings was evaluated using the TWEANN classifiers. Following this, an additional 24 hours of Netflow data was recorded, where all traffic flows were assumed to be of a normal nature. It was anticipated that the TWEANN classifiers would score normal traffic with high scores, in excess of 90 percent, while scoring the anomalous flows with much lower scores.

		Prediction outcome		
		p	**n**	
actual value	**p′**	154,022	20	P′
	n′	108	1,387,150	N′
		P	N	

Table 5.26: A confusion matrix depicting the evaluation results concluded by the nine TWEANN classifiers on the previously unseen test data set.

The nine TWEANN classifiers were used to evaluate the 154,130 assumed normal flow records of the test data set. The results achieved by the classifiers are summarised in Table 5.26, where 99.99 percent accuracy, 99.93 percent detection rate and 0.001 percent false positive rates were realised. While these were previously unseen flow samples, they were recorded at the same time as the training set.

Ensuring a greater diversity for evaluation purposes, the Netflow collector was once more configured to record flow data for a period of 24 hours on a normal week day, from midnight on Monday through to midnight of the following day. Figure 5.10 plots the total aggregate flows recorded during this period, for each time-slot, where a total of 100,085 flow samples were recorded. The statistical features listed in Table 5.19 were extracted for each aggregated flow, forming a new data set comprised of previously unseen data. The features were normalised using the bounds described in Table 5.21 and evaluated by each of the nine TWEANN classifiers. The testing once more operated under the assumption that the recorded flow samples consisted of only normal traffic flows. The results of this evaluation process revealed that 100,053 of the flow samples were marked as normal traffic, where a certainty score exceeding 90 percent triggered the flow sample to be marked as "normal". Only 32 of the flows considered were scored below 90 percent by all classifiers, resulting in a false positive "anomalous" mark being applied.

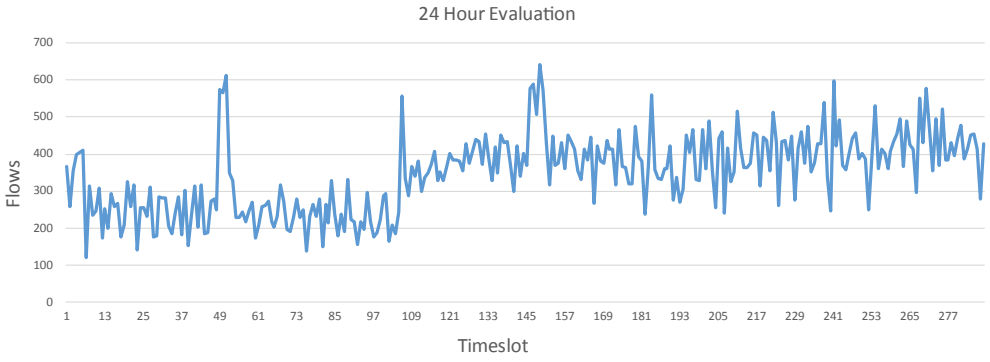


Figure 5.10: A chart describing total aggregated flow records observed over a 24-hour period during classifier evaluation. The aggregated flows are counted for each five-minute time-slot - 286 in total.

A controlled DDoS test was devised to test the classifier's ability to detect anomalous traffic flows on the network. A secondary network interface on the edge router was connected to a new *Virtual Local Area Network* (VLAN) on the corporate managed switch. A new private IP prefix was assigned to the interface, with a default class C scope[14]. This design allows simulated DDoS attacks to be sent to fictitious host addresses on the dedicated network prefix without routing the attacks to the Internet or legitimate hosts on the network. Directing the attack through the edge router ensures Netflow records are generated and forwarded to the APIC classifiers.

Two distinct attack simulations were crafted toward a fictitious host on the new VLAN from within the corporate *Local Area Network* (LAN). The first initiated TCP connections toward the new VLAN at a rapid rate, simulating a TCP connection attack. The attack targeted a single destination host, using the TCP protocol across a number of destination ports. The attack ran for a total of 15 minutes, after which the recorded flows were added to the "normal" data set between time-slots 46 through 48. In total, 2,795 anomalous flows were introduced and reported by the Netflow collector during this time, in addition to the 805 normal flows already reported. The attack flows were easy to manually identify and verify in the recorded data set using the destination host address as the key. The APIC method was successfully able to classify all 805 flows as

[14] A class C IP prefix scope contains 256 IP addresses: 254 are usable for host addressing.

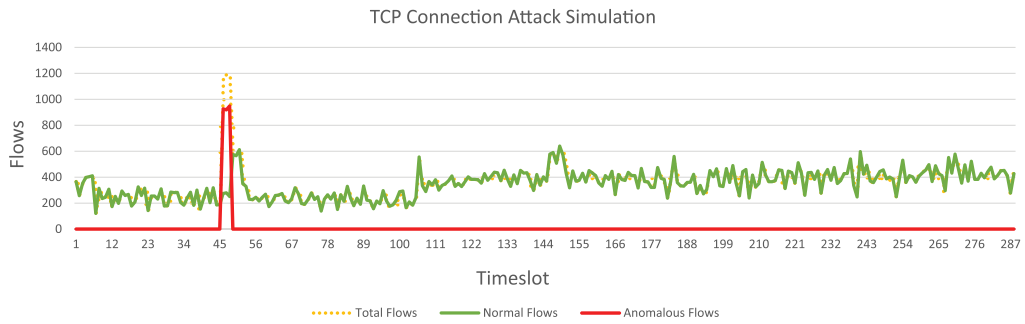


Figure 5.11: A chart showing both the total number of flows recorded during the evaluation window, and the simulated TCP connection attack traffic flows. Both normal (solid line) and anomalous (long-dash line) traffic are shown, as detected by the classifiers.

"normal", with an average probability score of 96.91 percent. The APIC method also successfully scored 2,790 flows as anomalous, where the highest probability score achieved for these flows, by any classifier, was 38.19 percent. A total of five anomalous flows were erroneously marked as normal by the classifiers. These test results are displayed as a confusion matrix in Table 5.27. The accuracy, detection rate and false positive rates were calculated using Equations 5.1, 5.2 and 5.3 as 99.86 percent, with a detection rate of 100 percent and false alarm rate of 0.18 percent.

The second attack simulation consisted of two attacks, designed to test each classifier's ability to detect volumetric attacks. In these tests, UDP packets were crafted with a payload consisting of 1480 bytes of ASCII "A" characters. These packets were sent at a high rate to a single destination host on the attack VLAN, to 30 distinct destination ports. Unlike the TCP connection attack, a significant number of new flows was not generated, as only 30 flows were observed and reported by the router's Netflow exporter per time-slot. Although the average number of flows did not increase significantly over each attack period, the $udpOutPerc$ and $rOutUDPPkts$ features increased significantly during these times (Figure 5.12b). The ICMP features (Figure 5.12a) during this time showed little change, however a distinct drop in the $tcpOutPerc$ feature is noted in Figure 5.12c. Finally, the $rOutTotalPkts$ feature (Figured 5.12d) shows a distinct increase in ratio, while the remainder of the general flow features remain constant.

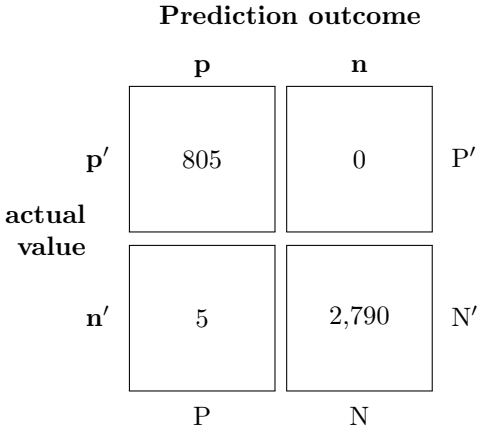
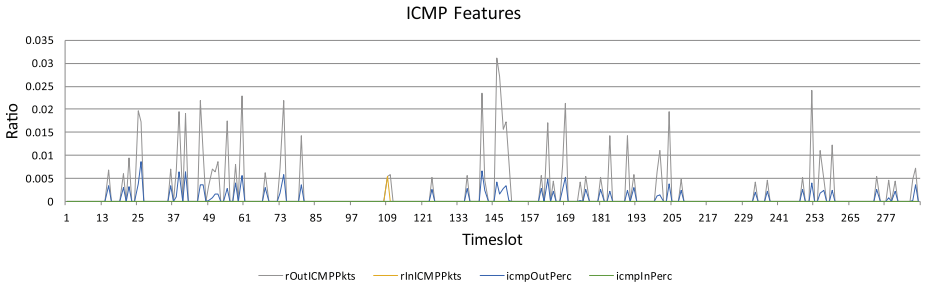
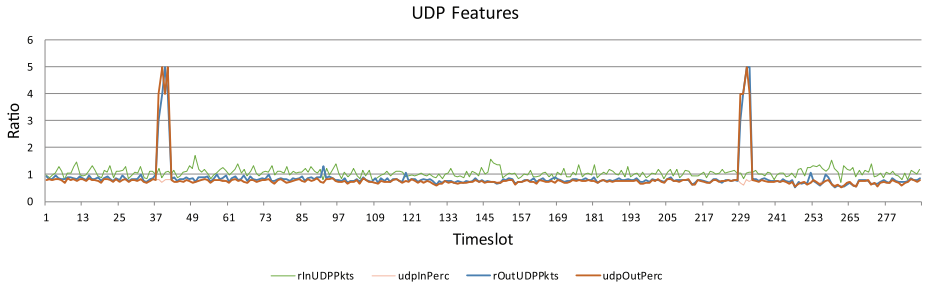


Table 5.27: A confusion matrix for the live TCP connection attack test evaluated by the trained TWEANN classifiers. A total of 805 samples were correctly identified as normal traffic profiles (TP), with 2,790 samples correctly identified as anomalous (TN). A total of five flow samples were incorrectly marked as normal traffic (FP), with no normal traffic profile samples erroneously marked as anomalous (FN).

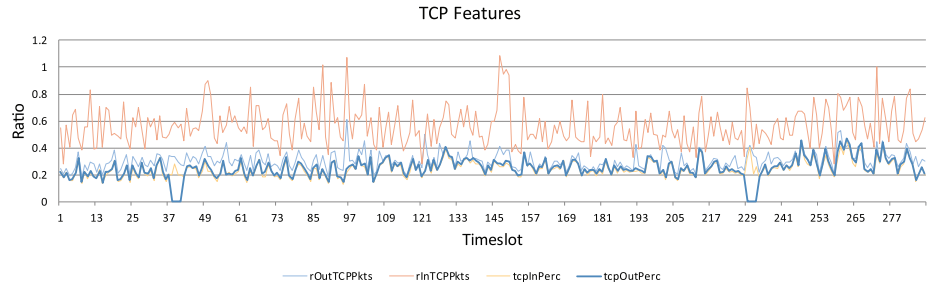
Each attack lasted no longer than 20 minutes, covering four time-slots. For each time-slot, a total of 30 attack flows were maintained, ensuring a consistent volumetric attack over the 20-minute period. The flows observed for each time-slot, along with the average probability score provided by the APIC classifiers, are outlined in Table 5.28.



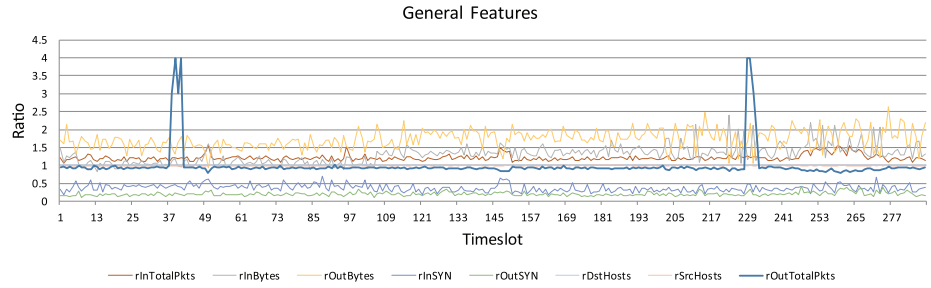
(a) *Internet Control Message Protocol* (ICMP) Features



(b) *User Datagram Protocol* (UDP) Features



(c) *Transport Control Protocol* (TCP) Features



(d) General Features

Figure 5.12: Charts depicting the skewing of flow features observed during a simulated volumetric attack toward a host on the attack VLAN.

Timeslot	Normal	Normal Score	Anomalous	Normal Score
03:05	366	97.24%	29	25.04%
03:10	258	99.01%	30	12.20%
03:15	354	98.97%	30	6.10%
03:20	399	99.20%	30	7.89%
19:00	276	99.44%	30	8.03%
19:05	416	99.05%	30	11.23%
19:10	459	96.50%	28	21.40%
19:15	376	98.80%	30	11.21%

Table 5.28: Results of the live volumetric anomalous flow test. The results show a high degree of probability (in excess of 95 percent) is achieved detecting normal traffic flows, while anomalous flows scored far below the 90 percent certainty mark. These scores indicate that the APIC classifiers were able to accurately discern normal traffic flows from volumetric attack flows using Netflow flow records.

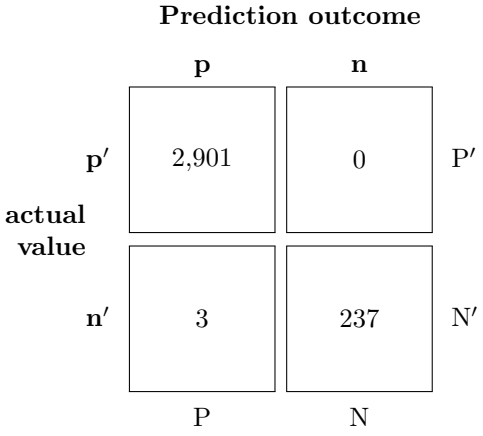


Table 5.29: A confusion matrix for the volumetric attack test evaluated by the trained TWEANN classifiers. A total of 2,901 samples were correctly identified as normal traffic profiles (TP), with 237 samples correctly identified as anomalous (TN). A total of three flow samples were incorrectly marked as normal traffic (FP), with no normal traffic profile samples errantly marked as anomalous (FN).

The results of this test were used to produce a confusion matrix (Table 5.29), where accuracy, detection and false positive rates were calculated as 99.90 percent, 99.89 percent and 0.096 percent respectively.

5.5.7 Discussion

The experiments conducted in this section demonstrate the efficacy of APIC in identifying anomalous (attack) traffic flows on a live network, where the normal network traffic has been profiled and learnt. The accuracy, detection and false positive scores achieved in each of the experiments conducted in this section indicate that the classifiers were able to accurately discriminate between normal and anomalous traffic flows.

The clustering process described in Section 5.5.4 failed to associate 0.12 percent (740) host profile records with any of the nine discovered clusters. It was later determined, through manual tracing, that this traffic was anomalous. An IP-enabled network camera was found to be sending an excessive number of unidirectional ICMP flows egress toward the Internet. The assumption that all recorded flows were normal was based on the commercial IDS and firewall on site not reporting any anomalous attack traffic during the seven-day recording period. The misclassified flow entries in Section 5.5.5 (highlighted in Table 5.25) were also manually traced, where it was determined that the false positive (FP) datum were marked as normal flow entries by another cluster. The close distance relationship between cluster datum caused false positives at the clustering level, but did not hamper detection of these flows as normal traffic by the system as a whole.

Bhuyan et al. (2013) asserted that it was better to stop a network attack at the source, rather than on the victim-end. This is because it prevents wasting of resources across the intermediary networks it traverses en route to the destination. Furthermore, once a flooding attack has entered the target's network, inhibiting the traffic at that point will not free the resources already consumed on their external network connectivity. Stopping an attack at the source, or close to the source, is not always possible, as it requires NBAD systems at each network egress point. By receiving Netflow records from multiple edge routers, the APIC method is ideally positioned to address this problem, providing a normal traffic profiling service for these networks, analysing and classifying all traffic flows traversing edge routers in an efficient and effective manner.

According to Jalili et al. (2005), a single network may exhibit various profiles over different periods of each day. The recommendation was that all

Method	Data Set	Accuracy	Detection	False Positive
APIC	Private Train	99.99%	99.99%	4.87E-06%
APIC	Private Test	99.90%	99.89%	0.096%
Chen (2009)	Private	96.00%	99.99%	5-7%
Jalili et al. (2005)	Private	94.90%	94.90%	2-5%
Bhaya and Manaa (2014)	CBR Train	99.93%	99.92%	0.09%
Bhaya and Manaa (2014)	CBR Test	99.77%	99.53%	0.46%
Bhaya and Manaa (2014)	CB Train	99.93%	99.91%	0.09%
Bhaya and Manaa (2014)	CB Test	97.67%	96.63%	3.23%

Table 5.30: A comparison of accuracy, detection and false alarm rates achieved by comparable, specifically crafted methods against those achieved by the generic APIC method during the testing process.

flow datum should be tested against baseline flows captured during the same time-slot. The experiments conducted in this section found that the average flow feature ratios remained consistent across various times of the day and days of the week, as demonstrated in Figure 5.8. While it is important to aggregate flows using fixed time-slots, these experiments provide evidence that factoring the time-slot itself into the evaluation process (as a feature, for example) when using the APIC method, was not required.

The simulated attacks executed in this section demonstrated the ability of APIC to profile source-end traffic, as it originated from the LAN. It is also argued, however, that the method will be equally effective classifying attack traffic ingress on the *Wide Area Network* (WAN) interface, as all communicating hosts ingress on this interface are also profiled in the same manner as those originating on the LAN. The APIC method is thus capable of acting as a victim-end and source-end NBAD system, identifying attack flows in near real-time and, according to the summarised results in Table 5.30, at least as accurately as comparable systems designed specifically for this purpose. The results highlight the possibilities for early detection of anomalous traffic using Netflow records exported from edge routers in real-time. The experiments demonstrate that the Netflow protocol provides adequate statistical data from which flow profiles can be derived, while placing near negligible additional load on the edge router.

5.6 Conclusion

This chapter presented a case study in which the efficacy of the APIC method was evaluated in the task of *Anomaly Intrusion Detection* (AID). Specifically, the method was tested as both a victim-end and source-end NBAD system, using both publicly available, pre-labelled static data sets and private data sets, recorded on a live enterprise network. The results achieved by the experiments conducted in this chapter indicate that the APIC general method achieves accuracy, detection and false positive rates that are comparable with, and often exceed, results produced by methods designed specifically for this purpose. The automated compilation of feature sets describing host communication profiles and the use of TWEANN classifiers to classify anomalous traffic on computer networks is a unique contribution of this research.

Chapter 6

Document Recognition: Handwritten Digits

Machine learning (ML) has, in recent years, played a significant role in computer vision tasks, including image recognition, video analysis and even playing complex games - defeating human experts at games such as Go[1]. According to LeCun et al. (1998), better pattern recognition systems can be built by relying on automatic learning and removing hand-designed heuristics. This ideal is shared with those of the APIC method, where previous manual feature selection, model development and hyper-parameter tuning processes are replaced by fully autonomous ML methods.

This chapter explores the efficacy of APIC in the task of handwritten character recognition. Using the popular *Mixed National Institute of Standards and Technology* (MNIST)[2] data set, the method automatically extracts features and produces an accurate classifier for a data set of handwritten numbers provided by over 500 contributors. The current state-of-the-art classifier for this data set is modelled manually by human experts and achieves an error score of 0.23 percent (Ciregan et al., 2012). The objective of this chapter is to demonstrate that APIC is capable of designing less complex classifiers automatically, while maintaining test set recall rates that are comparable with existing, manually-created systems. The ability of APIC to achieve this objective provides evidence of its efficacy as a general method for developing classifiers for a variety of tasks.

[1] `https://en.wikipedia.org/wiki/Go_(game)`

[2] `http://yann.lecun.com/exdb/mnist/`

The empirical test devised in this chapter demonstrates that, by including additional distortions or scaled samples of the original MNIST training data set, the APIC method is capable of producing models that classify handwritten characters with competitive degrees of accuracy (Table 6.5). The following sections provide background on image and, specifically, handwritten character-recognition techniques. An experiment follows, in which APIC is evaluated for its ability to construct less complex models for handwritten character-recognition tasks.

6.1 Image Recognition

Many new methods have been proposed lately for enhancing computer vision. In most cases, these methods divide the task into two main parts, namely the *feature extractor* and the *trainable classifier*. The feature extractor is rather specific to each task and thus requires the most design effort (LeCun et al., 1998). This process is responsible for extracting and encoding information about each problem as low-dimensional vectors, suitable for an ML classifier. Feature vectors should be representative of each pattern in the data set, providing a tolerance for transformation, translation and distortion of patterns within the input space. While the features extracted are specific to each task, the classifier is often more general and trainable. According to LeCun et al. (1998), a large number of commercial *Optical Character Recognition* (OCR) systems use some form of multi-layer *Artificial Neural Network* (ANN) (Gurney, 2003) trained using *backpropagation* (Rumelhart et al., 1988).

Over the years, a number of methods have been proposed and tested for recognising images, or objects, in an image. These include linear classifiers, *K-nearest Neighbour* (KNN), Boosted Stumps (Reyzin and Schapire, 2006), Non-Linear Classifiers, *Support Vector Machines* (SVM), ANNs and *Convolutional Neural Networks* (CNN). According to LeCun et al. (1998) and demonstrated by Ciregan et al. (2012), CNNs designed specifically to deal with the variability of 2D shapes outperform all other techniques for identifying handwritten digits.

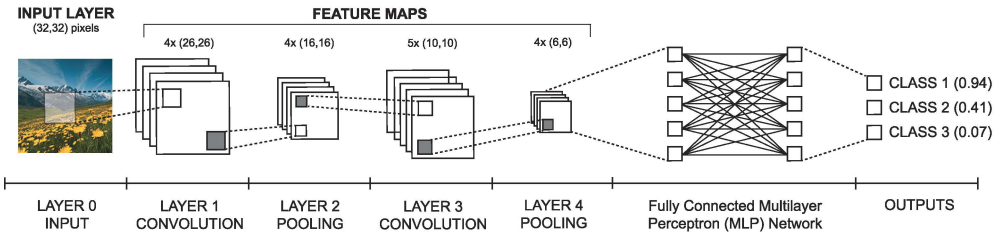
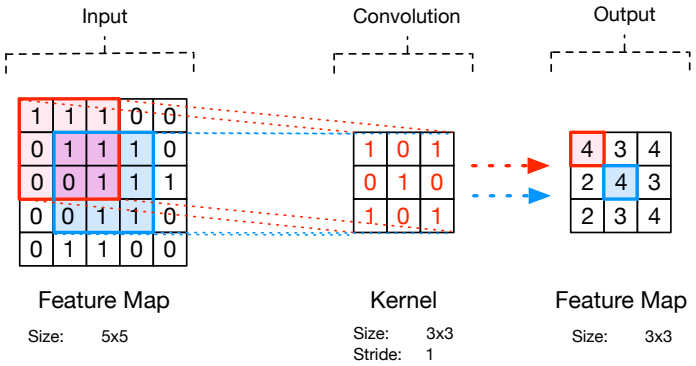


Figure 6.1: An example CNN, where each image (32x32 pixels) is presented to the first convolutional layer (layer 1), comprised of four 2-dimensional (26x26) feature vectors. Each feature vector is sub-sampled, or pooled by the four feature maps (16x16), before passing to a second layer of convolution feature maps (layer 3). The outputs of layer 3 are pooled by four additional feature maps, with a spacial neighbourhood of 6x6. The results of this final pooling process are passed to a fully-connected, MLP network. The outputs of the trained MLP network indicate the probability of the input image matching a particular class. In this case, the classifier identifies the image as class one, with 94 percent probability.

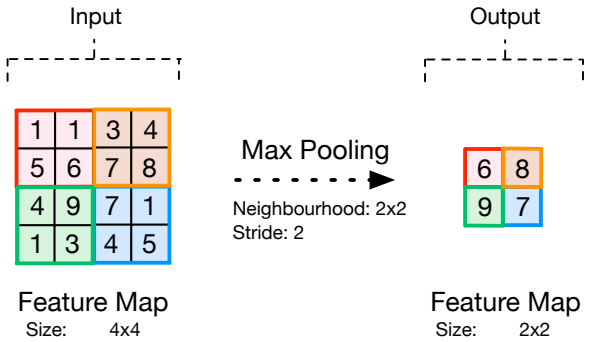
6.1.1 Convolutional Neural Networks

CNNs are a special type of feed-forward ANN, inspired by the organisation of the visual cortex in animals and are comprised of multiple layers of neurons that have tunable weights and biases. Contrary to most other classifiers, CNNs make an explicit assumption that the inputs to the network are images, allowing them to encode certain properties, or features, into the model. CNNs are comprised of three types of layers, namely *Convolutional Layer*, *Pooling Layer* and *Fully-Connected Layer* (Figure 6.1). The convolution and pooling layers may be repeated, where the output of the final pooling process is passed as input to the fully-connected network. CNNs are particularly successful at classifying images as they use special kernels designed to extract important features from an image, known as *convolutions*. These features are resilient, often impervious to transformations, translations or distortion. The ability for CNNs to extract valuable features from images, instead of treating them as a whole, allows CNNs to outperform conventional *multi-layer perceptron* (MLP) models.

Feature maps for each convolutional layer are constructed by translating specific kernels, or filters, over output from the previous layer. In the case of the first convolutional layer, this output is typically the set of image samples.



(a) An example kernel being translated, with a stride of 1, over an input to produce a feature map.



(b) An example of pooling, or down-sampling, used to reduce the values of a feature map and improve generalisation of the model.

Figure 6.2: Examples of convolution and pooling processes, integral components of the *feature extraction* process.

The kernel of each feature map is often chosen by an expert, based on the data set, using a heuristic approach. The kernel is translated across the input matrix, where the amount of overlap and the degree of translation is determined by the *kernel stride* parameter. Feature map values are derived by translating the associated kernel through the input (an image or feature map from a previous layer), calculating the *weighted sum* (Evangelos, 2000) at each location. This process is illustrated in Figure 6.2a. The number of feature maps configured for each layer is another parameter often determined by human experts through heuristic approaches.

Each feature map is reduced using a form of down-sampling, mitigating the effect of inadvertent translation and distortion of objects within images. This down-sampling, or *pooling* (Graham, 2014), improves the generalisation capabilities of the CNN, by averaging the results of specific regions within a feature map. The pooling layer produces new feature maps by translating a matrix, with a predefined spacial neighbourhood (receptive area) and stride, through each feature map of the preceding layer. There are many functions used to implement pooling, where the most common is *max pooling* (Graham, 2014). In max pooling, the value of each location is the largest value within the spacial neighbourhood of the pool at that position (Figure 6.2b).

The final pooling layer of a CNN connects directly to an MLP, expressing datum as single-dimensional input vectors. The number of hidden layers and neurons for each MLP is configured by human experts, based on the requirements for each classification task. The number of outputs present on a CNN matches the number of categories in the labelled training set. Each output neuron provides a value during recall, indicating the probability of an image matching a specific class (Figure 6.1).

The challenge of producing successful CNN classifiers is finding the best configuration for convolution and pooling layers, including the number of feature maps, kernel design and translation stride for each convolution layer and the spacial neighbourhood and stride for each pooling layer. Furthermore, it is necessary to consider the MLP topology design for each task. Typically, these parameters are determined by human experts, through empirical methods, for each task. This requires expert knowledge of both the problem domain and optimal CNN architecture design. Human designed models are often overly complex, requiring significant compute power to process large data sets. The APIC method, already successful at automatically producing efficient, accurate classifiers for *Internet Protocol* (IP) traffic classification (Chapter 4) and *Network Based Anomaly Detection* (NBAD) (Chapter 5) tasks, is a viable alternative to human expert customisation of CNNs for image recognition applications. The following sections provide an empirical evaluation of APIC as a suitable candidate for this task.

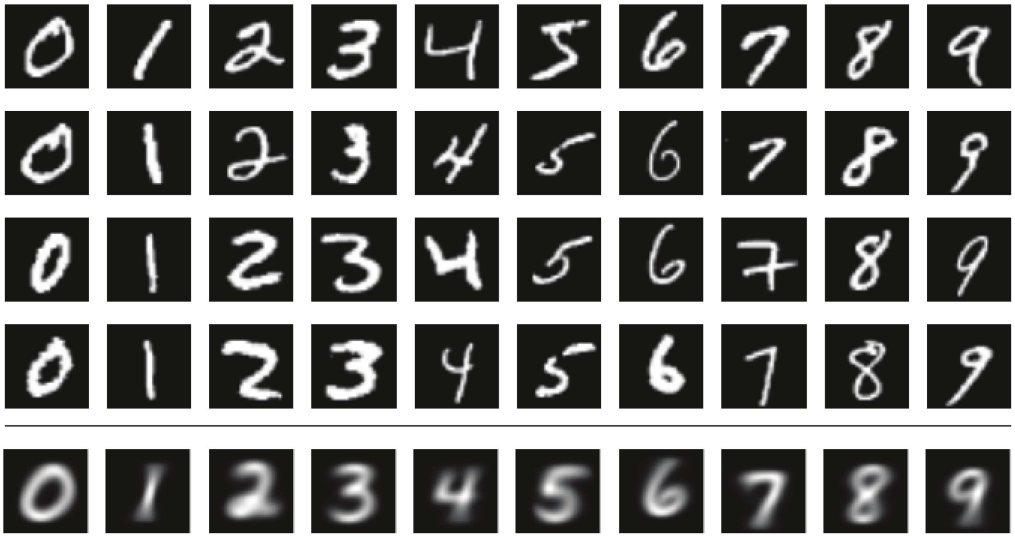


Figure 6.3: Example images retrieved from the MNIST training data set. Five samples of each digit are provided, showing diversity amongst the digits of the same type, and of others. The top four images of each column represent random, individual samples of the digit, while the bottom image shows the mean average of all samples of each digit in the MNIST training set.

6.2 Task Description

The current state-of-the-art method for classifying the MNIST data set was recently proposed by Ciregan et al. (2012), who recorded an error-rate of 0.23 percent. This result is close to the 0.2 percent error-rate expected of humans (LeCun et al., 1995). Some of the 23 misclassified samples were incorrectly labelled in the data set, while others contained random strokes appearing midway through images. To date, most works achieve error rates between 0.23 percent and 8 percent (Section 6.4). The objective of this case study was to determine if, when using APIC, similar error-rates could be realised when classifying the MNIST data set using a less complex CNN classifier, developed automatically by the method.

To succeed at this task, the method should automate the feature selection (feature extraction) and classifier development processes of a CNN, providing a topology that accurately classifies a data set with an error rate comparable with similar systems designed by human experts. The following subsections provide details of an experiment where the efficacy of APIC is tested using the MNIST data set.

6.3 MNIST Data Set

The MNIST data set (LeCun et al., 1998) contains handwritten samples of single-digit decimal numbers, ranging from nought to nine. The data set is divided into a training and testing set comprising 60,000 and 10,000 samples respectively. Each sample has been normalised by scaling the width and height to 20x20 pixels, while preserving the aspect ratio. The result is centred on a 28x28 pixel canvas, where the centre of the handwritten character image aligns with that of the larger 28x28 base image (Figure 6.3). Many image-recognition systems have used the MNIST data set for evaluating their methods. Some of these are discussed in Section 6.4.

As a significant contributor to the field of image and, more specifically, handwritten image classification, the APIC method should demonstrate production of automated classifiers (CNN) that produce error rates rivalling those of similar works, compiled empirically by human experts. Demonstrating the ability for a classifier to achieve competitive results without human intervention allows more efficient feature extraction process definition when developing a CNN. This allows CNNs to be applied more broadly across many image-recognition tasks, without first seeking the input of a human expert.

The first step in this development is for APIC to automatically determine the best features to extract from images for a particular classification task. The following subsection details how the APIC feature selection (Section 3.1) process was applied to achieve this.

6.3.1 Feature Selection

The accuracy achieved by image-recognition systems, much like any classification system, is highly dependent on appropriate features that best describe the problem space (LeCun et al., 1998). While CNNs are capable of extracting valuable features from images (Figure 6.2a) and reducing them through pooling operations (Figure 6.2b) to improve generalisation of the model, the design of the classifier is often determined empirically for each task by human experts. Choices concerning how many convolution and pooling layers to include, the kernel sizes, values and strides, and the MLP topology is ultimately decided this way. This is apparent in many

Feature	Bits	Min	Max
$conv\_layers$	3	1	7
fm_count	3	1	7
fm_stride	3	1	7
$fm\_magnitude$	3	2	7
fm_matrix	$fm\_magnitude^2$	N/A	N/A
$pool\_size$	3	2	7
$pool\_stride$	3	1	7

Table 6.1: Parameters for defining the configuration of the convolution and pooling layers in an APIC-tuned CNN classifier. The value of each parameter is encoded as a bit string and concatenated to form a variable-length genotype.

systems tested against the MNIST data set (Section 6.4).

The APIC feature selection process (Section 3.1) is well suited for determining the best feature sets for a variety of classification problems. The process can adapt to design and optimise the configuration of convolution and pooling layers for a CNN classifier, as demonstrated in this case study. First, the APIC method divides the CNN design process into two related search problems: the search for an optimal convolution and pooling configuration specific to the task (feature selection), and designing the topology of a more general, fully-connected MLP model for classifying them (classifier (MLP) development).

For this case study, the feature selection process used a *Genetic Algorithm* (GA) with variable-length genotypes to describe configurations of the convolution and pooling layers. The parameters governing the design of each configuration (Table 6.1) were encoded as bit strings and joined to form a complete genotype. The range for each parameter was determined heuristically, based on estimated requirements for classifying MNIST images and empirical testing of APIC on an MNIST training data subset. Each genotype was formed, or compiled, using algorithm 2.

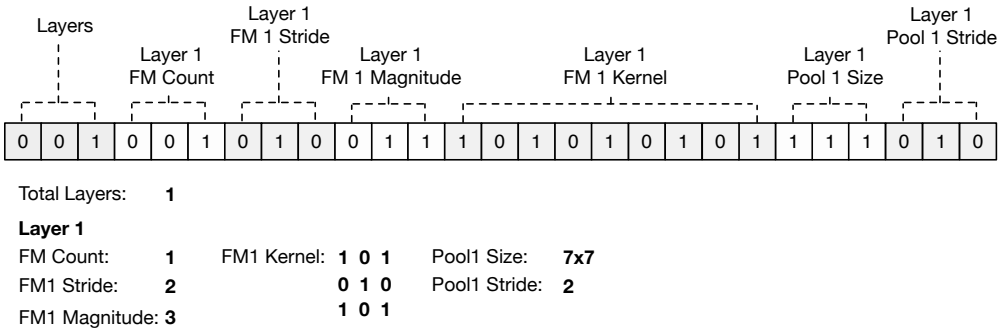


Figure 6.4: A convolution and pooling configuration encoded as an APIC feature selection genotype. The genotype describes a single convolution and pooling layer with a single feature map. The feature map's configuration is 3x3, with a stride of 2. The feature map's kernel is encoded in 9 bits (3x3). The kernel is followed by the size of the single pool (7x7) with a stride value of 2.

```
genotype = as_bits(layer_count)
foreach layer in layers do
    genotype += as_bits(layer.fm_count)
    foreach feature_map in layer.feature_maps do
        genotype += as_bits(feature_map.stride)
        genotype += as_bits(feature_map.magnitude)
        genotype += as_bits(feature_map.get_as_bitstring())
    end
    foreach pool in layer.pools do
        genotype += as_bits(pool.size)
        genotype += as_bits(pool.stride)
    end
end
```

Algorithm 2: Pseudo-code outlining the compilation process for encoding a CNN feature extraction configuration as a bit string for the APIC feature selection process (Section 3.1).

An example genotype describing a configuration with a single layer of convolution and pooling, with a single feature map and single pool is illustrated in Figure 6.4. The variable number of layers, feature maps and pools encoded makes it inefficient to encode each configuration as a fixed-length genotype. Instead, the size of each genotype is determined dynamically, where the number of genes is directly proportional to the size of the configuration being encoded.

Parameter	Description	Value
max_population	Maximum number of genotypes per generation	100
elitism	Number of best genotypes transferred to next population	10
max_generations	Maximum number of generations to traverse	100
mutation_rate	Mutation rate applied when breeding next generation	0.1
crossover_rate	Crossover rate applied when breeding next generation	N/A

Table 6.2: GA parameters used to control the feature selection (convolution and pooling configuration) search process.

Genotypes with random values were generated for the initial population, where the number of layers, feature maps per layer, kernel size, stride and values were selected at random, within the bounds listed in Table 6.1. The parameters guiding GA execution are listed in Table 6.2. All non-elite genotypes were evaluated at each generation, where successful configurations were used to train MLPs, where the best topology was found by a second GA (Section 6.3.2). Configurations were marked as successful if no errors were raised when extracting features from the MNIST training data set, according to the convolution and pooling configuration described by the genotype. Unsuccessful configurations, such as a multi-layer configuration, where a particular pool's spacial neighbourhood exceeds the dimensions of the source feature map, are marked with a fitness score of -1. Outputs derived from genotypes whose configuration successfully extracted features from all images were used to train MLPs (Section 6.3.2), where the recall score of the best scoring classifier was used as the fitness score for the feature extraction genotype. The topology of the best scoring MLP was recorded and stored with each genotype.

After scoring each genotype of the population, the highest achieving (most elite) genotypes were identified and, together with their existing MLP solution, transferred directly to the population of the succeeding generation. Per generation, ten of fittest genotypes were transferred to the subsequent population. This is an heuristic value, based on past experience with populations up to 100 total genotypes. The remainder of the new population was developed by selecting existing genotypes from the current generation, using roulette wheel selection (Al Jadaan et al., 2008), altered by application of a custom bit string mutation algorithm, using the

configured *mutation_rate* of 0.1. This heuristic, low value was chosen to prevent the GA from converging too quickly, which may cause the algorithm to converge on a local minima, ultimately representing an inadequate solution to the problem. Simply inverting the value of a particular bit, a common practice when applying bit string mutation, is not suitable for the bit strings in this experiment, as it may render an invalid configuration (corrupt genotype). Instead, a specialised mutation function was developed specifically for this task (algorithm 3), allowing the mutation operation to modify any of the encoded configuration parameters independently.

```
topology = get_from_bitstring(genotype)
/* Mutate the number of layers                                    */
if rand_float() > mutation_rate then
    /* Generate number of layers                                  */
    new_layers = random_int(low=min_layers, high=max_layers+1)
    /* Generate new and remove old layers                         */
    topology.reshape_layers(new_layers)
end
/* Manipulate each layer                                          */
foreach layer in topology.layers do
    /* Mutate the number of feature maps                          */
    if rand_float() > mutation_rate then
        new_fms = random_int(low=min_fms, high=max_fms+1)
        /* Add or remove FMs to this layer as dictated by new_fms */
        layer.reshape_fms()
    end
    /* Mutate the values of kernels                               */
    foreach kernel in layer.kernels do
        /* Mutate neighbourhood size                              */
        if rand_float() > mutation_rate then
            kernel.random_resize()
        end
        /* Mutate kernel stride                                   */
        if rand_float() > mutation_rate then
            kernel.random_stride()
        end
        /* Mutate kernel values                                   */
        foreach bit in kernel do
            if rand_float() > mutation_rate then
                bit.invert()
            end
        end
    end
end
```

```
/* Mutate the number of pools                                    */
if rand_float() > mutation_rate then
    new_pools = random_int(low=min_pools, high=max_pools+1)
    /* Add or remove pools to this layer as dictated by new_pools */
    layer.reshape_pools()
end
foreach pool in layer.pools do
    /* Mutate spacial neighbourhood size                         */
    if rand_float() > mutation_rate then
        pool.random_resize()
    end
    /* Mutate pool stride                                        */
    if rand_float() > mutation_rate then
        pool.random_stride()
    end
end
return compile_genotype(topology)
```

Algorithm 3: Pseudo-code demonstrating the mutation process, developed specifically for this case study, executed on each new genotype of the successive generation.

While it is possible to implement a custom crossover mutation function in a similar fashion, the potential gain realised from this process was deemed insignificant compared to the complexities related to ensuring the encoded configuration remained valid after mutation. The search for the best configuration continues until *max_generations* is reached (Table 6.2), or another predefined stop condition is triggered (algorithm 1). For this experiment, the search process was permitted to continue until *max_generations* was reached.

6.3.2 Classifier (MLP) Development

The genotypes produced in Section 6.3.1 were used to extract features from the MNIST training data set. Prior to any convolution or pooling operations, each MNIST sample image was resized (normalised) to a 20x20 pixel size image in accordance with Ciregan et al. (2012). Each image was duplicated four times, where the first copy was resized to 10x10 pixels. The images of the second copy were resized to 14x14 pixels. Both of these images were then up-scaled to 20x20 pixels. This resizing process was

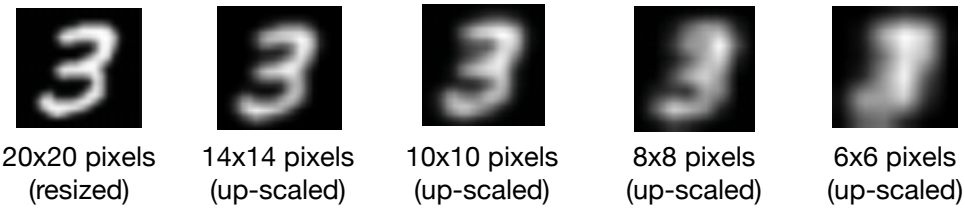
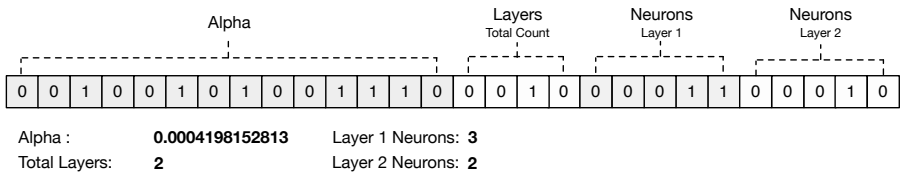


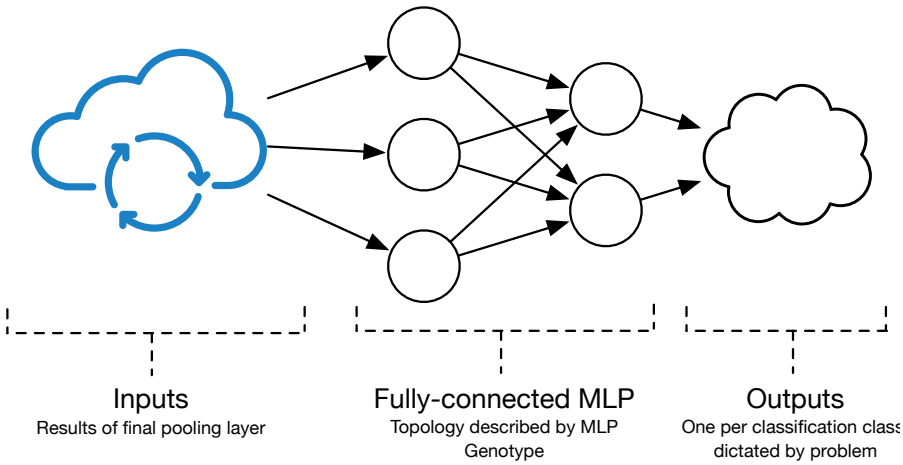
Figure 6.5: Each MNIST sample is down-scaled to a 20x20 image pixel. The original MNIST training data set is duplicated twice, where samples of the first copy are resized to 14x14 images, whereafter they are up-scaled to 20x20 images. The second copy is reduced to 10x10 size, before being up-scaled to 20x20. This process increases the distortion of the data set, while preserving the label for each sample. Reducing images below 8x8 size reduces the image quality to a state where the original handwritten digit is difficult even for humans to identify.

designed to increase generality of the total training set, by distorting the original images through sampling and up-scaling, while retaining the original classification label. Early testing revealed that resizing sample images below 10x10 pixels caused distortions that were too significant after up-scaling the image to its original 20x20 pixels size (Figure 6.5). Each image in the third and fourth data sets were rotated around the image centre five degrees clockwise and five degrees anti-clockwise, respectively. This rotation provided a final perspective for the classifier, improving generalisation and providing better classification accuracy. The third and fourth data sets were chosen for this process as a suitable level of distortion was already applied. Five degrees was a heuristic value, used in order to allow additional perspectives of the images to be presented to the classifier during the training process.

A second GA was configured to search for the best fully-connected MLP classifier topology for the CNN network. The 300,000 sample training data set was used to train each candidate MLP classifier designed by this GA. The focus of the MLP GA was on identifying the best scoring MLP topology for each feature extraction genotype, using the APIC classifier production process described in Section 3.3. The GA encodes the MLP topology and training parameters (Table 6.3) as bit string genotypes.



(a)



(b)

Figure 6.6: An example showing how an MLP topology is encoded as a bit string genotype using APIC. The variable-length encoded genotype (a) represents a fully-connected MLP classifier (b).

These genotypes are variable in length, where magnitude is proportional to the number of hidden layers present in the MLP topology. An example MLP topology, encoded as a variable-length genotype, is illustrated in Figure 6.6. The example genotype describes a two-layer fully-connected MLP, where layer one contains three neurons and layer two contains two neurons. The alpha value is derived by dividing the decimal representation of the 14 bit alpha bit string into one. Figure 6.6 illustrates the implementation of the encoded topology, where the magnitude of the input vector is determined by the output produced by the convolution and pooling configuration (Section 6.3.1) and the number of outputs by the number of classes of the problem. The parameters used to control the GA for training each MLP in this experiment are listed in Table 6.4.

Feature	Bits	Min	Max
alpha	14	1	16383
layers	4	1	15
neurons	5	1	31

Table 6.3: Parameters for defining the configuration of each MLP. The value of each parameter is encoded as a bit string and concatenated to form a variable-length genotype.

Parameter	Description	Value
max_population	Maximum number of genotypes per generation	100
elitism	Number of best genotypes transferred to next population	10
max_generations	Maximum number of generations to traverse	1000
mutation_rate	Mutation rate applied when breeding next generation	0.1
crossover_rate	Crossover rate applied when breeding next generation	N/A
train_iterations	Training iterations for the backpropagation algorithm	10,000

Table 6.4: GA parameters used to control the feature selection (convolution and pooling configuration) search process.

The weights of each MLP topology were randomly selected, prior to tuning by the APIC classifier production process (Section 3.3). The training process loaded all 300,000 MNIST training samples, including distortions and rotations, extracting features for each genotype produced by the feature selection process (Section 6.3.1). The backpropagation algorithm was used to tune the weights of the classifier over a maximum of 10,000 iterations (*train_iterations*, Table 6.4). This value was chosen heuristically, based on observations noted when training MLPs within a CNN on similar data sets. A recall test was performed on each trained MLP classifier, where the full MNIST test data set, comprising 10,000 previously unseen, labelled samples, was subjected to the feature extraction process and evaluated by the MLP. The best scoring MLP topology, determined by the recall accuracy of the MNIST test data set, was returned and used as the fitness score for each feature extraction genotype produced in Section 6.3.1.

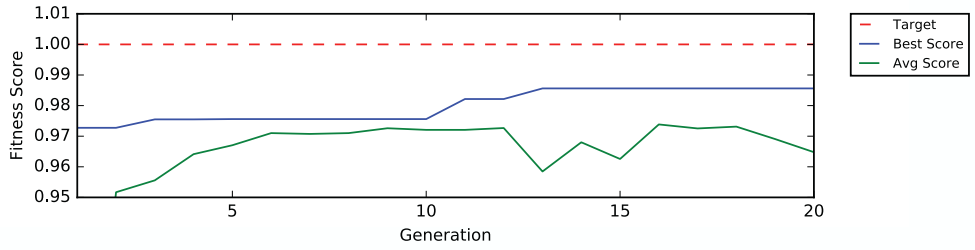


Figure 6.7: Results of the APIC feature extraction search process. The chart includes results for the first 20 generations as no improvements were observed after generation 13.

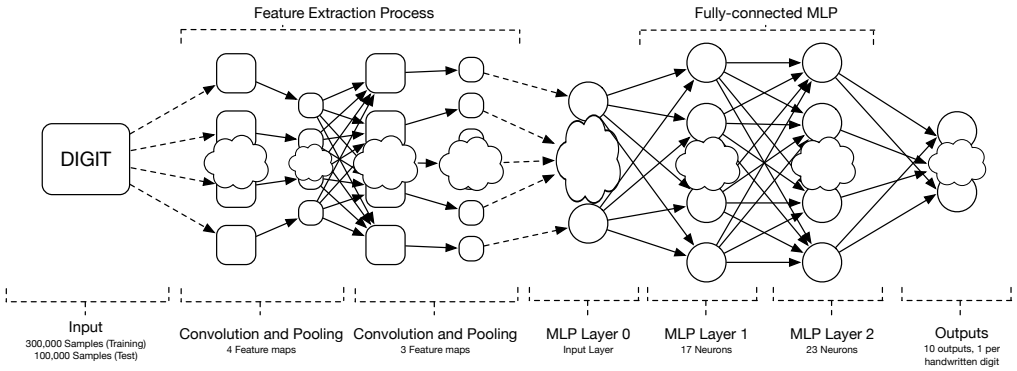


Figure 6.8: Best scoring CNN topology designed by APIC.

The best scoring feature extraction genotype and its associated convolution and pooling architecture, along with the best scoring MLP topology produced by this genotype, were combined to form the best scoring CNN produced by the APIC method for the MNIST data set using the parameters listed in Tables 6.1 and 6.3. The following subsection details the results achieved by the APIC method after the experiment concluded.

6.3.3 Results

The APIC method produced a top scoring CNN topology, reaching an accuracy rate of 98.86 percent (1.14 percent error rate). This was achieved by extracting features from the 300,000 MNIST training set samples (60,000 original, 120,000 resized and up-scaled samples, and 120,000 rotated samples) and following the steps outlined in Section 6.3.1 and 6.3.2.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

(a) Layer 1: Kernel 1
Stride 2

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

(b) Layer 1: Kernel 2
Stride 3

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

(c) Layer 1: Kernel 3
Stride 2

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

(d) Layer 1: Kernel 4
Stride 2

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(e) Layer 2: Kernel 1
Stride 1

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

(f) Layer 2: Kernel 2
Stride 1

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

(g) Layer 2: Kernel 3
Stride 1

Figure 6.9: A depiction of the seven kernels chosen by APIC for the feature extraction layer and their associated strides. The first four were assigned to the first convolution layer, with the last three forming the second convolution layer.

The best scoring feature extraction configuration was discovered at generation 13. The scores achieved by the feature extraction search process at each generation are shown in Figure 6.7. The best scoring feature extraction configuration included two convolution and pooling layers, illustrated in Figure 6.8. The best scoring MLP for this configuration consisted of two hidden layers and 40 total hidden neurons (Figure 6.8). The kernel matrices for each convolution layer are shown in Figure 6.9.

A pooling process was executed for each convolution layer of the best-scoring CNN. Layer 1 used four pools, with spacial neighbourhood (receptive area) and strides set to 3x3 stride 2, 3x3 stride 3, 3x3 stride 2 and 3x3 stride 2 respectively. Layer 2 included three pools, with configuration 4x4 stride 1, 2x2 stride 1 and 2x2 stride 1 respectively. The best-scoring MLP configuration was found with two hidden layers, with layer 1 and layer 2 consisting of 17 and 23 neurons respectively. The alpha value for the MLP was identified during the search process as 1/16026 or 6.2398e-5. The alpha value is used for regularisation, aiding to avoid overfitting by penalising weights with large magnitudes.

6.4 Discussion

Many approaches for handwritten digit identification have been published over recent years. The following section outlines a few approaches tested using the MNIST data set. Some of these approaches perform poorly, resulting in high error rates, while others define the state-of-the-art technique for classifying the data set. One clear observation is that methods achieving low error rates often exhibit complex models consisting of hundreds of feature maps (for CNNs) and hundreds of neurons (for ANN/MLPs). A summarised list of each method discussed, and their respective error rates, is recorded in Table 6.5.

LeCun et al. (1998) tested a variety of gradient-based methods for classifying the MNIST data set, including linear classifiers, K-nearest neighbour, *Principle Component Analysis* (PCA), *Radial Basis Function* (RBF), SVMs, ANNs and CNNs. For the purposes of this discussion, the focus is on the approaches and results achieved by the authors for their experiments on ANN and CNN classifiers. LeCun et al. (1998) produced

Method	Description	Error-rate
LeCun et al. (1998)	ANN Classifiers	1.6% - 4.7%
LeCun et al. (1998)	CNN Classifiers	0.7% - 1.7%
Ciregan et al. (2012)	CNN (Multi-column Deep Learning)	0.23%
Makhzani et al. (2015)	Semi-supervised AAE	0.85%
Makhzani et al. (2015)	Unsupervised AAE (16/30 Clusters)	9.55%/4.10%
Mohapatra et al. (2015)	CDCST Approach	1.2%
APIC	Automated CNN development	1.14%

Table 6.5: A list of recent competitive results for classifying the MNIST database of handwritten digits, ordered by date of publication.

multi-layer ANN topologies, experimenting with the number of hidden layers and neurons. The first test, on an ANN with a single hidden layer and 300 hidden neurons, resulted in an error-rate of 4.7 percent. This error-rate was improved to 4.5 percent, by increasing the number of hidden neurons to 1,000. Growing the training set by including distortions of the original images reduced the error rates of the 300 and 1,000 neuron networks by 3.6 percent and 3.8 percent respectively. The most significant impact on the error rate was achieved when slanted, or rotated, training images were used. Including these samples, the error rate for the 300 neuron network was reduced to 1.6 percent. The authors also found that, by expanding the network to two hidden layers and adding additional neurons per layer (300 and 100 respectively), only marginal improvements were observed, achieving 3.05 percent error-rate.

LeCun et al. (1998) expanded their ANN search to include defining CNN classifiers for the same task. For the first test, samples from the MNIST training data set were down-sampled to 16x16 pixels and centred in the 28x28 input layer. The authors did not go into detail on their design of the convolution and pooling layers for this classifier, however they stated that the classifier achieved an error-rate of 1.7 percent during test data set recall. These results led the authors to experiment with larger convolutional networks. The authors tested a network, *LeNet-4*, which contained four first-level feature maps, connected to eight sub-sampling (pooling) maps, followed by 16 feature maps and another 16 pooling maps. The results were passed through a single-layer MLP with 120 neurons, which achieved an error-rate of

1.1 percent. The authors next developed *LeNet-5*, by combining or *boosting* (Drucker et al., 1993) three LeNet-4 networks, training and operating them in an ensemble fashion. An error-rate of 0.7 percent was observed, the best of all classifiers tested by the authors (LeCun et al., 1998).

The method proposed by Ciregan et al. (2012) represents the current state-of-the-art technique for classifying images of the MNIST data set. The authors achieved an error-rate of 0.23 percent, producing the best scoring classifier for this data set to date. The method, like LeNet-5 (LeCun et al., 1998), uses *multi-column Deep Neural Networks* (MCDNN), with convolution and pooling layers as pre-processors. The authors normalised the MNIST training data set to 20x20 pixel images, from which six additional data sets were created with sizes 10x10, 12x12, 14x14, 16x16 and 18x18 pixels. The authors asserted that reducing the size of the images was akin to observing the same image from different angles and perspectives. The authors trained five CNNs for each data set, resulting in 35 total columns for the classifier. Samples from the data sets were distorted at each of the 800 epochs, where a learning rate decay of 0.993 was applied. This reduced the learning rate over the training process from 0.001 (initial) to 0.00003. The results of all five CNNs were combined, producing the final classification result.

Makhzani et al. (2015) proposed the use of the *Adversarial Autoencoder* (AAE), an *Autoencoder* (AE) (Bengio et al., 2009) that maps an AE output distribution $q(z|x)$ to an arbitrary prior distribution $p(z)$ using adversarial training, rather than traditional variational inference. Using this method, a batch of inputs were encoded and decoded by the AE, which was updated based on the standard reconstruction loss between input and expected output. A second batch of inputs was transformed by the AE encoder, after which they were joined with samples from the prior distribution $p(z)$. The discriminator was updated, based on its ability to separate samples generated by the AE and those of $p(z)$. Finally, a batch of inputs was transformed by the AE encoder and passed to the discriminator for evaluation. The discriminator predicted the source of the data, after which the AE encoder was updated. The reconstruction loss of the AE continued to decline, as did the adversarial loss, until such time as improvements in the discriminator, and that of the generator, caused the network to converge. The authors recorded an error-rate of 0.85 percent when

executing an AAE in a semi-supervised learning configuration. In this test generated, unlabelled samples were used to supplement labelled samples when developing the classifier. Finally, the authors removed labels altogether, testing an unsupervised model that resulted in error-rates of 9.55 percent and 4.10 percent for 16 and 30 clusters, respectively.

Mohapatra et al. (2015) proposed the *Discrete Cosine S-Transform* (DCST) method for classifying the MNIST data set. The authors removed the boundary pixels of each 28x28 pixel MNIST sample, by reducing each to 20x20 pixels. The authors used DCST to perform the feature extraction process for each image, resulting in a feature length of 400. These features were used to train an MLP classifier with 400 input neurons, ten output neurons and a single hidden layer. Unfortunately the authors failed to state how many neurons were present in the hidden layer, only stating that they were "appositely fixed". The authors also considered only 10,000 of the 70,000 available samples from the MNIST data set in their experiment. Using this data subset, they achieved an error-rate of 1.2 percent during their recall tests.

Application of the APIC method for classifying the MNIST data set, described in Section 6.3, automatically generated best scoring CNN designs within the parameters listed in Tables 6.1 and 6.3. The most elite (highest scoring) genotype from the feature extraction GA defined a CNN classifier that achieved 98.86 percent recall rate (1.14 percent error rate) when evaluated on the full MNIST data set. This value is comparable with the ANN and CNN results of LeCun et al. (1998), where the authors achieved error-rates between 0.7 percent and 4.7 percent, using complex topologies consisting of many neurons and feature maps. The recall and error rate achieved by the less-complex, APIC-generated CNN classifier rivalled those achieved by semi-supervised and unsupervised AAE, tested by Makhzani et al. (2015). The scores achieved by the APIC-generated CNN also compare against those of Mohapatra et al. (2015), where 1.2 percent recall rate was achieved when testing on the full MNIST data set.

While the results produced by the APIC classifier are competitive, the most noteworthy contribution of APIC to this task is the ability to generate less complex models for classifying data. These less complex models were demonstrated to achieve results comparable with more

complex, manually-defined classifiers. Complexity, in this case, is measured by the number of hidden layers and hidden neurons required by the MLP to achieve a model capable of generalising well, where models comprised of less neurons and layers are considered of lower complexity (Kon and Plaskota, 2006). For CNN classifiers, complexity extends to include the composition of the feature extraction layers, where less convolutional layers and feature maps of lower dimensions are considered less complex. The complexity of the classifiers produced by APIC was controlled by strict parameters (Tables 6.1 and 6.3), forcing the method to search for best feature extraction configurations, including kernel designs, to improve classification scores. An explanation for how the number of feature maps, kernels (filters) and pooling operations were chosen was excluded from all of the comparable CNN approaches discussed in this section. After reviewing all available information, this section concludes that these configurations were derived heuristically. There is no evidence to suggest significant effort was invested in optimising these configurations to improve task performance, an issue plaguing many systems evaluating MNIST and data sets of similar size (Kumar et al., 2010; Mohapatra et al., 2015). The APIC method demonstrated that competitive results could be achieved using far less complex models. The model achieved competitive results using seven feature maps and 40 neurons, while LeCun et al. (1998) used 20 feature maps and 120 neurons for *LeNet-4* and Ciregan et al. (2012) used 820 feature maps and 150 neurons for each DNN of MCDNN. Although the accuracy results achieved by the best scoring APIC classifier failed to challenge the current state-of-the-art design (Ciregan et al., 2012), they provided evidence to suggest that simplified models could be developed to achieve similar results to those of the 35 DNNs developed as part of the MCDNN classifier.

The MCDNN developed by Ciregan et al. (2012) is effectively 35 DNN classifiers working together to classify data samples. The same boosting approach was tested by LeCun et al. (1998), where the results of three classifiers were used to decide the classification result. LeCun et al. (1998) observed the best results of all classifiers using this strategy. The APIC method, as part of the feature extraction process, developed a population of suitable CNN candidates, of which ten were the most elite (best-scoring). It

is possible that by utilising a combination of these best-scoring CNN classifiers, the recall and error rates achieved by APIC at this task could be improved further. This evaluation, however, falls outside of the objectives of this chapter and will subsequently be evaluated in future research.

6.5 Conclusion

The objective of this case study was to demonstrate that, by applying methods like APIC, the complexity of image-recognition classifier models could be reduced, while maintaining competitive recall and error rates. This process was demonstrated using the full MNIST data set, where previously unseen handwritten digits were identified with an accuracy rate of 98.86 percent (1.14 percent error rate) using a single CNN classifier. This chapter found that accurate, low-complexity CNN models can be produced by paying attention to the optimisation of the feature extraction process. Using the APIC automated feature selection (Section 3.1) and classifier production (Section 3.3) processes, comparably accurate, less-complex CNN classifiers could be produced, where recall and error rates rivalled those of more complex, manually-defined classifiers.

Less complex models increase the reach of classification algorithms, especially in situations where computational overhead is critical, such as in embedded systems. As the increase in compute power continues to expand, the range of APIC-operating parameters can also be expanded, allowing the method to explore and test more complex classifiers. This, coupled with a boosting strategy positions APIC as a method that can produce classifiers that challenge state-of-the-art techniques for image recognition and similar classification tasks.

Chapter 7

Discussion and Future Work

Automated Pattern Identification and Classification (APIC) is a principled *Machine Learning* (ML) pipeline method that aims to automate many of the manual tasks currently associated with ML model development. During this process, efficient model phenotypes are discovered, which generalise well compared to topologies constructed manually by human experts. More efficient models lead to significant performance gains and reduce the chance of overfitting. Intuitively, overfitting occurs when the model fits the problem too well, often due to an overly complex model definition. Likewise, an underfitting situation occurs when the underlying trends of the data are unable to be captured, often due to an overly simplified model design. The APIC method prevents both of these situations by evolving models over a number of generations, evaluating each phenotype using previously unseen data. APIC is a general method, with efficacy extending beyond a single domain into a broad range of complex classification tasks, characterised by noisy, mixed data sets consisting of either static or streaming data.

This chapter begins with a general discussion of APIC's approach to reducing dependency on human experts for implementing ML technologies in a variety of complex classification tasks, improving accuracy, completeness, efficiency (performance) and generalisation compared to similar systems. The discussion focuses on implementation of the method in three complex tasks: the first two describe related tasks in the network domain, with the third a distinct task in the domain of computer vision. Successful application of the method in each of these tasks demonstrates the efficacy of APIC as a

general method for a broad range of classification tasks. The remainder of the chapter provides areas of future work, where additional research can be applied to further enhance the APIC method.

7.1 Automating Classifier Model Design

The APIC method provides several advances for implementing ML in many complex classification tasks. Although algorithms, such as *Artificial Neural Networks* (ANN), have proven effective in solving complex tasks, the design of a suitable model architecture can be challenging, even for human experts. Using a *Neuroevolutionary* (NE) approach, many of the human design requirements associated with implementing ML technologies in these areas are removed. This leads to a more pervasive deployment of ML technologies, especially in instances where no human experts are available, or where complexity of the task dictates that manual designs are not feasible. This is especially true for evolving data, where new patterns are frequently introduced, such as in real-time streaming data sets. In these cases, it is neither feasible nor cost effective for human experts to monitor and update classifiers manually.

Incorporating feature selection (Section 3.1), pattern discovery (Section 3.2) and classifier production (Section 3.3) tasks into a single pipeline method allows all aspects of a classification task to be tracked and tuned holistically by the APIC method. Rigorous self-evaluation of each model, automatically designed to classify identified patterns using optimised feature subsets discovered by the method and tested by cross-validation using previously unseen sample data, promotes the development of efficient, high-performance models that generalise well. The results achieved by these models are automatically improved using *Evolutionary Algorithms* (EA), which encode candidate solutions as bit strings. While candidate solutions, in this dissertation, were provided by EAs, it is possible to substitute alternative optimisation methods for each task. Likewise, alternative algorithms may be implemented in each process of the pipeline method, as the situation dictates. More detail on the topic of algorithm substitution is provided in Section 7.3.

In this dissertation, three complex classification tasks were chosen to demonstrate efficacy of the APIC method. Successful application of the method in these tasks demonstrates that it is broadly applicable, achieving accuracy, detection and error rates comparable with methods designed specifically for each task, with less human expert design input. The advantages provided by APIC do not imply that these classifiers will always outperform those manually developed for a particular task. Instead, the high-level objective of each case study was to demonstrate that even today, complex hand-designed classifiers could be replaced by high-performance models that achieve at least comparable accuracy, automatically. The following section provides a general discussion regarding the application and benefits of APIC for each case study considered in this dissertation.

7.2 Case Study Analysis

Three current, complex classification tasks were chosen as case studies to demonstrate the efficacy of the APIC method across a broad range of classification problems. The first was *Internet Protocol* (IP) traffic classification (Chapter 4), a task characterised by low degrees of completeness, owing to manual design processes often associated with IP traffic classification systems. APIC was also applied to a related task, *Network-Based Anomaly Detection* (NBAD), in Chapter 5, where the method was tested as a viable victim-end and source-end solution, classifying anomalous traffic early and accurately. Finally, Chapter 6 tested APIC for automatically developing more efficient (higher-performance) models for classifying images as accurately as more complex models, designed by human experts.

The following subsections discuss the benefits arising for each task through application of the APIC method.

7.2.1 IP Traffic Classification

The goal of an IP traffic classification system is to understand the type of traffic traversing networks as it evolves in both scope and complexity (Zhang et al., 2009a). The evolution of application protocols traversing IP networks

presents a significant problem for vendors, as unique signatures are required to classify instances of each application protocol to maintain completeness of the system. Completeness and accuracy are, according to Szabo et al. (2007), two performance metrics often used to evaluate IP traffic classification systems.

In Chapter 4, most classifiers for modern IP traffic classification systems were found to be designed by human experts. This process introduced delays and risk of error during signature development, reducing both the completeness and accuracy of the system. The APIC method, in contrast, demonstrated that higher degrees of completeness were achievable by identifying new application protocols in streaming, mixed, noisy data sets, producing efficient and accurate classifiers to identify future instances of each automatically. The case study demonstrated that using APIC, a self-sustaining classification framework that protects against innovation, remaining up-to-date and relevant, was achievable. APIC's ability to automatically identify new application protocols in a constantly evolving data set demonstrated that automatic, accurate identification of new patterns in unlabelled, mixed, noisy data sets was possible using a general ML pipeline method. Furthermore, APIC automatically produced and evolved customised classifiers, where accuracy results were comparable with, and often exceeded, those achieved by more complex classifiers, manually designed by human experts. Both of these findings support the objectives of this dissertation, set out in Section 1.3, providing evidence that ML-based pipeline methods, such as APIC, are poised to challenge state-of-the-art IP traffic classification systems in completeness and accuracy.

7.2.2 Network-Based Anomaly Detection (NBAD)

NBAD is a task closely related to IP traffic classification, where anomalous traffic flows on an IP network need to be identified and managed. APIC was tested as a viable candidate for this task in Chapter 5, where the method was found capable of discriminating between normal and attack (anomalous) traffic in both static and live data sets.

According to Tsai and Lin (2010), IP-based anomaly detection systems are often plagued with high rates of false alarms. The case study presented

in Chapter 5 showed that using APIC, false alarm rates were reduced to 0.08 percent, while maintaining accuracy and detection rates of 99.48 percent and 99.99 percent on a publicly-available, static data set. Comparably, the best scoring, manually-designed system constructed specifically for this task achieved accuracy, detection and false alarm rates of 99.96, 99.99 and 0.02 percent, respectively. These results showed that the APIC method was capable of automatically evolving efficient classifiers where accuracy, detection and false positive rates rivalled those of the best scoring classifier, designed specifically for this task by human experts. The remaining six comparable works achieved false alarms significantly greater than APIC, between 2.2 percent and 33.70 percent. Most noteworthy, the results achieved by APIC were achieved with very little human-design input compared to the other systems. Furthermore, the APIC method was designed as a broadly applicable, general method capable of exhibiting efficacy in a wide range of classification tasks, while the comparable methods were designed specifically for the purpose of discriminating between normal and attack flows in a static data set.

The second part of Chapter 5 evaluated the efficacy of APIC in profiling and detecting anomalies on a live enterprise network. Using statistics inferred from Netflow data records, the method was found capable of detecting anomalous (attack) traffic flows with accuracy, detection and false alarm rates of 99.90 percent, 99.89 percent and 0.096 percent, respectively. The accuracy and detection rates achieved by APIC exceeded those of comparable systems (Table 5.30), while the false alarm rate matched those of the best scoring methods.

The experiments performed in Chapter 5 provide further evidence to support the research objectives of this dissertation (Section 1.3), demonstrating the efficacy of APIC in a broad range of tasks. Interestingly, anomaly detection is not limited to computer networks, having already been applied in a number of domains, including intrusion detection (Garcia-Teodoro et al., 2009), fraud detection (Phua et al., 2010), fault detection (Idé and Kashima, 2004), system health monitoring (Niu et al., 2011) and event detection in sensor networks (Zhang et al., 2010). The design of APIC allows it to extend beyond anomaly detection in IP networks, where application in other domains is the subject of future work.

7.2.3 Handwritten Digit Recognition

The final case study of this dissertation was chosen to demonstrate the broad applicability of the APIC method. For this, the method was tested in the domain of *Computer Vision* (CV), a discipline attempting to analyse and understand scenes of the real world (Klette, 2014). Specifically, the task of handwritten digit recognition was chosen, where the APIC method evolved less-complex (higher-performance) image recognition classifiers to identify handwritten digits using the popular MNIST data set.

In Chapter 6, the APIC method developed *Convolutional Neural Network* (CNN) topologies to recognise instances of previously unseen handwritten digits from the MNIST data set. Comparable methods, including the current state-of-the-art method for MNIST classification, used complex models designed by human experts. These manually-designed models used heuristic values for the feature extraction and classification model architecture, resulting in complex, computationally-expensive classifiers. The APIC method was shown to improve the efficiency (performance) of classifiers for this task, by determining a more efficient feature extraction process and *Multi-layer Perceptron* (MLP) architecture through evolutionary processes.

This case study demonstrated that APIC was both capable of developing classifiers across a broad range of tasks across multiple domains, and able to evolve more than *Topology and Weight Evolving Neural Network* (TWEANN) classifiers, tested in Chapters 4 and 5. Using novel, intuitive mutation functions (algorithm 3), candidate solutions of variable complexity were evolved automatically. The accuracy achieved by the best scoring CNN rivalled scores of comparable methods, using a more efficient classifier. This was possible as APIC not only developed an optimised MLP classifier (Section 3.3), but also searched for the best kernel matrices and convolution design for the feature extraction component. This process was performed using the APIC feature selection process (Section 3.1), controlled by EAs. Kernel selection, according to Huang and Wang (2006), has a profound impact on accuracy achieved by classifiers. Contrary to similar methods, where human experts used heuristic values to develop complex models, the APIC method focused on selecting the best kernel parameters and features to describe each image, leading to more efficient

representations, resulting in less-complex classifiers that performed well compared to the more complex, manually-designed classifiers. The state-of-the-art method for identifying digits in the MNIST data set used a boosting strategy, incorporating the results of multiple complex classifiers to make a classification determination. This boosting strategy, and the development of a classifier ensemble could also benefit solutions provided by APIC. The implementation of a boosting strategy for APIC-evolved classifiers will be investigated in future works.

7.3 Future Work

The APIC general method has already demonstrated efficacy in a broad range of tasks, across a variety of domains. In addition to IP traffic classification (Chapter 4), NBAD (Chapter 5) and handwritten digit recognition (Chapter 6), many other tasks exist that will benefit from automated phenotype development using ML-based pipeline methods, such as APIC. These classification tasks include those associated with education, finance, medical and other technology fields.

The APIC method was designed as an ML pipeline method, specifically for removing dependence on human experts, improving the application of ML-based classification in a broad range of tasks. The method is modular, where specific algorithms may be substituted with others as the task dictates. For example, Bayesian Optimisation algorithms may be substituted in favour of EAs, or *Support Vector Machines* (SVM) may be substituted as the classifier model, instead of an MLP or TWEANN. The possibilities for these substitutions are vast, where further research is required to ascertain the benefits arising from these in various classification tasks. Another area of research might be to automatically evaluate algorithm substitutions, further reducing human-design input when implementing APIC in each task.

As the availability of compute power continues to increase, so the search space provided to APIC for each task can also be increased. This could be simply adjusting the search space of each algorithm, or alternatively, through a "complexification" (Stanley and Miikkulainen, 2002) process, where simplistic networks are expanded until a better-performing, more

complex network is discovered. The implication of such processes has yet to be tested and is subsequently subject to future work.

Finally, the concept of "boosting", where classifier ensembles are assembled to make a classification, is an interesting direction for the future of APIC. In this dissertation, the best performing classifier was used to make a classification determination, however the final population of each classifier production process (Section 3.3) contains a number of high-scoring candidate solutions. By including more of these models when classifying datum, additional aspects will be considered, leading to a classification process that generalises well. A boosting strategy using candidate solutions evolved by APIC has yet to be tested and will subsequently also be the subject of future research.

7.4 Conclusion

The APIC method has shown promise in a broad range of classification tasks, removing the dependency on human-design input in both related and disparate domains. None of the similar methods considered in this dissertation demonstrated the same degree of automation, nor were they capable of extending efficacy beyond the task for which they were originally designed. The APIC method was demonstrated to automatically perform feature selection, pattern identification and classifier production (model development) tasks, producing high-performing, accurate classifiers protected from innovation. Although in this dissertation, APIC was used to evolve ANN-based models, it is a general method that is suitable for evolving almost any structured classifier. It is anticipated that through future research, more automation - including the selection of algorithmic components for each task and the development of boosting models - will further increase the benefits already enjoyed by application of the method.

Chapter 8

Conclusion

In this dissertation, the *Automated Pattern Identification and Classification* (APIC) method for automatically evolving high-performance classifiers to solve complex classification problems was presented and evaluated. This chapter summarises the contributions of the dissertation, providing evidence that the APIC method improves the efficiency, accuracy, completeness and pervasiveness of *Machine Learning* (ML) in a broad range of classification tasks.

8.1 Research Contributions

A number of artefacts were produced in this dissertation - the most significant being a new method for automatically developing high-performance, accurate classifiers for a broad range of applications in a diverse set of scientific domains. The method promotes reducing the human input required to develop classifiers for each task, by incorporating feature selection, pattern identification and classifier production processes into a single ML-pipeline method. The combination of each of these processes into a single, general method is a novel contribution of this dissertation. The ability of the method to evolve its own classifiers protects against innovation, remaining effective by developing new classifiers automatically as new patterns are first detected.

The ability to automatically develop accurate, portable classifiers is a major contribution to the field of *Internet Protocol* (IP) traffic classification, where most other classification systems were shown in Chapter 4 to be heavily

dependent on human-design input. The APIC method allowed new classifiers to be developed and deployed in near real-time, providing a classification system that surpasses other comparable systems in *completeness*. The ability to select an optimal feature subset, identify patterns in a mixed, noisy data set and automatically produce portable, accurate classifiers for IP traffic classification tasks are each contributions to the field of IP networking. The application of *Topology and Weight Evolving Neural Networks* (TWEANN) networks as IP traffic classifiers is another novel contribution of this research.

In Chapter 5, APIC's efficacy as a *Network-Based Anomaly Detection* (NBAD) system was tested, where the method showed great promise detecting anomalous (attack) traffic in both static and dynamic (live) data sets. For these tasks, APIC evolved custom TWEANN classifiers, whose application in NBAD systems is also a unique contribution of this research. A single method capable of both identifying underlying application protocols in IP traffic and the detecting of anomalous (attack) traffic in IP flows is an important contribution to the field of IP networking, promoting increased control of today's complex, evolving networks. Only the APIC method was found capable of generalising between both of these tasks without requiring substantial core changes or significant design input by human experts.

Not only did the APIC method demonstrate efficacy by extending beyond a single task in the IP networking domain, but it also demonstrated the capability of addressing complex classification tasks in the diverse domain of computer vision. In Chapter 6, the method evolved both the feature extraction and classifier components of *Convolutional Neural Network* (CNN) classifiers, demonstrating that more efficient (higher-performance) classifiers were achievable by increasing focus on the features selected for these tasks. Using novel mutation processes (algorithm 3), the complexity of each CNN phenotype was adjusted automatically, until the best scoring classifier combination was found. This is contrary to comparative methods, where the structure of each phenotype was configured heuristically using substantially more complex designs. Both the custom mutation processes and evolutionary processes implemented by APIC are novel contributions of this dissertation.

The APIC method is an important contribution toward the automated development of efficient, accurate classifiers for addressing a broad range of classification tasks, as it makes no assumptions regarding the provided inputs (feature set) or the final phenotype (solution). Instead, the method automatically discovers the best feature set, identifies distinct patterns within the data set and develops optimised models in unison to determine the best configuration for each of these processes. APIC can, therefore, be generalised to develop solutions for a broad range of current and future classification tasks with little human-design input compared to current methods. The APIC method is thus a general methodology for automatically developing efficient and accurate classifiers for complex classification tasks in a variety of domains, characterised by mixed, noisy data sets.

Appendix A

Publications

This section provides a list of publications that were contributed to the scientific community in support of the method proposed by this dissertation.

R.G. Goss and G.S. Nitschke. Automated Pattern Identification and Classification: Anomaly Detection Case Study. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17). ACM, New York, NY, USA, 59-60, 2017

A short paper highlighting the results achieved by APIC in the task of anomaly detection on IP networks (Chapter 5).

R.G. Goss and G.S. Nitschke, Automating Network Protocol Identification, in Biju, I and Nauman, I. Case Studies in Intelligent Computing Achievements and Trends, 109-123, Taylor and Francis, 2014

A book chapter presenting a completely automated pipeline method, referred to herein as the DPCS method, and it's success in automating IP traffic classification. The DPCS method was later rebranded as APIC, after successful testing in more general contexts (outside of IP traffic classification).

R.G. Goss and G.S. Nitschke, Network Protocol Identification Ensemble with EA Optimization, in Proceedings of the Fifteenth Annual Conference Companion on Genetic and Evolutionary Computation Conference Companion, 1735-1736, ACM, 2013

A paper describing the application of GAs for automatically finding the most optimal hyper-parameters for tuning the k-means clustering algorithm. The identified clusters are used as annotated data sets for training ANN classifiers to identify future instances of each protocol.

R.G. Goss and G.S.Nitschke, Automated Network Application Classification : A Competitive Learning Approach, in Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence for Communication Systems and Networks, 45-52, IEEE, 2013

A paper describing the application of a hierarchical self-organising map to cluster and automatically annotate IP traffic flows.

R.G. Goss and R.A. Botha, Establishing Discernible Flow Characteristics for Accurate, Real-Time Network Protocol Identification, in Proceedings of the 2012 9th International Network Conference, 25-34, 2012

A paper exploring the combination of two previously distinct approaches to IP traffic classification: Statistical Analysis and Deep Packet Inspection. Experimentation demonstrated classification accuracy of trained models is improved by combining features of each into a single training set.

R.G. Goss and R.A. Botha, Traffic Management in Next Generation Service Provider Networks Are we there yet?, in Proceedings of the 2011 IEEE Information Security South Africa Conference, 2011

This paper evaluated the current state of IP traffic classification in next generation service provider networks, justifying the requirement for more accurate, automated approaches for tracking and managing data exchanges.

R.G. Goss and R.A. Botha, Deep Packet Inspection Fear of the Unknown, in Proceedings of the 2010 IEEE Information Security South Africa Conference, 2010

A paper establishing the effectiveness and limitations of Deep Packet Inspection, a technique that, at the time of publication, was the most progressive traffic management technique deployed by Internet service providers.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

O. Al Jadaan, L. Rajamani, and C. Rao. Improved selection operator for ga. *Journal of Theoretical & Applied Information Technology*, 4(4), 2008.

E. Alpaydin. *Introduction to machine learning*. MIT press, 2014.

R. Alshammari and A. Zincir-Heywood. A flow based approach for ssh traffic detection. In *Proceedings of the IEEE International Conference on System, Man and Cybernetics*, pages 296–301, Montreal, Que, 2007. IEEE Computer Society.

R. Alshammari and A. Zincir-Heywood. Investigating two different approaches for encrypted traffic classification. In *Sixth Annual Conference on Privacy, Security and Trust*, pages 156 – 166. IEEE Computer Society, 2008.

R. Alshammari and A. Zincir-Heywood. Machine learning based encrypted traffic classification: Identifying ssh and skype. In *Computational Intelligence for Security and Defence Applications*, pages 1–8. IEEE, CISDA, 2009.

R. Alshammari, P. I. Lichodzijewski, M. Heywood, and A. N. Zincir-Heywood. Classifying ssh encrypted traffic with minimum packet header features using genetic programming. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2539–2546. ACM, 2009a.

R. Alshammari, A. Zincir-Heywood, and A. Farrag. Performance comparison of four rule sets: An example for encrypted traffic classification. In *World Congress on Privacy, Security, Trust and the Management of e-Business*, pages 21–28. The IEEE Computer Society, 2009b.

M. Amini and R. Jalili. Network-based intrusion detection using unsupervised adaptive resonance theory (art). In *Proceedings of the 4th Conference on Engineering of Intelligent Systems (EIS 2004), Madeira, Portugal*, 2004.

I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

T. Auld, A. Moore, and S. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1), January 2007.

T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

T. Bäck, D. B. Fogel, and Z. Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*, volume 1. CRC Press, 2000.

L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6): 1554–1563, 1966.

D. Belson. Akamai's state of the internet q2 2015. 2015. URL `https://www.stateoftheinternet.com/downloads/pdfs/2015-cloud-security-report-q2.pdf`.

Y. Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305, 2012.

J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, et al. Algorithms for hyper-parameter optimization. In *NIPS*, volume 24, pages 2546–2554, 2011.

L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT Conference*, page 6. ACM, 2006.

W. Bhaya and M. E. Manaa. A proactive ddos attack detection approach using data mining cluster analysis. *Journal of Next Generation Information Technology*, 5(4):36, 2014.

M. H. Bhuyan, H. J. Kashyap, D. K. Bhattacharyya, and J. K. Kalita. Detecting distributed denial of service attacks: methods, tools and future directions. *The Computer Journal*, page bxt031, 2013.

T. Bourke. *Server load balancing.* " O'Reilly Media, Inc.", 2001.

T. Bujlow, V. Carela-Español, and P. Barlet-Ros. Extended independent comparison of popular deep packet inspection (dpi) tools for traffic classification. Technical report, Universitat Politècnica de Catalunya, 2014.

T. Bujlow, V. Carela-Español, and P. Barlet-Ros. Independent comparison of popular dpi tools for traffic classification. *Computer Networks*, 76:75–89, 2015.

J. Chambers, W. Cleveland, B. Kleiner, and P. Tukey. Graphical methods for data analysis. wadsworth int'l. *Group, Belmont, CA*, 1983.

V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: a bayesian classification system. In *Readings in knowledge acquisition and learning*, pages 431–441. Morgan Kaufmann Publishers Inc., 1993.

C.-L. Chen. A new detection method for distributed denial-of-service attack traffic based on statistical test. *Journel of Universal Computer Science*, 15(2):488–504, 2009.

Y. Chen, L. Dai, and X.-Q. Cheng. Gats-c4. 5: an algorithm for optimizing features in flow classification. In *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, pages 466–470. IEEE, 2008.

D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.

S. Darvishi, M. C. Ridding, D. Abbott, and M. Baumert. Investigation of the trade-off between time window length, classifier update rate and classification accuracy for restorative brain-computer interfaces. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 1567–1570. IEEE, 2013.

M. Dash and H. Liu. Feature selection for classification. *Intelligent data analysis*, 1(1):131–156, 1997.

W. De Donato, A. Pescape, and A. Dainotti. Traffic identification engine: an open platform for traffic classification. *Network, IEEE*, 28(2):56–64, 2014.

K. Deep and H. Mebrahtu. Combined mutation operators of genetic algorithm for the travelling salesman problem. *International Journal of Combinatorial Optimization Problems & Informatics*, 2(3), 2011.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, pages 1–38, 1977.

P. Dorfinger. Real-time detection of encrypted traffic based on entropy estimation. Master's thesis, Salzburg University of Applied Sciences, August 2010.

H. Drucker, R. Schapire, and P. Simard. Improving performance in neural networks using a boosting algorithm. *Advances in neural information processing systems*, pages 42–42, 1993.

A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer, 2003. ISBN 3540401849.

A. E. Eiben and M. Schoenauer. Evolutionary computing. *Information Processing Letters*, 82(1):1–6, 2002.

J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286. ACM, 2006.

M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery in Databases*, volume 96, pages 226–231, 1996.

T. Evangelos. Multi-criteria decision making methods: a comparative study. *Netherland: Kluwer Academic Publication*, 2000.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874, 2008.

F. Ferri, J. Inesta, A. Amin, and P. Pudil. *Advances in Pattern Recognition: Joint IAPR International Workshops SSPR 2000 and SPR 2000 Alicante, Spain, August 30 - September 1, 2000 Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000. ISBN 9783540679462. URL `https://books.google.co.za/books?id=6HhD5ZX990QC`.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

D. B. Fogel. The advantages of evolutionary computation. In *BCEC*, pages 1–11. Citeseer, 1997.

G. M. Foody. Status of land cover classification accuracy assessment. *Remote sensing of environment*, 80(1):185–201, 2002.

D. Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

Y. Freund and R. E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, Berlin Heidelberg, 1995.

P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

F. Gargiulo, L. Kuncheva, and C. Sansone. Network protocol verification by a classifier selection ensemble. In *Multiple Classifier Systems*, pages 314–323. Springer-Verlag, Berlin, Heidelberg, 2009.

M. Gebski, A. Penev, and R. Wong. Protocol identification of encrypted network traffic. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 957–960. IEEE Computer Society, 2006.

F. Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

F. Glover. Tabu searchpart ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

J. Goodman and D. Heckerman. Fighting spam with statistics. *Significance*, 1(2):69–72, 2004.

R. Goss and R. Botha. Traffic flow management in next generation service provider networks are we there yet? In *Information Security South Africa (ISSA), 2011*, pages 1–6. IEEE, 2011.

R. Goss and R. Botha. Establishing discernible flow characteristics for accurate, real-time network protocol identification. In *Proceedings of the 2012 International Network Conference (INC2012)*, pages 25–34, 2012.

R. Goss and G. Nitschke. Automated network application classification: A competitive learning approach. In *In Proceedings of the IEEE Symposium Series on Computational Intelligence (IEEE SSCI 2013)*, 2013a.

R. Goss and G. Nitschke. Network protocol identification ensemble with ea optimization. In *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2013)*, 2013b.

R. Goss and G. Nitschke. Automating network protocol identification. In B. Issac and N. Israr, editors, *Case Studies in Intelligent Computing: Achievements and Trends*. CRC Press, Cleveland, Ohio, 2014.

B. Graham. Fractional max-pooling. *CoRR*, abs/1412.6071, 2014. URL `http://arxiv.org/abs/1412.6071`.

S. S. Greene. *Security policies and procedures*. New Jersey: Pearson Education, 2006.

P. Gregory. *CISSP guide to security essentials*. Cengage Learning, 2014.

S. Grossberg. Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, 37:1–47, 2013.

K. Gurney. *An introduction to neural networks*. CRC press, 2003.

P. Harrington. *Machine Learning in Action*. Manning Publications Co., 2012.

S. Hendrickson. Getting started with hadoop with amazons elastic mapreduce. *EMR*, (1/43), 2010.

T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *Communications Surveys & Tutorials, IEEE*, 16 (4):2037–2064, 2014.

C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al. A practical guide to support vector classification. Technical report, 2003. URL `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`.

B. Hu and Y. Shen. Machine learning based network traffic classification: A survey. *Journal of Information and Computational Science*, 9(11):3161–3170, October 2012.

C.-L. Huang and C.-J. Wang. A ga-based feature selection and parameters optimization for support vector machines. *Expert Systems with applications*, 31(2):231–240, 2006.

K. Huang and D. Zhang. A byte-filtered string matching algorithm for fast deep packet inspection. In *The Nineth International Conference for Young Computer Scientists*, pages 2073 – 2078. IEEE Computer Society, 2008.

T. Idé and H. Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 440–449. ACM, 2004.

R. Jalili, F. Imani-Mehr, M. Amini, and H. R. Shahriari. Detection of distributed denial of service attacks using statistical pre-processor and unsupervised neural networks. In *Information Security Practice and Experience*, pages 192–203. Springer, 2005.

T. Jansen. *Analyzing Evolutionary Algorithms: The Computer Science Perspective*. Springer, 2013.

S. Juma, Z. Muda, and W. Yassin. Reducing false alarm using hybrid intrusion detection based on x-means clustering and random forest classification. *Journal of Theoretical & Applied Information Technology*, 68(2), 2014.

Y. Kassahun, M. Edgington, J. H. Metzen, G. Sommer, and F. Kirchner. A common genetic encoding for both direct and indirect encodings of networks. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1029–1036. ACM, 2007.

M. J. Kearns and U. V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0-262-11193-4.

H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best

practices. In *Proceedings of the 2008 ACM CoNEXT conference*, page 11. ACM, 2008.

R. Klette. *Concise computer vision*. Springer, 2014.

T. Kohonen. The self-organizing map. In *Proceedings of the IEEE*, volume 78, pages 1464–1480, 1990.

M. A. Kon and L. Plaskota. Complexity of predictive neural networks. In *Unifying Themes in Complex Systems*, pages 181–191. Springer, 2006.

V. V. Kumar, A. Srikrishna, B. R. Babu, and M. R. Mani. Classification and recognition of handwritten digits by using mathematical morphology. *Sadhana*, 35(4):419–426, 2010.

R. Leander. *Building application servers*. Number 21. Cambridge University Press, 2000.

Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, 261:276, 1995.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

W. Li. Using genetic algorithm for network intrusion detection. *Proceedings of the United States Department of Energy Cyber Security Group*, pages 1–8, 2004.

Z. Li, R. Yuan, and X. Guan. Accurate classification of the internet traffic based on the svm method. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 1373–1378. IEEE, 2007.

H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

B. G. Lindsay. Mixture models: theory, geometry and applications. In *NSF-CBMS regional conference series in probability and statistics*, pages i–163. JSTOR, 1995.

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967a.

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967b.

G. Maiolini, A. Baiocchi, A. Iacovazzi, and A. Rizzi. Real time identification of ssh encrypted application flows by using cluster analysis techniques. In *NETWORKING 2009*, pages 182–194. Springer, 2009.

A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.

A. Mcgregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. *Passive and Active Network Measurement*, pages 205–214, 2004.

M. Melanie. An introduction to genetic algorithms. 1999.

R. Miikkulainen. Evolving neural networks. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pages 2441–2460. ACM, 2010.

B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

J. Mirković, G. Prier, and P. Reiher. Source-end ddos defense. In *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pages 171–178. IEEE, 2003.

T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

J. Mockus. *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media, 2012.

R. K. Mohapatra, B. Majhi, and S. K. Jena. Classification performance analysis of mnist dataset utilizing a multi-resolution technique. In *Computing, Communication and Security (ICCCS), 2015 International Conference on*, pages 1–5. IEEE, 2015.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT Press, 2012.

A. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Network Measurement*, volume 3431, pages 41–54. Springer, 2005.

D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)*, 24(2):115–139, 2006.

Z. Muda, W. Yassin, M. N. Sulaiman, and N. I. Udzir. Intrusion detection based on k-means clustering and oner classification. In *Information Assurance and Security (IAS), 2011 7th International Conference on*, pages 192–197. IEEE, 2011.

Z. Nascimento, D. Sadok, and S. Fernandes. A hybrid model for network traffic identification based on association rules and self-organizing maps (som). In *The Nineth International Conference on Networking and Services (ICNS2013)*, pages 213–219, 2013.

T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

G. Niu, S. Singh, S. W. Holland, and M. Pecht. Health monitoring of electronic products based on mahalanobis distance and weibull decision metrics. *Microelectronics Reliability*, 51(2):279–284, 2011.

W. D. Nordhaus. The progress of computing. 2001.

S. H. Oh and W. S. Lee. An anomaly intrusion detection method by clustering normal user behavior. *Computers & Security*, 22(7):596–612, 2003.

J. Park, H.-R. Tyan, and C.-C. Kuo. Ga-based internet traffic classification technique for qos provisioning. In *Intelligent Information Hiding and Multimedia Signal Processing, 2006. IIH-MSP'06. International Conference on*, pages 251–254. IEEE, 2006.

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

D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys (CSUR)*, 39(1):3, 2007.

C. Phua, V. Lee, K. Smith, and R. Gayler. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*, 2010.

J. Quinlan. C4. 5: Programs for machine learning. *Morgan Kaufmann Series in Machine Learning*, 1993.

J. R. Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

T. C. Redman. *Data driven: profiting from your most important business asset*. Harvard Business Press, 2008.

L. Reyzin and R. E. Schapire. How boosting the margin can also boost classifier complexity. In *Proceedings of the 23rd international conference on Machine learning*, pages 753–760. ACM, 2006.

O. Roeva, S. Fidanova, and M. Paprzycki. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*, pages 371–376. IEEE, 2013.

S. Rogers and M. Girolami. *A first course in machine learning*. CRC Press, 2016.

P. Rousseeuw. Silhouettes: A graphical aid to the interpretation and valudation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

A. Rubin. *Statistics for evidence-based practice and evaluation*. Cengage Learning, 2012.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning representations by back-propagating errors*. MIT Press, Cambridge, MA, USA, 1988.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.

M. Sakamoto, T. D. Marcotte, A. Umlauf, D. Franklin Jr, R. K. Heaton, R. J. Ellis, S. Letendre, T. Alexander, J. McCutchan, E. E. Morgan, et al. Concurrent classification accuracy of the hiv dementia scale for hiv-associated neurocognitive disorders in the charter cohort. *Journal of acquired immune deficiency syndromes (1999)*, 62(1):36–42, 2013.

M. Schoenauer. Evolutionary computation published in control and cybernetics 26 (3) pp 307-338 marc schoenauer* and zbigniew michalewiczu. *Control and Cybernetics*, 26(3):307–338, 1997.

S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*, pages 512–521. ACM, 2004.

C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

G. I. Sher. Discover & explore neural network (dxnn) platform, a modular tweann. *arXiv preprint arXiv:1008.2412*, 2010.

G. I. Sher. *Handbook of neuroevolution through erlang*. Springer, 2012.

A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, 31(3):357–374, 2012.

J. Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer, 2005.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

S. V. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

J. M. Stewart. *Network Security, Firewalls and VPNs*. Jones & Bartlett Publishers, 2013.

G. Szabo, I. Szabo, and D. Orincsay. Accurate traffic classification. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–8. IEEE, June 2007.

P. Szor. *The art of computer virus research and defense*. Pearson Education, 2005.

C.-F. Tsai and C.-Y. Lin. A triangle area based nearest neighbors approach to intrusion detection. *Pattern Recognition*, 43(1):222–229, 2010.

H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *Tools with Artificial Intelligence, 1992. TAI'92, Proceedings., Fourth International Conference on*, pages 200–203. IEEE, 1992.

C. William et al. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

C. V. Wright, F. Monrose, and G. M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 7:2745–2769, 2006.

X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9): 1423–1447, 1999.

W. Yassin, N. I. Udzir, Z. Muda, and M. N. Sulaiman. Anomaly-based intrusion detection through k-means clustering and naives bayes classification. In *Proceedings of the 4th International Conference on Computing and Informatics (ICOCI), Sarawak, Malaysia*, pages 298–303, 2013.

S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 250–257. IEEE, 2005.

J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. Vasilakos. An effective network traffic classification method with unknown flow detection. *Transactions on Network and Service Management*, 10(2), June 2013.

M. Zhang, W. John, K. Claffy, and N. Brownlee. State of the art in traffic classification: A research review. In *PAM Student Workshop*, 2009a.

Y. Zhang, Z. Li, S. Mei, and C. Fu. Session-based tunnel scheduling model in multi-link aggregate IPSec VPN. In *Third International Conference on Multimedia and Ubiquitous Engineering*, 2009b.

Y. Zhang, N. Meratnia, and P. Havinga. Outlier detection techniques for wireless sensor networks: A survey. *Communications Surveys & Tutorials, IEEE*, 12(2):159–170, 2010.