2022-12

MACHINE LEARNING STATISTICAL DETECTION OF ANOMALIES USING NETFLOW RECORDS

Putman, Zachary W.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

MACHINE LEARNING STATISTICAL DETECTION OF ANOMALIES USING NETFLOW RECORDS

by

Zachary W. Putman

December 2022

Thesis Advisor:	Chad A. Bollmann
Second Reader:	George W. Dinolt

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2022	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
MACHINE LEARNING STATISTICAL DETECTION OF ANOMALIES USING NETFLOW RECORDS

5. FUNDING NUMBERS
REPQN

6. AUTHOR(S) Zachary W. Putman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NCWDG

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

NetFlow is a network protocol system that is used to represent an overall summary of computer network conversations. A NetFlow record can convert previously captured packet captures or obtain NetFlow session data in real time. This research examines the use of machine-learning techniques to identify anomalies in NetFlow records and classify malware behavior for further investigation. The intent is to identify low-cost solutions leveraging open-source software capable of deployment on computer hardware of currently in-use data networks.

This work seeks to determine whether expert selection of features can improve machine-learning detection algorithm performance and evaluate the trade-offs associated with eliminating redundant or excessive numbers of features. We identify the Random Forest algorithm as the strongest single algorithm across three of four metrics, with our chosen NetFlow features cutting the testing and training times in half while incurring minor reductions in two metrics. The experiment demonstrates that the chosen NetFlow features are sufficiently discriminative to detect attacks with a success rate higher than 94%.

14. SUBJECT TERMS statistical detection, machine learning, NetFlow, denial of service, CRISP-DM	**15. NUMBER OF PAGES** 87
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

MACHINE LEARNING STATISTICAL DETECTION OF ANOMALIES USING NETFLOW RECORDS

Zachary W. Putman
Lieutenant, United States Navy
BS, Georgia Institute of Technology, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2022

Approved by: Chad A. Bollmann
Advisor

George W. Dinolt
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

NetFlow is a network protocol system that is used to represent an overall summary of computer network conversations. A NetFlow record can convert previously captured packet captures or obtain NetFlow session data in real time. This research examines the use of machine-learning techniques to identify anomalies in NetFlow records and classify malware behavior for further investigation. The intent is to identify low-cost solutions leveraging open-source software capable of deployment on computer hardware of currently in-use data networks.

This work seeks to determine whether expert selection of features can improve machine-learning detection algorithm performance and evaluate the trade-offs associated with eliminating redundant or excessive numbers of features. We identify the Random Forest algorithm as the strongest single algorithm across three of four metrics, with our chosen NetFlow features cutting the testing and training times in half while incurring minor reductions in two metrics. The experiment demonstrates that the chosen NetFlow features are sufficiently discriminative to detect attacks with a success rate higher than 94%.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

List of Acronyms and Abbreviations

CSOC	Cybersecurity Operations Center
CRISP-DM	Cross-Industry Process for Data Mining
csv	comma separated value
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DR	Detection Rate
FAR	False alarm rate
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
IPS	Intrusion Protection System
IRC	Internet Relay Chat
K-NN	K-Nearest Neighbor

L7	Application Layer 7
MCC	Matthews Correlation Coefficient
ML	Machine Learning
NB	Naïve Bayes
NPS	Naval Postgraduate School
NHEITC	Naval Higher Education Information Technology Consortium
OSI	Open Systems Interconnection
OWASP	Open Web Application Security Project
pcap	packet capture
RF	Random Forest
SVM	Support Vector Machine
SQL	Structured Query Language
TN	True Negative
TP	True Positive
WAF	Web Application Firewall
Weka	Waikato Environment for Knowledge Analysis
XSS	Cross Site Scripting

Acknowledgments

First and foremost, I would like to thank my wife, Victoria, for her support. Taking care of three kids under four years old is hard work, and you fill all of our lives with joy. Thank you to all of the professors here at the Naval Postgraduate School. Your passion for the subject encouraged me to delve deeper into topics than I could have ever imagined. Finally, I would like to thank my advisor, Professor Chad Bollmann, for his guidance and support throughout this process.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Overview

This thesis focuses on improving network security, through open source software compatible with currently fielded equipment, against Denial of Service (DoS) attacks targeting Department of Defense networks. We identified suitable, publicly available datasets that simulate DoS and other application layer attacks. We then selected and compared the effectiveness of default and adapted machine learning algorithms to improve the ability to differentiate attacks from benign traffic.

1.2 Motivation and Objective

DoS attacks are a threat against two pillars of cybersecurity: integrity of data and availability of service. Attacks are accomplished by preventing legitimate user access through exhausting a given resource or overloading network infrastructure. DoS attacks are used in a variety of ways: for reconnaissance of network defenses, as a weapon to prevent legitimate user access, or a distraction to divert resources away from other cyberattacks. While these attacks have been around for decades, DoS attacks will remain very common and are part of cybercrime-as-a-service [1].

The Naval Higher Education Information Technology Consortium (NHEITC) is responsible for providing cybersecurity to the four higher learning schools, Naval Postgraduate School (NPS), Defense Language Institute, Naval War College, and the U.S. Naval Academy. Each school is responsible for defending data assets, isolating intrusion, and detecting attacks. We chose to target Application Layer 7 (L7) attacks is because at NPS, most attacks are conducted by cybercriminals and novice attackers, with application layer attacks being the most common [2]. DoS attacks are a cheap and effective attack vector and, when successful, consume resources and man-hours to determine the scope of the attack and remediate damage caused to internal infrastructure. While there are tools that are effective in stopping attacks, such as a Web Application Firewall (WAF), the cost of services becomes expensive

for the amount of traffic these institutions experience.

The objective of this thesis is to provide one-step detection of DoS attacks using NetFlow features and machine learning algorithms. As seen in Figure 1.1, a Cisco network flow is a summary of packet capture (pcap) data which is intended to provide a high-level view of network traffic. This process is less hardware intensive since administrators can choose specific flow attributes to collect instead of entire packets. Our approach analyzes NetFlow version 9 features to determine the most indicative features of application layer DoS and other web attacks in order to generate the highest accuracy from an array of ML algorithms. We identify these features using baseline datasets that simulate realistic network traffic. We then examine the extent to which feature selection centered on attack characteristics effect computation time and accuracy, as compared to using full feature sets.

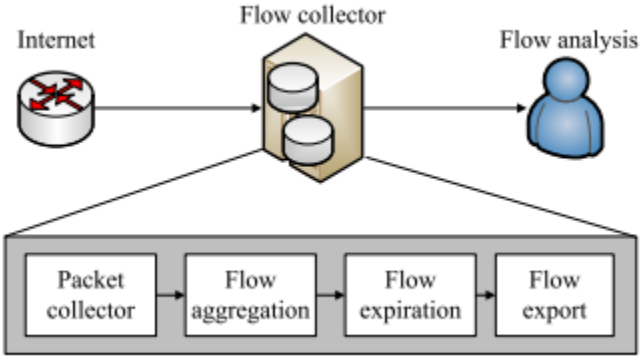


Figure 1.1. Packet Collection to Flow Generation. Source: [3].

1.3 Methodology Overview

To guide our approach, we employed the Cross-Industry Process for Data Mining (CRISP-DM) methodology, as seen in Figure 1.2, as the reference model for our research. The phases of CRISP-DM are:

1. Business understanding: In Chapters 1 and 2, we explore the nature of DoS attacks and how machine learning can be used to classify attacks. We assess some current methods which will define our success criteria in Section 2.1, and select the tools and algorithms we will use in Section 2.3 and 2.4.

2. Data understanding: Additionally, in Chapter 2, we review currently available data sources and describe the datasets that were chosen for the experiment.
3. Data Preparation: In Chapter 3, we describe the process to prepare the dataset for modeling, which consists of the following steps: data selection, data cleaning, data integration, data formatting, and how data will be split for training and testing.
4. Modeling: In Chapter 4, we apply our selected algorithms to conduct our experiment and record the results.
5. Evaluation: In Section 4.3, we interpret the results based on which algorithms meet the success criteria, and compare algorithm performance using our reduced feature set.
6. Deployment: In Chapter 5, we review our project and discuss future work.

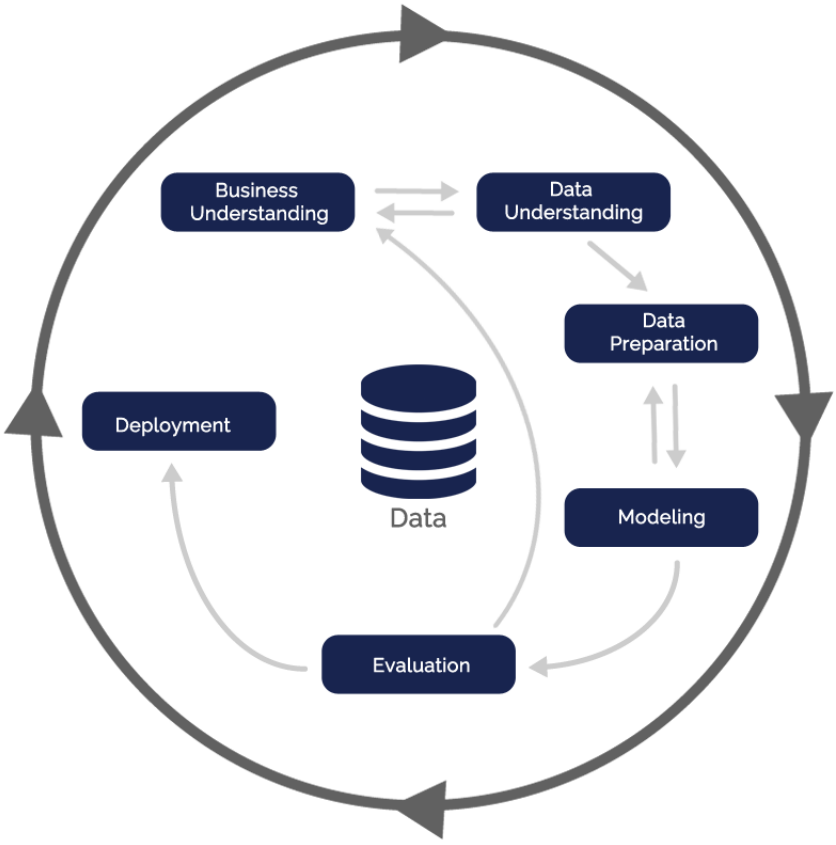


Figure 1.2. The CRISP-DM 6 Stage Life Cycle. Source: [4].

1.4 Overview of Targeted Attacks

For this research we focus on the attacks targeting the seventh layer of the Open Systems Interconnection (OSI) model, known as L7. This layer is responsible for end user interaction with web application services, such as email or web browsing. Some common protocols at this level are Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Internet Relay Chat (IRC), and Domain Name System (DNS). Cyberattacks at this layer target specific applications or vulnerabilities which can occur due to overloading of specific application functions or mis-configuration of servers. We chose to target these attacks because the Open Web Application Security Project (OWASP) Foundation [5] has studied the effectiveness of these attacks and concluded application layer attacks have higher obscurity, higher efficiency, and higher lethality compared to other types of DoS attacks.

DoS attacks can be used to hide other legitimate attacks. While DoS attacks utilize application protocols, other application layer attacks can occur through manipulating databases or utilizing scripts and queries. As such, in this work we also study the impact on detection accuracy of DoS attacks in the presence of other L7 attacks. This work specifically seeks to detect the following types of L7 DoS attacks: HTTP Flood, Slowloris, and R.U.D.Y. To examine the impact other application layer attacks have on detection of these DoS attacks, we also tested detection in the presence of Structured Query Language (SQL) injects, Cross Site Scripting (XSS) attacks, and brute force attacks.

1.4.1 HTTP Flood

HTTP flood attacks are a form of volumetric DoS, where an attacker attempts to overwhelm services with high volumes of malicious traffic. This type of attack is successful when a service is no longer able to respond to other users. The two most common types of HTTP flood attacks are:

1. HTTP GET attacks: These attacks consume server resources by posing as normal users requesting access to a web page. The difficulty of detection comes from the use of standard URL requests, which look no different from normal traffic.
2. HTTP POST attacks: As explained in RFC 7231 [6], "POST requests perform resource-specific processing on requested payloads." These attacks take advantage of the Content-Length field of an HTTP header. Because it is a legitimate request to

a server, resources are set aside for the incoming message.

1.4.2 Slowloris

Slowloris is a L7 cyberattack tool that performs "low and slow" DoS attacks, where an attacker attempts to overwhelm services by maintaining open HTTP connections, as shown in Figure 1.3. Similar to an HTTP GET attack, this attack takes advantage of GET request headers and is effective because it mimics slow internet connectivity and gradually occupies server resources until all resources have been used.

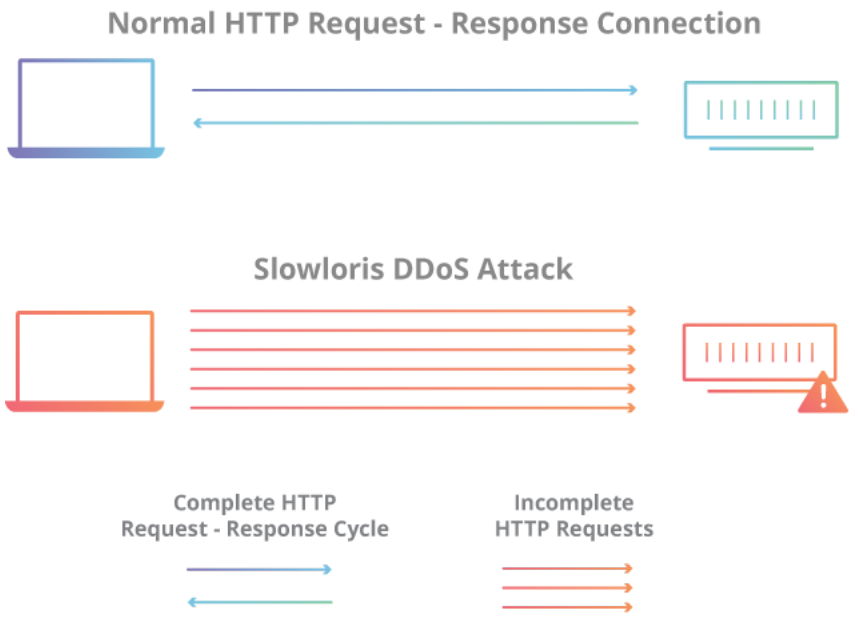


Figure 1.3. Methodology of the Slowloris Attack Tool. Source: [7].

1.4.3 R.U.D.Y Attacks

Similar to Slowloris, the L7 attack tool, "R U Dead Yet?" performs DoS attacks by sending HTTP POST packet requests slowly. Because there is no minimum speed a web server can support, this attack will use up server resources as it waits for requested information of the given form, such as logging in or uploading a document.

1.5 Scope

The scope of this research is constrained to utilizing known application layer DoS attacks to identify attacks using NetFlow records. We chose to use only open source software compatible with currently fielded equipment, with the intent of identifying solutions that are easy to implement and have no additional cost considerations.

1.6 Research Questions

In this work, we examined the following research questions:

1. How has ML been used to classify L7 DoS attacks?
2. Does ML algorithm performance vary when detecting DoS attacks alone vs in the presence of other attack vectors?
3. What features does ML assess as most important for each attack type?
4. Can expert selection be used to eliminate redundant or unnecessary features initially selected by an ML algorithm?
5. What are the advantages and disadvantages of this reduced feature set? How does the reduced feature set compare to the full feature set at detecting attacks/false alarms?
6. Which additional features would lead to the greatest reduction of false alarms from benign traffic?
7. Can expert selection be used to eliminate redundant or unnecessary features initially selected by an ML algorithm?

1.7 Organization

The thesis is organized as follows. Chapter 2 provides a literature review of past research efforts, an overview of machine learning algorithms and publicly available datasets, and discuss the analytical tools we used. Our methodology and experimental design for our work is presented in Chapter 3. The results of the experiment and analysis are found in Chapter 4. Chapter 5 concludes with a summary of our findings and recommendations for future work.

CHAPTER 2:
Literature Review and Machine Learning

This chapter describes the most pertinent prior work from a large pre-existing body of network anomaly detection. We begin with analyzing works that focus only on L7 DoS attacks and then look at research that focus on L7 DoS attacks in the presence of other L7 attacks. Finally, we discuss the benefits and limitations of publicly available benchmark datasets, discuss the factors that went into tool analysis, and provide an overview of the machine learning algorithms that we use to conduct our experiments.

2.1 Literature Review

Since this research focuses on improving network security, it is important to analyze the methodology of previous efforts. An overview of what we believe to be the most pertinent research can be found in the beginning of each subsection, as seen in Table 2.1 and Table 2.2. We first consider how others have utilized NetFlow to detect only DoS attacks.

2.1.1 Single Attack Types

Table 2.1. Summary of Literature Review Based on Only DoS Attacks.

Research Work	Features Used	Detection Approach	Strengths	Limitations
Kemp et al. [8]	Protocol, Packets, Bytes, TCP flags, Initial flags, Session flags, Attributes, Packets/s, Bytes/s, Bytes/packet, Duration, Label	Machine Learning	High accuracy for detection of Slow Read attack traffic, utilizing live web server data	Only tested for simple cases of this attack. Did not include other types of anomalies.
Feng et al. [9]	Bytes/message, Traffic size from IP block, Average behavior interval, Average absolute deviation of behavior interval, Number of messages sent/time, Number of similar received messages, Request consumption, and Ratio of incoming to outgoing traffic	Reinforcement learning	Self-evolving according to interactions with the environment, actively minimizes false positive rates and maximizes true positive rates to prevent attacks.	Initial training based upon new environmental factors heavily impact accuracy.
Beckett et al. [10]	Number of Databases: Opened, Closed, Queried, Commits; Total database query time, average query time per database open, average number of queries per database open	Novel resource consumption sensor and C4.5 decision tree	Can detect sophisticated DDoS attackers targeting large database resources.	Proposed system is only designed for use while under a DDoS attack.

Kemp et al. [8] [11] studied the effectiveness of eight supervised Machine Learning (ML) techniques, focusing on 11 NetFlow features chosen to detect SlowHTTP DoS attacks. Their

experimental design focused on an infected machine targeting a single victim using only a SlowHTTP DoS attack that lasted one hour. They concluded that Random Forest (RF) performed better than other models. Their experiment showed a high accuracy against DoS attacks, but only tested simple cases of the attacks. This work studies similar attack vectors and builds upon it with other L7 attacks.

Feng et al. [9] utilized NetFlow data to mitigate Distributed Denial of Service (DDoS) attacks using reinforcement learning. Rather than using existing public datasets, they constructed a simulation environment and generated attacks using various DDoS tools. They shows that through reinforcement learning, using features designed specifically in his experiment, L7 DDoS attack detection and the mitigation accuracy can be maintained independent of system load. Their method showed a 98.73% DDoS attack detection rate. Our work shows that we can achieve this level of detection through the use of our chosen features, explained more in Chapter 3.

Beckett et al. [10] proposed a new sensor that analyzes net flow data to classify L7 DDoS attacks. The system used 30-second data windows, a decision tree classifier, and empirically chose seven features which focused on database resources. Instead of using a benchmark dataset, they used a test bed with seven days of real traffic logs from a private cloud environment. Their method detected attacks with an overall accuracy of 97.9%, although their team acknowledged higher false positive results would occur in benign-only traffic. We show that the use of our reduced feature set would produce similar overall accuracy under all circumstances.

2.1.2 Multiple Attack types

Recent attacks have shown that DoS and web-based attacks are often coordinated activities [1]. As such, we examine the methodology that researchers have used to detect DoS in the presence of other L7 attacks.

Table 2.2. Summary of Literature Review Based on DoS Attacks in the Presence of Other L7 Attacks.

Research Work	Features Used	Detection Approach	Strengths	Limitations
Zoppi et al. [12]	Varied depending on dataset, but most used features were: Duration of connection, Number of bytes exchanged, Number of packets	Machine Learning	Studied the effectiveness of unsupervised algorithms on zero-day attacks.	Current threat landscape is not reflected in all datasets.
Ahmin et al. [13]	79 features, Full list found in the Appendix	Novel intrusion detection system utilizing hierarchical classification	Proposed IDS detected DoS attacks better than other machine learning methods.	Hierarchical model setup and how the features were used within the model was not explained.
Borisenko et al. [14]	Amount of bytes, Amount of packets, Source IP address, Source IP port, Destination IP address, Destination IP port	Machine learning	Proposed architecture can be used to strengthen cloud defense.	Only tested for simple cases of external and internal attacks.
Ullah et al. [15]	Source IP address, Destination IP address, Destination Port, Protocol, Flow duration, Flow bytes/s, Flow Packets/s, Flow IAT mean, Flow IAT std, Flow IAT max, Flow IAT min, Fwd IAT total, Fwd IAT mean, Subflow Fwd Packets, Subflow Fwd bytes, Subflow Bwd packets, Subflow Bwd bytes	Novel two-level flow detection system using machine learning	Two-level system can enhance security of Internet of Things devices.	Empirical selection of features and criteria used for models selection were not explained.

Zoppi et al. [12] employed the use of 17 unsupervised ML algorithms testing the Information Gain feature selection strategy on 11 publicly available datasets. His research found that regardless of dataset, three features were selected a majority of the time: duration of connection, number of bytes exchanged, and number of packets. While his experiment was not focused on any singular type of attack, 9 of his 11 datasets did include DoS of some kind, which could explain the features found. While not solely focused on the same attacks, we use his method of feature selection to determine common NetFlow features of our studied attacks. Comparing attack types, irrespective of dataset, he concluded an average Matthews Correlation Coefficient (MCC) score of 0.50 and overall accuracy of 80%.

Ahmin et al. [13] proposed a hybrid Intrusion Detection System (IDS) system that used three supervised models to conduct multi-class identification. By utilizing 79 NetFlow features, their team analyzed the effectiveness of hierarchical classification of anomalies against all attacks found in the CIC-IDS 2017 dataset. They concluded their method could classify DoS attacks with high accuracy, greater than 93%, but classification of other application layer attacks was average compared to other models. Our work shows that our reduced feature set would produce similar one-step classification results.

In cloud environments, Borisenko et al. [14] studied the performance of four supervised ML algorithms to detect internal and external application-layer DDoS attack. Rather than using a baseline dataset, Borisenko created two virtual networks and simulated benign traffic and the DDoS attacks. Their research examined the impact of six NetFlow features using data

mining techniques, and determined that a decision tree algorithm performed best with a False Positive Rate (FPR) of 0.05%. Their research concluded that performance was not significantly affected whether the compromised systems which launched the application layer attacks were located within network or outside the network. This makes sense because their features focused on flow identifiers, such as source/destination Internet Protocol (IP) addresses and tested only simple cases of attacks. Our work shows that DoS attacks can be found independent of these features.

In Internet of Things (IoT) environments, Ullah et al. [15] proposed using an anomaly based, two-level flow detection system to detect DDoS attacks and other application layer attacks within IoT networks. Using a benchmark dataset, they empirically selected 17 features. Although their research did not provide a detailed performance breakdown of supervised learning algorithms for either layer one or two, the authors final results achieved an overall classification accuracy of 98.80%. Our work shows similar results using less features.

As seen from the literature review, there are many proposed methods to detect DoS attacks. While some ways are more successful, the selection of dataset, features, and ML algorithm play a significant factor in classification accuracy. We consider our reduced features set a success if it maintains a higher than 94% DoS detection rate, the median of the peer study performance, while keeping false alarms to a minimum. The next section provides a deeper look at the datasets we used.

2.2 Dataset Analysis

Publicly available datasets are a useful tool in determining the usefulness of our experiment. Known as benchmark datasets, they provide a useful comparison tool for researchers to assess their design against other methods. In most cases, these datasets label the type of traffic that is recorded as benign or attack. Labeling facilitates comparing and contrasting the effectiveness of the various algorithms through our chosen metrics.

In ML applications, each data point is commonly called an instance or case. Each case has a set of values associated with it, known as features. The analyzed datasets have the features listed by column, with each case occupying a new row. The number of features depends on the software that is used to record and extract data.

We investigated many different datasets (approximately 11) but selected the most recent and attack specific datasets for further analysis. The following benchmark datasets were examined for use in the experimental design: CTU-13, CIDDS-001, CIDDS-002, CIC-IDS 2017, CSE-CIC-IDS 2018, and CIC-IDS 2019. An overview of these datasets can be found in Figure 2.3. The criteria for use in this experiment were as follows: first, the dataset needed to contain L7 DoS attacks as well as other attacks. Second, as the nature of DoS attacks have changed over the years, only datasets that contained attack profiles currently seen today were used. Finally, the network topology should approximate a real world network topology closely related to what is commonly seen today. Give these criteria, the datasets that were selected are a combination of attacks from CIC-IDS 2017 and CSE-CIC-IDS 2018.

Table 2.3. Overview of Publicly Available Datasets.

Dataset	Creation date	Realistic traffic	Data Labeled	Problem Specific Data?	# of attacks	Diverse Scenarios	PCAP	# of features
CTU-13	2013	Y	Y	N	9	Y	Y	15
CIDDS-001	2017	Y	Y	N	8	Y	Y	14
CIDDS-002	2017	Y	Y	N	15	Y	Y	14
CIC-IDS 2017	2017	Y	Y	Y	7	Y	Y	80
CSE-CIC-IDS 2018	2018	Y	Y	Y	8	Y	Y	80
CIC-DDOS 2019	2019	Y	Y	Y	14	Y	Y	80

2.2.1 CTU-13 (2013)

CTU-13 is a dataset of botnet traffic that was captured by the Czech Technical University, Czech Republic [16]. The dataset consists of botnet traffic mixed with both normal traffic and background traffic in a real network environment. The dataset includes 13 different scenarios of nine different attack combinations. Each scenario was recorded in bidirectional NetFlow format, consisting of 40 features and one of three classification labels. While the dataset includes some application layer attacks, this dataset was not selected because it did not look at L7 DoS attacks.

2.2.2 CIDDS-001 (2017)

CIDDS-001 (Coburg Intrusion Detection Data Set) emulates a small business environment [17]. The dataset consists of four attacks: Port Scan, Ping Scan, HTTP DoS, and Brute Force attacks. Each attack targeted an external server or one of four subnets within their simulated environment. These attacks were executed over the course of 92 scenarios over a four-week period. The data was captured in unidirectional NetFlow format, consisting of ten NetFlow features and four classification labels. The features focus on flow identifiers and flow duration. The limited number of features and attack types led to the exclusion of this dataset.

2.2.3 CIDDS-002 (2017)

CIDDS-002 emulates a small business environment, focusing solely on attacks against the three subnets [17]. The dataset consists of four attacks (Port Scan, Ping Scan, HTTP DoS, and Brute Force) against three virtual networks. These attacks were executed over 43 different scenarios over a two-week time period. The data was captured in unidirectional NetFlow format, consisting of ten NetFlow features and four classification labels. While technically an updated version of the previous dataset, CIDDS-002 has the same limitations of lack of features and attack types.

2.2.4 CIC-IDS 2017

CIC-IDS-2017 is an intrusion detection evaluation dataset that emulates naturalistic, benign background traffic [18]. Eight types of malicious attacks were executed against a virtual network. The data was captured over a period of five days, with each day hosting a combination of attacks. Extraction was done in bi-directional NetFlow format using the publicly available CICFlowMeter software. Seventy-nine features were calculated; a full list of features can be found in the Appendix. Because of the wide variety of attacks, the DoS attack data was included for use in our experiment.

2.2.5 CSE-CIC-IDS 2018

CSE-CIC-IDS 2018 is an Intrusion Protection System (IPS)/IDS dataset that emulates real traffic and is labeled as either benign data or an attack type [18]. Unlike CIC-IDS 2017, this dataset was a joint effort by the Communications Security Establishment and the Canadian

Institute for Cybersecurity to study anomaly detection on the AWS computing platform, seen in Figure 2.1. The entire dataset contains seven different attacks and was extracted using an updated version of CICFlowMeter, resulting in 84 features. From this dataset, we use the full DoS and web attack data.

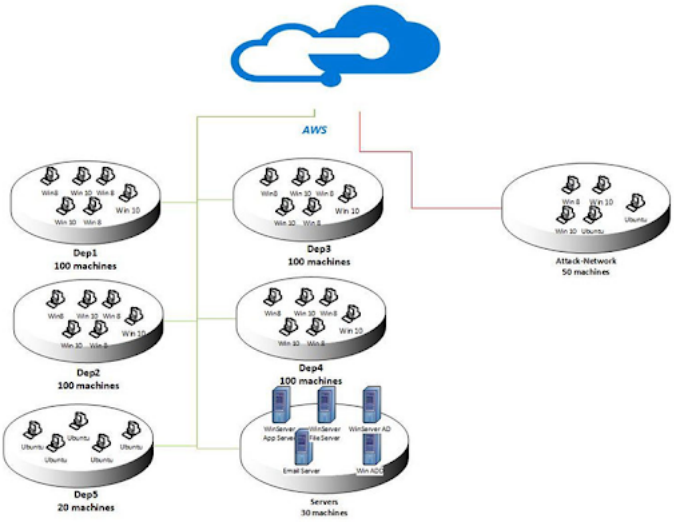


Figure 2.1. Network Topology of CSE-CIC-IDS 2018. Source: [18].

2.2.6 CIC-DDOS 2019

CIC-DDOS 2019 is a DDoS evaluation dataset containing benign data and most common DDoS attacks, mostly focused on reflection and exploitation attacks [19]. The data was captured over a period of two days, recorded in bidirectional NetFlow format, and contains 12 different DDoS attacks. This dataset was not chosen due to the low volume of Web DDoS traffic.

To conclude, there are many datasets that are available for use. While each dataset has its advantages, a common disadvantage is imbalanced data and feature availability. Due to the assumptions each dataset made on how an attack was performed, there is an imbalance between recorded benign traffic and attack traffic. This data imbalance can affect algorithm performance towards the minority class, which in most cases is the attack traffic. The features available for study also vary greatly, as each dataset creator uses different software tools and underlying equations when exporting the flows. However, we blend the DoS attacks from

the selected datasets to show detection is determined by features common to the DoS attack itself, not from where an attack is originating from.

2.3 Tool Comparison

Similarly, while dataset selection is important, the data analysis software choice also plays a critical role for our experiment. Since one of our goals is to use open source software in conjunction with already fielded equipment, the software should be able to: load datasets in comma separated value (csv) format, execute the machine learning algorithms, and provide results in an easily understood manner. Ease of use and simplicity of user interfaces were also considered.

2.3.1 R

R is an open source programming language mainly used for statistical analysis [20]. The R community is active and the ecosystem provides a large amount of machine learning algorithms. R provides the tools needed for data preparation, analysis, and experimental reporting. The disadvantages are a steep learning curve and poor memory management. Because the programs and functions are spread across different packages, R is slower to run than some of the other programs listed.

2.3.2 Python

Python is a computer language that can be used for statistical analysis and machine learning [21]. One of the advantages of Python is that it is an open-source language, so there are a wide variety of libraries and packages that are freely available. The coding is interpreted line by line, leading to longer execution times and large memory usage. With the many databases available for use in Python, it can be hard to choose which packages to use for experimentation.

2.3.3 MATLAB

MATLAB is a programming platform that contains application toolboxes that can perform a variety of machine learning tasks. MATLAB claims it is capable of handling big data and that the processing and extraction techniques allow for iterative tuning on machine learning

algorithms [22]. However, the limited big data packages make this tool difficult to compare results to other findings.

2.3.4 Weka

Waikato Environment for Knowledge Analysis (Weka) is open-source software that contains a library of machine learning algorithms used for data mining [23]. The main advantage is the easy-to-use graphical user interface and the ability to easily compare algorithms. The disadvantage is the amount of memory the program requires to process large datasets. After experimenting with the various tools, Weka was chosen due to the ease of use, short learning curve, and well-designed features. An example Weka graphical user interface is shown in Figure 2.2.

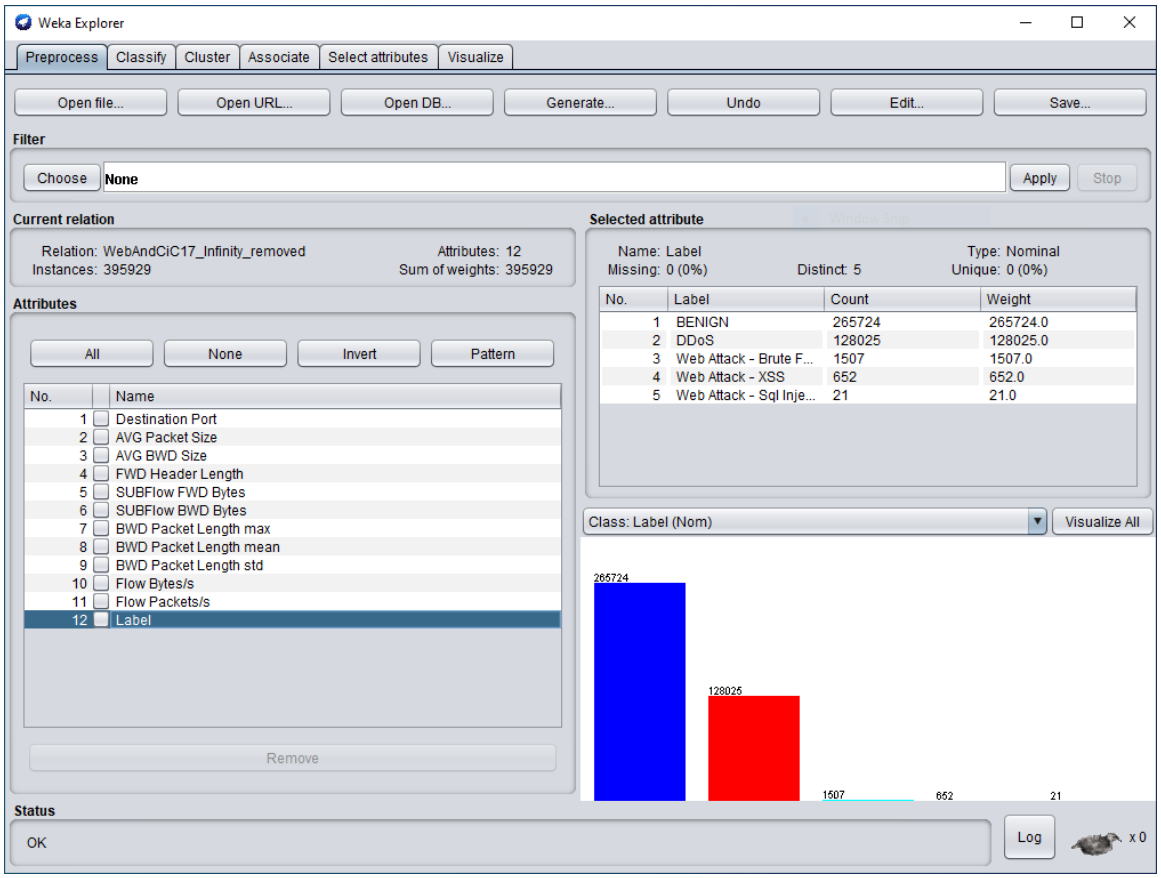


Figure 2.2. Example of Weka's Graphical User Interface

The Weka analysis process starts with loading the dataset as a csv file and performs error checking and initial summarization of each column. Although Weka is able to use many different data attributes, such as nominal and numerical data, not every algorithm supports every type of data. This makes the pre-processing, described in Chapter 3, very important for ensuring the algorithm can accept the dataset. After Weka runs the algorithms, the user interface provides easy-to-read information regarding the results, as shown in Figure 2.3. The user interface is explained in more detail in Chapter 3.

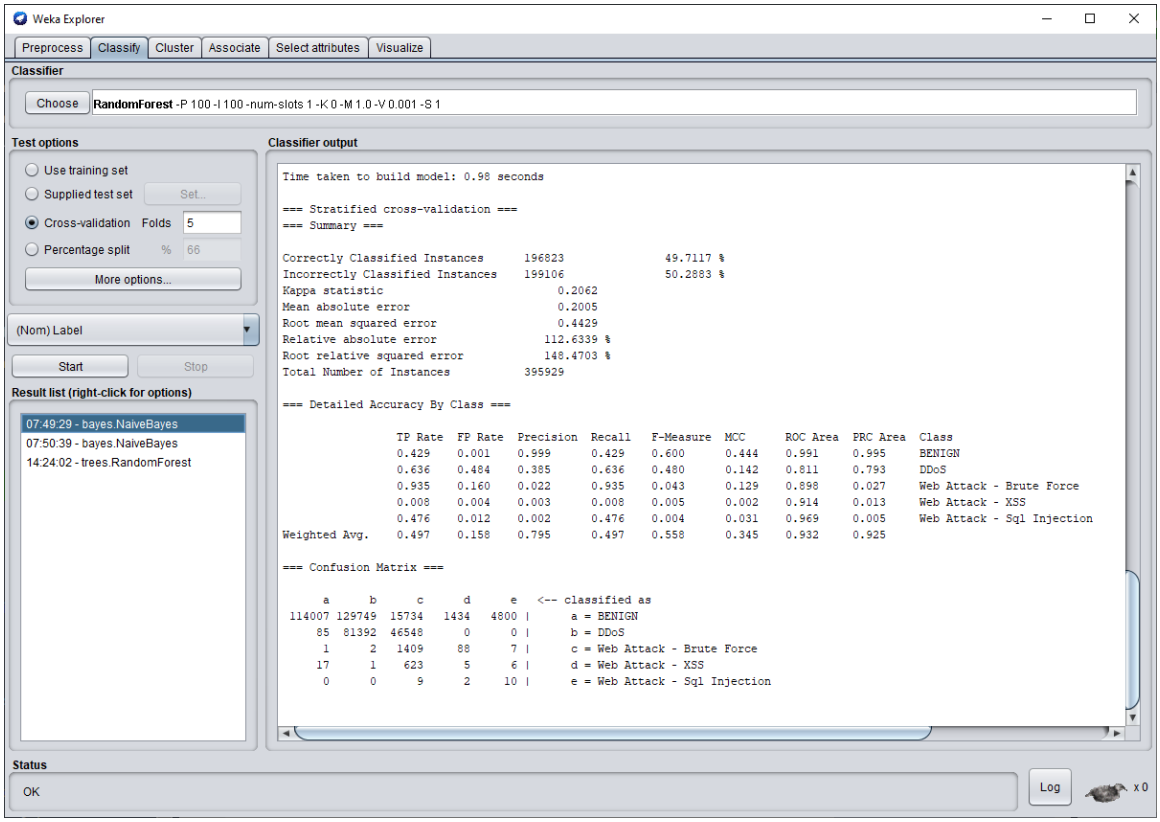


Figure 2.3. Example of Weka's Classifier Output

2.4 Algorithm Overview

While there are a variety of tools that can be used to conduct statistical analysis and anomaly detection, a frequent common factor is the usage of machine learning algorithms. Machine learning is a subset of artificial intelligence, which uses various programming techniques

to process data and extract useful information. There are three types of machine learning algorithms: supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning is used to find relationships between various features. Supervised learning algorithms are trained on labeled datasets to create a basis for making predictions of unlabeled data, which an algorithm then uses to assign a class label. Unsupervised learning uses unlabeled data to find hidden relationships that may not be initially seen. Unsupervised learning classifies input data into groups and assigns class labels based on groupings. Semi-supervised learning takes a middle approach and uses a small amount of labeled data to help predict unlabeled data. Reinforcement learning analyzes reactions in response to environment inputs. Some commonly used algorithms for each method can be found in Figure 2.4.

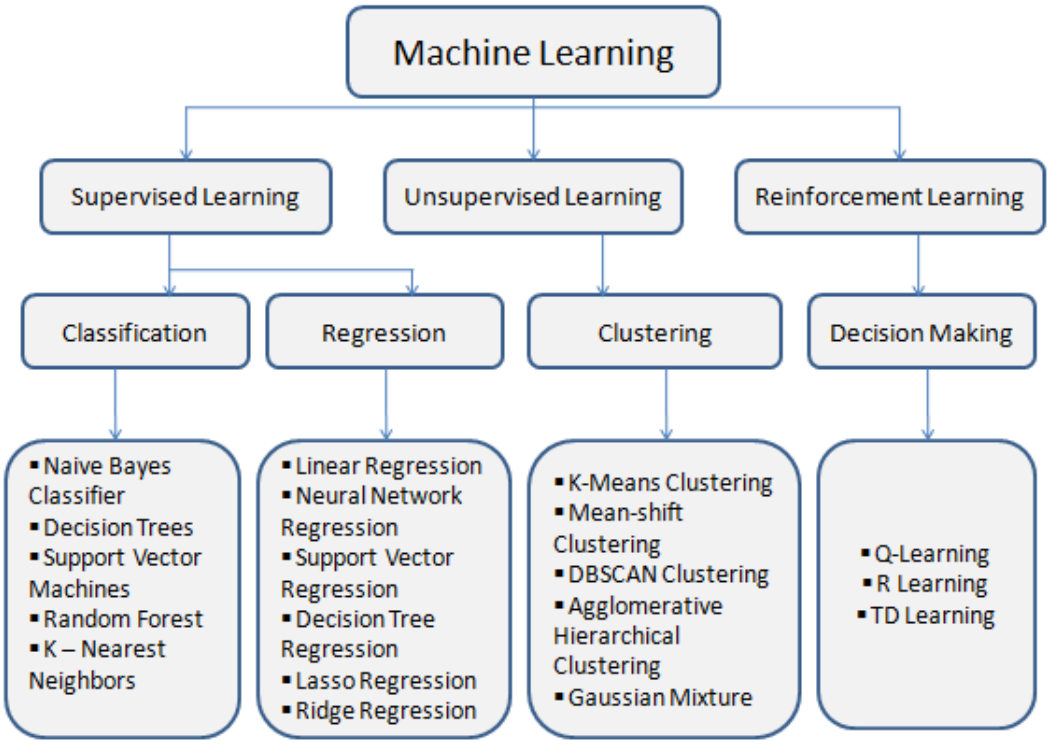


Figure 2.4. Overview of Machine Learning Methods. Source: [24].

We utilize supervised learning as the basis for our research, because as signatures for DoS attacks change over time, new baselines can be used to update training data. This post deployment process is key to maintaining or improving detection of anomalies over time. Supervised machine learning based approaches allow for operators to analyze small samples of new attacks and re-train algorithms once deployed.

Each machine learning algorithm has strengths and weaknesses, with the trade-off usually being accuracy or speed. Because there is no single algorithm that is best suited for all problems, four supervised classification algorithms were chosen for testing the datasets: RF, K-Nearest Neighbor (K-NN), Naïve Bayes (NB), and Support Vector Machine (SVM). These four algorithms are well established and are commonly used in literature for anomaly detection. For a better understanding of the Random Forest model, decision trees were also studied.

While there are other supervised learning classification algorithms, such as logistic regression or neural networks, the research using these algorithms did not appear often during literature review. A study of the top 10 algorithms in data mining for NxN comparisons showed the usefulness of our chosen models [25]. The four chosen algorithms represent a balance of speed vs accuracy, with SVM, RF, and K-NN focusing on accuracy while NB focuses on speed.

2.4.1 Decision Tree

Decision trees are represented by a series of binary splits based upon evaluation of characteristics of the data, with the terminal nodes determining the classification label. The benefit of this method is that it is easy to interpret the decisions the algorithm is making, as shown in Figure 2.5.

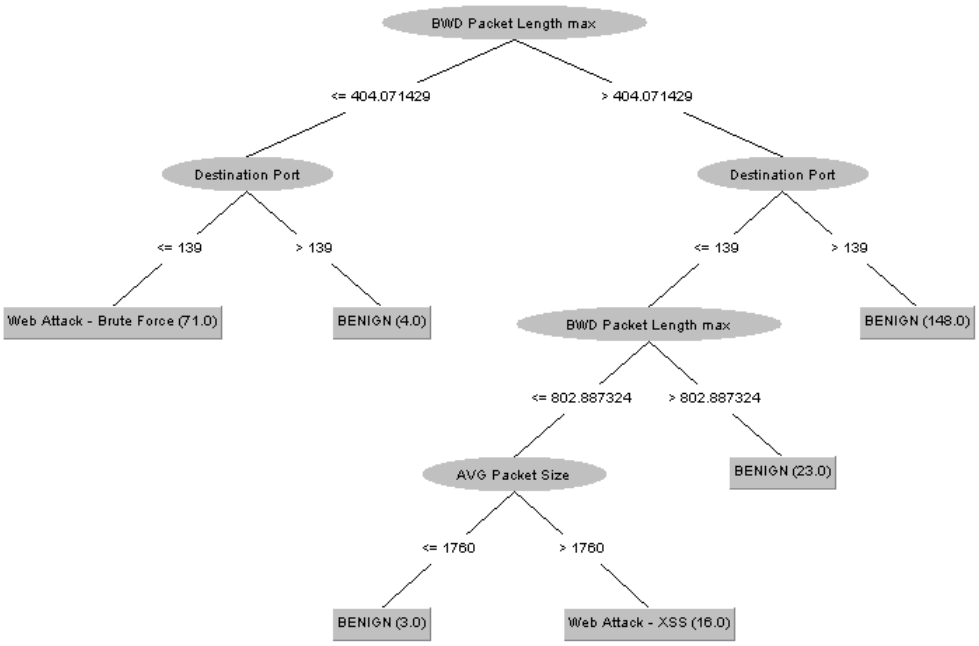


Figure 2.5. Example of a Decision Tree Splitting and Class Label Output

The interpretation becomes more complicated with larger trees; however, techniques such as boosting and bagging help improve classification. Boosting is a method that repeatedly trains a model and provides a weight for each prediction and test. Bagging is a method that trains classifiers on partitions of training data, with final classification being determined by majority vote. The Weka algorithm *J48* is the implementation of Quinlain's original C4.5 decision tree algorithm [26].

2.4.2 Random Forest

The Random Forest algorithm is an ensemble classifier consisting of a collection of decision trees that uses a random selection of features at each node to determine binary splits [27]. Individual trees are constructed and create outputs, with the final classification being decided through majority voting, as seen in Figure 2.6. This algorithm is an effective tool in classification, since the large number of trees that are used reduces over-fitting. The Weka algorithm *RandomForest()* is the implementation of Breiman's original algorithm, with the default settings specified as bagging.

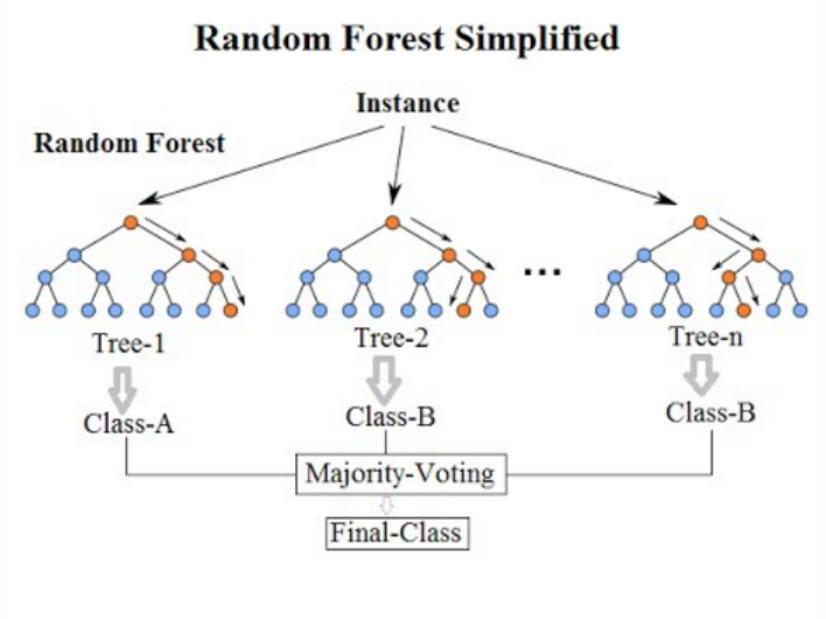


Figure 2.6. Example of Random Forest Output. Source: [28].

2.4.3 Support Vector Machine

The Support Vector Machine algorithm is a supervised learning method that can be used for classification analysis [29]. Given a set of training examples, a SVM training algorithm builds a model that assigns new examples to a category. The algorithm maps training data to points in space, attempting to maximize the width between two categories, and classifies testing samples based on where they fall [29]. In Weka, the *SMO* algorithm implements sequential minimal optimization using polynomial kernels.

Using SVM for mutli-class scenarios, classification is solved using pairwise classification, also known as 1 vs 1. In this approach, the classifier uses $m * (m - 1)/2$ hyperplanes, with m being equal to the number of classes. Using this approach, a multi-class classification problem is changed into one binary classification problem for each class. In this case, the SMO algorithm divides the flows into the following problem sets:

- Benign vs DoS
- Benign vs SQL

- Benign vs Web Attack
- Benign vs XSS
- DoS vs SQL
- DoS vs Web Attack
- DoS vs XSS
- SQL vs Web Attack
- SQL vs XSS
- Web Attack vs XSS

Figure 2.7 shows an example of this with three classes. Using the colors as different classes, a hyperplane separates each pair of classes while ignoring the third class. A flow is then classified based upon a majority vote.

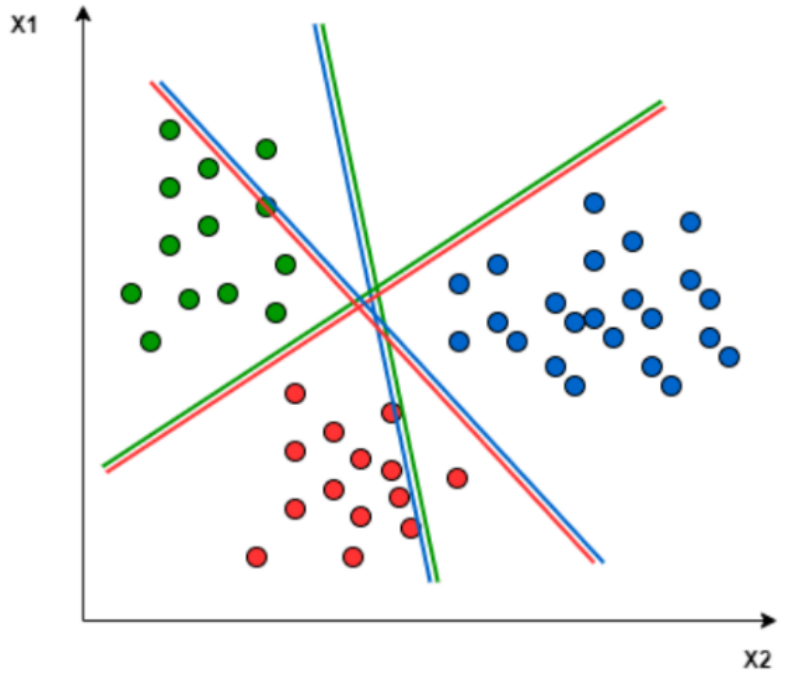


Figure 2.7. Example of 1 vs 1 SVM With Three Classes. Source: [30].

2.4.4 K-Nearest Neighbor

The K-Nearest Neighbor algorithm is a supervised learning method used for classification analysis [31]. The input consists of the K closest training examples from the training data, and the output is the class that the testing data is predicted to be. As more data is input, a case is assigned the class which is most common among its K nearest neighbors [31]. For our combined dataset, there is a large imbalance between the categories. As such, a smaller K-value was chosen in relation to the smallest attack type. Using the Weka algorithm *IBk*, we empirically choose the value of K to be equal to one and then two.

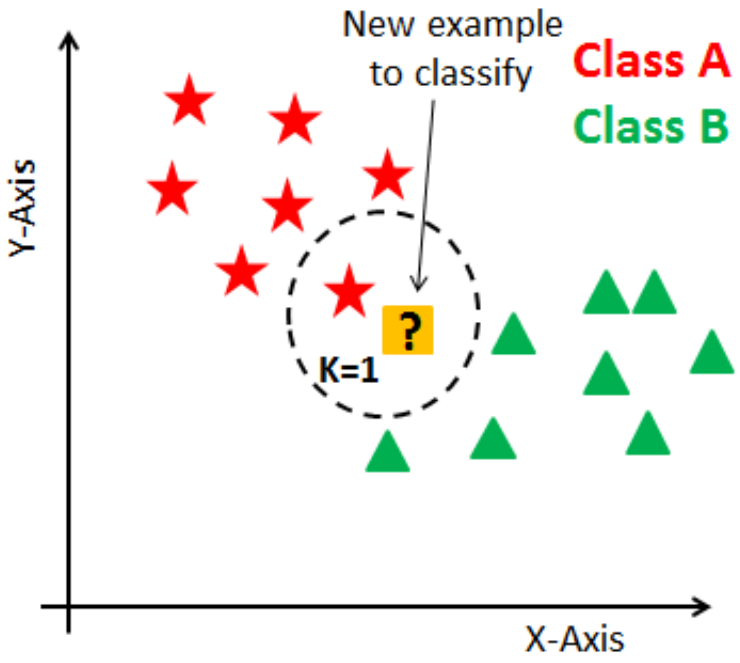


Figure 2.8. Example of K-NN decision making. Source: [32].

2.4.5 Naïve Bayes

The Naïve Bayes algorithm is a supervised learning method that uses the Bayes' theorem to predict a class [33]. The assumption is that a feature is independent of the value of any other feature. To determine the class of a specific flow, Weka uses the following equation, where P is the probability, c is the class, and fl is a flow:

$$P(c|fl) = \frac{P(fl|c) * P(c)}{P(f)} \tag{2.1}$$

The algorithm will assign a class label based on the highest probability class score for each case. While the weight of any feature can be directly input, the Weka algorithm *NaïveBayes* analyzes the training data to automatically determine the weight of each feature.

In summary, literature review has shown that NetFlow features can be used to conduct anomaly detection. The choices we made on dataset, analysis tool, and algorithm selection were influenced by the literature and impacted the experimental design. Chapter 3 discusses our proposed approach to network anomaly detection using supervised learning.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Experimental Design

Using the CRISP-DM methodology, we design our experiment with the following steps:

1. Prepare the data to use in the experiment. Explore the data, apply feature selection, and reduce data that is 'NaN' or listed as infinity.
2. Use Weka to execute the supervised algorithms on our prepared dataset. Default settings were used on all algorithms to facilitate comparison with future work.
3. Record and collect metric scores for each experimental run.
4. Evaluate results, review process, and determine next steps.

These steps are further explained in Sections 3.1–3.3.

3.1 Data Preparation

As discussed in Chapter 2, we selected the CIC-IDS 2017 and CSE-CIC-IDS 2018 datasets for this experiment. The flow data was downloaded from the Canadian Institute for Cybersecurity website [34] in csv format. The flows for the specific attack types were concatenated to create a single dataset.

To process the combined dataset in Weka the feature names had to be changed. This is because the names of some features are repeated, and within Weka each category must be unique. We changed each feature name to the letter of the corresponding column. We then removed all rows that had values equal to 'Infinity' or 'NaN'. Because the 2018 dataset had more flow identifiers than the 2017 dataset, such as 'Flow ID', 'Source IP', 'Source Port', 'Destination IP', 'Protocol', and 'Time Stamp', these features were deleted to maintain consistency with the 2017 dataset features.

3.2 Feature Selection

We then pre-processed to remove excessive and unimportant features. As shown in Section 2.1, the selected features play an important role in detecting attacks. The dataset begins with

a total of 79 features and one classification label. Using the methods described by [12], we analyzed each attack independently. The Weka search method, *Ranker*, ranks attributes by their individual evaluations. This helps us understand the relevance of the features respective to identifying an attack. Each feature is given a score, ranging from 0 to 1, with larger values indicating more relevant features.

There are two methods to select features: selecting all features exceeding a given threshold or selecting the number of features to retain that have the highest rank. The first option may result in a different number of selected features for each attack, so we choose to look at the top 10 features. Table 3.1 shows the top ten features of each attack and the average information gain score. Weka calculates entropy using MDL-based discretization as seen in (3.1), as described in [35]. Information gain by measuring the entropy of the feature with respect to the class, as seen in (3.2) [23]. In (3.2), Ig is information gain, c is class, and f is feature.

$$H(X) = \sum_{i=1}^{k} P(x_i) log(P(x_i)) \tag{3.1}$$

$$Ig(c, f) = H(c) - H(c|f) \tag{3.2}$$

The closer the average value is to 0, the more we expect to see false negatives and false positives occurring within that attack type.

Further analysis of the features in each attack showed three sets of features were identical in terms of values recorded by the NetFlow exporter but had different names. The features 'Total Length of Fwd Packets', 'Total Length of Bwd Packets', and 'Fwd Header Packet Length' were approximately the same as 'Subflow Fwd Bytes', 'Subflow Bwd Bytes', and 'Fwd Header Byte Length'. Although the equations used to generate the values are unavailable, it could be that the packet feature values were kept in units of bytes. After removing these features from DoS attack, the next largest features were input: 'Backward Packet Length Mean', 'Flow Bytes/s', and 'Flow Packets/s'. This brought the average information gain from 0.815 bits to 0.782 bits for DoS attacks.

A Venn diagram was created to visually map the features of each attack, as seen in the Appendix. An interesting observation from the ranked features showed that XSS and Brute

force attacks were separated by a single feature. The overlapping features between these two attack types could cause an algorithm to mistake XSS attacks as Brute Force, or vice versa. With XSS attacks having a much smaller average information gain than Brute Force attacks, we expect that XSS will have more false positives results.

Although Weka defaults to using all features, several ranked features from the other application layer attacks overlapped the ranked features from the DoS attack set. The overlapping features showed that other application layer attacks utilized similar features as DoS attacks. We predict that removing non-overlapping features from the other non-DoS application layer attacks should have a minimal impact in algorithm performance. The overlapping features were chosen as the focus of the study, to determine if our selected features could be used to differentiate DoS attacks from other non-DoS application layer attacks and reduce our features down to a minimum. As shown in [14] and [15], the 'Destination Port' feature can improve detection and was kept for a feature of study.

Our selected features listed in Table 3.2 represent the focus our experiment. Compared to past research, this reduced feature set offers a novel, streamlined approach to detecting application layer DoS attacks.

Table 3.1. Top Ten Features from Weka of Each Attack Type and Average Information Gain

Attack Type	Top Ten Features	Average Information Gain
DoS Attack	Total length of fwd packets, Subflow fwd bytes, Avg packet size, Total length of bwd packets, Subflow bwd bytes, Bwd packet length mean, Avg bwd segment size, Fwd header packet length, Fwd header byte length, backward packet length mean	0.815 bits
Brute Force	Fwd IAT min, Fwd packets, Fwd IAT mean, Flow IAT mean, Flow IAT max, Flow IAT std, Fwd Header Length, Flow bytes/s, Initial window bytes backward, Bwd Packets/s	0.038 bits
XSS	Fwd IAT min, Fwd packets, Fwd IAT mean, Flow IAT mean, Flow IAT max, Flow IAT std, Fwd header length, Flow bytes/s, Initial window bytes backward, Fwd IAT std	0.004 bits
SQL Injection	Fwd packet length max, Max packet length, Bwd IAT mean, Fwd packet length mean, Avg fwd segment size, Bwd packet length max, Bwd packet length std, Destination port, Flow bytes/s, Initial window bytes backward	0.02 bits

Table 3.2. Reduced Feature Set Used for Experimentation. Source: [36].

Feature	Summary
Avg Bwd Segment Size	Average number of bytes bulk rate in the backward direction.
Avg Packet Size	Average size of packet
Bwd Packet Length Max	Maximum size of packet in backward direction.
Bwd Packet Length Mean	Mean size of packet in forward direction.
Bwd Packet Length Std	Standard deviation size of packet in backward direction.
Destination Port	Port destination of flow
Flow Bytes/s	Number of flow bytes per second
Flow Packets/s	Number of flow packets per second
Fwd Header Length	Length (bytes) of the forward packet header.
Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction.
Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction.
Label	The type of traffic recorded, listed either as Benign or one of the attack categories.

3.3 Experiment Setup

To determine the metrics of our reduced feature dataset, our experiment was run in two parts. First, we tested detection of only DoS attacks in the presence of benign traffic, using the reduced feature set. We then tested the detection of DoS attacks in the presence of other application layer attacks, using the reduced feature set. After the completion of the experiment, we compared results of the first two trials against the results obtained from using all available features to determine the extent to which feature selection impacted computation time and accuracy.

Within Weka, each algorithm was executed in three steps. First, the dataset was loaded into Weka in the Pre-processing tab, shown in Figure 3.1.

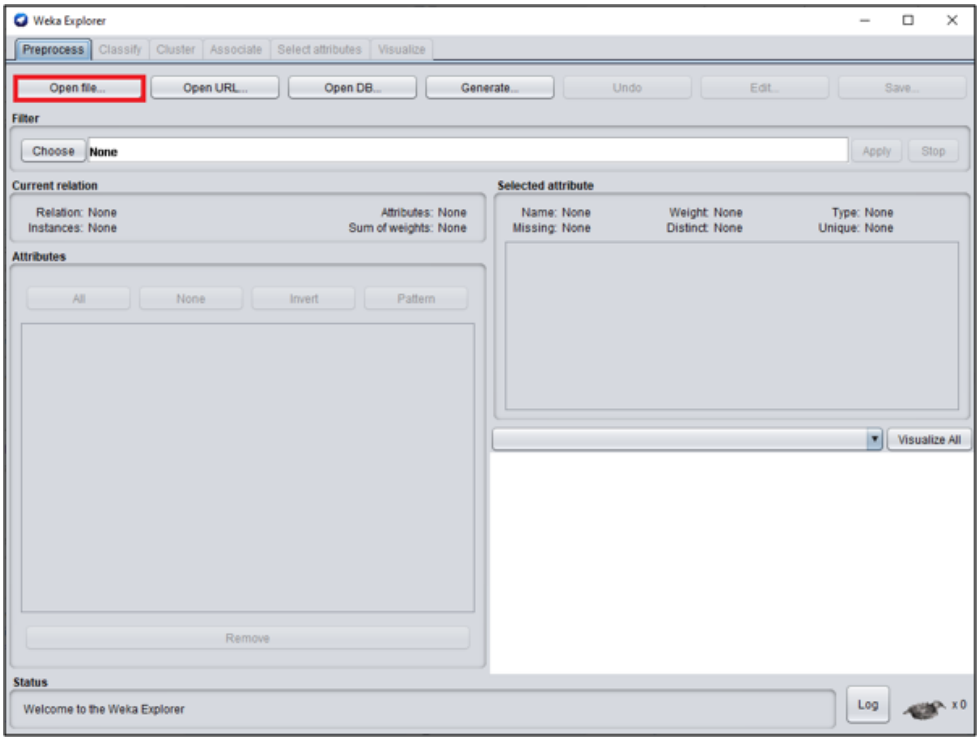


Figure 3.1. Loading Files Into Weka for Testing

Next, the appropriate algorithm was chosen from the list of options in the Classify tab, shown in Figure 3.2.

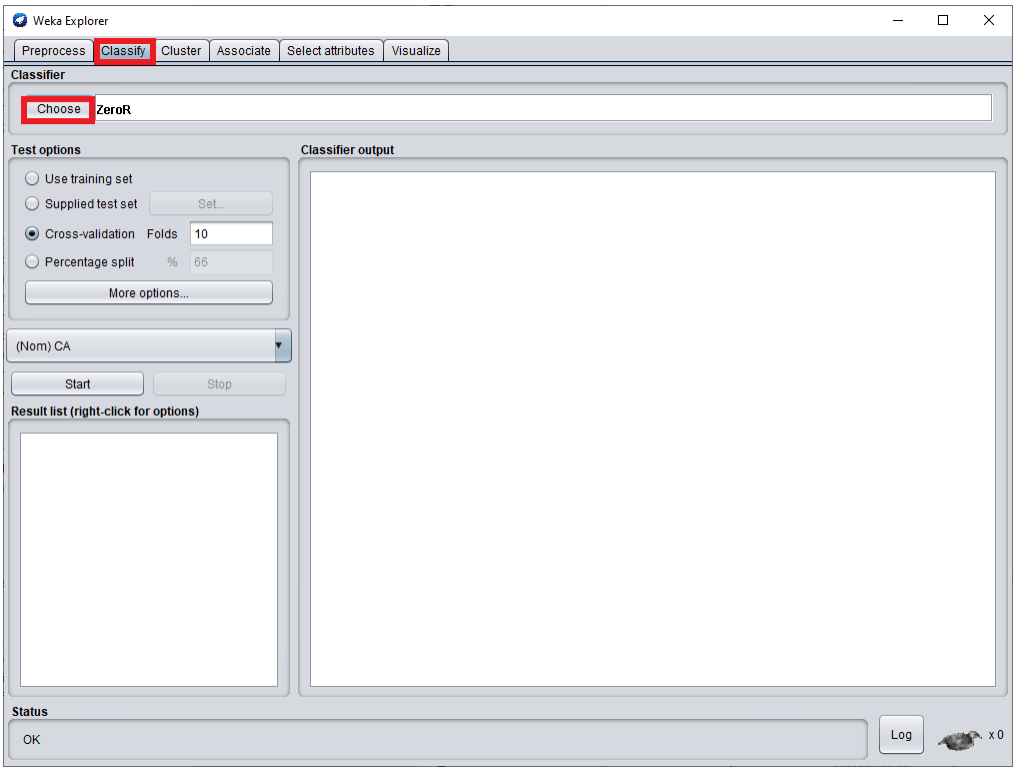


Figure 3.2. Choosing Appropriate Algorithm to Model

Lastly, the cross-validation value was chosen in the test options box and the experiment was run, shown in Figure 3.3.

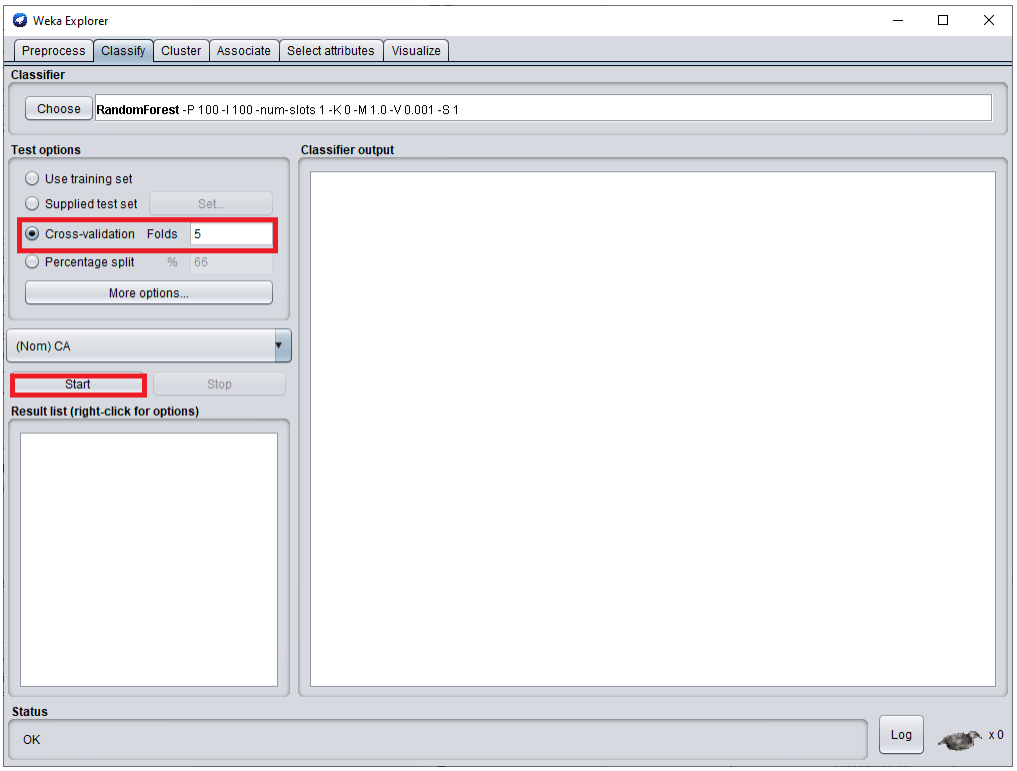


Figure 3.3. Selecting Cross-Validation and Model Start

Cross-validation is a method to evaluate machine learning models. This method splits data into k groups, where each case is used as a test set exactly once and is used in the training data $k - 1$ times, seen in Figure 3.4.

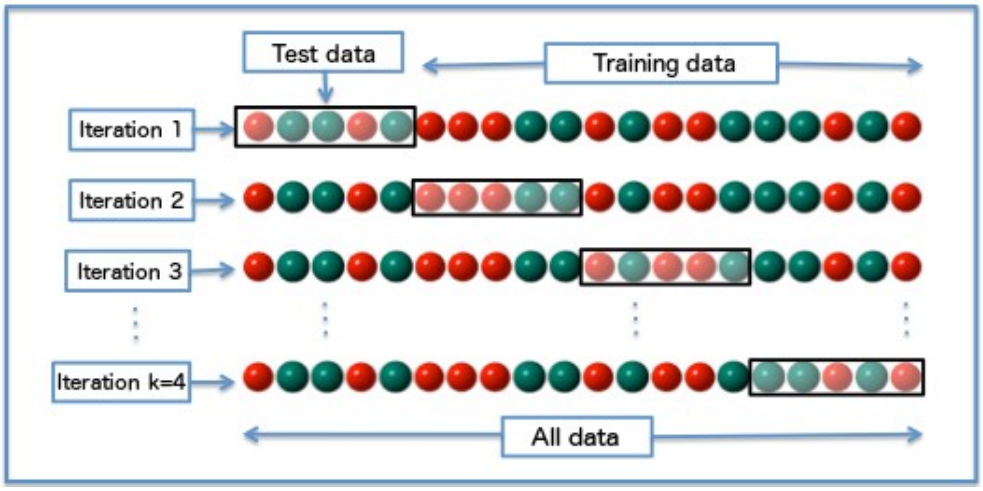


Figure 3.4. Cross-Validation Example with k = 4. Source: [37].

Weka goes one step further and implements stratified k-fold cross-validation. Similar to k-fold cross-validation, stratified k-fold cross-validation keeps the percentage of samples of each target class the same. This is significant because of the imbalance within the combined dataset, as discussed at the end of Section 2.2.6. In our experiment, the value of k was set to 5. This means that 80% of the data is used for training and 20% of the data is used for testing in each iteration. While there is no widely accepted standard for training and testing sizes, an 80% / 20% train-test split is commonly used in machine learning if not performing cross validation. For our experiment, the training and test record splits for each experiment are shown in Table 3.3 and 3.4. At the end of each iteration, the results are saved, with the final score being the average of the results.

Table 3.3. Distribution of Training and Test Data Subsets for Experiment 1

Label	Total	Training	Testing
Benign	97718	78174	19544
DoS	128025	102420	25605
Total	225745	180595	45150

Table 3.4. Distribution of Training and Test Data Subsets for Experiment 2

Label	Total	Training	Testing
Benign	265724	212579	53145
DoS	128025	102420	25605
Brute Force	1507	1206	301
Web Attack	652	522	130
SQL Inject	21	17	4
Total Attack	130205	104165	26040
Total	395929	316744	79185

3.4 Performance Metrics

Weka outputs the results of each algorithm into a confusion matrix, shown in Table 3.5, which shows the four cases of classification: True Positive (TP), True Negative (TN), False Positive (FP), or False Negative (FN).

- True Positive: correct classification of attack traffic as attack. For example, a flow is predicted as a DoS attack and it is a DoS attack.
- True Negative: correct classification of benign traffic as benign.
- False Positive: incorrect classification of benign traffic as a different attack class. For example, a flow is predicted as benign traffic and it is a DoS attack.
- False Negative: incorrect classification of attack traffic as benign.

Table 3.5. Confusion Matrix Illustration

		Predicted Class	
		Attack Class	Benign Class
True Class	Attack Class	$TruePositive$	$FalseNegative$
	Benign Class	$FalsePositive$	$TrueNegative$

We use these values to calculate the accuracy, detection rate (DR), false alarm rate (FAR), and the MCC of each algorithm, seen in (3.3) – (3.6).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

$$DR = \frac{TP}{TP + FN} \tag{3.4}$$

$$FAR = \frac{FP_{Benign}}{TP_{Attack} + FP_{Benign}} \tag{3.5}$$

$$MCC = \frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{3.6}$$

Accuracy is the ratio of predictions that the algorithm correctly classified. However, accuracy alone does not give a complete picture of algorithm performance. MCC returns a value between -1 and 1, with a higher value indicating a better overall accuracy. The MCC is a reliable statistical metric for imbalanced datasets, because a high value will only be obtained if all four categories performed well, as shown in (3.5) [38].

Detection Rate (DR) is an important metric in this experiment, because if an attack is classified as benign then it still has the potential to compromise the system. False alarm rate (FAR) (from benign traffic) is also important because of the impact it has on the operator. Irrelevant alerts lead to wasted time and alert fatigue, because for each alert the operator must consider the source, detection logic being used, and apply their own logic process to determine if tools need to be revamped, calibrated, or updated with new data. While there are no standards for acceptable false alarm rates current survey data reports that an estimated 20 – 40% of security alerts are false positives [39].

For the first experiment, maximizing overall accuracy and the MCC score were our objective. For the second experiment, a multi-class confusion matrix was used to compare each supervised learning classification result with overall accuracy, DR, and FAR as the measures of performance.

The metrics discussed were chosen because they are a good method of testing whether

the selected features could provide one step-detection of in both experiments. A high detection rate combined with a low false alarm rate would show that the selected features would minimize operational risk while maximizing resources.

CHAPTER 4:
Results and Analysis

This chapter will discuss the performance of each algorithm against application layer attacks. We analyze the results, followed by general observations. These experiments were carried out using an Intel(R) Xeon(R) CPU E5-2640 v3 CPU running at 2.60 GHz with 32 Gb of RAM, utilizing default parameters on Weka.

For each of the following subsections we list the algorithm used, the calculated metrics, and a colored confusion matrix. The matrix utilizes three shades of green and red: a tinted color for values less than 10%, a pure color for values between 10% and 90%, and shaded color for values greater than 90%.

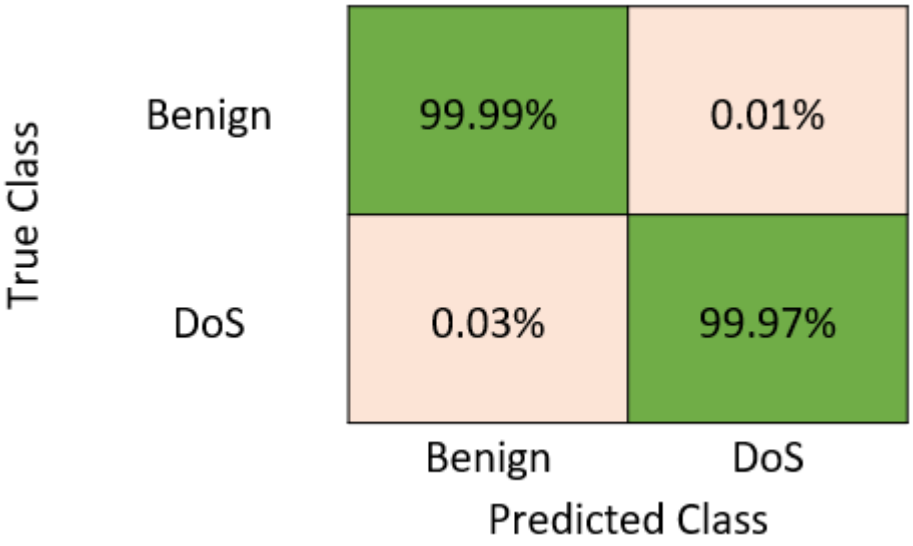
4.1 Experiment 1–DoS Detection in Benign Traffic

Experiment 1 tested detection of only DoS attacks in the presence of benign traffic, using the reduced feature set.

4.1.1 Random Forest

Table 4.1 shows the results of the first experiment using the RF algorithm. RF showed 0% of the DoS flows (35 of 128 025) wrongly classified as normal traffic while 0% of the normal flows (27 of 97 718) were wrongly classified as DoS traffic. This performance yields a MCC value of 0.99 and overall accuracy of 99.97%.

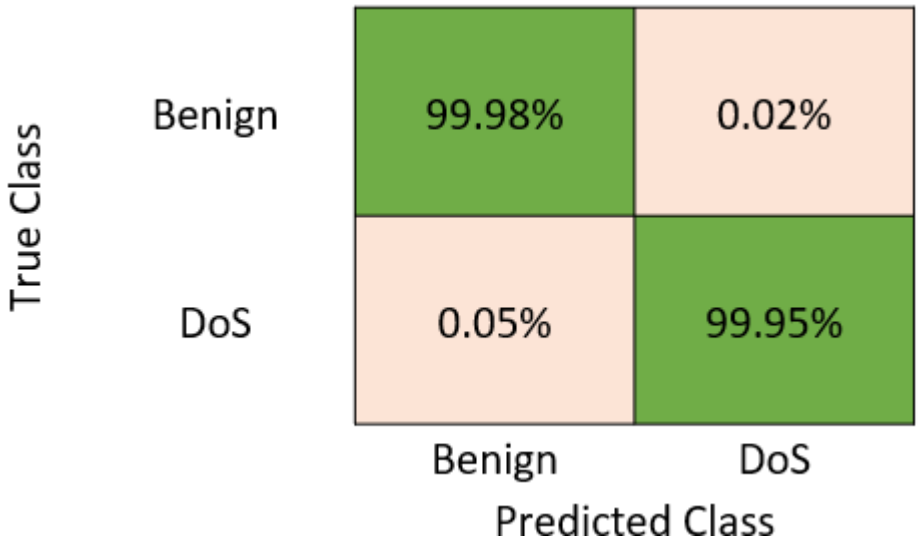
Table 4.1. Confusion Matrix for Random Forest, Experiment 1



4.1.2 K-NN, K=1

Table 4.2 shows the results of the first experiment using the K-NN algorithm with 1-nearest neighbor. 1-nearest neighbor showed 0.02% of the DoS flows (68 of 128 025) wrongly classified as normal traffic while 0.05% of the normal flows (16 of 97 718) were wrongly classified as DoS traffic. This performance yields a MCC value of 0.99 and overall accuracy of 99.96%.

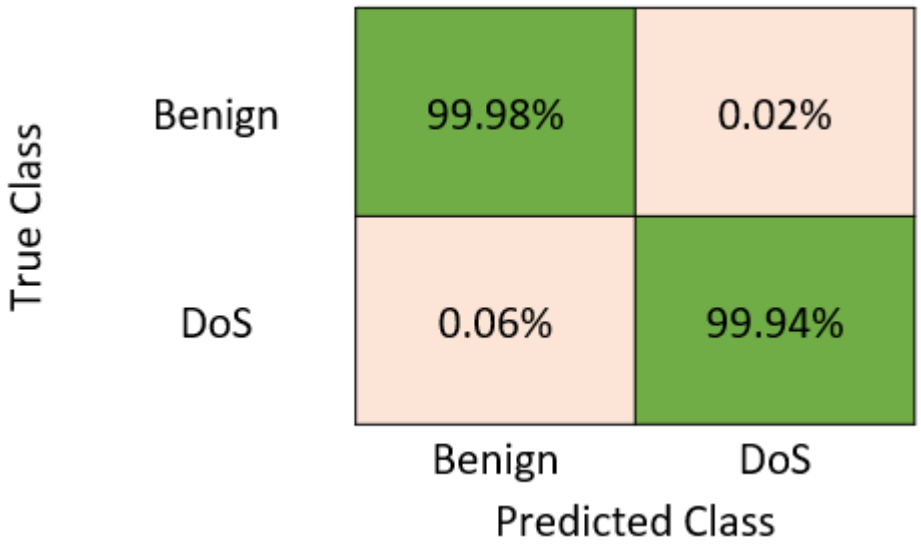
Table 4.2. Confusion Matrix for K-Nearest Neighbor, K=1, Experiment 1



4.1.3 K-NN, K=2

Table 4.3 shows the results of the first experiment using the K-NN algorithm with 2-nearest neighbor. 2-nearest neighbors showed 0.02% of the DoS flows (76 of 128 025) wrongly classified as normal traffic while 0.06% of the normal flows (22 of 97 718) were wrongly classified as DoS traffic. This performance yields a MCC value of 0.99 and overall accuracy of 99.95%.

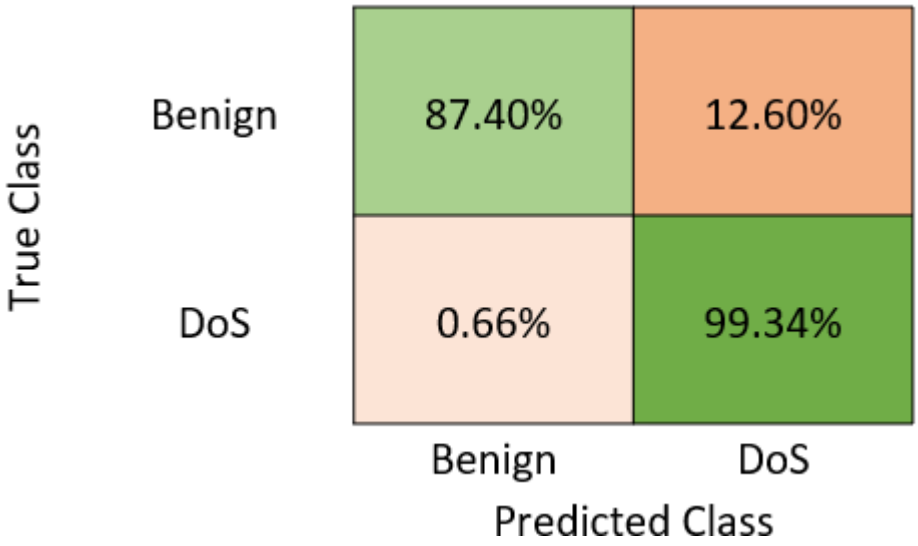
Table 4.3. Confusion Matrix for K-Nearest Neighbor, K=2, Experiment 1



4.1.4 SVM

Table 4.4 shows the results of the first experiment using the SVM algorithm. SVM showed 0.66% of the DoS flows (841 of 128 025) wrongly classified as normal traffic while 12.60% of the normal flows (12 313 of 97 718) were wrongly classified as DoS traffic. This performance yields a MCC value of 0.88 and overall accuracy of 94.17%.

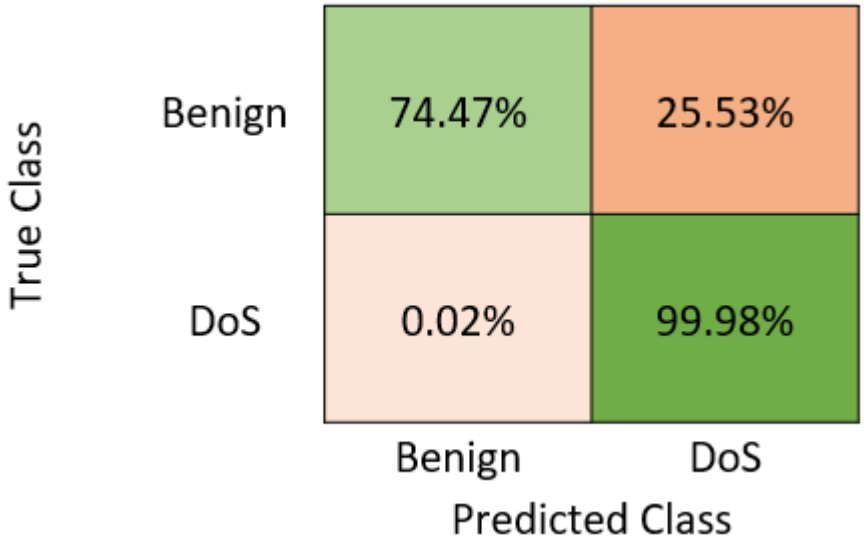
Table 4.4. Confusion Matrix for Support Vector Machine, Experiment 1



4.1.5 NB

Table 4.5 shows the results of the first experiment using the NB algorithm. NB showed 0.02% of the DoS flows (26 of 128 025) wrongly classified as normal traffic while 25.53% of the normal flows (24 943 of 97 718) were wrongly classified as DoS traffic. This performance yields a MCC value of 0.79 and overall accuracy of 88.94%.

Table 4.5. Confusion Matrix for Naïve Bayes, Experiment 1



4.1.6 Experiment 1 Analysis

From this experiment we show that the RF algorithm performed the best at detecting DoS attacks using our reduced feature set, while SVM performed the worst. While the MCC scores were 0.99 for both the RF and K-NN algorithms, the RF algorithm yielded less false positives which leads to a smaller amount of false alerts being generated. Although the DoS detection rate was 0.68% greater in NB than SVM, the number of false alarms generated by the NB algorithm were double that of SVM. This trade-off makes the SVM algorithm a better choice.

4.2 Experiment 2–DoS Detection in the Presence of Other L7 Attacks

Experiment 2 tested the impact of mixing other attacks in with DoS attacks, using the reduced feature set. From the literature review, we expect that the addition of other L7 DoS attacks will decrease the accuracy of our models due to an increase in false positive and false negative counts. The results from [13] suggests that the large class imbalance of XSS and SQL attacks will have the biggest impact on our chosen metrics.

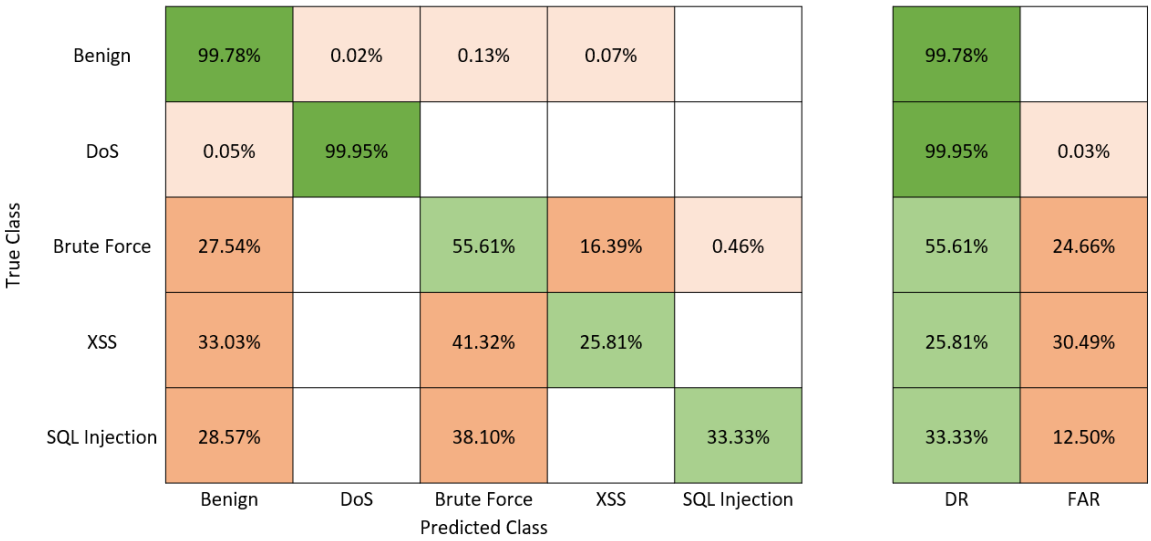
When comparing the overall performance of our algorithms, we focused on how many attacks went undetected by our reduced feature set and how many false alarms are generated. An attack classified as benign traffic could lead to a compromise, while an attack incorrectly classified as a different attack would still be evaluated as dangerous. Benign traffic classified as an attack would increase the FAR and negatively impact user experience and security operators.

In each confusion matrix listed in Section 4.2, we provide the FAR for each attack. However, because of the population imbalance of the other application layer attacks, a high FAR for an attack may have a disproportionate impact on the overall FAR. As such, we determine the overall FAR by calculating the median value.

4.2.1 Random Forest

Table 4.6 shows the results of the second experiment using the RF algorithm, with an overall accuracy of 99.54% and a MCC of 0.9996. RF showed 0.05% of the DoS flows (58 of 128 025) wrongly classified as normal traffic, showed 30% of other application layer attacks wrongly classified as normal traffic (636 of 2153), while 0.22% (590 of 265 724) of the benign flows were wrongly classified as application layer attacks. RF had a median FAR of 18.58%.

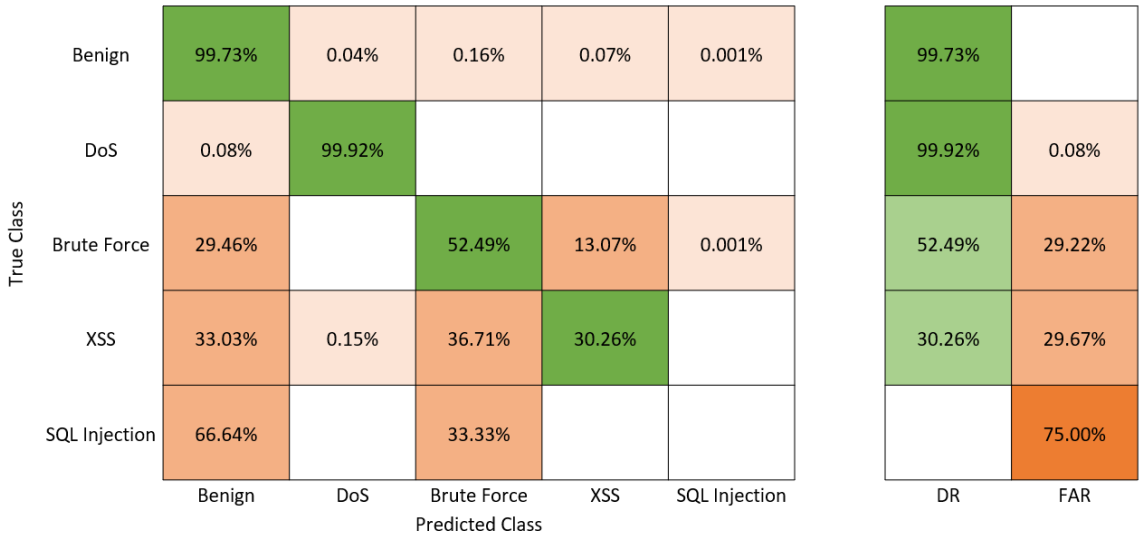
Table 4.6. Confusion Matrix for Random Forest, Experiment 2



4.2.2 K-NN, K=1

Table 4.7 shows the results of the second experiment using the K-NN algorithm with 1-nearest neighbor, with an overall accuracy of 99.49% and a MCC of 0.9886. K-NN showed 0.08% of the DoS flows (98 of 128 025) wrongly classified as normal traffic, showed 31% of other application layer attacks wrongly classified as normal traffic (673 of 2153), while 0.27% (727 of 265 724) of the benign flows were wrongly classified as application layer attacks. K-NN with 1-nearest neighbor had a median FAR of 29.45%.

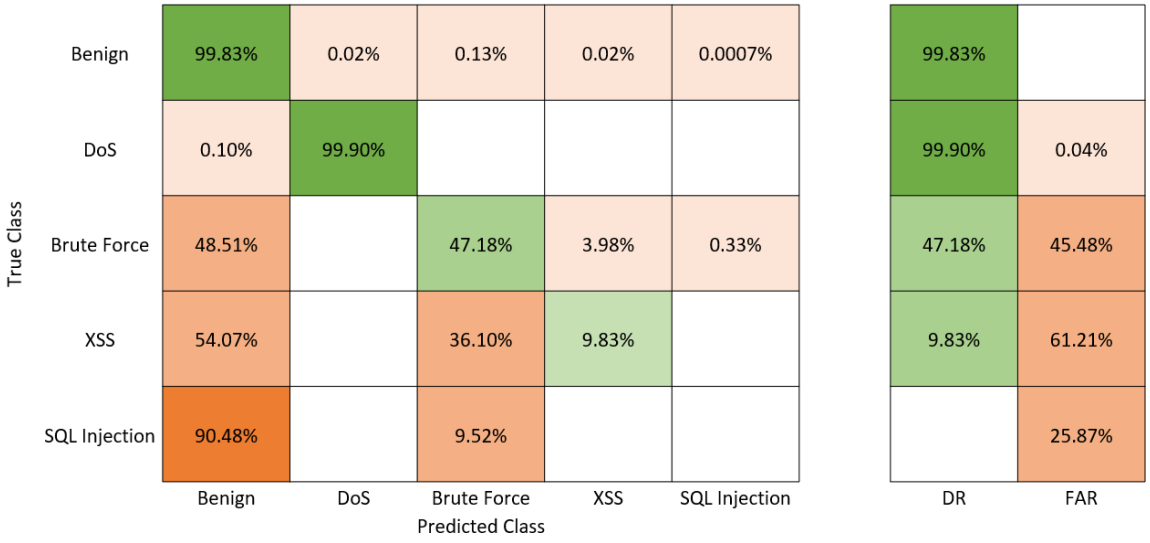
Table 4.7. Confusion Matrix for K-Nearest Neighbor, K=1, Experiment 2



4.2.3 K-NN, K=2

Table 4.8 shows the results of the second experiment using the K-NN algorithm with 2-nearest neighbors, with an overall accuracy of 99.50% and a MCC of 0.9887. K-NN showed 0.10% of the DoS flows (126 of 128 025) wrongly classified as normal traffic, showed 51% of other application layer attacks wrongly classified as normal traffic (1102 of 2153), while 0.17% (449 of 265 724) of the benign flows were wrongly classified as application layer attacks. K-NN with 2-nearest neighbors had a median FAR of 35.68%.

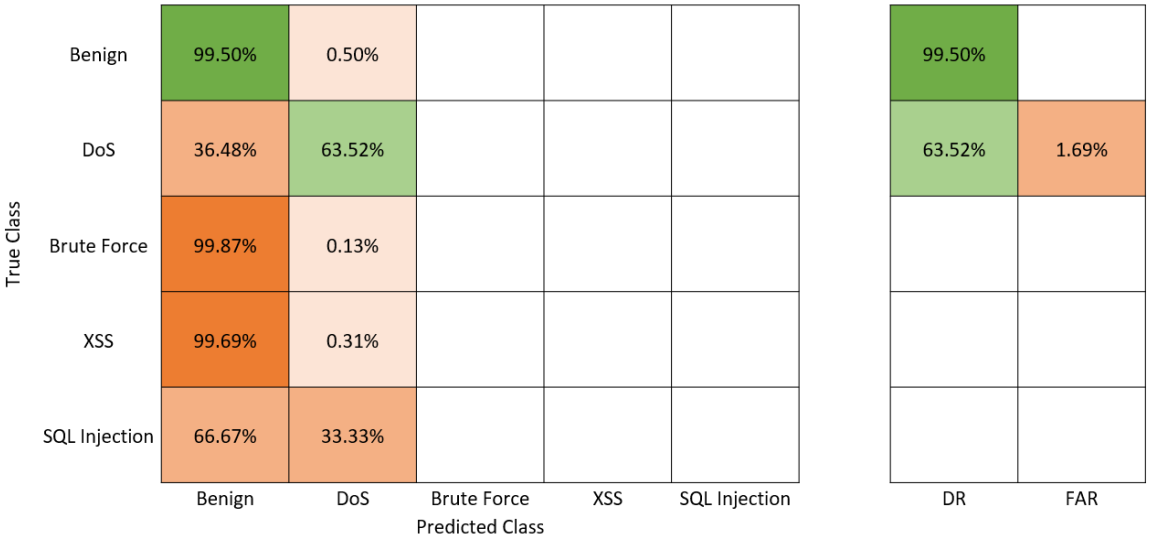
Table 4.8. Confusion Matrix for K-Nearest Neighbor, K=2, Experiment 2



4.2.4 SVM

Table 4.9 shows the results of the second experiment using the SVM algorithm, with an overall accuracy of 87.30% and a MCC of 0.7159. SVM showed 36.48% of the DoS flows (46 701 of 128 025) wrongly classified as normal traffic, showed 99% of other application layer attacks wrongly classified as normal traffic (2142 of 2153), while 0.50% (1390 of 265 724) of the benign flows were wrongly classified as application layer attacks. SVM had a median FAR of 0%.

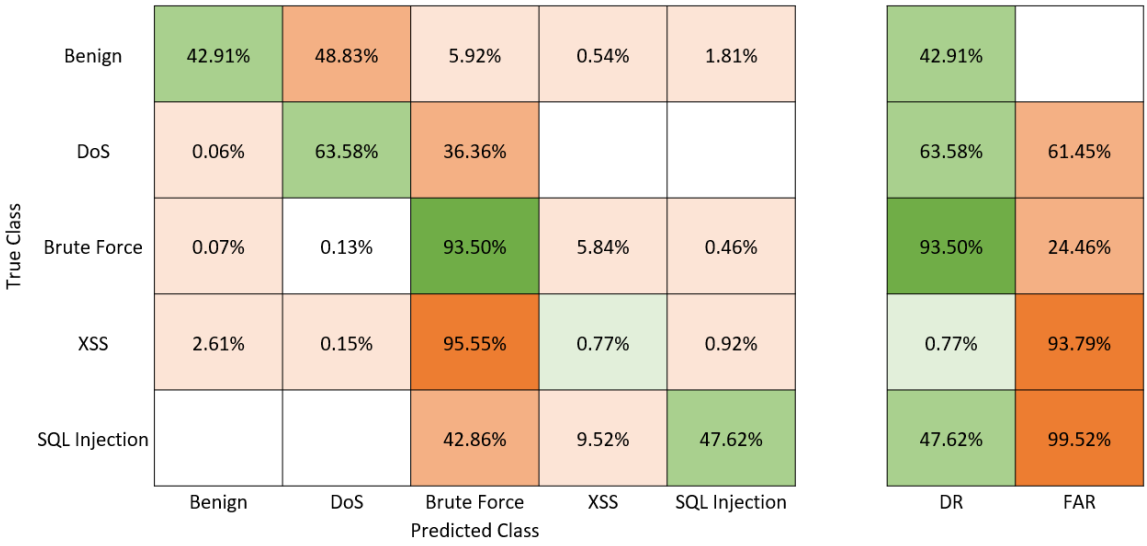
Table 4.9. Confusion Matrix for Support Vector Machine, Experiment 2



4.2.5 NB

Table 4.10 shows the results of the second experiment using the NB algorithm, with an overall accuracy of 49.71% and a MCC of 0.2159. NB showed 0.06% of the DoS flows (85 of 128 025) wrongly classified as normal traffic, showed 0.83% of other application layer attacks wrongly classified as normal traffic (18 of 2153), while 47.19% (60 415 of 265 724) of the benign flows were wrongly classified as application layer attacks. NB had a median FAR of 77.62%.

Table 4.10. Confusion Matrix for Naïve Bayes, Experiment 2



4.2.6 Experiment 2 Analysis

From this experiment we show that the RF algorithm performed the best at detecting DoS attacks using our reduced feature set, while the SVM and NB algorithms performed the worst. However, while SVM had a higher MCC score and overall accuracy than NB, the SVM algorithm had the worst detection of other application layer attacks. Shown in Table 4.9, the SVM algorithm produced the least amount of benign false alarms relative to the other algorithms, leading to a FAR of 0%. But the algorithm also had the most false negatives. In comparison, as shown in Table 4.10, the NB algorithm had the fewest false negatives, resulting in less attacks being successful. However, the algorithm also had the highest benign false positives, leading to the highest FAR of all four algorithms.

These results are consistent with our findings from Experiment 1. The effect of adding other attacks reduced the overall accuracy of each algorithm, although the large drop in performance of the SVM and NB algorithms was interesting. The RF algorithm had the smallest decrease of 0.43% and the NB algorithm had the largest decrease of 39.23%.

With the population imbalance of Brute Force, XSS, and SQL Injection attacks, a low non-DoS attack detection rate did not impact the overall detection rate significantly. This is most easily seen when looking at the reduction of performance with the SVM algorithm: although there was a 0% detection rate for the other application layer attacks, the overall detection rate was still 87.30%. However, as discussed in Section 3.4, the MCC gives a better representation of algorithm performance. Table 4.11 compares the MCC score of each algorithm using the full feature set or the reduced feature set. The MCC score for the RF and K-NN algorithms were high for both feature sets, while the SVM and NB algorithms experienced a large decrease in score with the addition of the other application layer attacks using the reduced feature set.

Table 4.11. Algorithm MCC Score Comparison

	MCC Scores			
	All Features		Reduced Features	
Algorithm	Experiment 1	Experiment 2	Experiment 1	Experiment 2
RF	0.9998	0.9949	0.9994	0.9996
K-NN, K=1	0.9996	0.9957	0.9992	0.9886
K-NN, K=2	0.9996	0.9920	0.9991	0.9887
SVM	0.9791	0.9471	0.8845	0.7159
NB	0.9547	0.9112	0.7892	0.2159

4.3 Comparisons Between Reduced and Full Feature Sets

While we have focused on the algorithm results of using reduced features, in this next section we make comparisons based on the same experimental design but while using all features.

4.3.1 Overall Detection Accuracy Comparisons

Each algorithm showed a decrease in overall detection accuracy when using the reduced feature set, shown in Table 4.12. The RF algorithm had the smallest decrease in accuracy, while the NB algorithm had the largest decrease in accuracy between the two feature sets.

Table 4.12. Algorithm Overall Detection Accuracy

	Classification Success Accuracy			
	All Features		Reduced Features	
Algorithm	Experiment 1	Experiment 2	Experiment 1	Experiment 2
RF	99.99%	99.75%	99.97%	99.54%
K-NN, K=1	99.98%	99.78%	99.96%	99.49%
K-NN, K=2	99.98%	99.65%	99.95%	99.50%
SVM	98.97%	97.54%	94.17%	87.30%
NB	97.75%	95.53%	88.94%	49.71%

4.3.2 DoS Only Detection Rate comparisons

We highlight the DoS only detection rate to show that the reduced feature set is sufficiently discriminative to detect attacks accurately. As seen in Table 4.13, comparing the DoS detection rate of the reduced feature set to the DoS detection rate of the full feature set, the algorithm choice had a clear impact on performance.

Table 4.13. Algorithm DoS Detection Accuracy Comparison

	DoS Detection Accuracy			
	All Features		Reduced Features	
Algorithm	Experiment 1	Experiment 2	Experiment 1	Experiment 2
RF	99.99%	100%	99.97%	99.95%
K-NN, K=1	99.99%	100%	99.95%	99.92%
K-NN, K=2	99.98%	100%	99.94%	99.90%
SVM	99.95%	99.90%	99.34%	63.52%
NB	99.92%	99.90%	99.34%	63.58%

In regards to the RF and K-NN algorithms, the detection of DoS attacks are comparable between all features and the reduced features, with 100% detection vs greater than 99.9% detection. The SVM and NB algorithms had the most significant decrease in performance, with DoS detection decreasing by more than 33% between the two feature sets. This would indicate additional features may be needed to help distinguish between the two classes for the SVM and NB algorithms.

4.3.3 Comparison of Testing Times

We compared the algorithm computation times using the reduced features dataset to the algorithm computation times using the full feature dataset. Table 4.14 shows the time required to test and train each algorithm, per fold, for each experiment. Table 4.15 and Table 4.16 show the total testing time for each algorithm and the percent reduction time between the two feature sets, for each experiment.

Table 4.14. Algorithm Comparison of Time Required to Test and Train Algorithms, per Fold

	Testing and Training Time per Fold (s)			
	All Features		Reduced Features	
Algorithm	Experiment 1	Experiment 2	Experiment 1	Experiment 2
RF	181.2	215.15	86	167.61
K-NN, K=1	1392.6	3967	1078.6	2988
K-NN, K=2	1284.4	3960	21 155.2	2945
SVM	180.6	97.23	67.2	3,610
NB	4.6	11.22	0.8	2.81

Table 4.15. Algorithm Total Testing and Training Time Comparisons, Experiment 1

	Total Test and Training Time (s)			
Algorithm	All Features	Reduced Features	Time Difference	% Time Reduced
RF	906	430	476	52.54%
K-NN, K=1	6936	5393	1543	22.25%
K-NN, K=2	6442	5776	666	10.34%
SVM	903	336	567	62.79%
NB	23	4	19	82.61%

Analyzing the full feature dataset, the NB algorithm had the fastest computation times for both experiments. The slowest algorithm was the K-NN algorithm in both experiments. SVM and RF had similar computation times for Experiment 1, with a less than 0.6 second difference for each fold, but SVM was the faster of the two algorithms in Experiment 2. Comparing to our reduced feature dataset, the NB algorithm still had the fastest computation times for both experiments. There was a greater deviation between the SVM and RF algorithms times for Experiment 1, with SVM being the faster of the two. However, the SVM algorithm had a large increase in computation time in Experiment 2. Due to the methods described in Section 2.4.3, the increased build time was most likely due to the calculation of each pairwise function.

Table 4.16. Algorithm Total Testing and Training Time Comparisons, Experiment 2

	Total Computation Time (s)			
Algorithm	All Features	Reduced Features	Time Difference	% Time Reduced
RF	1354	1060	294	21.71%
K-NN, K=1	19 837	14 942	4895	24.68%
K-NN, K=2	19 800	14 725	5075	25.63%
SVM	648	23 282	-22 634	-3492.90%
NB	60	15	45	75%

We next compared the percent reduction of computation time between the reduced feature set and the full feature set in each experiment, as shown in Table 4.16. For Experiment 1, the use of the reduced feature set had the greatest improvement in the NB algorithm, which improved by 82.61%. The SVM and RF algorithms had the next largest build time reductions, decreasing by 62.79% and 52.54% respectively. The K-NN algorithm had the smallest total build time reduction, decreasing as K increased. For Experiment 2, the greatest reduction in build time was again the NB algorithm, which improved by 75%. The K-NN algorithm had the second largest build time reductions, although the total build time was much longer than the RF algorithm. Because of the large increase in build time for the SVM algorithm, there was a performance decrease of 3491%.

4.3.4 Reduced Feature Set Performance Comparison

As seen in Table 4.17, reviewing the overall performance of our reduced feature set, we see a performance gain in three of the four algorithms, and a DoS detection accuracy reduction below 1% in two algorithms. Performance gain was calculated by dividing the computation time difference by the total computation time using all features.

Table 4.17. Comparison of Reduced Feature Set Performance against Full Feature Set Performance

	Performance Gain		DoS Detection Reduction	
Algorithm	Experiment 1	Experiment 2	Experiment 1	Experiment 2
RF	52.54%	21.71%	0.02%	0.05%
K-NN, K=1	22.25%	24.64%	0.02%	0.08%
K-NN, K=2	10.34%	25.63%	0.03%	0.10%
SVM	62.79%	-3492.90%	4.80%	36.47%
NB	82.61%	75%	8.81%	36.32%

If algorithm computation speed is the main factor of consideration for an organization, the NB algorithm would be the best algorithm choice using our reduced feature set while the SVM algorithm would be the worst choice. If algorithm accuracy is the main factor affecting performance, the RF algorithm would be the best choice using our reduced feature set while the NB algorithm would be the worst choice. Although the K-NN algorithm takes longer to build, the overall accuracy was above 99% for both experiments. The RF algorithm is the best compromise solution, offering faster computation times while maintaining high overall accuracy. Depending on the specific real-world implementation, this trade-off could be acceptable.

In conclusion, our reduced feature set cut testing and training times in half for the most accurate detection algorithm, RF, while incurring only minor reductions (0.02% and 0.005%) in accuracy and MCC against DoS attacks, respectively.

CHAPTER 5:
Conclusion

The security of Navy networks relies on tools and capabilities to detect and stop cyberattacks. As application layer attacks continue to become more sophisticated, machine learning can be a valuable tool to enhance the security of Navy networks by allowing analysts to detect and respond faster to network threats.

Using our reduced feature set, we have shown the detection of DoS attacks both alone and in the presence of other attack vectors maintained above 99% in the two best performing algorithms, with a DoS FAR of less than 0.08%. Another advantage of our reduced feature set is the testing and training times being cut by over 20% for three algorithms. Detection of non-DoS application layer attacks were only partially effective, which are close to [13] using full feature sets.

However, the performance of the reduced feature set is highly algorithm-dependent. In the two top performing algorithms, there were a slight decrease in overall accuracy, less than 0.3%, and a decrease of less than 0.1% in DoS detection. The addition of other application layer attacks had a significant negative impact on SVM performance and NB overall accuracy decreased by 45%. Additionally, two algorithms had a median FAR of over 20%. This shows that more features would be necessary to reduce the false alarm rate for these algorithms. NB had the highest median FAR of the four algorithms, while SVM had the most false negatives.

Our objective was to provide one-step detection of DoS attacks using a reduced set of NetFlow features. We showed that our selected features accomplished this goal. We executed our experiment in two phases and compared the results of each algorithm after cross validation. We used the metrics of accuracy, detection rate, false alarm rate, and MCC to evaluate our results.

5.1 Future Work

Cybercrime as a service has enabled DoS attacks to occur against more targets than ever before [1]. Sections 5.1.1–5.1.3 provide key areas that may yield interesting outcomes if further examined.

5.1.1 Additional Dataset Attack Types

We are limited to the assumptions from the publicly available datasets regarding benign background traffic and malicious actors operate. As such, the datasets may not be representative of behavior that is currently seen in real-world networks. A dataset generated from the NPS Cybersecurity Operations Center (CSOC) that can be used for training and testing would be of practical benefit.

Testing against recent attack data would further inform the usability of the NetFlow features. As new attacks occur, the training data can be updated to aid in prevention of similar attacks. As new attack data is analyzed, test datasets can be updated to aid detection of specific attacks. This could lower the dataset imbalance to aid detection of less frequent attack types.

5.1.2 Feature Selection Increase

Additional features from each of the other application layer attacks could help improve the detection of those attacks. This can be seen when comparing to the full feature set, where the overall accuracy was greater than 95% for all algorithms. Further analysis would be needed to optimize tradeoffs regarding which features would provide the greatest accuracy of less frequent attacks while also maintaining faster testing time.

5.1.3 Compare Results to WAF Protected Network

In addition, proprietary tools currently exist to aid in network defense against DoS attacks. A comparison of detection results between various WAF and our NetFlow features will allow for a cost-saving analysis to be performed.

5.1.4 Testing of Other Algorithms

Although the features presented in the experiment were used with four supervised algorithms, more research will be needed to evaluate whether these features can be effectively used to detect application layer DoS attacks with other supervised algorithms or with semi-supervised algorithms.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: Data

Figure A.1. Common Features Between Attack Types

Table A.1. Full List of Features From CICFlowMeter. Source: [36].

Feature Name	Description
Destination Port	Destination port of flow
Flow Duration	Duration of the flow in microseconds
Total Fwd Packet	Total packets in the forward direction
Total Backward Packets	Total packets in the backward direction
Total Length of Fwd Packets	Total size of packet in forward direction
Total Length of Bwd Packets	Total size of packet in backward direction
Fwd Packet Length Max	Maximum size of packet in forward direction
Fwd Packet Length Min	Minimum size of packet in forward direction
Fwd Packet Length Mean	Mean size of packet in forward direction
Fwd Packet Length Std	Standard deviation size of packet in forward direction
Bwd Packet Length Max	Maximum size of packet in backward direction
Bwd Packet Length Min	Minimum size of packet in backward direction
Bwd Packet Length Mean	Mean size of packet in backward direction
Bwd Packet Length Std	Standard deviation size of packet in backward direction
Flow Bytes/s	Number of flow packets per second
Flow Packets/s	Number of flow bytes per second
Flow IAT Mean	Mean time between two packets sent in the flow
Flow IAT Std	Standard deviation time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Total	Total time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction

Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Fwd PSH Flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the backward direction
Fwd Packets/s	Number of forward packets per second
Bwd Packets/s	Number of backward packets per second
Min Packet Length	Minimum length of a packet
Max Packet Length	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet

FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWE Flag Count	Number of packets with CWE
ECE Flag Count	Number of packets with ECE
Down/Up Ratio	Download and upload ratio
Average Packet Size	Average size of packet
Avg Fwd Segment Size	Average size observed in the forward direction
Avg Bwd Segment Size	Average number of bytes bulk rate in the backward direction
Fwd Header Length	Length of the forward packet header
Fwd Avg Bytes/Bulk	Average number of bytes bulk rate in the forward direction
Fwd Avg Packets/Bulk	Average number of packets bulk rate in the forward direction
Fwd Avg Bulk Rate	Average number of bulk rate in the forward direction
Bwd Avg Bytes/Bulk	Average number of bytes bulk rate in the backward direction
Bwd Avg Packets/Bulk	Average number of packets bulk rate in the backward direction
Bwd Avg Bulk Rate	Average number of bulk rate in the backward direction
Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction

Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction
Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction
act_data_pkt_fwd	Count of packets with at least 1 byte of TCP data payload in the forward
min_seg_size_forward	Minimum segment size observed in the forward direction
Active Mean	Mean time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Min	Minimum time a flow was active before becoming idle
Idle Mean	Mean time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Min	Minimum time a flow was idle before becoming active
Label	Classification of flow

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] C. Ardagna, S. Corbiaux, A. Sfakianakis, and C. Douligeris, *ENISA Threat Landscape report 2021*, A. Malatras, M. Theocharidou, I. Lella, and E. Tsekmezoglou, Eds. European Union Agency for Cybersecurity, October 2021.

[2] J. Brown, "Cybersecurity operations center (CSOC)," presented at Naval Postgraduate School, Monterey, CA, USA, 2021.

[3] X. Jing, Z. Yan, and W. Pedrycz, "Security data collection and data analytics in the internet: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 586–618, 2019.

[4] G. Mafra, "CRISP-DM and what i did wrong," Magrathea Labs, blog, Dec. 14, 2018 [Online]. Available: https://wrywhisker.pulpfriction.net/wallcrust/linear-colinear-felinear.html

[5] W. O. Chee and T. Brennan, "H.....t.....t....p....p....o....s....t," presented at OWASP AppSec DC 2010, Washington, D.C., USA, 2010.

[6] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Semantics and content," *RFC*, no. 7231, June 2014.

[7] Cloudflare, "What is a slowloris DDoS attack?" Accessed Aug. 25, 2021 [Online]. Available: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/

[8] C. Kemp, C. Calvert, and T. Khoshgoftaar, "Utilizing netflow data to detect slow read attacks," in *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, 2018 [Online]. Available: doi:10.1109/IRI.2018.00023

[9] Y. Feng, J. Li, and T. Nguyen, "Application-layer DDoS defense with reinforcement learning," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1–10.

[10] D. Beckett, S. Sezer, and J. McCanny, "New sensing technique for detecting application layer DDoS attacks targeting back-end database resources," in *2017 IEEE International COnference on Communications (ICC)*, 2017 [Online]. Available: doi:https://doi.org/10.1109/Icc.2017.7997376

[11] C. Kemp, C. Calvert, T. Khoshgoftaar, and M.Najafabadi, "Detecting slow HTTP POST DoS attacks using netflow features," in *Florida Artificial Intelligence Research Society Conference*, 2019.

[12] T. Zoppi, A. Ceccarelli, T. Capecchi, and A. Bondavalli, "Unsupervised anomaly detectors to detect intrusions in the current threat landscape," *ACM/IMS Trans. Data Sci.*, vol. 2, no. 2, Apr. Available: 2021[Online].doi:https://doi.org/10.1145/3441140

[13] A. Ahmim, L. Maglaras, M. A. Ferrag, M. Derdour, and H. Janicke, "A novel hierarchical intrusion detection system based on decision tree and rules-based models," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2019, pp. 228–233. Available: doi:https://doi.org/10.1109/DCOSS.2019.00059

[14] K. Borisenko, A. Smirnov, E. Novikova, and A. Shorov, "DDoS attacks detection in cloud computing using data mining techniques." in *Advances in Data Mining. Applications and Theoretical Aspects*, 2016, pp. 197–211. Available: https://doi.org/10.1007/978-3-319-41561-1_15

[15] I. Ullah and Q. Mahmoud, "A two-level flow-based anomalous activity detection system for IoT networks," *Electronics*, vol. 9, no. 3, Apr 2020 [Online]. Available: doi:https://doi.org/10.3390/electronics9030530

[16] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers and Security Journal*, vol. 45, pp. 100–123, Apr 2014 [Online]. Available: doi:http://dx.doi.org/10.1016/j.cose.2014.05.011

[17] M. Ring, S. Wunderlich, D. Grued, D. Landes, and A. Hotho, "Flow-based benchmark data sets for intrusion detection," in *16th European Conference on Cyber Warfare and Security (ECCWS)*, 2017, pp. 361–369.

[18] I. Sharafaldin, A. Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *4th International Conference on Information Systems Security and Privacy (ICISSP)*, 2018.

[19] I. Sharafaldin, A. Lashkari, S. Hakak, and A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *IEEE 53rd International Carnahan Conference on Security Technology*, 2019.

[20] B. Boehmke and B. Greenwell, *Hands-On Machine Learning with R*, 1st ed. Chapman and Hall, 2019. Available: https://doi.org/10.1093/acprof:oso/9780199343638.003.0004

[21] G. Lakshmi and M. Shang, *Hands-on Supervised Learning with Python: Learn How to Solve Machine Learning Problems with Supervised Learning Algorithms Using Python (English Edition)*. BPB PUBN, 2021.

[22] The MathWorks, Inc. *Statistics and Machine Learning Toolbox User's Guide*. MathWorks, Natick, MA, USA, 2021 [Online]. Available: https://www.mathworks.com/help/releases/R2021a/pdf_doc/stats/stats.pdf

[23] E. Frank, M. Hall, and I. Witten. *The WEKA Workbench: Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf

[24] S. Rajbanshi, "Everything you need to know about Machine Learning," Analytics Vidhya, blog, Mar. 25, 2021 [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/

[25] N. Settouti, M. E. A. Bechar, and M. A. Chikh, "Statistical comparisons of the top 10 algorithms in data mining for classification task," vol. 4, no. 1, 2016. Available: doi:http://doi.org/10.9781/ijimai.2016.419

[26] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann Publishers, 1993.

[27] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct. Available: 2001[Online].doi:https://doi.org/10.1023/A:1010933404324

[28] G. Gao, M. Wang, H. Huang, and W. Tang, "Agricultural irrigation area prediction based on improved random forest model," *Research Square*, Jan. 2021 [Online]. Available: doi:http://doi.org/10.21203/rs.3.rs-156767/v1

[29] J. Platt, *Fast Training of Support Vector Machines Using Sequential Minimal Optimization*, B. Schoelkopf, C. Burges, and A. Smola, Eds. Cambridge, MA, USA: MIT Press, 1998.

[30] Baeldung, "Multiclass Classification Using Support Vector Machines," Aug. 25, 2021 [Online]. Available: https://www.baeldung.com/cs/svm-multiclass-classification

[31] D. Aha and D. Kibler, "Instance based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.

[32] A. Navlani, "KNN classification using scikit-learn," Artificial Intelligence in Plain English, Aug. 26, 2020 [Online]. Available: https://ai.plainenglish.io/knn-classification-using-scikit-learn-efb34151a8b9

[33] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo, CA, USA, 1995.

[34] Canadian Institute of Cybersecurity, "Datasets," Aug. 1, 2021 [Online]. Available: https://www.unb.ca/cic/datasets/index.html

[35] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuousvalued attributes for classification learning," in *Thirteenth International Joint Conference on Articial Intelligence*. Morgan Kaufmann Publishers, 1993, vol. 2, pp. 1022–1027.

[36] A. H. Lashkari, "CICFlowMeter," Jun. 7, 2021 [Online]. Available: https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt

[37] D. Shulga, "5 reasons why you should use cross-validation in your data science projects," Towards Data Science, blog, Sep. 27, 2018 [Online]. Available: https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79

[38] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (MCC) over f1 score and accuracy in binary classification evaluation," vol. 21, no. 6, 2020. Available: doi:https://doi.org/10.1186/s12864-019-6413-7

[39] Security, "One-fifth of cybersecurity alerts are false positives," Security Magazine, blog, Mar. 15 2022 [Online]. Available: https://www.securitymagazine.com/articles/97260-one-fifth-of-cybersecurity-alerts-are-false-positives#:~:text=Eighty%2Done%20percent%20of%20surveyed

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California