Spring 5-2021

Analysis of Theoretical and Applied Machine Learning Models for Network Intrusion Detection

Jonah Baron

ANALYSIS OF THEORETICAL AND APPLIED MACHINE LEARNING MODELS FOR NETWORK INTRUSION DETECTION

A dissertation submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Cyber Operations

May 2021

By
Jonah Baron

Dissertation Committee:

Dr. Kyle Cronin, PhD
Dr. Austin O'Brien, PhD
Dr. Christopher Olson, PhD

DISSERTATION APPROVAL FORM

This dissertation is approved as a credible and independent investigation by a candidate for the Doctor of Science in Cyber Security degree and is acceptable for meeting the dissertation requirements for this degree. Acceptance of this dissertation does not imply that the conclusions reached by the candidate are necessarily the conclusions of the major department or university.

Student Name: Jonah Baron

Dissertation Title: Analysis of Theoretical and Applied Machine Learning Models for Network Intrusion Detection

Dissertation Chair/Co-Chair: Dr. Kyle Cronin Date: _____

Dissertation Chair/Co-Chair: _____ Date: _____

Committee member: Dr. Austin O'Brien Date: _____

Committee member: Dr. Christopher Olson Date: _____

Committee member: _____ Date: _____

Committee member: _____ Date: _____

ACKNOWLEDGMENTS

The completion of this Ph.D. program and dissertation would not have been possible with the support of those around me. I will be forever grateful for the generous support I received from all my family, friends, and other associates during this long journey. There have been numerous people around me who have heard me express my excitement and frustration towards completing this degree. I cannot list everyone here, but I can note the most influential advocates that sincerely helped me throughout this process. First, I want to thank my parents, James Baron and Marlene Baron, who have been my biggest supporters ever since my initial application and continued all the way through the completion of this dissertation. Next, I want to thank my previous RIT professors, Dr. Sumita Mishra and Dr. Bo Yuan, for their continued interest in my research and wellbeing throughout this program. I also want to thank my workplace supervisor, Andrew Guyan, who was the largest workplace advocate for my completion of this degree. Several friends were also unbelievably helpful and supportive throughout this program, especially those from RIT. Specifically, I want to note Jared Stroud, as he was one of the first to discuss the idea of seriously pursuing a Ph.D., even while we were still undergraduate students together at RIT. Additionally, I want to thank the friends I gained while attending my first DSU residency. I specifically want to thank Rick Matthews and TJ Nelson, who provided a fantastic support group as fellow classmates in the program. Next, I want to thank the excellent DSU faculty. Dr. Wayne Pauli is a spectacular guide and resource to all of those enrolled in this program and has genuinely made this process more manageable. I am also incredibly thankful for my first two committee members, Dr. Kyle Cronin and Dr. Austin O'Brien, who were unbelievably supportive of me and my research topic despite my initial lack of background knowledge on particular concepts of this study. Additionally, I want to thank Dr. Chris Olson, who joined my committee after my proposal and had since provided valuable feedback for my dissertation. Lastly, I want to thank my dog, Nyx, who has silently been there with me during all the ups and downs; he has truly been my greatest comfort throughout this whole journey.

ABSTRACT

Network Intrusion Detection System (IDS) devices play a crucial role in the realm of network security. These systems generate alerts for security analysts by performing signature-based and anomaly-based detection on malicious network traffic. However, there are several challenges when configuring and fine-tuning these IDS devices for high accuracy and precision. Machine learning utilizes a variety of algorithms and unique dataset input to generate models for effective classification. These machine learning techniques can be applied to IDS devices to classify and filter anomalous network traffic. This combination of machine learning and network security provides improved automated network defense by developing highly-optimized IDS models that utilize unique algorithms for enhanced intrusion detection. Machine learning models can be trained using a combination of machine learning algorithms, network intrusion datasets, and optimization techniques. This study sought to identify which variation of these parameters yielded the best-performing network intrusion detection models, measured by their accuracy, precision, recall, and F1 score metrics. Additionally, this research aimed to validate theoretical models' metrics by applying them in a real-world environment to see if they perform as expected. This research utilized a quantitative experimental study design to organize a two-phase approach to train and test a series of machine learning models for network intrusion detection by utilizing Python scripting, the scikit-learn library, and Zeek IDS software. The first phase involved optimizing and training 105 machine learning models by testing a combination of seven machine learning algorithms, five network intrusion datasets, and three optimization methods. These 105 models were then fed into the second phase, where the models were applied in a machine learning IDS pipeline to observe how the models performed in an implemented environment. The results of this study identify which algorithms, datasets, and optimization methods generate the best-performing models for network intrusion detection. This research also showcases the need to utilize various algorithms and datasets since no individual algorithm or dataset consistently achieved high metric scores independent of other training variables. Additionally, this research also indicates that optimization during model development is highly recommended; however, there may not be a need to test for multiple optimization methods since they did not typically impact the yielded models' overall categorization of

success or failure. Lastly, this study's results strongly indicate that theoretical machine learning models will most likely perform significantly worse when applied in an implemented IDS ML pipeline environment. This study can be utilized by other industry professionals and research academics in the fields of information security and machine learning to generate better highly-optimized models for their work environments or experimental research.

DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

_____

Jonah Baron

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This section introduces the topic of this dissertation on machine learning and network intrusion detection. This study evaluates a series of machine learning models for network intrusion detection to identify how the combination of machine learning algorithms, network intrusion detection datasets, and optimization parameters impacts the final models' outcome and performance. Additionally, this research created an intrusion detection system (IDS) with a machine learning (ML) capabilities pipeline such that trained ML models can be applied and evaluated in a real-world environment. A quantitative experimental study design effectively organizes the tests for this research and analyzes their results. This section discusses the background, goals, and significance of this research and briefly introduces how this study was designed and conducted.

Background of the Problem

Detecting network anomalies and malicious traffic is a significant concern and challenge for organizations today. Organizations must adequately protect private data from external threats; however, attackers continue to develop new techniques to bypass standard security practices and protocols. Organizations may employ security operation center (SOC) analysts to review network traffic and alerts to track any potential compromise within the network to address this ongoing threat (Schinagl, Schoon, & Paans, 2015). Typically, SOC analysts will review network traffic and system logs whenever they identify an anomaly within the network (Aijaz, Aslam, & Khalid, 2015). Many of these analysts rely on alerts from different systems to warn them about potential threats or anomalies found on particular devices. If a SOC analyst believes a system has been legitimately compromised, they will likely try to correlate events to establish a complete timeline of how a threat actor obtained initial access, established persistence, and exfiltrated data (Deyang & Dedong, 2011). This extensive adversary tracking may require the analyst to verify and correlate logs from

numerous systems tediously. These analysts may easily miss malicious traffic or related events while combing through the logs and security alerts between devices, as this process can be extremely challenging. Experienced threat actors will utilize different tools, techniques, and procedures (TTPs) to effectively bypass security systems and protection mechanisms. Several different methods and tools provide obfuscation and evasion capabilities that prevent detection within networks and devices. Advanced Persistent Threat (APT) groups are highly technical adversaries that will almost certainly utilize these evasion techniques to effectively blend into a victim organization's network traffic to avoid detection over an extended period.

Intrusion Detection System (IDS) devices are security systems that monitor the network for malicious or anomalous activity and generates alerts based on their configuration. There are two primary detection techniques utilized by IDS devices, misuse detection and anomaly detection (Nassar, El-Bahnasawy, Ahmed, Saleeb, & El-Samie, 2019). Misuse detection, also known as signature-based detection, will flag malicious traffic based on a vendor's unique signatures. The biggest problem for misuse detection is maintaining an up-to-date signature ruleset or database to flag known bad traffic. This type of detection method struggles to identify unknown, or zero-day, network attacks that have not had signatures developed. The benefit of misuse detection is that it has a low false alarm rate, meaning that the alerts are typically very accurate. Anomaly detection will establish a baseline of normal traffic behavior to identify abnormal traffic behavior based on the normal profile deviation. Anomaly detection, or behavior-based detection, can strongly generalize traffic, thereby enabling the capability to flag unknown zero-day attacks. The major challenges for anomaly detection are the high false alarm rate and inaccuracies. Regardless of either misuse detection or anomaly detection mode, a fine-tuned IDS should effectively flag malicious and anomalous traffic while also allowing innocuous traffic to pass through to enable normal business operations. It should also be noted that open-source IDS software operates by utilizing a primary detection engine made up of various processes that apply filters to the ingested network traffic.

An IDS configured for misuse detection will observe traffic and send alerts when specific network traffic criteria match individual signatures configured on the device. The rules or signatures on an IDS must be manually set up and regularly maintained by a user who knows what is considered "bad" traffic and "good" traffic on their network. The challenge for

these IDS misuse detection configurations is properly defining and maintaining the rules to alert on network traffic that may be malicious. Over time, threat actors and their attacks change as more vulnerabilities and exploits for systems become published; thus, many signature-based IDS devices will not utilize a relevant or up-to-date ruleset to alert against new attacks unless analysts regularly review and update the IDS alert ruleset (Sopan, Berninger, Mulakaluri, & Katakam, 2018).

An IDS configured for anomaly detection will require both setup and tuning. In a two-phase process, the IDS must be trained to build a normal behavior profile of the network and then tested to compare the current traffic to the previously generated normal behavior profile (Anson, 2020). Historically, many IDS devices and software used strict statistical models to analyze deviations from the normal traffic profile to determine if a piece of traffic is anomalous. Newer techniques for anomaly detection incorporate artificial intelligence and machine learning techniques, which will be further discussed later in this paper. An anomaly-based IDS device aims to flag anomalous traffic while maintaining a low degree of false positives false negatives. This type of configuration's primary benefit is the potential to detect unknown, or zero-day, network attacks before vendors even have published signatures. The issue is that current anomaly detection techniques employed by most IDS devices do not utilize ML classification algorithms to categorize pieces of traffic. A poorly tuned anomaly-based IDS will generate numerous false alarms until a network security analyst decides to ignore specific alerts due to the low confidence levels in the mismanaged IDS device or software.

There are numerous ways to set up and configure an IDS on a network. One of the first decisions that must be made when installing an IDS is where to set up the device on the network (Bhuyan, Bhattacharyya, & Kalita, 2014). IDS devices can be installed inline on the network, where live network traffic must pass through the system to reach the next device. The benefit of an inline setup is live traffic monitoring analysis; a potential challenge with this setup is traffic disruption due to excessive analysis on the device, thereby resulting in traffic queues. Alternatively, IDS devices can be setup on a mirror or span port, a dedicated port on a networking device configured to forward network traffic or logs. The benefit of this configuration is the lack of potential traffic flow disruption; however, since the network analysis is not live, threats may reach vulnerable devices on the network by the time the IDS

alert goes off. There are also hardware-based and software-based IDS setups. Hardware-based IDS devices are physical systems that provide dedicated machines and processing power for traffic analysis. Software-based IDS devices can be installed on a system that enables the device to perform IDS capabilities in addition to its original function; administrators may install IDS plugins onto existing networking systems or may stand up dedicated virtualized devices with IDS software to act as a dedicated virtual IDS device. Lastly, it should be mentioned that an Intrusion Detection System is closely related and commonly installed alongside an Intrusion Prevention System (IPS). IDS devices are dedicated systems intended to alert if specific network traffic patterns are detected on the network; IPS devices can be used to drop malicious packets or alter security configurations or rules on systems (such as routers or switches) if a threat is detected on the network. IPS devices can be set up just like IDS devices but are meant to react to malicious threats on the network rather than just alert (Johansen, 2020). IPS devices can be potent systems; however, these devices' primary concern is the potential disruption of innocuous traffic. IPS devices may identify a "threat" and close critical ports or restrict communication between segmented networks, thereby disrupting standard business operations. This automatic network reconfiguration could cost the organization with the IPS a significant amount of money and reputation due to downtime of communications between critical systems. It is even more challenging to properly configure an IPS to identify and react based on a static ruleset due to the potential risk of business downtime. Many organizations use a well-configured IDS to send alerts to security analysts, who then manually investigate the alerts to determine if the threat was legitimate.

While there are several different IDS hardware and software solutions, they all contain a similar syntax for IDS rules. Each IDS rule will have a set of fields configured to perform a particular action when identifying specific network criteria (Collins, 2014). For example, an IDS rule will contain specific fields to filter network packets such as a source IP, destination IP, direction of traffic, source port, destination port, and protocol. These rules can become even more granular by filtering packets based on fields unique to the protocol of the traffic or even other layers in the Open Systems Interconnection (OSI) model. In addition to these filter criteria, IDS rules will also be configured with specific rule actions (such as an alert or log if the criteria are identified in a packet), rule numbers, rule priority, rule revision, and rule classification. IDS rules can be highly specialized or extremely broad; it is up to the analyst to

finetune the IDS to properly configure these rules to alert on legitimate threats to the network correctly, or else the IDS will alert on innocuous traffic.

IDS devices can be highly effective security systems that alert on threats within a network. However, it can be an arduous and tedious process to keep the IDS ruleset updated regularly. When initially setting up an IDS, it may take weeks or sometimes months to train an IDS to create a proper baseline for the system to identify "normal" traffic (Bejtlich, 2013). However, even after the initial baseline training, security analysts will need to continuously review the network to identify typical traffic and then manually update the IDS rules to match what they find. A misconfigured IDS ruleset can result in an overly sensitive IDS that generates excessive alerts (incorporating both true positives and false positives); similarly, a misconfigured IDS can also be configured with a loose ruleset that will not alert on legitimate malicious traffic (involving false negatives and true negatives). It can take several months for a security analyst to set up and adequately fine-tune an IDS, and even after it has been configured, someone will still need to monitor the network traffic to generate up-to-date IDS configurations continuously. In an ideal world, there would be a way to automate this process of analyzing the network traffic to determine what is considered "normal" traffic and then generate an effective IDS ruleset that alerts anomalous and malicious traffic while maintaining a high level of true positive and true negative alerting.

Machine learning (ML) may provide a solution to this ongoing effort to detect new malicious traffic. Machine learning uses algorithms and statistical models to analyze or learn from training data to make decisions or predictions observed in other ingested datasets. There are several different ML algorithm types along with numerous optimization techniques to best train the ML model. ML algorithms can be categorized in different ways. There are unsupervised learning, supervised learning, and semi-supervised learning techniques (Maseer, Yusof, Bahaman, Mostafa, & Foozy, 2021). Unsupervised learning occurs when an algorithm is given an unlabeled dataset and will attempt to identify patterns or structures within the data. Human experts will label a portion of the semi-supervised learning dataset to assist the algorithm in identifying patterns better. If a dataset is completely labeled, then this can be used for supervised learning algorithms to identify a function or model that explains the data. Additionally, ML algorithms can be grouped between shallow learning methods and deep learning methods, which will be discussed later. These numerous ML learning techniques and

classification models will be further discussed in the Literature Review section. With the adequately selected ML algorithm and training dataset, a machine learning IDS device can be configured to automate the process of detecting anomalous network traffic through the use of misuse or anomaly detection. This type of IDS with ML capabilities will potentially automate detecting emerging threats and zero-day attacks utilized by attackers before the signatures are published by vendors. However, there are several challenges with the proposed IDS devices with ML capabilities due to the wide variety of possible ML algorithms and network intrusion detection datasets' quality. These variables will need to be tested and evaluated to identify the best and most effective IDS ML configuration that produces the highest true positive detection rate while keeping false positive alerts at a minimum.

Datasets are required to train and test ML algorithms. In the network intrusion detection realm, the datasets can consist of three primary types of data that make up the network traffic dataset (Ring, Wunderlich, Scheuring, Landes, & Hotho, 2019). First is packet-based data; this is commonly obtained in a standard packet capture format and contains full packet headers, fields, payloads, and associated metadata. The second is flow-based data; this is a more condensed format which stripes out unique packet properties and primarily maintains packet metadata. This type of data aggregates packets that share specific properties within a given time window into a single flow and does not contain payload information. Lastly, there is a hybrid category of network data; this data contains a mixture of both packet-based and flow-based data. An example of this hybrid traffic would be flow-based data that has been enriched to contain specific packet-based fields such as payloads and unique header information. Additional details and examples of these types of datasets will appear in the literature review of this paper.

Several evaluation metrics can be applied to machine learning algorithm results that enable researchers to compare and contrast the models' results. Before getting into these unique metrics, the four base metrics used to classify sets of labeled data are true positive (TP), true negative (TN), false positive (FP) or a statistical type I error, and false negative (FN) or a statistical type II error (Vinayakumar et al., 2019). True positive data are correctly labeled positive results; true negative data are correctly labeled false results; false-positive data are incorrectly labeled positive results; false-negative data are incorrectly labeled negative results. Machine learning researchers commonly utilize the following evaluation

metrics to compare the effectiveness of ML classification models: accuracy or proportion correct [(TP + TN) / (TP + TN + FP + FN)], positive predictive value [TP / (TP + FP)], sensitivity or recall or true positive rate or probability of detection or detection rate [TP / (TP + FN)], negative predictive value [TN / (TN + FN)], specificity or TN rate [TN / (TN + FP)], and false alarm rate or false positive rate or fall-out [FP / (TN + FP)] (Chiba, 2019). These terms and metrics will appear across several papers in the Literature Review section and will be utilized to analyze this study's results.

Deep learning (DL) is a subset of machine learning that analyses successive and meaningful layers of representations from the original data input. The "deep" in deep learning refers to the layers, or depth, of the modeled data. Modern deep learning models may incorporate hundreds or even thousands of layers, whereas other learning models (such as typical machine learning models) may only utilize one or two layers, which is sometimes called shallow learning (Thapa, Liu, Kc, Gokaraju, & Roy, 2020). These layered representations of data are typically referred to as a neural network (NN). Each layer in the NN represents the original input data in an increasingly altered and informative perspective. Applied weights, also referred to as parameters, modify the data representations at each NN layer to slightly transform the data (Akashdeep, Manzoor, & Kumar, 2017). There can be millions of parameters at each layer, and learning, in this context, actually means finding the desired values for the weights applied in the NN. Next, after all the layers have transformed the original data and produced a final result, a loss function, or objective function, is used to take the output of the NN along with the true target value (the desired outcome) to calculate the distance score, or loss score, to see how far off the NN was from the true target value. Once this distance score has been calculated, a feedback signal, called an optimizer, is used to implement a backpropagation algorithm to adjust the NN weights and hopefully lower the loss score. This process will repeat several times; the weights are initially random values, but each iteration of the NN will modify the parameter values and minimize the loss value. This training loop process repeats a sufficient number of times until NN yields a minimal loss value.

Data processing and optimization are additional significant factors when training machine learning models. There are numerous data transformation steps required to properly format the data such that it can be appropriately ingested while training an ML model (Buczak

& Guven, 2016). Additionally, each machine learning algorithm contains a series of highly configurable parameters that can be fine-tuned to optimize the trained model to yield the highest performance within a particular scoring method. As a result, models can be better optimized for particular scoring methods, such as accuracy over precision or vice versa. The optimization of ML models can dramatically change and affect the outcome of trained models. Optimizing a model for a particular scoring method may yield higher results in that specific metric; however, it may also severely impact the other evaluation metrics, both positively and negatively. It is crucial to consider the impact of optimizing a particular scoring method as it may dramatically affect the performance and other metrics of the model.

Artificial intelligence and machine learning are the future for several professional fields. The research and integration of these automation techniques into different professions are occurring now and are only bound to increase as time progresses. Properly trained ML models will identify anomalies, outliers, patterns, and trends much better than humans. This type of analysis will save organizations time, human resources, and money. The integration of ML into IDS network environments is only a small subset and implementation of the whole topic of ML. However, this research seeks to assist organizations that utilize IDS devices on their network. This research reviews different ML classification models using a series of network intrusion datasets to identify which ML algorithms most effectively identify network traffic anomalous activity. Additionally, this research develops a proof of concept of an IDS and Python scripting pipeline that applies the previously trained ML models to unseen network traffic for evaluation. The machine learning models selected for this research will include Naïve Bayes (NB), Decision Tree (DT), Random Forest (RT), Ada Boost (AB), Bagging Classifier (BC), Logistic Regression (LR), and Stochastic Gradient Descent (SGD). The datasets used to train and test each of these ML algorithms will include the KDD 99, NSL-KDD, UNSW-NB 15, CICIDS 2017, and CIDDS-001 network traffic datasets. Additionally, each model will be optimized for the following scoring methods: accuracy, precision, and F1 score. These ML algorithms, network datasets, and scoring methods will be further discussed later. Organizations can use this study's results to implement highly effective IDS pipelines by utilizing basic scripting with ML capabilities along with standard IDS logging to enable strong automated network defenses against various threat actors and attacks.

Statement of the Problem

The problems addressed by this study are the inefficiencies of current IDS configurations (Sopan et al., 2018). One of the biggest challenges for managing an IDS is investigating false positive alerts. With a properly configured IDS, every alert from the system should be thoroughly investigated by an analyst. When there is a false positive alert, this causes an organization to waste valuable time and money on analysts investigating an innocuous event; additionally, there is also a huge opportunity cost to this alert because it also delays time-sensitive investigations on true positive alerts. IDS devices can be extremely challenging to set up and maintain. When an IDS is configured for misuse detection, it can be difficult for analysts to maintain an up-to-date database of signatures. Security analysts will still need to monitor the network and manually enter IDS rules to refine further the configurations to match any network changes. Additionally, when an IDS is configured for anomaly detection, creating a baseline profile of regular traffic can take an excessive amount of time. The typical anomaly detection engine only applies fundamental statistical analysis to the data to identify deviations from the normal traffic profile. Even after tuning the IDS for misuse detection or anomaly detection, security analysts will need to ensure that the IDS ruleset is not too strict or too loose as either option will result in over-alerting or under-alerting. Machine learning provides a solution to these IDS challenges as it can automate the process of training, configuring, and maintaining an IDS device with a high level of confidence for identifying anomalous traffic (Dangi et al., 2020).

An organization should better identify malicious network attacks by utilizing an intelligent IDS with ML capabilities. An IDS with ML capabilities will still generate alerts for security analysts to review. However, this system will correlate and identify minute anomalies found in the network traffic at a much more in-depth level than any individual. This implementation of ML techniques for IDS devices should maintain a high degree of confidence for detecting legitimate threats or anomalies within the network. Additionally, machine learning models for network intrusion detection observed in the literature have not been adequately trained and tested against multiple datasets in a standardized and repeatable test environment. Evaluating the machine learning models and datasets in these controlled

experimental settings is extremely important to determine the most effective way to detect malicious traffic. This research has the potential to impact any organization that utilizes IDS devices to secure its networks. This study's results can be used by organizations to identify which ML classification models are the most effective at intrusion detection and then implement an IDS device with ML capabilities to create a highly efficient detection engine. This implementation within organizations will enable higher levels of anomaly and malware detection across the network while requiring significantly less workforce to maintain the IDS and investigate alerts. Additionally, this study's results could affect the future development and implementation of IDS devices or software worldwide. Developing a machine learning, or intelligent, IDS that applies ML classification techniques for network intrusions will significantly improve IDS alerts' reliability and reduce the human resources necessary to maintain proper IDS devices and rulesets within organizations. Over time and with proper refinement, these types of intelligent IDS ML pipelines could potentially flag zero-day attacks before vendors even publish signatures.

Objectives of the Project

This research compares multiple machine learning algorithms against a series of datasets associated with network intrusion detection and different optimization methods to determine which classification models are the most effective for detecting network attacks. In addition to this goal, this research also created an IDS pipeline to apply the machine learning techniques for intrusion detection on unseen network traffic. The first phase of this research involved training multiple ML models by utilizing a series of machine learning algorithms, multiple training datasets, and different optimization methods to evaluate each models' different properties. The datasets used in this research are taken from multiple organizations that have produced popular datasets and are used for academic papers and research within the machine learning and network security communities. A common trend later seen in the literature review of ML intrusion detection techniques is a lack of consistent experimentation and critical analysis due to the researchers' selection of ML models and the datasets. It is later observed in the literature review that several researchers implement their custom ML algorithm and training dataset to highlight their own techniques' effectiveness. Using multiple

datasets across a series of controlled and repeatable experiments that evaluate ML algorithms and intrusion detection datasets provides a fair comparison to determine which IDS ML implementation is the most effective.

This study includes each experiments' results and analysis, which evaluate each ML model's selected metrics. Each experiment for the first phase involved developing a series of ML models that combine different ML classification algorithms, networking datasets, and scoring techniques. The results of these tests were collected and analyzed to identify the strengths and weaknesses of each ML model. Additionally, this research investigated the feasibility of applying trained ML models on unseen network traffic and flagging malicious traffic. The research deliverables include an in-depth analysis comparing and contrasting the ML models and results and applying them to new traffic. Several evaluation metrics are discussed in the literature review. However, this research primarily focuses on accuracy, precision, recall, and F1 score as evaluation metrics. This research aims to create a better anomaly detection IDS that utilizes machine learning over statistical analysis. Additionally, this study identifies which combination of ML classification algorithms, IDS training datasets, and different optimization methods generate the highest performing ML models based on the selected evaluation metrics. This research also showcases the need to utilize various algorithms, datasets, and optimization methods during ML model development. Another goal of this study was to validate the ML models from other researchers, as seen throughout the literature review. Lastly, a major goal of this study was to test the theoretical ML models in an applied environment to validate their theoretical accuracy for identifying malicious content on previously unseen network traffic.

Nature of the Study

This research study compared multiple ML models to determine which combination most effectively identifies anomalous or malicious network behavior to be implemented into an applied IDS environment with ML augmentation. The ML algorithm experiments consist of training an ML classification algorithm against a series of network intrusion detection datasets and scoring methods. The ML classification algorithms selected for this research include Naïve Bayes (NB), Decision Tree (DT), Random Forest (RF), Ada Boost (AB),

Bagging Classifier (BC), Logistic Regression (LR), and Stochastic Gradient Descent (SGD). Additionally, the datasets selected to train and test each of these ML models include the KDD 99, NSL-KDD, UNSW-NB 15, CICIDS 2017, and CIDDS-001 datasets. Lastly, the optimization methods used for each of these models include accuracy, precision, and F1 score. These ML models, IDS datasets, and scoring techniques were selected based on the research experiments, field surveys, and authors' recommendations found in the Literature Review section. These ML experiments were conducted through Python's scikit-learn library and utilized the same system resources within a virtual environment ("scikit-learn," 2020). A significant priority for this research was to maintain consistent testing and resource utilization between each experiment. This research seeks to achieve reliable and repeatable results where others can utilize the same ML algorithms, datasets, and scoring techniques to generate the same models. Once the ML models were generated and analyzed with a set of evaluation metrics, the next step in this research was to apply the trained models in an IDS pipeline environment to see if the models' metrics hold up in an applied network environment filtering previously unseen traffic. The IDS software selected for this research was Zeek, a popular open-source IDS software ("zeek," 2020).

This study sought to achieve repeatability and consistency between each ML model experiment. Additionally, this study strived to identify the most practical and efficient IDS implementation that utilizes ML techniques for anomaly detection on network traffic. Several researchers have conducted ML experiments to showcase the efficiency of their proposed algorithm. Many of these research papers, seen later in the literature review, select a single dataset, typically the KDD 99 or NSL-KDD dataset, and between one to five popular ML algorithms to compare their results and highlight their newly proposed ML algorithm. There have been several issues identified with the KDD 99 and NSL-KDD datasets despite being the most popular datasets in this field of ML and network intrusion detection (Creech & Jiankun, 2013). The KDD 99 dataset was initially produced in 1999 as part of a data mining competition and provided the community with an acceptable IDS training dataset. This dataset is still used today as it provides several essential proofs of concept found in various networking techniques and applications. However, despite its popularity, researchers have found that the KDD 99 dataset contains several issues (Haider, 2017). Some of these issues include redundant records that skew ML algorithms' training, imbalanced attack categories

and distribution, poorly defined attack categories and techniques, old and impractical network traffic, and nonstandard or hybrid data that consists of flow-based network traffic that has been manually enriched with payloads. The NSL-KDD dataset was built on the KDD 99 dataset. This updated dataset addressed some of these issues, but some of the core problems persisted. Certain researchers have strongly encouraged others in this field to utilize multiple network intrusion datasets, especially ones that have been recently produced, to evaluate ML classification algorithms effectively (Ring et al., 2019). Over the past few years, numerous organizations have produced robust datasets intended to be used for ML training and IDS evaluation. The Canadian Institute for Cybersecurity (CIC), based out of the University of New Brunswick (UNB) in Fredericton, has produced multiple IDS evaluation datasets intended to be reliable ML test and validation datasets for academic research. CIC also created the NSL-KDD dataset, which patched some of the significant issues with the dataset. Additionally, in Germany, Coburg University has been producing and updating the Coburg Intrusion Detection Data Sets (CIDDS) repository, which contains multiple ML datasets designed for network intrusion detection research. The University of South Wales in Australia produced an intrusion detection dataset in 2015 that was also well accepted by the ML and network security communities. These datasets are all public and provide a highly needed update for network intrusion datasets. The KDD 99 and NSL-KDD datasets are still considered the most popular. However, the datasets produced by these schools and organizations provide the academic community stronger datasets that contain well-balanced and adequately distributed network attacks, modern network traffic and attacks, and properly labeled data. Again, this research utilizes the KDD 99, NSL-KDD, UNSW-NB 15, CICIDS 2017, and CIDDS-001 datasets for consistent analysis between ML models. These datasets provide an extensive and unique comparison between each tested ML model experiment since each dataset consists of different data types (packet-based, flow-based, hybrid), network traffic, and attack techniques. Ideally, this study's results show other professionals in the ML and network security communities the need to utilize multiple datasets as a standard best practice for future ML IDS research and evaluations. There will never be a "perfect" intrusion detection dataset that will always be the best dataset to train a particular model. Multiple datasets should be used to compare and identify the ideal dataset for an ML algorithm. This research finds multiple options to implement an IDS with ML capabilities. This study applies

the trained ML models from the previous experiments to see if the same results are observed in the detection models' applied implementations. The following section's Literature Review reveals that many academic researchers only evaluate ML algorithms using a single dataset, yielding purely proof of concept or theoretical detection results. Most researchers fail to apply their ML models in a real-world scenario to validate the results they obtained. This research sought to validate the theoretical results by implementing an ML IDS and comparing those results with the previous ML experimental results.

While this research can benefit numerous industry organizations and academia's security development, this study also has several significant limitations. The largest limitation of this research is that, while a primary goal is to maintain consistency between ML datasets, each organizations' network will consist of highly unique traffic. These variations of networks mean that even though the testing and validation phases in this research may yield effective detection results, that does not necessarily mean this type of IDS with ML capabilities can be easily implemented into another network with different network traffic and maintain its results without finetuning and training multiple ML models. Additionally, even if an organization successfully implements an IDS with ML augmentation like this research, security analysts will still be required. This type of ML IDS environment benefits from the high confidence of automated IDS alerts and anomaly detection. This high confidence IDS configuration means that organizations will not need to hire and maintain as many SOC analysts. However, some analysts will still be required to investigate the alerts further, update or modify the ML models, and check the IDS ruleset regularly. In theory, this should require significantly fewer analysts, thereby saving organizations a substantial amount of manpower and money on an automated IDS with ML capabilities. It should also be noted that this research focuses on utilizing only supervised machine learning techniques that require labeled datasets. That means that all other machine learning methods, including deep learning techniques, a major field of research, are not incorporated into this particular research. There are extensive research and funding in the subset field of deep learning; however, this is excluded from this research due to the intense resource and time requirements to develop and configure deep learning models properly.

This concludes the Introduction section of this paper, which reviewed the background, goals, significance, and study implementation. This study evaluates the effectiveness of

multiple ML models when trained using various ML algorithms, IDS datasets, and optimization techniques. Additionally, this study identifies multiple ways to apply trained IDS ML models. This research performs an in-depth quantitative analysis of ML models' outcomes and performance across two primary research phases. This study highlights the need for researchers to utilize multiple algorithms, datasets, and optimization methods during ML model development. Additionally, this study's results identify which combination of ML algorithms, datasets, and scoring methods yields the best evaluation metrics in terms of accuracy, precision, recall, and F1 score. Academic researchers and industry professionals can use this research to implement highly effective and autonomous ML IDS environments that utilize the most effective ML classification models to identify anomalies and threats on their unique networks. The next section of this paper will provide a literature review for the existing research conducted in network security and machine learning.

Terms and Definitions

- Intrusion Detection System (IDS) – Device or software that monitors systems for malicious activities and generates alerts if anomalous traffic is detected
- Intrusion Prevention System (IPS) – Device or software that changes network or host behavior or security rules if anomalous traffic is identified
- IDS Ruleset – A series of IDS rules that are used for signature detection; each rule contains a set of filter criteria to alert on certain observed traffic or behaviors
- Misuse detection / Signature-based detection – IDS detection technique that requires the utilization of an IDS ruleset database to alert based on strict rule criteria; usually highly accurate, but suffers from successful zero-day attack detection
- Anomaly detection / Behavior-based detection – IDS detection technique that requires proper baselining of normal network traffic to identify anomalies, typical IDS engines will utilize statistical standard deviation detection to identify anomalous behavior; more capable of detecting zero-day attacks but suffers from inaccurate or low detection

- Hybrid detection – utilizes both signature-based and behavior-based detection methods to provide enhanced detection; both of these methods can be implemented at the same time on the same IDS
- Flow-based datasets – condensed packets that are grouped based on different header field information or network protocols; no payloads are included in network flows
- Packet-based datasets – entire IP packets that include full header information and payloads
- Hybrid/Other NID datasets – enriched network packet capture that contain both flow-based and packet-based data, typically requires manual alteration of the data
- Machine learning (ML) – algorithms that can perform a task by producing a model and inferring future data without the need for explicit instructions
- Dataset features – unique properties or attributes belonging to a dataset; these unique features can be identified and labeled for ML algorithms
- Supervised learning – a type of ML algorithm that ingests fully labeled datasets
- Unsupervised learning – a type of ML algorithm that can ingest unlabeled datasets
- Semi-supervised learning – a type of ML algorithm that ingests partially labeled datasets
- Shallow learning – typical machine learning algorithms that only utilize a single layer or round of analysis on ingested data; typically considered supervised learning that utilizes labeled datasets
- Deep learning – machine learning algorithms that perform a series of transformations or analysis on ingested data across multiple layers or rounds; typically considered unsupervised or semi-supervised learning that utilizes unlabeled or partially labeled datasets
- Training set – a subset of a full dataset used for training the ML models
- Testing set – a subset of a full dataset dedicated for testing and validating the previously produced ML models
- Type I error = FP
- Type II error = FN
- Accuracy / Portion correct = $\frac{TP + TN}{TP + TN + FP + FN}$

- Precision / Positive predictive value = $\frac{TP}{TP + FP}$
- Recall / Sensitivity / True positive rate / Probability of detection / Detection rate = $\frac{TP}{TP + FN}$
- Negative predictive value = $\frac{TN}{TN + FN}$
- Specificity / TN rate = $\frac{TN}{TN + FP}$
- False alarm rate / False positive rate / Fall-out = $\frac{FP}{TN + FP}$
- F1 Score = $2\left(\frac{precision * recall}{precision + recall}\right)$

CHAPTER 2

LITERATURE REVIEW

There have been significant developments in machine learning techniques for network intrusion detection in recent years. Several researchers worldwide have conducted valuable experiments and produced exciting papers that perform a critical analysis of their results. There are several essential literature pieces to review in this research field to understand modern machine learning capabilities and limitations. This section will review impactful conference proceedings and articles in the fields of machine learning and network intrusion detection.

Network Intrusion Detection Datasets

A dataset is required before the selection of any machine learning model. A machine learning algorithm cannot do anything unless it has multiple datasets to train, validate, and test the models. Every research paper covered in the upcoming subsection will utilize a dataset to input into a particular machine learning algorithm. This subsection will cover the most popular datasets used for training machine learning algorithms focused on network intrusion detection. Over the years, there have been individual datasets have become more prominent than others.

The paper titled “A survey of network-based intrusion detection data sets” provides an extremely comprehensive literature review of 34 datasets used for ML algorithms and network intrusion detection (Ring et al., 2019). The authors of this paper begin by introducing network attacks and network intrusion detection systems. They then review a series of other papers that have analyzed network intrusion detection datasets and highlight the need for a more thorough review of available datasets. The authors then discuss the type of data that makes up certain NID datasets. Packet-based data is commonly obtained in a packet capture format and contains payloads and associated metadata depending on the protocols utilized. Next, flow-based data is a more condensed format and primarily contains meta-information

about network connections. Flow-based data aggregates packets that share certain properties within a time window into an individual flow and does not contain payload information. The authors categorize the last set of data as “other data,” which incorporates all datasets that are neither purely packet-based nor flow-based. An example of these datasets may include flow-based data enriched with additional information from packet-based data or host-based log files. The authors state that the well-known KDD CUP 1999 dataset falls under this “other” data category because each data point contains both network-based attributes as well as enriched host-based attributes. Next, the authors identify a set of properties to compare and contrast datasets. Each of the 15 properties can be grouped into one of five categories. The authors identified the following properties in datasets: general information (year of creation, public availability, normal user behavior, and attack traffic), nature of data (metadata, format, and anonymity), data volume (count and duration), recording environment (kind of traffic, type of network, and complete network), and evaluation (predefined splits, balanced, and labeled). The researchers describe each of the properties and then display a three-page long table that displays 34 datasets associated with network intrusion detection and their 15 properties. They also have another large table in the paper that lists the 34 datasets and information about whether they are labeled and the data format (packet, flow, or other). There is another table that also lists all the datasets and describes the attacks contained within each. Lastly, the authors list another figure that shows certain relationships between associated datasets. After reviewing each dataset’s properties, the authors then move into the bulk of their paper, which briefly describes each dataset. Next, they review each of the data sources that may have generated certain datasets and different traffic generator programs that could have also been used to generate synthetic traffic data in certain datasets. The researchers then provide a series of observations and recommendations based on their comprehensive comparative analysis of collected datasets. They state that the perfect dataset does not exist and probably never will be created. Rather, they suggest researchers utilize multiple datasets to prevent overfitting with a single particular dataset, reduce the influence of artificial artifacts within a certain dataset, and evaluate the ML algorithm methods in a more general context. They also recommend using CICIDS 2017, CIDDS-001, UGR’16, and UNSWNB15 datasets as they may be suitable for evaluation settings due to their wide variety of attacks and data formats. They also recommend for researchers to make use of predefined subsets. Many ML

and data mining methods often use 10-fold cross-validation, which divides the dataset into ten equal subsets. One subset is used for testing, and the other nine are used for training; this procedure is repeated ten times such that every subset has been used once for testing. In an IDS evaluation dataset, this splitting technique would cause each attack to appear in the training data set, resulting in an improper generalization of attacks since they should detect previously unseen threats. This paper's researchers recommend creating meaningful training and test splits to facilitate comparisons of different approaches evaluated on the same data set. The authors also call for a closer collaboration of the security community to create and better new intrusion detection datasets. The community could benefit from a single generally accepted platform for sharing intrusion detection datasets without access restrictions. The authors also state that all the mentioned datasets and repositories mentioned in this paper can be found on the researchers' websites and future datasets. Next, the researchers found that most NIDS require standard input data formats and cannot handle preprocessed (enriched) data, and it is unclear if datasets from the "other" datatype category can be calculated in real-time. They recommend the community generating network-based datasets in standard packet-based or flow-based formats as they can be captured in real network environments. However, if researchers still produce "other" datatypes, the authors of this paper recommend publishing both the network-based data in a standard format as well as the scripts for transforming the dataset to another format. Lastly, the authors call for further anonymization of data within datasets and the publication of all future datasets so that other parties can utilize and validate them. This paper was one of the best sources for reviewing datasets associated with network intrusion detection. The authors conducted an extensive and comprehensive survey of public and private datasets and presented all 34 datasets in apparent formats for researchers to easily pick and choose the datasets that will work best for their research.

In his 2000 publication, researcher and author McHugh conducts an extremely in-depth analysis of the popular 1998 and 1999 DARPA datasets used for network intrusion detection and identifies a series of flaws associated with them (McHugh, 2000). This particular publication is the most referenced document by other authors and researchers in this literature review. At the time, the most comprehensive evaluation of research on intrusion detection systems was performed by MIT's Lincoln Laboratory and sponsored by DARPA. The author claims that while the research conducted contains many flaws, it was only a large-

scale attempt at evaluating IDS devices at that time, and it provides a basis for comparing other systems and datasets. This paper also primarily focuses on the 1998 evaluation and only briefly discusses the 1999 evaluation. The author then reviews other similar evaluation and dataset generation attempts conducted by other organizations and researchers. Next, he then moves into the actual evaluation of the dataset. The background and the attack data within the dataset were heavily synthesized. This generated data was intended to simulate data for present and future traffic, and as a result, this may have caused a bias towards unrealistic expectations for IDS devices. The data generated for the 1998 evaluation consists of two parts, background data that is supposed to be regular noise traffic and attack data that is intended to consist purely of attack scenarios. These two individual subsets of data were then further discussed, and the author reviewed the impracticality of each of these sets of traffic due to a large amount of synthesized data generation. He also mentions a severe lack of validation of the experiments and observed traffic for both the background and attack datasets. Much of the background and attack data were generated from scripts and programs collected from several sources, and there was no attempt to distribute the synthetic attacks in the background noise realistically. The simulated background data was supposed to represent the network traffic found in a typical Air Force Base. However, Lincoln Labs' information seems to differ based on the observed hosts within the traffic. The author states that it is unclear if the actual complexity of the network devices matters. He explains that the burden is typically on the researchers conducting the experiments to prove that the artificial environment did not affect the outcome by typically performing a controlled pilot study. The Lincoln Lab evaluators never conducted this type of controlled validation study. The experiment evaluators prepared datasets for training and testing. The training set consisted of seven weeks of data covering 22 hours per day, five days per week. The author then reviews the taxonomy of the attacks performed in this dataset, including denial of service, remote to user, user to superuser, and surveillance/probing attacks. Additionally, these attacks were further characterized by certain mechanisms used, including masquerading, abuse of a feature, implementation bug, system misconfiguration, and social engineering. The author then explains that this taxonomy describes the kinds of attacks that can be conducted on systems or networks, but it is not useful in describing what an IDS might see. This attacker-centric taxonomy creates a highly unrealistic evaluation bias. The author proposes alternative

taxonomies for future researchers where attacks could be classified at the protocol layer. This type of approach will lead to a better understanding of what one must do to detect attacks on a network. He also proposes another approach to classify attacks based on whether a completed protocol handshake is necessary out the attack. This classification technique will separate attacks into two distinct classes, one that may reveal a spoofed address or something that requires the attacker to reveal an actual location on the internal network. The author discusses how the Lincoln Lab evaluators' attacker-centric taxonomy results in unclear attack scenarios within the network. For example, a packet that causes a buffer overflow may not necessarily be an intentional attack. Next, the author discusses the evaluation of the 1998 dataset. He explains that the dataset consists of raw TCPdump data collected from a sniffer device within the network. He also explains that this tool's usage could also be problematic because it can become overloaded and drop packets; however, this is not a major concern for this evaluation because of the low data rates within this experiment. The Lincoln Lab team used a scoring metric referred to as the Receiver Operating Curve or Relative Operating Characteristic (ROC) method to present their results. The author explains this metric and states that it is commonly used for measuring signal-to-noise and alarm detection. He also reviews and critiques a series of other metrics used by the Lincoln Labs team to evaluate their experiments. The author then briefly reviewed the 1999 evaluation dataset; however, he did not go into much detail because the preliminary results were just released, and he did not have time to review the data before this publication. He states that the scoring method for constructing ROC curves, used for the 1998 and 1999 datasets, is inappropriate due to the detection process used by many IDS devices. He also states that the scanning/probing scenarios should not always be associated with attacks nor always labeled as intrusions. The author concludes this paper by restating that the Lincoln Lab IDS evaluation program was a major and impressive undertaking, but it still contained several experimental flaws, and the results remain unclear. He states that several other researchers attribute their success to this IDS evaluation publication, whereas other researchers believe this evaluation harmed their research efforts. The 1999 and 1998 evaluations demonstrate that IDS devices, at the time, are inferior at detecting new attacks. The author and an anonymous review state that DARPA could have obtained the same results with much less effort and resources than what was dedicated to this research. DARPA failed to obtain significant IDS results and breakthroughs

with this research. However, the author hopes that this critique will lead to a rethinking of the evaluation process and a recreation of its form to help DARPA reach its future IDS development goals.

In the year 2000, researchers McHugh, Christie, and Allen publish a paper that reviews IDS devices in-depth and includes an extensive discussion of why, how, and where they are utilized in an enterprise organization (McHugh, Christie, & Allen, 2000). Additionally, the authors discuss the two primary intrusion detection methods consisting of signature-based and anomaly-based detection techniques. They review each technique's pros and cons and discuss how IDS tools can be configured to operate at the network-level or host-level. After this in-depth discussion of IDS devices, they then discuss organizations' resources to set up and configure these systems correctly and their role as part of an overarching defense-in-depth strategy. The IDS lifecycle includes evaluation and selection, deployment, operation and use, and maintenance. The authors then review popular IDS tools and review intrusion detection experiments conducted by different organizations. The authors state that, at the time, the most comprehensive evaluations of IDS devices were conducted in 1998 and 1999 and performed by MIT's Lincoln Laboratory. These IDS evaluation experiments were funded by DARPA and had researchers utilize a packet capture of sniffed network traffic containing simulated (virtualized) and physical machines that were launching attacks. The attacks were divided into four categories: denial of service, remote to local, user to root, and probing/surveillance. In the 1998 evaluation of select IDS devices, the best IDS could only detect 75% of the attacks and generated multiple false alarms. The 1999 IDS evaluation produced slightly better results for detection and false alarm rates for the tested devices. The 1998 and 1999 evaluations indicated that IDS devices, at the time, were only moderately successful at identifying known intrusions or attack patterns and were much less successful at identifying previously unseen attacks. The authors conclude this paper by stating that new intrusion detection techniques and systems are being heavily researched, and they anticipate positive improvements for IDS devices in terms of enhanced detection and false alarm performance in the future.

In the paper titled "A Detailed Analysis of the KDD CUP 99 Data Set," the authors review one of the most popular datasets used for evaluating ML intrusion detection methods (Tavallaee, Bagheri, Wei, & Ghorbani, 2009). This dataset is the most commonly selected

dataset for testing, as seen throughout this literature review. The paper introduces the KDDCUP'99 dataset and how other researchers have used it to develop and evaluate machine learning methods to obtain a high detection rate while maintaining a low false alarm rate. The authors of this paper identify two primary issues with this KDDCUP'99 dataset. The first problem they identify is that the KDD dataset has many redundant records, which causes learning algorithms to become biased towards more frequent records during the training phase. Additionally, the authors identify another issue with the dataset: labeling the KDD dataset records. They found that even when using straightforward machine learning methods, there will be a minimum classification rate of 86% for properly labeling records meaning that the comparison of IDS devices will always be between 86% and 100% when using this dataset. The authors then propose a solution to the two identified problems. Developing a new dataset made up of selected records from the KDDCUP'99 dataset; they call this new dataset NSL-KDD. The authors then review the KDD dataset and discuss the type of content within the dataset. The KDDCUP'99 dataset contains attacks that fall into the following categories: Denial of Service (DoS) Attacks, User to Root (U2R) Attacks, Remote to Local (R2L) Attacks, and Probing Attacks. DoS attacks are when an attacker disrupts or denies system resources to legitimate users. U2R attacks are a form of privilege escalation where the attacker starts as a regular user with standard permissions and can exploit a vulnerability to gain root access on a device. R2L attacks occur when the attacker can send packets to a target device and exploits the system to gain access. Lastly, probing attacks are a way to gather information about a network or system. Additionally, the KDD dataset contains three primary features: basic, traffic, and content features. Basic features incorporate all attributes that can be extracted from a TCP/IP connection. Traffic features are time-based and are generated within a certain window timeframe from a specific host or service, and content features are individual events or unique fields that occur across the network or individual connection. The authors explain some of the existing issues that remain in both the KDDCUP'99 and DARPA'98 datasets, both of which are popular datasets for evaluating intrusion detection techniques. The first issue they identify in the datasets is the amount of synthesized data, which does not correctly reflect traffic observed in the real networks. The second issue is the traffic collection tools used to collect these datasets can be overloaded and dropping packets, and there was no work done to check the possibility of dropped packets. The third issue is the

lack of exact definitions for the attacks performed in the network traffic. The authors then review the two primary issues they want to address with their new dataset: redundant records and the level of difficulty for classification. The authors provided a solution to these problems by removing all the redundant records to remove the bias towards frequent records. They also randomly sample records from each difficulty-level group, such as the selected records are inversely proportional to the percentage of records in the original KDD dataset. This sampled selection of records enables the classification rates of machine learning methods to vary in a broader range and have a more accurate evaluation of learning techniques. This paper provided the most comprehensive analysis and valuable commentary on the KDDCUP'99 dataset. The KDD'99 and NSL-KDD datasets proposed by these authors will be observed in several upcoming experiments and research papers.

Machine Learning Research Surveys

The following publications consist of surveys or reviews that compare machine learning techniques used for network intrusion detection. The authors of these surveys do not conduct any experiments that produce results, but rather, they extract the results from several other publications and analyze that data. Some of these surveys are more comprehensive than others, but they do a great job showcasing the most popular machine learning techniques and network intrusion detection datasets.

The research paper titled "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection" provides an extremely in-depth analysis of the available machine learning models and datasets used for network intrusion detection (Buczak & Guven, 2016). This comprehensive paper first reviews three main types of detection analytics used by IDS devices: misuse-based (or signature-based), anomaly-based, and hybrid. The researchers then introduce machine learning and data mining (DM) and how they relate to the field of cybersecurity. They also discuss several processes for implementing machine learning and data mining techniques. They identify certain metrics used to evaluate binary classification models: accuracy or proportion correct [(TP + TN) / (TP + TN + FP + FN)], positive predictive value [TP / (TP + FP)], sensitivity or recall or true positive rate or probability of detection or detection rate [TP / (TP + FN)], negative predictive value [TN /

(TN + FN)], specificity or TN rate [TN / (TN + FP)], and FAR or FP rate or fall-out [FP / (TN + FP)]. The researchers also identify certain metrics used for multi-classification models: overall accuracy (exemplars classified correctly, all exemplars), class detection rate (exemplars from a given class classified correctly, all exemplars from a given class), and class FAR or class FP rate (exemplars from a given class classified incorrectly, all exemplars not from a given class). Next, the researchers briefly review the available cybersecurity datasets used for ML; this includes packet-level data, NetFlow data, and public datasets. Packet-level data includes the 144 Internet Protocols listed by the Internet Engineering Task Force (IETF) and incorporates all data across all OSI model layers stored within an individual packet. NetFlow data was introduced by Cisco and contained compressed and preprocessed data of actual network packets. The public dataset section mentions the DARPA 1998 and 1999 datasets, KDD 1999 dataset, and NSL-KDD dataset. MIT's Lincoln Laboratory created the DARPA 1998 and 1999 sets and contained a mixture of simulated and real network traffic, systems, and attacks. The DARPA 1998 data were collected for nine weeks, the first seven weeks assigned as the training set and the last two assigned as the testing set. The DARPA 1999 dataset was collected for five weeks, with the first three weeks assigned as the training set and the last two assigned as the testing set. The KDD 1999 dataset was created for the KDD Cup challenge in 1999 and is based on the DARPA 1998 dataset. The KDD 1999 dataset was found to have limitations, including synthesized network and attack data (due to actual traffic sampling), an unknown number of dropped packets, vague attack definitions, and a huge amount of redundant records. The NSL-KDD dataset builds off the KDD 1999 dataset and consists of selected records to address some of the issues identified. All three of these datasets contain four types of attacks as defined by DARPA, denial of service (DoS), user to root (U2R), remote to local (R2L), and probe or scan attacks. DoS attacks attempt to deny or disrupt systems or resources, U2R attacks grant privileged access to an attacker, R2L attacks grant local network access to an attacker, and scan attacks collect information about the network resources. The next section of the paper reviews ML and DM methods used for cybersecurity. The authors provide an extensive explanation of how the following methods operate: artificial neural networks, association rules and fuzzy association rules, Bayesian network, clustering, decision trees, ensemble learning, evolutionary computation, hidden Markov models, inductive learning, Naïve Bayes, sequential pattern mining, and support

vector machine. Within each of the descriptions for these ML methods, the authors also cite two to three sources that utilize the technique for misuse, anomaly, and hybrid detection. Artificial neural networks are composed of interconnected artificial neurons that are capable of certain computations on their inputs. Each layer passes its output to the next layer. Additionally, each layer performs a series of intrusions or computations on the input data such that it is slightly transformed layer to layer. Association rules describe a relationship among different attributes found among the data. Association rule mining attempts to discover previously unknown associate rules from the input data. A Bayesian network is a probabilistic, acyclic graphical model representing the variables and associated relationships from input data. Clustering is a set of techniques for finding patterns in unlabeled data. This method is considered unsupervised pattern discovery, where the input data are group together based on a similarity measure. There are several different approaches and algorithms for clustering input data, including connectivity models, distribution models, density models, and graph models. A decision tree is a tree-like structure that has leaves and branches. The leaves represent classifications, and the branches represent the conjunctions of features that lead to those classifications. Data input is labeled (classified) by testing its feature (attribute) values against the decision tree's nodes. According to this author, the best-known methods for automatically building decision trees are the ID3 and C4.5 algorithms. Ensemble learning methods combine the output of multiple weak learner models to form a stronger learner. There are several ensemble methods to develop and pick a strong learning output. Adaptive boosting (AdaBoost), bagging (or bootstrap aggregating), and random forests are all ensemble techniques that choose the best output developed from a series of weaker models. Evolutionary computation encompasses genetic algorithms, genetic programming, evaluation strategy, particle swarm optimization, ant colony optimization, and artificial immune systems. Evolution computation builds models based on systems observed in nature. Genetic algorithms and genetic programming algorithms are the two most widely used evolutionary computation methods. These algorithms are based on the principle of survival of the fittest. They operate on a population of individuals or chromosomes, which are evolved using certain operators. Markov chains and hidden Markov models are within the same category. A Markov chain is a set of states interconnected through transition probabilities that determine the model's topology. A hidden Markov model is a statistical model where the system is assumed

to be a Markov process with unknown parameters, and the goal is to identify the hidden parameters derived from the observable parameters. Inductive learning starts with specific observations and measurements, begins detecting patterns and regularities, formulates a tentative hypothesis to explore, and develops some general conclusions, theories, or models. Several ML algorithms are inductive, but researchers typically refer to Repeated Incremental Pruning to Product Error Reduction (RIPPER) and quasi-optimal (AQ) when discussing inductive learning. Naïve Bayes classifiers are probabilistic classifiers where input features are assumed to be independent, and the conditional probabilities form the classifier model. This model assigns classification labels to the input data features. Sequential pattern mining is a data mining method associated with transactional databases where each transaction has a temporal ID, a user ID, and an itemset. An itemset is a set of distinct items purchased in a transaction, and a sequence is an ordered list of itemset values. The length of a sequence is defined as the number of itemset values within the sequence. Additionally, the time ID determines the order. A sequence is considered maximal in sequences if it is not contained in any other sequence. A support vector machine is a classifier based on separating a hyperplane in the feature space between two classes so that the distance between the hyperplane and the closest data points of each class is maximized. This technique is based on a minimized classification risk rather than an optimal classification. After this discussion of ML techniques, the author then analyzes and charts the collected sources to generate figures that depict the ML method, listed paper/author, intrusion detection technique (misuse, anomaly, or hybrid), and the dataset utilized by the researchers. The researchers for this research survey did a wonderful job identifying the datasets and ML techniques available for network intrusion detection. This published research is one of the most comprehensive papers that categorizes and explains ML methods in addition to collecting, categorizing, and charting numerous other research papers related to ML network intrusion detection.

In their 2018 paper, researchers Shashank and Balachandra collect and analyze various research publications based on ML intrusion detection techniques (Shashank & Balachandra, 2018). They begin their paper by discussing types of machine learning algorithms and divide them into supervised learning, unsupervised learning, and semi-supervised learning techniques. They then discuss the exact ML algorithms selected for the research: Bayesian network, SVM classifier, K nearest neighbor, Naïve Bayes, fuzzy logic, genetic algorithm,

decision tree, random forest, and artificial neural network. They also briefly discuss a set of optimization techniques used within neural networks, including particle swarm optimization, backpropagation, and adaptive moment optimizer. The researchers then discuss three primary intrusion detection datasets: KDD DataCup 1999, GureKDDcup, and NSL-KDD. They also discuss the metrics they use to measure and evaluate ML techniques, including accuracy, detection rate, and false alarms. The authors then present a table containing the ML methods, authors who utilized the specified method in a published paper, the dataset selected, and the ML method results. This paper's authors could have conducted an additional analysis comparing and contrasting the experiments and results between each of the ML methods taken from author researchers' published work.

Researchers Liu and Lang conduct a thorough review of modern machine learning and deep learning models for intrusion detection while discussing their strengths and weaknesses in their 2019 publication (Liu & Lang, 2019). The authors begin the paper by discussing IDS devices and their limitations, such as high false alarm rates and inability to detect unknown attacks. They then explain how machine and deep learning can provide a solution to these problems. The researchers then develop a clear graphic that displays the components that make up an IDS. This figure included different detection methods like anomaly detection and misuse detection and also sources of data such as log-based detection for host-based IDS devices and packet-based, flow-based, and session-based detection for network-based IDS devices. The authors discuss the differences between misuse detection and anomaly detection. They compared the detection performance, detection efficiency, dependence on domain knowledge, interpretation, and unknown attack detection factors in their comparison. Additionally, they review the differences between host-based IDS and network-based IDS by discussing the source of data, deployment, detection efficiency, intrusion traceability, and limitations. The authors then move on to the bulk of their survey, the machine learning algorithms used for network intrusion detection. The researchers developed a well-made graphic that reviews several models they discuss in the paper; this figure divided machine learning models into two groups, shallow models and deep learning models, and then further divided that by discussing supervised and unsupervised learning models. The authors create another chart that reviews the pros and cons of several shallow models used for intrusion detection; their table compared the algorithm, advantages, disadvantages, and improvement

measures. They discussed shallow models, including artificial neural networks, support vector machine, K nearest neighbor, Naïve Bayes, logistic regression, decision tree, and K-means. The authors then further discuss each of these algorithms and how they operate. Next, they compare various deep learning models, also presented in a chart. This figure included the algorithm, suitable data types, supervised or unsupervised, and functions. The deep learning models discussed in this section include autoencoder, restricted Boltzmann machine, deep brief network, deep neural network, convolution neural network, recurrent neural network, and generative adversarial network. The researchers also identify five major differences between shallow and deep models; this includes the running time, number of parameters, feature representation, learning capacity, and interpretability. Next, they identify a series of metrics used to evaluate machine learning models; this includes accuracy, precision, recall, F-measure, false negative rate, and false positive rate. They then move on to briefly discuss the datasets they found in their literature review for their survey. They review the DARPA 1998, KDD99, NSL-KDD, and UNSW-NB15 datasets. Additionally, they discuss the traffic and attacks within these datasets and how IDS devices can detect them, another major focus of this survey paper. They start by discussing packet-based attack detection (consisting of packet parsing-based detection and payload analysis-based detection), flow-based attack detection (consisting of feature engineering-based detection, deep learning-based detection, and traffic grouping-based detection), session-based attack detection (consisting of statistic-based feature detection methods and sequence feature-based detection), and log-based attack detection (consisting of rule and machine learning-based hybrid methods, log feature extraction-based detection, and text analysis-based detection). They then display a large table that compares the methods and papers on machine learning IDS devices; the categories in this table included the methods, papers, data sources, machine learning algorithms, and datasets. The authors then move on to discuss the challenges and future direction of machine learning-based IDS devices. They state that there is a lack of available datasets (and there are several problems with the KDD99 dataset, which is the most widespread), there is inferior detection accuracy in actual environments, and there is low efficiency for implemented machine learning-based IDS devices to detect attacks in real-time. Their reviewed papers identify three major IDS research trends: utilizing domain knowledge, improving machine learning algorithms, and developing practical models. This paper concludes by stating that deep learning models are becoming

more popular in IDS research. Deep learning models can improve IDS devices' performance and, compared to shallow learning models, deep learning models generate stronger fitting and generalization abilities. Additionally, deep learning algorithms do not require manual feature engineering or domain knowledge, giving deep learning a huge advantage over shallow learning methods. However, the authors also state that the biggest downside for deep learning models is the required running time, which is often too long to meet the real-time requirement of implemented IDS devices. This survey paper also did a fantastic job extensively reviewing and charting several machine learning and deep learning techniques. The authors also did a wonderful job reviewing the defensive detection methods employed by IDS devices to identify attacks, which many other researchers do not discuss.

Researchers discuss popular machine learning techniques and available datasets for network intrusion detection in the paper "A Review of Machine Learning Methodologies for Network Intrusion Detection" (Phadke, Kulkarni, Bhawalkar, & Bhattad, 2019). The researchers begin the paper by introducing the rise of cyberattacks and explain how network-based and host-based IDS devices can be used to assist this problem. They then introduce ML and then explain that algorithms can be classified into supervised learning, unsupervised learning, or semi-supervised learning models. They then introduce several ML methodologies that can be used for network intrusion detection. This section is the bulk of the paper where the authors list out and explain the following algorithms: support vector machine, an unnamed algorithm (proposed by another author that seems to build upon SVM), min-max k-means clustering, intelligent intrusion detection system (also proposed by another author and makes use of random forest and clustering techniques), artificial neural networks, and back propagation neural network. They then briefly discuss the three datasets used by other researchers, including the KDD Cup 1999 dataset, the UNSW-NB15 dataset, and the custom-made dataset to showcase their ANN. This paper's authors then compare each algorithm's accuracy and false positive rate and their selected dataset. The authors could have done a much more thorough analysis of the extracted results and incorporated several other papers and ML methodologies. It seems like the authors of this paper handpicked particular research publications and did not cover a wide range of available ML algorithms or datasets used for network intrusion detection.

Machine Learning Research Experiments

This section reviews paper publications that focused on developing ML models utilizing datasets for network intrusion detection. These papers' researchers had designed custom experiments, evaluation metrics, test results, and analysis. The experiments and results from these publications should be repeatable and consistent. While each paper is unique, specific ML algorithms, metrics, terms, and trends that appear across each of these experiments were incorporated into this dissertation research.

In a 2013 paper, authors Thaseen and Kumar evaluate several different tree-based classification algorithms using the NSL-KDD 99 dataset (Thaseen & Kumar, 2013). The algorithms selected for the experiments include ADTree, C4.5, LADTree, NBTree, RandomTree, RandomForest, and REPTree. The researchers also utilize a series of classifier performance metrics, which include the following: true positives, false positives, true negatives, false negatives, accuracy, precision, recall, and F-measures. Error metrics, such as Mean Absolute Error, Root Mean Squared Error, and Kappa Statistics. The experiments' results indicate that the RandomTree, RandomForest, and REPTree models achieved the most effective accuracy scores.

Researchers for another publication, titled "Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection," perform a comparative analysis between three types of machine learning methods using the NSL-KDD dataset (Ahmad, Basheri, Iqbal, & Rahim, 2018). The researchers first discuss the type of dataset that fits the needs of this research. They choose to use the NSL-KDD dataset, perform data pre-processing, select a set of ML classifiers (SVM, RF, and ELM) for testing, and then evaluate the results using specific metrics (accuracy, precision, and recall) and dataset sample size (full NSL-KDD dataset, half dataset, or one-fourth dataset). The results show that ELM (or hidden layer feedforward neural networks) outperform the other machine learning techniques in every measured metric category (accuracy, precision, and recall) when using the entire dataset.

Researchers Choudhury and Bhowal published a paper in 2015 where they conduct several experiments by selecting various ML classification algorithms trained using the NSL-KDD dataset (Choudhury & Bhowal, 2015). The paper begins by mentioning network-based and host-based IDS devices along with their associated anomaly-based and signature-based

detection classifiers. The authors then transition into data mining and machine learning by discussing the WEKA tool and several ML classifiers (Bayes, function, lazy, meta, mi, misc, rules, and trees classifier), classification algorithms (BayesNet, logistic, IBK, JRip, PART, J48, random forest, random tree, and REPTree), and ensemble algorithms (AdaBoost, bagging, and stacking). The authors conduct experiments on all the mentioned classification algorithms and the ensemble algorithms using the NSL-KDD dataset and WEKA tool. They also mention their performance metrics for this research, which includes true positive, false negative, false positive, true negative, receiver operating characteristics (ROC), sensitivity, specificity, precision, accuracy, kappa, mean absolute error, F1 score, false positive rate, false discovery rate, negative predictive rate, and training time. The researchers then review their classification tests and ensemble tests, which indicate that random forest, BayesNet, and boosting algorithms are the most efficient for intrusion detection.

The researchers who published "Machine-Learning-Based Feature Selection Techniques for Large-Scale Network Intrusion Detection" propose novel feature selection techniques and compare their results against three other well-known ML feature sets using the NSL-KDD dataset (Al-Jarrah et al., 2014). The authors start the paper by discussing ML methods and the challenges of feature selection; they propose two custom ensemble methods using the Random Forest algorithm and compare their technique with other well-known ML techniques. They also introduce the idea of big data IDS environments and their associated challenges: large volumes of IDS data require efficient data storage methods, big data flows at high velocity, which requires expensive storage and processing systems, and big data has various types of structures and different sources. The researchers then explain their ensemble algorithm using Random Forest and discuss the experiments conducted for this research. They start by discussing the KDD 99 dataset explain why they chose the NSL-KDD dataset. The authors pre-process the dataset by removing redundant records and then enumerating, normalizing, discretizing, and balancing them. They then discuss feature selection and state its significant impact on intrusion detection system performance since it reduces computation costs, removed data redundancy, increases the detection algorithm's accuracy, facilitates data understanding, and improves generalization. The standard feature selection process includes subset generation, evaluation, and validation. Different feature selection models are also categorized into three groups based on the evaluation criteria: filter model, wrapper model,

and hybrid model. The authors then discuss their ML model selection and explain that ensemble classifiers combine multiple models to generate a single model with better prediction accuracy. They state that several popular ML classifier algorithms can be used for an ensemble classifier, but they selected Random Forest (RF) for their research. They also discuss their evaluation metrics for their experiments, including detection rate or sensitivity, accuracy, training time, Mathew's correction coefficient, and false alarm rate. The researchers then review their conducted experiments using their Random Forest – Forward Selection Ranking and Random Forest – Backward Elimination Ranking ensemble models and compare their results to other researchers' ensemble classifier results.

Researchers conduct a comparative analysis of five ML algorithms while using the KDD Cup 99 dataset in the paper "Intrusion Detection in Computer Networks via Machine Learning Algorithms" (Ertam, Kilinçer, & Yaman, 2017). The authors briefly introduce IDS devices and five ML algorithms selected for this research, including Naïve Bayes, Bayes NET, random forest, multilayer perception, and sequential minimal optimization. The authors never explain how each of their selected ML classifiers operates. They then introduce the KDD Cup 99 dataset and state how other researchers have used it in their experiments. They also discuss the four attack categories and distribution within the dataset. Next, the authors discuss their evaluation metrics for this study: true positive, false positive, false negative, true negative, false positive rate, precision, recall, f-measure, and accuracy. They also state the physical machine resources used to perform their experiments and their usage of the WEKA data mining tool to classify the data. The authors then display and discuss their results. Each experiment was broken up by the attack class category and the ML classifier algorithm based on the tables. The results were highly distributed across each selected ML algorithm and attack category.

In 2018, researchers Maniriho and Ahmad published a paper where they compared the results of four ML classification algorithms used in conjunction with two different feature selection techniques while utilizing the NSL-KDD dataset (Maniriho & Ahmad, 2018). The paper begins by introducing the topics of network IDS devices and machine learning techniques. The authors also briefly discuss the NSL-KDD dataset and why it was used for these experiments. They also state that they selected four ML algorithms that were a part of the WEKA data mining tool. The four ML algorithms selected for this research are random

forest, decision stump, Naïve Bayes, and stochastic gradient descent combined with two different feature selection techniques (correlation ranking filter and gain ratio feature evaluator). This research's performance evaluation metrics include accuracy, detection error, true positive rate, true negative rate, precision, recall, and f-measure. The experiments showcase the dramatically varying results between each ML algorithm and feature selection test.

In the 2011 paper, researcher Yu-Xin conducts tests on three ML algorithms utilizing the KDD Cup 1999 dataset and WEKA data mining tool (Yu-Xin, 2011). The author introduces NIDS and their two primary classification categories of misuse detection and anomaly detection. The author then discusses how ML can assist network intrusion detection categorization and introduces the three ML techniques selected for their research experiments, including neural networks, support vector machines, and decision trees. The KDD 1999 dataset is selected for this research because it showcases the purpose of this research on network intrusion detection and machine learning. This research also introduces feature selection and reviews three attribute evaluation methods and four search methods implemented in the experiments. The evaluation methods are: CfsSubsetEval, InfoGainAttributeEval, and GainRatioAttributeEval; the search methods include: BestFirst, GreedyStepwise, Ranker, and GeneticSearch. Next, the author reviews the exact setup for each experiment and explains that the RBF network is used as the neural network algorithm, SMO is used as the SVM algorithm, and J48 is used as the decision tree algorithm. Each of these selected algorithms is used in conjunction with selected feature selection evaluation methods and search methods. The author then reviews these tests' results and explains how these ML classifiers and feature selection techniques showcase ML schemes' challenges in intrusion detection. The key challenges identified are fluctuant capability, false alarm rate, and difficulties in training. The results between each experiment vary or fluctuate dramatically based on the selected ML algorithm and feature selection technique and, because of these variations, the false alarm rate results are also affected. Training ML models is also a challenge because more precise detection results require larger training datasets. However, proper ML training becomes hard to achieve because the required training datasets will need to be so enormous that they require too many system resources to store and process. The author admits that their tests only include a small amount of ML algorithms and feature

selection techniques, but they wanted to showcase certain key challenges found in other ML intrusion detection environments and research.

The research paper “Machine Learning Algorithms In Context of Intrusion Detection” selects four supervised ML algorithms to detect anomalies found in the KDD99 dataset (Mehmood & Rais, 2016). The authors begin the paper by introducing host-based and network-based intrusion detection along with the two primary intrusion detection classifications of signature-based detection and anomaly-based detection. They then introduce ML and explain how it can assist the classification process of network intrusion detection. This research’s four ML algorithms are support vector machine, Naïve Bayes, J.48 (decision tree), and decision table. The researchers select the KDD99 dataset to compare their four ML classification algorithms. They briefly mention the KDD99 attack classes: root to local, denial of service, probe, user to root, and normal. They also explain that they utilize the WEKA data mining tool to implement their selected ML algorithms and compare the true positive and false positive rate values as evaluation metrics between tests. The results indicate that each ML algorithm had varying results for each KDD 99 dataset’s attack classes. The researchers state that feature selection algorithms will help generate better results for future work.

Researchers Rahat and Ahsan conduct a comparative study of five ML classifier algorithms using the KDDCUP ‘99 dataset to analyze classification metrics in their 2015 paper (Rahat & Ahsan, 2015). The paper starts by introducing the topic of network intrusion detection and how machine learning techniques can assist with this challenge. The authors review a series of publications that discuss different ML classification techniques used for network intrusion detection. They also mention class imbalance and feature selection as challenging issues found within ML algorithms. Class imbalance is caused when there is an unequal sample distribution within datasets. The authors review other researchers’ work and how they went about trying to resolve the challenges of class imbalance and feature selection. After this literature review, the authors discuss their performance metrics for this research, including true positive, true negative, false positive, false negative, accuracy, precision, and recall. They also briefly discuss the KDD’99 dataset, which contains 23 labels for different attack types and 41 labeled features to detect network intrusions. Next, the researchers discuss the methodology for their experiments. They first discuss two data sample methods that are used to remove the class imbalance in the dataset. The researchers then perform feature set

reduction to find the minimum number of features that can effectively represent the data in a classification problem. They utilize five ranking methods in conjunction with ranker search to reduce the feature set. The ranking methods they use are PCA, information gain, gain ratio, chi square, and filtered attribute. Lastly, they discuss the five ML classifiers used for their experiments, including J48 (decision tree), Naïve Bayes, AdaBoost, bagging, and nearest neighbor. Lastly, they note the four different experiment scenarios where they mix and match the process of data sampling, feature selection, and ML classifiers. Scenario 1 only uses the classifiers that are applied, scenario 2 performs feature reduction followed by applied classifiers, scenario 3 performs feature reduction followed by data sampling and then classifiers, and scenario 4 performs data sampling followed by applied classifiers. The researchers then display and discuss their experiments, which were highly variable based on each selected scenario, ML classifier, and evaluation metric.

In the paper titled "Class Imbalance Problem in the Network Intrusion Detection Systems," researchers compare four ML classification algorithms and use the NSL-KDD dataset to showcase the class imbalance problem found within the popular dataset (Rodda & Erothi, 2016). The authors briefly introduce IDS devices and explain that ML techniques can assist with classification. Several research publications and ML techniques have been developed to create an effective intrusion detection system; however, there is a class imbalance problem found in the most popular datasets. The class imbalance problem occurs when the size of a specific class or category within a particular dataset is too small or too large and thereby not adequately represented. They conduct a very brief literature review that discusses ML research and experiments conducted on the NSL-KDD or KDD 99 datasets performed by other researchers. The researchers explain why they selected the NSL-KDD dataset and discuss the attack categories and their distribution. The authors then immediately discuss their results without explaining how they set up each experiment or their evaluation metrics. The ML algorithms evaluated for this research included Naïve Bayes, Bayes Net, J48, and Random Forest. The results and analysis section indicate that the authors only really compared the accuracy metric between each experiment. These results still showcase their primary argument that the class imbalance problem because all of the ML classification algorithms seemed to fail on the user to root attacks due to the minimal amount of U2R traffic found within the NSL-KDD dataset.

CHAPTER 3

QUANTITATIVE RESEARCH METHODOLOGY

This section reviews the selected research methodology and design for this study. This research performed a quantitative research study, which consisted of two primary phases. The first phase involved developing machine learning models based on machine learning classification algorithms and intrusion detection datasets. The second phase then attempted to implement an ML IDS pipeline environment by applying the ML detection models from the first phase to unseen network traffic fed through Zeek IDS software. The results gathered from the second phase tests were then compared against the first phase results to determine if the ML models operated differently between the theoretical and applied environments. Both phases incorporated quantitative research methodologies, results, and analysis.

Purpose Statement

One purpose of this research was to showcase the need to utilize multiple intrusion detection datasets. The large majority of researchers only selected a single dataset for developing their ML models, typically incorporating the KDD 99 or NSL-KDD datasets. Additionally, this research sought to help identify which ML classification algorithms are the most effective at network intrusion detection in applied environments. This research study implements an IDS ML pipeline that augments IDS logs with trained ML models to validate the theoretical best ML algorithms, datasets, and optimization methods. This research benefits both academic researchers as well as industry security professionals. As seen in the previous Literature Review section, many academic researchers fail to train their ML algorithms using multiple datasets. Numerous public datasets are associated with network intrusion detection, each of which is unique and comes with associated pros and cons. Due to the wide variety of intrusion detection datasets, it is best practice to train and evaluate ML algorithms using multiple datasets (Ring et al., 2019). This research achieves and verifies this best practice by

conducting numerous research experiments that train multiple ML algorithms using multiple datasets.

Additionally, as seen in the previous Literature Review section, most ML researchers never design or explain practical implementations to validate their models in a real-world environment. This research implements a live IDS ML pipeline and then compares each ML model's evaluation metric results in both the theoretical and implemented environments. Additionally, this research documents the creation of an IDS with ML classification capabilities. This research could greatly benefit security professionals because they can use this as a guide to set up a highly effective IDS ML environment. There are very few guides or papers online that document the process of configuring the IDS anomaly detection engine utilizing ML techniques. An IDS with ML capabilities should incorporate anomaly detection and standard signature detection using a single application for enhanced detection.

Research Questions and Hypotheses

Research Question 1

How does the selection of a single network intrusion dataset impact machine learning models' outcomes and performance when trained using multiple machine learning algorithms and optimization methods?

Hypothesis 1

Not all machine learning models will achieve high accuracy when trained using any network intrusion dataset.

Research Question 2

How does the selection of a single machine learning algorithm impact machine learning models' outcomes and performance when trained using multiple network intrusion datasets and optimization methods?

Hypothesis 2

Not all machine learning models will achieve high accuracy when trained using any machine learning algorithm.

Research Question 3

How does the selection of a single optimization method impact machine learning models' outcomes and performance when trained using multiple network intrusion datasets and machine learning algorithms?

Hypothesis 3

Not all machine learning models will achieve high respective performance metrics when trained using any optimization method.

Research Question 4

How does the performance of theoretical machine learning models change when tested in an applied environment?

Hypothesis 4

The accuracy of theoretical machine learning models will perform significantly worse in an applied environment.

Quantitative Method – Experimental Study Design

Design

This study was divided into two quantitative research phases. The first phase focused on testing and quantitatively evaluating each of the selected ML algorithms, network intrusion datasets, and optimization methods by producing a series of models. The machine learning algorithms selected for this research include Naïve Bayes (NB), Decision Tree (DT), Random Forest (RF), Ada Boost (AB), Bagging Classifier (BC), Logistic Regression (LR), and

Stochastic Gradient Descent (SGD). All of these algorithms are considered supervised learning algorithms, which means they require fully labeled datasets.

These ML algorithms were identified and selected due to a few reasons. One part of their selection was due to their popularity throughout the literature review. Individual algorithms and entire algorithm categories were tallied during the literature review, and these particular algorithms were selected due to their regular appearances throughout the research. Additionally, these algorithms were selected based on their availability within the Python scikit-learn library. The scikit-learn library is a publicly available Python module that contains a wide variety of machine learning models and functions (Garreta, Moncecchi, Hauck, & Hackeling, 2017). Lastly, the resource requirements and speed at which the algorithms generate their models were considered factors. These algorithms' speed and resource requirements were major considerations due to the number of tests conducted for this study. Certain algorithms were removed from this research if they took longer than three days to produce a single model.

The Naïve Bayes (NB) algorithm belongs to its own Naïve Bayes category and is a probabilistic classifier that applies Bayes' theorem between features (Tsai, Hsu, Lin, & Lin, 2009). This classifier assumes strong or naïve independence between data features and calculates conditional probabilities for different classes, then used to label the data.

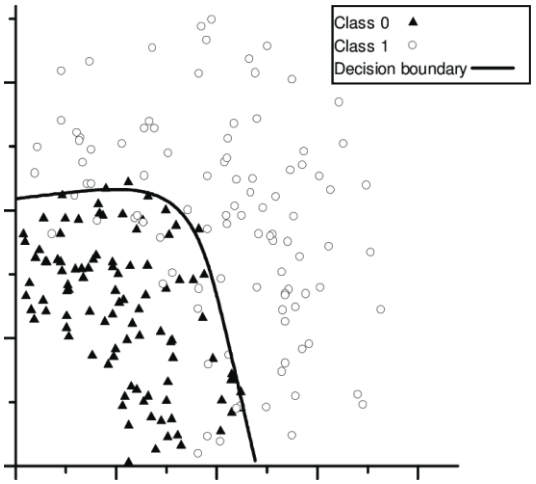

Figure 1: Naïve Bayes Algorithm Example

The Classification and Regression Trees (CART) algorithm falls under the decision tree category and develops tree-like structures where the leaves represent classifications and

branches represent conjunctions of features that lead to classifications (Tsai et al., 2009). This algorithm is unique to scikit-learn as it is a type of decision tree algorithm based on the popular C4.5 decision tree algorithm with very slight alterations.

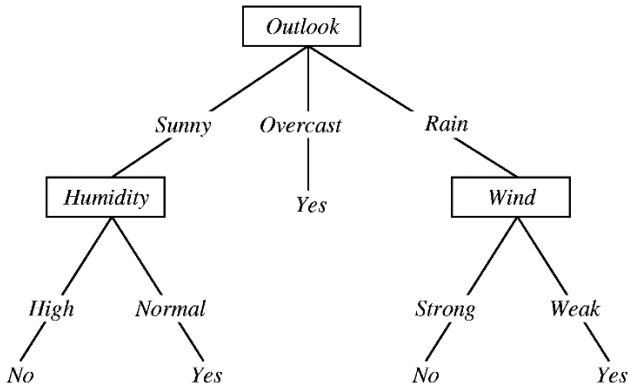

Figure 2: Classification and Regression Trees Algorithm Example

The Random Forest (RF) algorithm belongs to the ensemble learning category (Aburomman & Reaz, 2017). Ensemble learning constructs several models and then selects the best model based on majority or weighted voting. This algorithm combines decision trees and ensemble learning to produce several decision trees that use randomly selected data features or attributes as their input, such that a forest is generated with trees with controlled variance.

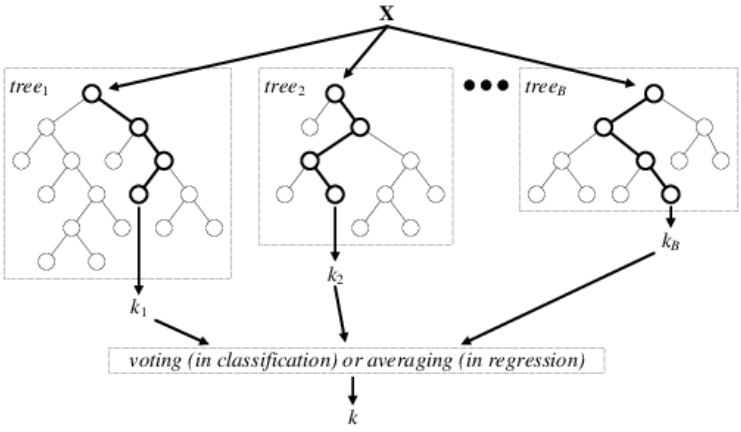

Figure 3: Random Forest Algorithm Example

Adaptive Boosting, or AdaBoost (AB), is another type of machine learning algorithm that falls under ensemble learning (Aburomman & Reaz, 2017). Boosting is a form of machine learning ensemble algorithm where models are sequentially added, and later models

in this sequence can correct the predictions made by earlier models. The AdaBoost algorithm is considered adaptive because it combines, or adapts, multiple weak classifiers into a single strong classifier while boosting the sequence of models. As a result, this algorithm is considered quite sensitive to noisy data and outliers. Additionally, this ensemble learning technique makes the algorithm less susceptible to overfitting algorithms since it reduces variance.

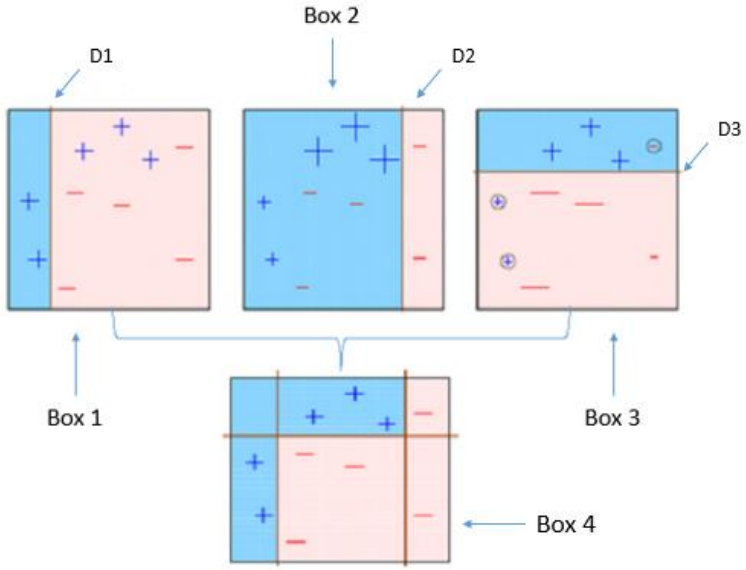

Figure 4: Adaptive Boosting Algorithm Example

Bagging Classifier (BC) is another machine learning algorithm that belongs to the ensemble learning category (Aburomman & Reaz, 2017). Bagging, also known as bootstrap aggregating, is another type of machine learning ensemble algorithm that generates subsets of the original dataset through sampling. Bagging Classifier fits base classifiers (such as Decision Tree) on random subsets of the original dataset and then aggregates their predictions, by voting or averaging, to create a final model and prediction. Similar to the other ensemble learning methods, this ensemble technique reduces variance and thereby helps avoid overfitting.

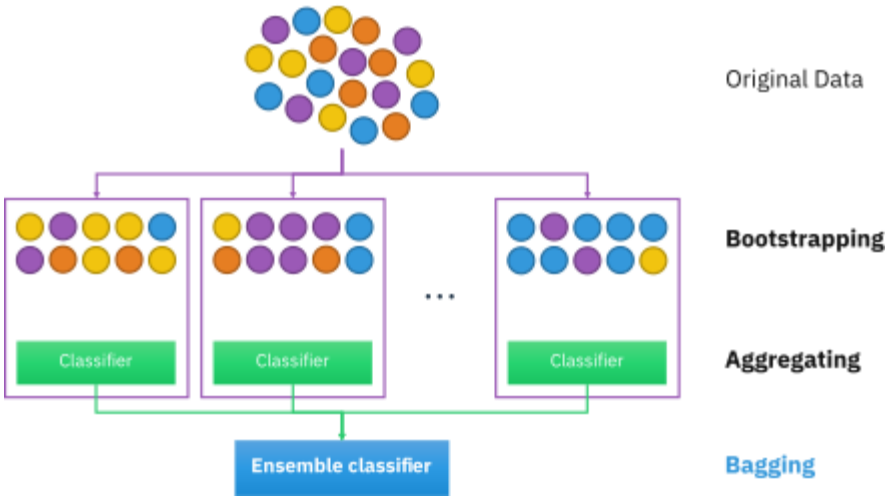

Figure 5: Bagging Classifier Algorithm Example

The Logistic Regression (LR) algorithm is a well-known algorithm in machine learning and falls under the linear regression category (Yihunie, Abdelfattah, & Regmi, 2019). Logistic regression applies a statistical model that uses the logistic function to model binary dependent variables. After ingesting data, this algorithm calculates a particular threshold coefficient to make classifications and predictions thereby.

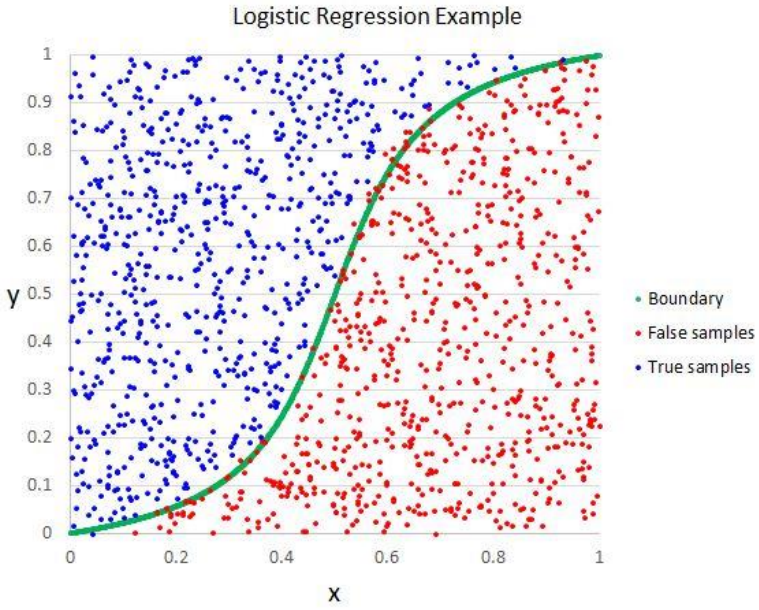

Figure 6: Logistic Regression Algorithm Example

Stochastic Gradient Descent (SGD) is another form of linear regression. This algorithm may also be known as a stochastic approximation of gradient descent optimization

(Yihunie et al., 2019). Gradient descent optimization is an iterative optimization algorithm used for identifying the local minimum, or lowest point, on a function by taking repeated steps in the opposite direction of the gradient. The term “stochastic” is another word for “random.” As a result, SGD applies randomness in the gradient descent algorithm by selecting a single random data point out of the whole dataset for each iteration of gradient descent optimization. This random selection of data points dramatically reduces the necessary computation.

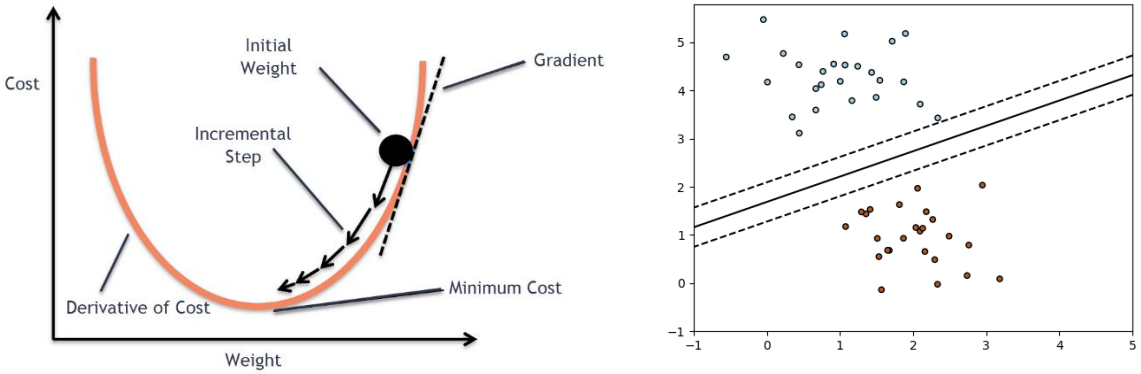

Figure 7: Stochastic Gradient Descent Algorithm Example

Numerous machine learning algorithms were tested and discarded while investigating a variety of algorithms for this research. In addition to the algorithms officially selected, the following algorithms were tested and removed from this study: Bayesian Network, K-Nearest Neighbor, Support Vector Machine (Linear, Radial Basis Function, and Sigmoid), Ridge Classifier, One-Class SVM, Isolation Forest, and Local Outlier Factor. These algorithms were considered and tested due to their popularity and success seen throughout the literature review. The Bayesian Network algorithm was removed due to its absence within the scikit-learn library, whereas all the other listed algorithms were discarded due to their intense resource and time requirements. Many of these algorithms are extremely sensitive to feature input, and each new feature exponentially increases the amount of time to develop a model (Lee, 2019). These algorithms may have had difficulties ingesting the datasets due to their immense size, especially when considering the high number of labeled features and the total number of packets. If any of these algorithms took longer than three days to generate a single model in the online virtual environment, they were discarded. Many of these algorithms may have produced a highly successful machine learning model after several days of development.

However, it was decided to remove these intense algorithms due to the high number of models required for this research. Using a single selected algorithm, 15 models (one algorithm, five datasets, and three optimization methods) would need to be developed to be adequately analyzed for this study. This considerable time and resource requirement for developing these particular models did not fit this study's proposed timeline.

Towards the end of investigating machine learning algorithms to include in this research, a focus on the inclusion of novelty and outlier detection algorithm categories was heavily considered. These two machine learning categories were considered due to their extreme outlier sensitivity and success with anomaly detection ("scikit-learn," 2020). The outlier detection category operates by identifying regions within the dataset observed as far or different from other regions. The algorithms that fall under this outlier detection category include Isolation Forest, Local Outlier Factor, and One-Class SVM. All of these algorithms were tested for this study. However, they were all dropped due to their high resource and time requirements. The outlier detection estimators fit the regions where the data is most concentrated, thereby ignoring deviant observations. The category of novelty detection operates by treating none of the training data as outliers and determining if new observations are considered an outlier, or rather, a novelty. The One-Class SVM and Local Outlier Factor algorithms could be configured to operate under this category of novelty detection but still belong under the outlier detection category. Again, both these algorithms were tested and dropped due to their requirements; they took longer than three days to develop a single model. With more time and more robust computational resources, these algorithms may have generated the most effective anomaly detection models for this study.

The datasets used to train and test each of these ML algorithms include the KDD 99, NSL-KDD, UNSW-NB 15, CICIDS 2017, and CIDDS-001 datasets. The KDD 99 and NSL-KDD datasets were selected because of their popularity in the literature review. Nearly all of the literature utilized at least one of these datasets. The UNSW-NB 15, CICIDS 2017, and CIDDS-001 datasets were selected at the recommendation of a comprehensive intrusion detection datasets survey paper where the authors suggested these particular datasets due to their wide variety of attack categories and network traffic composition (Ring et al., 2019). Each experiment in this first phase of research applied a combination of machine learning models, network intrusion datasets, and optimization methods.

The KDD 99 dataset was created as part of the International Knowledge Discovery and Data Mining Competition (KDD) Cup in 1999. This dataset was based on the DARPA 1998 dataset, produced by MIT's Lincoln Labs. This dataset is the most popular dataset for network intrusion detection despite its several flaws previously discussed in the literature review (Creech & Jiankun, 2013). The NSL-KDD dataset was created at the University of New Brunswick in Fredericton, Canada, in 2009. This dataset essentially updated the KDD 99 dataset by purging many redundant records and improperly distributed attacks (Tavallaee et al., 2009). The UNSW-NB15 dataset was produced at the University of New South Wales in Sydney, Australia, in 2015. The CICIDS2017 dataset was also produced at the University of New Brunswick, Fredericton, Canada, in 2017. This university is partnered with the Canada Institute for Cybersecurity. The CIDDS-001 dataset was developed at the Coburg University of Applied Sciences in Coburg, Germany, in 2017. Each of these datasets is fully labeled, meaning that they can operate with supervised learning algorithms properly. These datasets provide various data format types (packet-based, flow-based, hybrid/other), metadata, size, attack scenarios, and features. A comparison of these selected datasets can be found in Tables 1 and 2. These tables were taken and modified from the previously referenced academic research survey of available network intrusion datasets (Ring et al., 2019).

The second phase of research focused on modifying the Zeek anomaly detection engine to enable ML classifications of network traffic ("zeek," 2020). This second phase aimed to apply the ML models initially trained and tested in the first phase to a real-world application of an IDS with ML capabilities. The implemented IDS ML pipeline results were quantitatively analyzed and compared to the previous phase results to determine if there was a difference in evaluation metrics between conceptual and applied environments.

	General Information				Nature of the Data		
Dataset	**Year Created**	**Public Availability**	**Normal Traffic**	**Attack Traffic**	**Metadata**	**Format**	**Anonymity**
KDD 99	1998	Yes	Yes	Yes	No	Hybrid/other	None
NSL-KDD	1998 (2009)	Yes	Yes	Yes	No	Hybrid/other	None
UNSW-NB15	2015	Yes	Yes	Yes	Yes	Packet, Hybrid/other	None
CICIDS2017	2017	Yes	Yes	Yes	Yes	Packet, Flow (bi)	Yes (IPs)
CIDDS-001	2017	Yes	Yes	Yes	Yes	Flow (uni.)	None

Table 1: Dataset Comparison – General Information and Nature of the Data

	Data Volume		Recording Environment			Evaluation		
Dataset	**Count**	**Duration**	**Kind of Traffic**	**Type of Network**	**Complete Network**	**Predefined Splits**	**Balanced**	**Labeled**
KDD 99	5M points	Not specified	Emulated	Small enterprise	Yes	Yes	No	Yes
NSL-KDD	150K points	Not specified	Emulated	Small enterprise	Yes	Yes	No	Yes
UNSW-NB15	2M points	31 hours	Emulated	Small enterprise	Yes	Yes	No	Yes
CICIDS2017	3.1M flows	5 days	Emulated	Small enterprise	Yes	No	No	Yes
CIDDS-001	32M flows	28 days	Emulated and real	Small enterprise	Yes	No	No	Yes

Table 2: Dataset Comparison – Data Volume, Recording Environment, and Evaluation

Instruments and Tools

The first phase of this study was conducted using five cloned Ubuntu 20.04 virtual machines (VMs) in the DSU Information Assurance (IA) Lab, an online virtual environment. Each VM utilized 24 cores with a 2.30 GHz processing speed, 100 GB of RAM, and 500 GB of hard disk space. These five VMs were labeled after the five datasets selected for this research. This study utilized several free and open-source tools. Ubuntu was selected as the underlying operating system on each VM due to its lightweight and open-source availability ("Ubuntu," 2021). Additionally, Ubuntu comes with a series of preinstalled tools that were used for this research. One primary tool requirement necessary for this study was Python version 3. Python is a very well documented and popular high-level scripting language with a large community of supporters ("Python," 2021). This study also used the Anaconda suite, a free platform-agnostic package, and environment manager ("Anaconda," 2021). The Anaconda suite provided access to the Scientific Python Development Environment, or Spyder Integrated Development Environment (IDE). Spyder is a free and open-source IDE specializing in advanced data analytics and debugging ("Spyder," 2020). Anaconda was utilized as a simple package manager for updating tools and libraries, including Python, Spyder, and scikit-learn. The scikit-learn module can be imported into Python and contains various machine learning capabilities and analytics ("scikit-learn," 2020). Scikit-learn successfully trained, tested, and evaluated each of the ML models in this research. Each of the previously mentioned datasets was downloaded from the developer organizations' website for free and were successfully ingested within the Python environment and scikit-learn functions. Additionally, scikit-learn provides standardized measuring techniques for evaluating the results of each ML experiment. The evaluation metrics analyzed between each ML experiment included accuracy, precision, recall, and F1 score. Additionally, it should be noted that during the development of the Python scripts for this research, several different online resources were utilized. Numerous technical books were referenced during code development through the use of the O'Reilly Online Learning library, which was previously known as Safari Books Online ("O'Reilly," 2021). The second phase of this study was conducted using the same set of online VMs. One of the five online VMs had Zeek installed, which acted as the open-source IDS for this research. The prediction dataset and associated raw PCAPs

selected for the second phase of research were fed into this Zeek IDS to produce standardized Zeek connection logs of the ingested traffic ("zeek," 2020). The Wireshark tool called mergecap was also utilized in the second phase of research and combined multiple raw PCAPs into a single PCAP format for Zeek ingestion ("mergecap," 2021).

Some alternative tools were considered to be used in conjunction with or possibly replace previously mentioned tools. Some other online virtual environments that were considered include Amazon Web Services (AWS), Google Cloud Platform (GCP), or Digital Ocean (DO). It should also be noted that VMware Workstation was a considered tool for this research as it would allow for virtualization on a host desktop; however, it was not utilized due to the intense resource requirements and the high number of experiments for this research study. Additionally, the previously mentioned open-source IDS software, Zeek (previously known as Bro), was just one open-source IDS option; Suricata or Snort was also considered. A handful of other ML algorithms, intrusion detection datasets, or evaluation metrics were removed or replaced while conducting this research.

Procedures

The first phase of this study involved developing a Python script that imports the scikit-learn module and associated datasets to produce a series of ML models. The Python script was configured to optimize and create a series of models based on the combination of ML models, network datasets, and scoring methods. Additionally, this Phase One script would first optimize each model to tune each of them to the highest associated metric score based on the selected optimization method. This Phase One script was distributed to each of the five VMs in the DSU Information Assurance (IA) Lab virtual environment. This first phase trained 105 models based on seven ML algorithms, five datasets, and three scoring methods. Each of these models, or experiments, utilized parameter optimization for each algorithm to then develop fully trained and optimized models. The selected evaluation metrics were used to compare each model effectively and consistently. A series of data transformations are applied to each of the imported datasets. The steps involved in conducting these transformations include label extraction and grouping, feature mapping and dropping, feature encoding and standardization, feature scaling and normalization, subset creation, and feature reduction. Additionally, a complete walkthrough of the Phase One Python script is

included in a later subsection. Essentially though, this phase's goal was to develop a series of highly optimized models that yield high evaluation metrics.

The second phase of this research involved applying the previously trained models to filter through unseen network traffic and identify malicious behavior. This process involved taking the raw PCAPs of the CICIDS 2017 dataset and feeding them through the Zeek IDS to produce connection logs. This PCAP merging and Zeek ingestion were performed using just one of the five available online VMs. These Zeek connection logs and the previously trained models were then fed into a new Python script that applied the models and yielded each model's detection output. The Phase Two Python script was downloaded and executed on each online VM, and the Zeek connection log output was generated from the raw PCAPs of CICIDS 2017. This entire process could be considered an IDS ML pipeline environment with additional scripts that push the necessary data between each script continuously. Below is a diagram that shows the overall procedures and flow of data between each phase of this research.

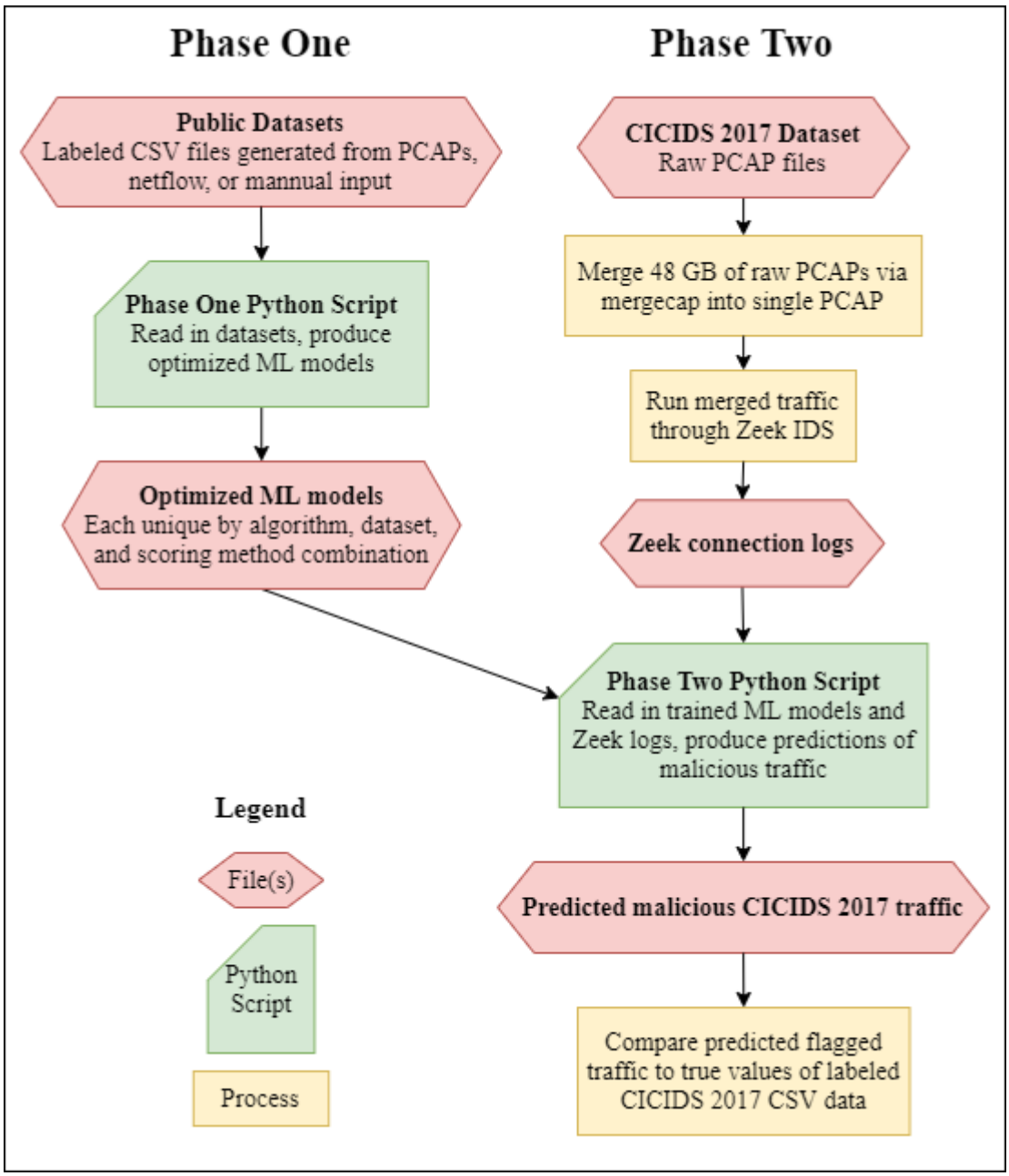

Figure 8: Phase One and Phase Two Python Scripts – Flowchart

Data Analysis and Interpretation of Results

The first phase of quantitative research produced a series of trained ML models with associated evaluation metrics. The evaluation metrics included in Phase One include accuracy, precision, recall, F1 score, and training time. These metrics were included due to their appearance throughout the literature review and their availability within the scikit-learn

library. Accuracy measures the fraction of correct predictions (true positives and false positives) out of the total results (true positives, true negatives, false positives, and false negatives). Precision measures the amount of true positive results over the actual results (true positives and false negatives), whereas recall measures the amount of true positive results over the model's predicted results (true positive and false negative). Perfect precision corresponds to no false positives (Type I errors); meanwhile, perfect recall corresponds to no false negatives (Type II errors). Additionally, precision and recall considered inverses of each other; if precision increases, recall decreases and vice-versa. The F1 score metric is considered to be the harmonic mean between these precision and recall metrics.

This research primarily focuses on the accuracy metric due to this study's research questions and hypotheses. Ideally, each trained model's goal was to achieve 95% or higher in each of these metric categories. This 95% threshold was selected because, while the realm of network security seeks to achieve 100% detection of attacks, the realm of machine learning must tolerate lower metrics to create an acceptable model depending on the problem at hand. The 95% threshold of success for evaluation metrics provides a robust network intrusion detection model. The second phase of this study involved applying the models and only reviewing each model's accuracy metric. The true values of the CICIDS 2017 detection rate (normal or malicious) were extracted from the CICIDS 2017 CSV data. The goal of each trained model in Phase Two was to match, as closely as possible, the same detection rate as dictated by the true values observed in the CICIDS 2017 CSV dataset. This study aimed to showcase the need for utilizing multiple datasets and testing models in an applied environment. The focus of this research will be the comparison of ML models' results between each of the phases to determine how the models operate between theoretical environments and implemented environments.

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

$$Precision = \frac{TP}{TP + FN}$$

$$Recall = \frac{TP}{TP + FP}$$

$$False\ alarm\ rate = \frac{FP}{TN\ +\ FP}$$

$$Specificity = \frac{TN}{TN\ +\ FP}$$

$$Negative\ predictive\ value = \frac{TN}{TN\ +\ FN}$$

$$F1\ Score = 2 * \frac{precision * recall}{precision + recall}$$

Limitations and Challenges

A primary goal for this research was actually to create and implement a real-world IDS with ML capabilities. There are very few guides and papers that review this process, so a significant hurdle for this study was investigating this task's feasibility. There were only three identified options for feasibly creating an IDS with ML capabilities. The first option was to create a pipeline that takes the IDS output and feeds it into an ML processing script. A second option was to modify the preprocessors of the IDS detection engine. However, after investigating this option, it was determined that this route might take too long for this proposed research timeline. Lastly, a third option was to create an IDS plugin using the appropriate programming language. However, the biggest reason why the IDS preprocessor modification and IDS plugin creation options were not utilized in this research is that both of these methods required writing in the native language of the IDS. Some IDS software is written in programming languages with very limited support for machine learning libraries, meaning that machine learning algorithms would have to be manually coded, which was outside the scope of this research. Several other significant challenges and limitations for this research are further discussed in-depth in Chapter 5.

Python Scripts – Commentary Walkthrough

Two Pythons scripts were created for each phase of research. This subsection will include a walkthrough of the scripts and review the significant coding choices. While extremely enlightening, there were several challenges encountered while developing these scripts. Below

are pseudocode walkthroughs of the two primary Python scripts developed for each phase of research. The entirety of each of these two scripts is included in the Appendices. The Phase One Python Script is found in Appendix A, and the Phase Two Python Script is included in Appendix B. Additionally, the following subsection will review certain significant decisions made while developing and testing these Python scripts.

Phase One – Python Script

Python Script 1 Walkthrough

1. Import libraries
2. Loop through datasets
    a. Import dataset (CSV format with labels)
    b. Apply label extraction and grouping
    c. Rename and drop features
    d. Apply appropriate data transformations
        i. Convert dataset to str/object
        ii. Identify and save unique values via fullset.values.ravel() - saved as *_unique.joblib
        iii. Encode dataset via LabelEncoder() - saved as *_encoder.joblib
        iv. Scale dataset via StandardScaler() - saved as *_scaler.joblib
        v. Create subsets via train_test_split() - returns X_train, X_test, Y_train, and Y_test
            1. The CICIDS-001 dataset is unique in that train_test_splt() is performed twice to only utilize only 10% of the total dataset
        vi. Perform feature reduction via PCA() - saved as *_pca.joblib
    e. Return transformed X_train, X_test, Y_train, and Y_test subsets
    f. Optimize models - First run-through
        i. Configure optimizations settings
            1. Set scoring methods- accuracy, precision_macro, and f1_macro
            2. Set classifiers/algorithms to optimize - Decision Tree, Random Forest, Ada Boost, Bagging Classifier, Logistic Regression, Stochastic Gradient Descent
            3. Set parameter grid for each algorithm (unique parameters for each classifier)
        ii. Loop through scoring methods and classifiers
            1. Optimize models via GridSearchCV, which will perform an exhaustive search based on the provided classifiers and parameter grids, which will be optimized for maximum scores based on the selected scoring methods
            2. Save output with best scores and parameters to CSV (this was just for recording purposes - these optimized parameters were manually inputted back into the Python script)

g. Model production - Second run-through
    i. Set scoring parameters again - accuracy, precision_macro, f1_macro
    ii. Loop through scoring methods
        1. Obtain the optimized parameters for each algorithm
        2. Loop through each algorithm
            a. Produce official models via clf.predict() while utilizing the optimized parameters for different ML algorithms, different scoring methods, and selected dataset
        3. Save output with produced models and evaluation metrics (accuracy, precision, recall, and F1 score) to CSV

Phase Two – Python Script

Python Script 2 Walkthrough

1. Import libraries
2. Loop through datasets
    a. Import the prediction dataset (merged raw PCAPs of CICIDS 2017)
    b. Store the dataset in a CSV format with the same matching columns/features as the other trained models and their corresponding datasets
    c. Based on the selected dataset, transform the prediction set in the same exact procedure the previous model was trained
        i. Load the *_unique.joblib, *_encoder.joblib, *_scaler.joblib, and *_pca.joblib to the prediction set
        ii. Add "-" to the encoder schema (a default value in case an unseen value is in the prediction set)
        iii. Convert dataset to str/object
        iv. Identify and save unique values via predictionset.values.ravel() - loaded and applied via *_unique.joblib
        v. Iterate through the encoder schema and replace any unseen values with "-" (all the values in the new unseen data must exist in the previously trained encoder schema)
        vi. Encode dataset via LabelEncoder() - loaded and applied via *_encoder.joblib
        vii. Scale dataset via StandardScaler() - loaded and applied via *_scaler.joblib
        viii. Create subsets via train_test_split() - returns X_train, X_test, Y_train, and Y_test
        ix. Perform feature reduction via PCA() - loaded and applied via *_pca.joblib
    d. Set the scoring methods (accuracy, prevision_macro, and f1_macro) and classifiers/algorithms (NB, DT, RF, AB, BC, LR, and SGD)
    e. Iterate through the scoring methods and classifiers
        i. Utilize the clf.prediction() function to predict malicious traffic in the unseen data

ii. Perform computations to determine the accuracy of the predicted value
iii. Store and save the output of the prediction values in a CSV

Major Development Decisions

Consistent Experimentation

A major goal of this research was to maintain consistency and repeatability. In Phase One, a combination of machine learning classification algorithm, network dataset, and optimization went through the same exact data transformation process except for a few items taken from the datasets. The major difference between each dataset was the inclusion of certain features. Features are unique labeled properties belonging to each dataset; the list of features included for each of the datasets used in this study were various packet or NetFlow fields. This study attempted to map as many Zeek IDS Connection Log fields to features identified within each dataset. This mapping meant that specific datasets had as little as six features mapped to the Zeek log fields when performing feature reduction, whereas other datasets had as many as twelve features mapped. The complete list of features from each dataset and Zeek connection log fields can be seen in Appendix C. There is a table below containing each of the fully mapped features. The only mapped feature dropped was the connection state (conn_state) field because each dataset had its own nonstandard format for that particular attribute. In addition to this initial feature reduction, some more features may have been dropped when conducting automated dimensionality reduction via the PCA function. The only other distinction between the data transformations was reducing the size of the CIDDS-001 dataset. Only 10% of the total CIDDS-001 was ingested due to its immense size. When reading in the entire CIDDS-001 dataset, the Python script would crash due to limited memory capacity despite the 100 GB of dedicated RAM for the VM. Aside from these two unique data transformations of selective feature mapping and CIDDS-001 size reduction, all the ML models went through the exact data ingestion and transformation process.

Zeek Conn Log	KDD 99	NSL-KDD	UNSW-NB15	CICIDS 2017	CIDDS-001
id.orig_h			srcip		Src IP Addr
id.orig_p			sport		Src Pt
id.resp_h			dstip		Dst IP Addr

id.resp_p			dsport	DestinationPort	Dst Pt
proto	protocol_type	protocol_type	proto		Proto
conn_state*	flag	flag	state		Flags
duration	duration	duration	dur	FlowDuration	Duration
orig_ip_bytes	src_bytes	src_bytes	sbytes	TotalLengthofFwdPackets	Bytes
resp_ip_bytes	dst_bytes	dst_bytes	dbytes	TotalLengthofBwdPackets	
service	service	service	service		
orig_pkts			Spkts	TotalFwdPackets	Packets
resp_pkts			Dpkts	TotalBackwardPackets	

Table 3: Mapped Features – Dataset Features to Zeek Connection Log Fields

Data Transformation Process

Several significant decisions were made while determining the transformations of data imported into the Python scripts for this research. There are numerous steps involved in machine learning to transform data into a proper ingestible format (Gron, 2017). A series of steps are taken whenever a dataset is read into the scripts to perform a consistent data transformation.

Data Preprocessing

1. Label extraction and grouping
2. Feature mapping and dropping
3. Feature encoding
4. Feature scaling
5. Subset creation and cross-validation
6. Automated feature reduction

The transformation process's first steps involve the labels that classify the associated packet as either malicious or innocuous. Many datasets have numerous subsets of malicious traffic and classify packets as a particular type of attack. A decision was made to group all these types of network attacks as only "malicious" since they are all unwanted network traffic forms. This decision dramatically increased metric evaluation results during testing.

The next step in the transformation process involves dataset features. With the labels in the previous step successfully extracted and grouped, the rest of each dataset's unique features are transformed. First, the applicable features' names are mapped and modified to match those belonging to the corresponding Zeek connection log fields. With the names of the features changed to match the Zeek logs, the rest of the features are then dropped since they will not be used. A significant decision here was to map as many features as possible between each unique dataset and the Zeek logs. Specific datasets only mapped six features, whereas others mapped as many as ten features.

Next, the feature data is then encoded. Feature encoding is vital to machine learning as it is used to create a simple dictionary of sorts. This encoding schema is used to standardize and parse through the feature data (Chio & Freeman, 2018). There are numerous encoding methods available within the scikit-learn module; however, the well-known and standard LabelEncoder function was selected for this step.

Feature scaling and regularization of the data is the next major data transformation. This transformation step takes the encoded data and scales it in a particular way such that the feature data is normalized (Albon, 2018). Regularization will scale the data such that each feature and individual value is scaled and thereby have a fixed and proportional weight. Similar to encoding, there are also several scaling methods available within the scikit-learn library. The StandardScaler function is a popular option and was selected for this step. Another scaler that was heavily considered was the MinMaxScaler. Both the StandardScaler and MinMaxScaler functions are relatively sensitive to outlier data, which is essential since many network attacks and anomalous traffic are typically considered infrequent, thereby considered outlier packets.

The next transformation of data involved splitting the dataset into subsets for cross-validation. Cross-validation takes the original full dataset and divides it into a series of multiple subsets to be used for training and testing, or validation (Sarkar & Natarajan, 2019). One type of cross-validation technique is called Leave-One-Out-Cross-Validation (LOOCV). One widespread implementation of the LOOCV scheme is k-fold cross-validation, where the original data is split into *k* equal-sized folds and, over a series of rounds, each fold becomes the testing set and the rest act as the training set. This k-fold cross-validation process is repeated *k* times, with each fold acting as the designated testing set once. This study utilized

the train_test_split function, which performed 10-fold cross-validation and outputs the dedicated training and testing subsets. This function has several configurable parameters, and one of the major decisions was to perform a 90/10 split, where 90% of the original data was used for training, and 10% of the original data was split off as a dedicated test set. Additionally, the parameters were also configured to incorporate the same proportion of unique feature data between the training and testing datasets.

Feature reduction is the final data transformation step in this process. Feature reduction is also crucial in machine learning as it removes the features deemed unhelpful for categorization (Chollet, 2017). There are several different feature reduction methods; however, the Principal Component Analysis (PCA) function was selected for this transformation step. The utilization of PCA for feature reduction was a significant step because it allowed for automated feature reduction, which was decided by the function. The selection of PCA and automated feature reduction allowed for consistency between each unique dataset, rather than applying highly unique and configured reduction methods.

Model Optimization

The optimization process was a significant step during the development of models for Phase One. This stage involved the usage of scikit-learn's GridSearchCV function, which performs an exhaustive search for the best scoring models developed by a cross-validated grid-search over a submitted parameter grid (Gron, 2017). This function will develop numerous models based on each combination of parameters to identify which parameters yield the best models with the highest scoring metrics. This optimization process is applied while developing machine learning models to create models with ideal parameters and yield the highest possible performance. Each of the algorithms selected for this study had a series of configurable parameters that could drastically alter the models' outcome. This optimization process was identified which set of parameters were best suited for each combination of datasets and algorithms. It should be noted that different models using different optimization methods may select the same algorithm parameters, thereby leading to the same models with identical metric scores; this is later seen in the results of this study. It should also be noted again that the Naïve Bayes algorithm is the only algorithm in this study that cannot be optimized since it

does not have any optimizable parameters. Each algorithm and its corresponding parameter grid can be seen below or in the Phase One Python Script in Appendix A.

- Decision Tree - criterion: [gini, entropy], max_features: [auto, sqrt, log2]
- Random Forest - n_estimators: [100, 1000, 3000, 5000], criterion: [gini, entropy], max_features: [auto, sqrt, log2]
- Ada Boost - n_estimators: [100, 1000, 3000, 5000], learning_rate: [0.5, 1, 1.5]
- Bagging Classifier - n_estimators: [100, 1000, 3000, 5000], max_features: [0.5, 1, 5], max_samples: [0.1, 0.5, 1]
- Logistic Regression - penalty: [l2, l1, elasticnet], C: [0.1, 1, 10], max_iter: [2500, 5000, 7500, 10000]
- Stochastic Gradient Descent - alpha: [5, 10, 15], penalty: [l2, l1, elasticnet], max_iter: [100, 1000, 3000, 5000]

Overfitting and Underfitting Avoidance

When training ML models, two major concerns are to avoid overfitting and underfitting the models (Kumar, 2019). Overfitting means that that produced model is over-tailored for the particular training data. Overfit models perform well during the training and testing phase; however, they severely underperform when ingesting new data. Underfitting is the exact opposite problem of overfitting, meaning that models have not been trained enough. Underfit models poorly perform during the testing phase as well as when ingesting new data. It is typically much easier to detect underfitting because the ML models will yield poor metric results during the testing phase.

Numerous experiments were conducted with various selected and discarded algorithms, datasets, and optimization parameters. It can be seen later in the Results and Discussion section that underfitting was never really a problem since the large majority of trained ML models from Phase One were highly optimized with very positive metric results. However, as a result, one primary concern for this study was overfitting due to the genuine possibility of excessive model optimization.

When reviewing Phase One results, it could be argued that the models were overfit based on how they performed when predicting malicious traffic in Phase Two. However, there

are some techniques to avoid or reduce the potential for overfitting. These overfitting reduction techniques were considered and applied when initially training the ML models.

The first method to reduce overfitting is to apply more training data (Ameisen, 2020). The more training data utilized, the harder it is for a model to learn too much from a single small dataset. The size of all five of the datasets for this study was quite large and diverse. Only 10% of the total CIDDS-001 dataset was utilized because the total dataset size was causing the VM to exceed its 100 GB of RAM and crash the Python script for excessive memory allocation.

The second technique to reduce overfitting is to apply regularization, also known as normalization or scaling (Patterson & Gibson, 2017). Regularizing the ingested data while training a model will scale the data such that each feature and individual value is scaled and thereby have a fixed and proportional weight. Many different regularization methods can cause the resulting ML models to be more sensitive or resistant to outlier data. This study utilized the StandardScaler regularization method, which is considered reasonably sensitive to outlier data input.

The third method for overfitting reduction is applying cross-validation when training models (Bonaccorso, 2020). Cross-validation splits the original full dataset and divides it into dedicated subsets for training, testing, or validation. Cross-validation helps reduce variability, thereby limiting overfitting. This study applied 10-fold cross-validation techniques when initially training the ML models by utilizing the train_test_split function in scikit-learn and splitting each dataset into ten equal-sized folds, with nine folds used for training and one fold used for validation.

The fourth method for overfitting avoidance is feature selection and dimensionality reduction (Halder & Ozdemir, 2018). High-dimensional data, or data with several features, is computationally expensive and prone to overfitting due to higher complexity. Feature selection picks a particular subset of features as the most influential properties of the dataset that more accurately correlates to the best classification of new data, reducing the total number of features, overall complexity, and potential for overfitting. This research performed dimensionality reduction two times when training ML models. First, a set of attributes found in Zeek IDS logs were used to filter the available features found in each dataset; this initial manual feature reduction removed the greatest number of features from each dataset. Second,

the Principal Component Analysis function was also applied to each dataset, which automatically dropped features that did not directly correlate to the network traffic's overall categorization. Overall, the combination of these four overfitting avoidance techniques was applied to reduce the potential for overfitting. The results of Phase Two may still indicate that the ML models from Phase One were still overfitted. However, the overfitting avoidance precautions were still taken nonetheless.

Prediction Dataset Selection and Development

The second phase of this research involved applying the trained models to predict malicious traffic on unseen datasets. However, to verify the predicted values' effectiveness and accuracy, the research required labeled information about the prediction dataset to know what portion of the traffic was considered malicious versus innocuous. Due to this verification requirement, the second phase of research utilized one of the previously selected datasets for training and applied it as a labeled prediction and verification dataset. This prediction dataset utilized the raw PCAP files fed into the Zeek IDS to produce connection logs. These IDS logs were then merged into a single PCAP and fed into the Phase Two Python script with the trained models to make predictions of malicious traffic based on the parsed logs. It should be noted that this Phase Two Python script utilized a custom Python package called parsezeeklogs which was installed from the Python Package Index (PyPI) repository ("parsezeeklogs 2.0.1," 2019). This package is just a lightweight utility for reading Zeek IDS log files and outputting them into CSV format. Everything else in the Phase One and Phase Two scripts utilizes public Python libraries and includes custom-written code for this research. The Phase Two script's prediction values were compared to the CSV files' labels from the same prediction dataset.

The initial dataset selected as the prediction and verification dataset was the UNSW-NB15 dataset. This dataset was selected because it includes publicly available labeled CSV data as well as raw PCAPs. The UNSW-NB15 dataset contains roughly 100 GB of PCAP files. However, there were numerous errors while merging the PCAP files to then feed into the Zeek IDS. These errors may be due to synthetic packets produced by particular tools for the UNSW-NB15 dataset during its development. Initially, the mergecap tool was utilized to combine the PCAPs ("mergecap," 2021). However, while this tool is preinstalled on Ubuntu

and several other Linux systems, it has very poor error handling. While using mergecap, the tool would crash due to malformed packets and would not produce any single PCAP output if encountering an error. As a result, another public tool was identified called joincap, available on GitHub (assafmo, 2020). This tool is very similar to mergecap; however, it focuses on graceful error handling and will skip over malformed packets when combining PCAP files. Through joincap, the 100 GB of PCAPs were joined together into a single 60 GB PCAP. Due to the extreme loss of raw PCAP files when merging the files, the UNSW-NB15 dataset was not selected for prediction and verification.

The final and officially selected dataset utilized for prediction comparison was the CICIDS 2017 dataset. This dataset is publicly available online and includes both labeled CSV files as well as raw PCAP files. The PCAPs were fed through the mergecap tool and successfully produced a single PCAP file with no errors ("mergecap," 2021). Additionally, the combined PCAP was the same size as the total split individual PCAP files, indicating an entirely successful merge. As a result, the CICIDS 2017 was selected as the prediction and verification dataset for Phase Two of this research. This dataset acted as a form of control group since it was used in both research phases. The CICIDS 2017 dataset was utilized in the first phase of research in the form of labeled CSV data as one of the five datasets to train the 105 ML models. Additionally, the raw PCAP files of the CICIDS 2017 dataset were utilized for the second phase of research to be ingested into Zeek IDS to produce Zeek connection logs and be ingested into the Phase Two Python script.

CHAPTER 4

RESULTS AND DISCUSSION

This section will review each phase of the research results and explain the different experiments' outcomes. As a quick review, this dissertation's research utilized the quantitative experimental study design, which was divided into two primary phases. The first phase of this research involved generating ML models based on a series of datasets, ML algorithms, and scoring methods. The first ML models are trained and evaluated on labeled data and then fed into the second research phase. This second phase of research applies the previously trained models on unseen network traffic to observe if the theoretical detection rates still apply when scanning for malicious traffic in new network data.

Phase One – Results

For the first phase of research, a Python script was utilized to read in datasets and then trained a series of ML models based on a combination of ML algorithms, datasets, and scoring methods. The first time the script was executed, the algorithms were initially optimized for specific scoring methods – accuracy, precision, and F1 score. The Grid Search Cross-Validation function (GridSearchCV) operates by testing a series of parameters and values to identify which parameters yield the best score based on the selected scoring method. Note that the Naïve Bayes algorithm cannot be optimized due to its nature and the lack of parameter options. After optimizing the parameters, the script was rerun and produced a total of 105 models. The script output contained a series of CSV files, which contained the evaluation metrics of each model.

5 datasets * 7 ML algorithms * 3 scoring optimization methods = **105 total trained models**

Datasets – KDD 99, NSL-KDD, UNSW-NB15, CICIDS 2017, and CICIDS-001

ML algorithms – Naïve Bayes, Decision Tree, Random Forest, Ada Boost, Bagging Classifier, Logistic Regression, and Stochastic Gradient Descent

Scoring (optimization) methods – accuracy, precision_macro (precision), and f1_macro (F1 score)

Evaluation metrics – accuracy, precision, recall, and F1 score

Phase One – Parameter Optimization

Score Method	Algorithm	Best Parameters	Best Score	Train Time
accuracy	Decision Tree	criterion='entropy', max_features='log2'	0.99796	0.48292
accuracy	Random Forest	criterion='gini', max_features='log2', n_estimators=100	0.99806	210.16277
accuracy	Ada Boost	learning_rate=1.5, n_estimators=5000	0.99793	814.39637
accuracy	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=5000	0.99801	243.03303
accuracy	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.96866	0.95317
accuracy	Stochastic Gradient Descent	alpha=15, max_iter=1000, penalty='l2'	0.83157	1.83523
precision_macro	Decision Tree	criterion='entropy', max_features='auto'	0.99572	0.49719
precision_macro	Random Forest	criterion='gini', max_features='log2', n_estimators=100	0.99577	213.23284
precision_macro	Ada Boost	learning_rate=1.5, n_estimators=5000	0.99557	790.43513
precision_macro	Bagging Classifier	max_features=5, max_samples=0.5, n_estimators=5000	0.99566	256.45741
precision_macro	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.96980	1.27933
precision_macro	Stochastic Gradient Descent	alpha=15, max_iter=1000, penalty='l2'	0.59331	2.53445
f1_macro	Decision Tree	criterion='entropy', max_features='auto'	0.99682	0.48747
f1_macro	Random Forest	criterion='gini', max_features='sqrt', n_estimators=100	0.99695	214.75251
f1_macro	Ada Boost	learning_rate=1.5, n_estimators=5000	0.99676	796.15304
f1_macro	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=100	0.99689	246.88263
f1_macro	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.94873	1.28646
f1_macro	Stochastic Gradient Descent	alpha=15, max_iter=1000, penalty='l2'	0.53052	2.66449

Table 4: Phase One – KDD 99 Models – Parameter Optimization

Score Method	Algorithm	Best Parameters	Best Score	Train Time
accuracy	Decision Tree	criterion='entropy', max_features='auto'	0.98169	0.02364
accuracy	Random Forest	criterion='gini', max_features='log2', n_estimators=5000	0.98371	33.80612
accuracy	Ada Boost	learning_rate=1.5, n_estimators=3000	0.97926	21.86760
accuracy	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=1000	0.98243	40.31787
accuracy	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.90567	0.05589
accuracy	Stochastic Gradient Descent	alpha=5, max_iter=1000, penalty='l2'	0.56294	0.06785
precision_macro	Decision Tree	criterion='gini', max_features='auto'	0.98071	0.02837
precision_macro	Random Forest	criterion='entropy', max_features='sqrt', n_estimators=100	0.98384	32.76877
precision_macro	Ada Boost	learning_rate=1.5, n_estimators=3000	0.97929	21.31620
precision_macro	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=3000	0.98285	40.37516
precision_macro	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.90559	0.06648
precision_macro	Stochastic Gradient Descent	alpha=5, max_iter=3000, penalty='l2'	0.73119	0.08599
f1_macro	Decision Tree	criterion='entropy', max_features='log2'	0.98161	0.03075
f1_macro	Random Forest	criterion='gini', max_features='log2', n_estimators=1000	0.98376	33.18701
f1_macro	Ada Boost	learning_rate=1.5, n_estimators=3000	0.97924	21.55623
f1_macro	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=1000	0.98261	40.31422
f1_macro	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.90562	0.06172
f1_macro	Stochastic Gradient Descent	alpha=5, max_iter=5000, penalty='l2'	0.44130	0.08950

Table 5: Phase One – NSL-KDD Models – Parameter Optimization

Score Method	Algorithm	Best Parameters	Best Score	Train Time
accuracy	Decision Tree	criterion='gini', max_features='sqrt'	0.98941	0.43377
accuracy	Random Forest	criterion='entropy', max_features='log2', n_estimators=100	0.99123	247.63014
accuracy	Ada Boost	learning_rate=1, n_estimators=3000	0.98921	605.48672
accuracy	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=1000	0.99115	189.27455
accuracy	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.97423	0.66619
accuracy	Stochastic Gradient Descent	alpha=15, max_iter=100, penalty='l2'	0.87683	0.98321
precision_macro	Decision Tree	criterion='entropy', max_features='sqrt'	0.97708	0.35475
precision_macro	Random Forest	criterion='entropy', max_features='sqrt', n_estimators=1000	0.98003	253.54329
precision_macro	Ada Boost	learning_rate=1, n_estimators=5000	0.97499	631.19595
precision_macro	Bagging Classifier	max_features=1, max_samples=0.5, n_estimators=100	0.98453	185.29858
precision_macro	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.94536	0.82141
precision_macro	Stochastic Gradient Descent	alpha=10, max_iter=100, penalty='l2'	0.53703	1.35347
f1_macro	Decision Tree	criterion='gini', max_features='log2'	0.97607	0.40691
f1_macro	Random Forest	criterion='entropy', max_features='auto', n_estimators=100	0.98013	251.29138
f1_macro	Ada Boost	learning_rate=1, n_estimators=3000	0.97550	613.18410
f1_macro	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=3000	0.97984	188.67667
f1_macro	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.94090	0.79907
f1_macro	Stochastic Gradient Descent	alpha=15, max_iter=1000, penalty='l2'	0.48656	1.22815

Table 6: Phase One – UNSW-NB15 Models – Parameter Optimization

Score Method	Algorithm	Best Parameters	Best Score	Train Time
accuracy	Decision Tree	criterion='entropy', max_features='sqrt'	0.98867	0.51552
accuracy	Random Forest	criterion='gini', max_features='sqrt', n_estimators=1000	0.99160	364.72154
accuracy	Ada Boost	learning_rate=1.5, n_estimators=5000	0.92834	552.42350
accuracy	Bagging Classifier	max_features=5, max_samples=0.5, n_estimators=5000	0.99115	257.60872
accuracy	Logistic Regression	C=1, max_iter=2500, penalty='l2'	0.84413	0.48867
accuracy	Stochastic Gradient Descent	alpha=15, max_iter=3000, penalty='l2'	0.80258	0.94233
precision_macro	Decision Tree	criterion='entropy', max_features='auto'	0.98147	0.45460
precision_macro	Random Forest	criterion='gini', max_features='sqrt', n_estimators=1000	0.98662	366.67431
precision_macro	Ada Boost	learning_rate=1.5, n_estimators=5000	0.89797	558.84882
precision_macro	Bagging Classifier	max_features=5, max_samples=0.5, n_estimators=5000	0.98653	260.66536
precision_macro	Logistic Regression	C=0.1, max_iter=2500, penalty='l2'	0.77686	0.65171
precision_macro	Stochastic Gradient Descent	alpha=15, max_iter=1000, penalty='l2'	0.47620	1.42750
f1_macro	Decision Tree	criterion='entropy', max_features='log2'	0.98226	0.46069
f1_macro	Random Forest	criterion='entropy', max_features='auto', n_estimators=1000	0.98678	371.29855
f1_macro	Ada Boost	learning_rate=1.5, n_estimators=5000	0.88283	560.62389
f1_macro	Bagging Classifier	max_features=5, max_samples=0.5, n_estimators=3000	0.98603	256.17446
f1_macro	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.69967	0.64171
f1_macro	Stochastic Gradient Descent	alpha=15, max_iter=1000, penalty='l2'	0.46995	1.31777

Table 7: Phase One – CICIDS 2017 Models – Parameter Optimization

Score Method	Algorithm	Best Parameters	Best Score	Train Time
accuracy	Decision Tree	criterion='entropy', max_features='sqrt'	0.99882	0.49638
accuracy	Random Forest	criterion='entropy', max_features='sqrt', n_estimators=1000	0.99927	259.87577
accuracy	Ada Boost	learning_rate=1.5, n_estimators=5000	0.99836	779.99958
accuracy	Bagging Classifier	max_features=5, max_samples=0.5, n_estimators=1000	0.99910	223.35500
accuracy	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.94686	0.74863
accuracy	Stochastic Gradient Descent	alpha=5, max_iter=100, penalty='l2'	0.89727	1.12009
precision_macro	Decision Tree	criterion='entropy', max_features='auto'	0.99703	0.46554
precision_macro	Random Forest	criterion='entropy', max_features='log2', n_estimators=3000	0.99817	264.88818
precision_macro	Ada Boost	learning_rate=1.5, n_estimators=5000	0.99516	769.69732
precision_macro	Bagging Classifier	max_features=0.5, max_samples=0.5, n_estimators=1000	0.99821	228.25839
precision_macro	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.89943	0.95949
precision_macro	Stochastic Gradient Descent	alpha=5, max_iter=100, penalty='l2'	0.44863	1.55976
f1_macro	Decision Tree	criterion='entropy', max_features='log2'	0.99690	0.47622
f1_macro	Random Forest	criterion='entropy', max_features='auto', n_estimators=3000	0.99801	265.96089
f1_macro	Ada Boost	learning_rate=1.5, n_estimators=5000	0.99549	785.13437
f1_macro	Bagging Classifier	max_features=5, max_samples=0.5, n_estimators=1000	0.99756	225.29216
f1_macro	Logistic Regression	C=10, max_iter=2500, penalty='l2'	0.83344	0.95891
f1_macro	Stochastic Gradient Descent	alpha=5, max_iter=100, penalty='l2'	0.47293	1.63497

Table 8: Phase One – CIDDS-001 Models – Parameter Optimization

Phase One – KDD 99 Dataset Results

Model	**Accuracy**	Precision	Recall	F1 Score	Train Time
Naïve Bayes	**0.97303**	0.96586	0.94853	0.95689	0.19895
Decision Tree	**0.99816**	0.99600	0.99824	0.99712	0.36601
Random Forest	**0.99819**	0.99603	0.99831	0.99716	2.47200
Ada Boost	**0.99794**	0.99554	0.99804	0.99678	949.45873
Bagging Classifier	**0.99816**	0.99592	0.99831	0.99711	33.93054
Logistic Regression	**0.96855**	0.96973	0.93048	0.94854	0.43249
Stochastic Gradient Descent	**0.80108**	0.64298	0.50011	0.44508	0.67686

Table 9: Phase One – KDD 99 Models – Accuracy Optimization

Model	Accuracy	**Precision**	Recall	F1 Score	Train Time
Naïve Bayes	0.97303	**0.96586**	0.94853	0.95689	0.19037
Decision Tree	0.99812	**0.99598**	0.99814	0.99706	0.32635
Random Forest	0.99818	**0.99602**	0.99830	0.99715	2.49991
Ada Boost	0.99794	**0.99554**	0.99804	0.99678	966.79733
Bagging Classifier	0.99816	**0.99598**	0.99829	0.99713	70.83681
Logistic Regression	0.96855	**0.96973**	0.93048	0.94854	0.35421
Stochastic Gradient Descent	0.80108	**0.80108**	1	0.88956	0.19012

Table 10: Phase One – KDD 99 Models – Precision Optimization

Model	Accuracy	Precision	Recall	**F1 Score**	Train Time
Naïve Bayes	0.97303	0.96586	0.94853	**0.95689**	0.15929
Decision Tree	0.99816	0.99602	0.99822	**0.99712**	0.20516
Random Forest	0.99818	0.99602	0.99830	**0.99715**	1.78166
Ada Boost	0.99794	0.99554	0.99804	**0.99678**	943.96914
Bagging Classifier	0.99814	0.99589	0.99830	**0.99709**	1.05218
Logistic Regression	0.96855	0.96973	0.93048	**0.94854**	0.42434
Stochastic Gradient Descent	0.80108	0.80108	1	**0.88956**	0.18991

Table 11: Phase One – KDD 99 Models – F1 Score Optimization

Phase One – NSL-KDD Dataset Results

Model	**Accuracy**	Precision	Recall	F1 Score	Train Time
Naïve Bayes	**0.89173**	0.89252	0.89246	0.89173	0.00726
Decision Tree	**0.98613**	0.98608	0.98616	0.98612	0.01268
Random Forest	**0.98828**	0.98831	0.98824	0.98827	3.80120
Ada Boost	**0.98155**	0.98154	0.98153	0.98154	15.72710
Bagging Classifier	**0.98734**	0.98742	0.98725	0.98733	0.58591
Logistic Regression	**0.90648**	0.90636	0.90653	0.90643	0.01096
Stochastic Gradient Descent	**0.51387**	0.51387	1	0.67888	0.00446

Table 12: Phase One – NSL-KDD Models – Accuracy Optimization

Model	Accuracy	**Precision**	Recall	F1 Score	Train Time
Naïve Bayes	0.89173	**0.89252**	0.89246	0.89173	0.00478
Decision Tree	0.98741	**0.98739**	0.98741	0.98740	0.00858
Random Forest	0.98815	**0.98817**	0.98811	0.98814	0.05314
Ada Boost	0.98155	**0.98154**	0.98153	0.98154	15.61126
Bagging Classifier	0.98734	**0.98742**	0.98725	0.98733	0.87895
Logistic Regression	0.90648	**0.90636**	0.90653	0.90643	0.00951
Stochastic Gradient Descent	0.51387	**0.51387**	1	0.67888	0.00392

Table 13: Phase One – NSL-KDD Models – Precision Optimization

Model	Accuracy	Precision	Recall	**F1 Score**	Train Time
Naïve Bayes	0.89173	0.89252	0.89246	**0.89173**	0.00491
Decision Tree	0.98647	0.98643	0.98649	**0.98646**	0.01002
Random Forest	0.98822	0.98825	0.98817	**0.98821**	0.81297
Ada Boost	0.98155	0.98154	0.98153	**0.98154**	15.61979
Bagging Classifier	0.98734	0.98742	0.98725	**0.98733**	0.63036
Logistic Regression	0.90648	0.90636	0.90653	**0.90643**	0.00933
Stochastic Gradient Descent	0.54612	0.71213	0.55799	**0.45311**	0.00573

Table 14: Phase One – NSL-KDD Models – F1 Score Optimization

Phase One – UNSW-NB15 Dataset Results

Model	**Accuracy**	Precision	Recall	F1 Score	Train Time
Naïve Bayes	**0.89293**	0.78412	0.64256	0.68054	0.08435
Decision Tree	**0.99170**	0.98080	0.98154	0.98117	0.37921
Random Forest	**0.99306**	0.98372	0.98481	0.98426	1.49134
Ada Boost	**0.98971**	0.97442	0.97912	0.97675	466.55119
Bagging Classifier	**0.99317**	0.98403	0.98496	0.98449	19.65784
Logistic Regression	**0.97419**	0.94521	0.93655	0.94082	0.35886
Stochastic Gradient Descent	**0.87405**	0.87405	1	0.93279	0.08537

Table 15: Phase One – UNSW-NB15 Models – Accuracy Optimization

Model	Accuracy	**Precision**	Recall	F1 Score	Train Time
Naïve Bayes	0.89293	**0.78412**	0.64256	0.68054	0.08946
Decision Tree	0.99187	**0.98118**	0.98195	0.98156	0.29828
Random Forest	0.99303	**0.98386**	0.98449	0.98418	12.24554
Ada Boost	0.98989	**0.97491**	0.97942	0.97715	778.32787
Bagging Classifier	0.97446	**0.98575**	0.89864	0.93638	1.08145
Logistic Regression	0.97419	**0.94521**	0.93655	0.94082	0.35494
Stochastic Gradient Descent	0.87405	**0.87405**	1	0.93279	0.08048

Table 16: Phase One – UNSW-NB15 Models – Precision Optimization

Model	Accuracy	Precision	Recall	**F1 Score**	Train Time
Naïve Bayes	0.89293	0.78412	0.64256	**0.68054**	0.08273
Decision Tree	0.99126	0.97967	0.98066	**0.98016**	0.25876
Random Forest	0.99305	0.98386	0.98459	**0.98422**	1.38001
Ada Boost	0.98971	0.97442	0.97912	**0.97675**	480.33834
Bagging Classifier	0.99318	0.98397	0.98507	**0.98452**	54.13052
Logistic Regression	0.97419	0.94521	0.93655	**0.94082**	0.52767
Stochastic Gradient Descent	0.87405	0.87405	1	**0.93279**	0.09553

Table 17: Phase One – UNSW-NB15 Models – F1 Score Optimization

Phase One – CICIDS 2017 Dataset Results

Model	**Accuracy**	Precision	Recall	F1 Score	Train Time
Naïve Bayes	**0.85553**	0.81139	0.68245	0.71751	0.09344
Decision Tree	**0.99318**	0.98845	0.99007	0.98926	0.42589
Random Forest	**0.99415**	0.99032	0.99124	0.99078	20.96378
Ada Boost	**0.92856**	0.89923	0.86886	0.88291	723.04974
Bagging Classifier	**0.99440**	0.99115	0.99117	0.99116	176.10695
Logistic Regression	**0.84409**	0.77625	0.67020	0.70009	0.19176
Stochastic Gradient Descent	**0.80256**	0.80256	1	0.89047	0.12037

Table 18: Phase One – CICIDS 2017 Models – Accuracy Optimization

Model	Accuracy	**Precision**	Recall	F1 Score	Train Time
Naïve Bayes	0.85553	**0.81139**	0.68245	0.71751	0.09795
Decision Tree	0.99309	**0.98826**	0.98997	0.98911	0.47588
Random Forest	0.99415	**0.99033**	0.99122	0.99077	20.17999
Ada Boost	0.92856	**0.89923**	0.86886	0.88291	734.92824
Bagging Classifier	0.99440	**0.99113**	0.99121	0.99117	171.72592
Logistic Regression	0.84409	**0.77625**	0.67020	0.70009	0.16185
Stochastic Gradient Descent	0.80256	**0.80256**	1	0.89047	0.27912

Table 19: Phase One – CICIDS 2017 Models – Precision Optimization

Model	Accuracy	Precision	Recall	**F1 Score**	Train Time
Naïve Bayes	0.85553	0.81139	0.68245	**0.71751**	0.08759
Decision Tree	0.99318	0.98842	0.99010	**0.98926**	0.41032
Random Forest	0.99416	0.99029	0.99132	**0.99080**	18.56984
Ada Boost	0.92856	0.89923	0.86886	**0.88291**	735.96754
Bagging Classifier	0.99440	0.99116	0.99119	**0.99117**	107.24517
Logistic Regression	0.84409	0.77625	0.67020	**0.70009**	0.18086
Stochastic Gradient Descent	0.80256	0.80256	1	**0.89047**	0.08245

Table 20: Phase One – CICIDS 2017 Models – F1 Score Optimization

Phase One – CIDDS-001 Dataset Results

Model	**Accuracy**	Precision	Recall	F1 Score	Train Time
Naïve Bayes	**0.90131**	0.74686	0.88045	0.79145	0.16439
Decision Tree	**0.99951**	0.99877	0.99855	0.99866	0.50578
Random Forest	**0.99963**	0.99908	0.99893	0.99900	14.92514
Ada Boost	**0.99852**	0.99571	0.99627	0.99599	1203.30446
Bagging Classifier	**0.99959**	0.99904	0.99872	0.99888	22.60481
Logistic Regression	**0.95024**	0.91176	0.79929	0.84416	0.37160
Stochastic Gradient Descent	**0.89727**	0.89727	1	0.94585	0.14127

Table 21: Phase One – CIDDS-001 Models – Accuracy Optimization

Model	Accuracy	**Precision**	Recall	F1 Score	Train Time
Naïve Bayes	0.90131	**0.74686**	0.88045	0.79145	0.15782
Decision Tree	0.99948	**0.99869**	0.99847	0.99858	0.37718
Random Forest	0.99963	**0.99908**	0.99893	0.99900	40.93319
Ada Boost	0.99852	**0.99571**	0.99627	0.99599	1116.44332
Bagging Classifier	0.99956	**0.99909**	0.99851	0.99880	15.54522
Logistic Regression	0.95024	**0.91176**	0.79929	0.84416	0.35591
Stochastic Gradient Descent	0.89727	**0.89727**	1	0.94585	0.13498

Table 22: Phase One – CIDDS-001 Models – Precision Optimization

Model	Accuracy	Precision	Recall	**F1 Score**	Train Time
Naïve Bayes	0.90131	0.74686	0.88045	**0.79145**	0.13131
Decision Tree	0.99950	0.99873	0.99853	**0.99863**	0.31852
Random Forest	0.99963	0.99908	0.99893	**0.99900**	39.92271
Ada Boost	0.99852	0.99569	0.99627	**0.99598**	1051.30573
Bagging Classifier	0.99959	0.99903	0.99877	**0.99890**	22.53338
Logistic Regression	0.95024	0.91176	0.79929	**0.84416**	0.36495
Stochastic Gradient Descent	0.89727	0.89727	1	**0.94585**	0.13185

Table 23: Phase One – CIDDS-001 Models – F1 Score Optimization

Phase One – Discussion

Parameter Optimization

The first time the script was executed, the goal was to identify optimized parameters for each model. It should be noted that the Naïve Bayes algorithm cannot be optimized due to the nature of the algorithm and a lack of parameters available. Additionally, when reviewing the parameter optimization scores, it can be seen that some of the ideal parameters remain consistent between different tests. The impact of these same selected and optimized parameters can be better seen when reviewing the actual trained models' evaluation metrics as they yield consistent evaluation metric scores.

Many of the models were considered highly optimized as most of them were able to obtain roughly 99% within their specified scoring method. Out of the 90 total trained models (Naïve Bayes cannot be optimized), only 30 were considered poorly optimized. Regarding this phase's goals, anything lower than a 95% score within the respective scoring method is considered poorly optimized. There was a noticeable trend identified by reviewing the results. All the algorithms produced highly optimized models and scores except for three particular algorithms. The Stochastic Gradient Descent, Logistic Regression, and Ada Boost algorithms and associated models consistently underperformed.

The worst performing algorithm was the Stochastic Gradient Descent algorithm, which consistently failed to reach a score higher than 95% across every single test, regardless of the particular dataset or scoring method. The second worst performing algorithm was the Logistic Regression algorithm. This algorithm consistently failed to reach a proper optimization score across each dataset when the specified scoring method was set to f1_macro. Additionally, there were multiple instances where the Logistic Regression algorithm failed to optimize appropriately and yielded inconsistent metrics. Lastly, the only other algorithm that failed to reach proper optimization was the Ada Boost algorithm. This algorithm only failed when interacting with the CICIDS 2017 dataset. However, it failed all three scoring methods.

Model Production

Similar to the parameter optimization phase, the actual production of the models yielded promising results. The large majority of models obtained 95% or better scores across

each evaluation metric, including accuracy, precision, recall, and F1 score. It should be noted that some of the different scoring methods utilized the same parameter configurations. The reusing of the same function parameters caused some of the models to yield remarkably similar, if not exact, metrics between tests with different scoring methods.

The only algorithm that yielded consistently low evaluation metrics was Stochastic Gradient Descent. Across every test, regardless of dataset or scoring method, the SGD models always produced evaluation metrics that were well below the 95% threshold. There were large fluctuations of metric scores when trained using the KDD 99 and NSL-KDD datasets. However, the UNSW-NB15, CICIDS 2017, and CIDDS-001 datasets yielded consistently low metric scores.

Additionally, another interesting observation of these results indicates that the differing scoring or optimization methods did not generate significantly different results. Each model produced using different scoring methods generated slightly different metrics. However, the models never caused the metrics to change enough to cross the threshold between success (>95%) and failure (<95%). There were some instances where the optimization method caused significant fluctuations, as seen in the tests involving the Stochastic Gradient Descent algorithm and the KDD 99 and NSL-KDD datasets. Therefore, while the differing scoring methods may have produced better respective metrics, they were still not impactful enough to change the score's overall category (success or failure).

While reviewing the models trained using the KDD 99 dataset, it appears that the Naïve Bayes, Decision Tree, Random Forest, Ada Boost, and Bagging Classifier algorithms maintained consistent metric scores greater than 95% across all the different scoring methods. The Logistic Regression algorithm achieved high scores (>95%) in accuracy or precision across all scoring methods; however, the algorithm also produced low scores (<95%) in recall and f1 values across all scoring methods.

When trained using the NSL-KDD dataset, the Decision Tree, Random Forest, Ada Boost, and Bagging Classifier algorithms produced positive metrics of >95% regardless of the scoring method. The Naïve Bayes and Logistic Regression algorithms yielded consistently below 95% across each scoring method.

The models trained off the UNSW-NB15 dataset produced highly consistent metrics with hardly any fluctuations between differing scoring methods. The Decision Tree, Random

Forest, Ada Boost, and Bagging Classifier algorithms produced positive results above the 95% threshold regardless of the scoring method. Unfortunately, the Naïve Bayes algorithm consistently yielded low metrics (<95%) across each scoring method. The Logistic Regression algorithm produced models that achieved successful metrics (>95%) in terms of accuracy metrics; however, it also maintained poor results (<95%) in terms of precision, recall, and F1 score metrics across all scoring methods.

The CICIDS 2017 dataset yielded interesting metrics because it was the only dataset that caused the Ada Boost algorithm to produce low metrics consistently. These low Ada Boost metrics may have been due to the dataset's size or possibly features selected for training. Additionally, this dataset also generated highly consistent metrics between each of the scoring method tests. The Decision Tree, Random Forest, and Bagging Classifier algorithms all produced models that generated positive metrics above the 95% threshold. However, the Naïve Bayes and Ada Boost algorithms regularly stayed below the 95% threshold across all scoring methods.

The models trained using the CIDDS-001 dataset also maintained remarkably consistent results between algorithms. The Decision Tree, Random Forest, Ada Boost, and Bagging classifier algorithms achieved positive metrics (>95%) across all scoring methods. However, the Naïve Bayes algorithm maintained low metrics (<95%) regardless of the scoring method. The Logistic Regression algorithm achieved positive results (>95%) in the accuracy metric; however, it also generated negative results in precision, recall, and F1 score across all scoring methods.

Phase Two – Results

This second phase of research attempted to apply the trained ML models from the previous phase and see how the theoretical models handle unseen network traffic. For this phase of research, the raw PCAPs of the CICIDS 2017 dataset were utilized. The CICIDS 2017 dataset was used in the first phase for training a model. However, this same dataset was also used in this second phase as a sort of control group and a known true value indicator for flagged traffic predictions. It should be noted for clarification that the CICIDS 2017 dataset was used in both phases of research. The first phase utilized the CSV data, which consisted of PCAP traffic and manual input for labels, whereas the second phase utilized only the raw PCAP traffic only which were then fed into the Zeek IDS for log generation. A second Python script was utilized for this second phase as well. This second script imported the Zeek logs along with the previously trained ML models. After parsing and mapping Zeek log fields to the applicable features for each trained ML model, the script successfully produced a prediction value of malicious traffic. This script allowed for previously trained ML models to predict malicious traffic on unseen data. However, the accuracy of these results appears to vary dramatically. The results are, again, divided up by trained dataset, ML algorithm, and scoring method.

Phase Two – CICIDS 2017 – True Values

	Packets	Percentage
Normal	2273097	0.80300
Malicious	557646	0.19699

Table 24: Phase Two – CICIDS 2017 – True Values

Phase Two – Trained Models' Predictions on CICIDS 2017 Raw Traffic

Model	Score Method	Malicious	Normal	Total	Flagged Traffic
Naïve Bayes	Accuracy	0	2119207	2119207	0
Decision Tree	Accuracy	1692378	426829	2119207	0.79859
Random Forest	Accuracy	0	2119207	2119207	0
Ada Boost	Accuracy	0	2119207	2119207	0
Bagging Classifier	Accuracy	0	2119207	2119207	0
Logistic Regression	Accuracy	1731105	388102	2119207	0.81686
Stochastic Gradient Descent	Accuracy	2119207	0	2119207	1
Naïve Bayes	Precision	0	2119207	2119207	0
Decision Tree	Precision	2119207	0	2119207	1
Random Forest	Precision	0	2119207	2119207	0
Ada Boost	Precision	0	2119207	2119207	0
Bagging Classifier	Precision	426829	1692378	2119207	0.20141
Logistic Regression	Precision	1731105	388102	2119207	0.81686
Stochastic Gradient Descent	Precision	1731105	388102	2119207	0.81686
Naïve Bayes	F1 Score	0	2119207	2119207	0
Decision Tree	F1 Score	0	2119207	2119207	0
Random Forest	F1 Score	0	2119207	2119207	0
Ada Boost	F1 Score	0	2119207	2119207	0
Bagging Classifier	F1 Score	0	2119207	2119207	0
Logistic Regression	F1 Score	1731105	388102	2119207	0.81686
Stochastic Gradient Descent	F1 Score	2119207	0	2119207	1

Table 25: Phase Two – KDD 99 Models – Predictions on CICIDS 2017 Raw Traffic

Model	Score Method	Malicious	Normal	Total	Flagged Traffic
Naïve Bayes	Accuracy	2080480	38727	2119207	0.98173
Decision Tree	Accuracy	38727	2080480	2119207	0.01827
Random Forest	Accuracy	0	2119207	2119207	0
Ada Boost	Accuracy	38727	2080480	2119207	0.01827
Bagging Classifier	Accuracy	0	2119207	2119207	0
Logistic Regression	Accuracy	1731105	388102	2119207	0.81686
Stochastic Gradient Descent	Accuracy	1731105	388102	2119207	0.81686
Naïve Bayes	Precision	2080480	38727	2119207	0.98173
Decision Tree	Precision	0	2119207	2119207	0
Random Forest	Precision	38727	2080480	2119207	0.01827
Ada Boost	Precision	38727	2080480	2119207	0.01827
Bagging Classifier	Precision	0	2119207	2119207	0
Logistic Regression	Precision	1731105	388102	2119207	0.81686
Stochastic Gradient Descent	Precision	1731105	388102	2119207	0.81686
Naïve Bayes	F1 Score	2080480	38727	2119207	0.98173
Decision Tree	F1 Score	0	2119207	2119207	0
Random Forest	F1 Score	0	2119207	2119207	0
Ada Boost	F1 Score	38727	2080480	2119207	0.01827
Bagging Classifier	F1 Score	0	2119207	2119207	0
Logistic Regression	F1 Score	1731105	388102	2119207	0.81686
Stochastic Gradient Descent	F1 Score	1731105	388102	2119207	0.81686

Table 26: Phase Two – NSL-KDD Models – Predictions on CICIDS 2017 Raw Traffic

Model	Score Method	Malicious	Normal	Total	Flagged Traffic
Naïve Bayes	Accuracy	2119207	0	2119207	1
Decision Tree	Accuracy	2081422	37785	2119207	0.98217
Random Forest	Accuracy	2084743	34464	2119207	0.98374
Ada Boost	Accuracy	3451	2115756	2119207	0.00163
Bagging Classifier	Accuracy	1	2119206	2119207	0
Logistic Regression	Accuracy	2119207	0	2119207	1
Stochastic Gradient Descent	Accuracy	0	2119207	2119207	0
Naïve Bayes	Precision	2119207	0	2119207	1
Decision Tree	Precision	38208	2080999	2119207	0.01803
Random Forest	Precision	2084403	34804	2119207	0.98358
Ada Boost	Precision	0	2119207	2119207	0
Bagging Classifier	Precision	0	2119207	2119207	0
Logistic Regression	Precision	2119207	0	2119207	1
Stochastic Gradient Descent	Precision	2119207	0	2119207	1
Naïve Bayes	F1 Score	2119207	0	2119207	1
Decision Tree	F1 Score	0	2119207	2119207	0
Random Forest	F1 Score	2119194	13	2119207	0.99999
Ada Boost	F1 Score	3451	2115756	2119207	0.00163
Bagging Classifier	F1 Score	0	2119207	2119207	0
Logistic Regression	F1 Score	2119207	0	2119207	1
Stochastic Gradient Descent	F1 Score	0	2119207	2119207	0

Table 27: Phase Two – UNSW-NB15 Models – Predictions on CICIDS 2017 Raw Traffic

Model	Score Method	Malicious	Normal	Total	Flagged Traffic
Naïve Bayes	Accuracy	531212	1587995	2119207	0.25067
Decision Tree	Accuracy	236062	1883145	2119207	0.11139
Random Forest	Accuracy	360301	1758906	2119207	0.17002
Ada Boost	Accuracy	783534	1335673	2119207	0.36973
Bagging Classifier	Accuracy	369295	1749912	2119207	0.17426
Logistic Regression	Accuracy	8948	2110259	2119207	0.00422
Stochastic Gradient Descent	Accuracy	0	2119207	2119207	0
Naïve Bayes	Precision	531212	1587995	2119207	0.25067
Decision Tree	Precision	415476	1703731	2119207	0.19605
Random Forest	Precision	361007	1758200	2119207	0.17035
Ada Boost	Precision	783534	1335673	2119207	0.36973
Bagging Classifier	Precision	368940	1750267	2119207	0.17409
Logistic Regression	Precision	8948	2110259	2119207	0.00422
Stochastic Gradient Descent	Precision	0	2119207	2119207	0
Naïve Bayes	F1 Score	531212	1587995	2119207	0.25067
Decision Tree	F1 Score	400550	1718657	2119207	0.18901
Random Forest	F1 Score	360171	1759036	2119207	0.16996
Ada Boost	F1 Score	783534	1335673	2119207	0.36973
Bagging Classifier	F1 Score	368985	1750222	2119207	0.17411
Logistic Regression	F1 Score	8948	2110259	2119207	0.00422
Stochastic Gradient Descent	F1 Score	0	2119207	2119207	0

Table 28: Phase Two – CICIDS 2017 Models – Predictions on CICIDS 2017 Raw Traffic

Model	Score Method	Malicious	Normal	Total	Flagged Traffic
Naïve Bayes	Accuracy	0	2119207	2119207	0
Decision Tree	Accuracy	0	2119207	2119207	0
Random Forest	Accuracy	0	2119207	2119207	0
Ada Boost	Accuracy	30	2119177	2119207	1.42E-05
Bagging Classifier	Accuracy	0	2119207	2119207	0
Logistic Regression	Accuracy	2119207	0	2119207	1
Stochastic Gradient Descent	Accuracy	0	2119207	2119207	0
Naïve Bayes	Precision	0	2119207	2119207	0
Decision Tree	Precision	0	2119207	2119207	0
Random Forest	Precision	0	2119207	2119207	0
Ada Boost	Precision	30	2119177	2119207	1.42E-05
Bagging Classifier	Precision	0	2119207	2119207	0
Logistic Regression	Precision	2119207	0	2119207	1
Stochastic Gradient Descent	Precision	0	2119207	2119207	0
Naïve Bayes	F1 Score	0	2119207	2119207	0
Decision Tree	F1 Score	44	2119163	2119207	2.08E-05
Random Forest	F1 Score	0	2119207	2119207	0
Ada Boost	F1 Score	30	2119177	2119207	1.42E-05
Bagging Classifier	F1 Score	0	2119207	2119207	0
Logistic Regression	F1 Score	2119207	0	2119207	1
Stochastic Gradient Descent	F1 Score	0	2119207	2119207	0

Table 29: Phase Two – CIDDS-001 Models – Predictions on CICIDS 2017 Raw Traffic

Phase Two – Discussion

Predictions

The results of Phase Two were the most surprising of this research. Despite the high metric scores (>95%) yielded by the large majority of optimized and trained models, most of them failed to obtain a score resembling that of the CICIDS 2017 labels. According to the CICIDS 2017 labeled CSV data, roughly 19.7% of the total CICIDS 2017 raw PCAP traffic should be considered malicious. Ideally, adequately trained models should reach a similar target score with a +/- 5% range. Only 13 of the 105 trained models reached a score close to the target value of 19.7%. This low number of successful models was even with the acceptable range expanded to +/- 10%. Since only 13 models achieved their target scores, that means that the other 92 models failed. Of the 13 successful models, 12 of those models were trained using the CICIDS 2017 dataset, which means they were operating off previously trained data. The 12 trained models produced off the CICIDS 2017 dataset yielded successful results when trained via the Naïve Bayes, Decision Tree, Random Forest, and Bagging Classifier algorithms regardless of the optimization method. It makes sense for the models trained using the CICIDS 2017 CSV data to yield better results than the models trained using other datasets sets since all the CICIDS 2017 data was previously seen and the models were fitted and optimized. The only successful model trained outside of the CICIDS 2017 dataset was the model trained using the KDD 99 dataset, precision optimization, and Bagging Classifier algorithm. This KDD 99, precision optimized, Bagging Classifier model managed to obtain a score of 20.1% flagged malicious traffic. It is impressive that this particular model achieved such highly accurate results; however, this particular model's results may have just been largely coincidental. There is certainly a possibility that if this unique model is applied to a new prediction dataset, it will fail. There is no indication that this model achieved accurate results due to any particular algorithm, dataset, or scoring method since all the other models (not trained using the CICIDS 2017 dataset) failed.

The results of this phase were quite surprising. The large majority of models probably failed due to overfitting or underfitting despite being optimized in the first phase. Overfitting or underfitting is indicated by several of the models' results that flag either 100% or 0% of the CICIDS 2017 traffic. Additionally, another major factor that likely affected this phase's outcome was the selected features for training. As mentioned earlier, specific datasets only

included particular labeled features for training models. Many unique dataset features were dropped when mapping them to the Zeek connection logs' available features. Lastly, another major probable factor that resulted in many of these models' failure was likely due to unseen data. While machine learning should perform better than typical IDS anomaly detection, this phase's results indicate that standard machine learning still needs to be highly optimized and tuned to operate on unseen traffic effectively. Deep learning is likely to perform better when determining if new or unseen traffic is malicious due to the nature of the algorithms.

CHAPTER 5

CONCLUSIONS

This section will review the significant findings, initially proposed research, and conclusions for this research study. There were numerous challenges and setbacks encountered throughout this research. As a result, certain aspects of the original proposal had to be altered. Additionally, some of the results of this study were very surprising yet also enlightening. It is hoped that this study's resulting data and analysis can validate or assist other industry professionals or academic researchers. Overall, the goals and intent of the initially proposed research were successfully met. Some key findings and take-aways could undoubtedly apply to real-world machine learning model development and IDS configuration. There are numerous ways to expand upon this combined field of machine learning and network security for future research and development.

Major Findings Review

Phase One

The results of Phase One yielded highly informative data. Interestingly enough, some particular algorithms consistently underperformed during this phase of research. The linear machine learning algorithms, such as the Logistic Regression and Stochastic Gradient Descent algorithms, produced low metrics during optimization and model production. It can be seen that this underperformance acted as a snowball effect. It could be argued that poor optimization leads to inadequate model production and performance, leading to inaccurate predictions. However, while these two algorithms performed inadequately, most other algorithms did well to optimize and produce highly positive evaluation metrics.

This research phase also revealed that the optimization of parameters did not significantly affect the metrics' overall pass or fail categorization of the trained models. The difference in scoring methods between datasets and algorithms never changed the overall categorization with the threshold of 95% or higher in their respective scoring metrics. The

optimization seemed to have the most considerable impact on the worst-performing algorithms, such as Logistic Regression and Stochastic Gradient Descent. The produced models with different optimization methods using these two underperforming algorithms yielded highly fluctuant results; however, they still did never meet the threshold of 95% or higher.

One last interesting observation from this phase is the one-off instance where the Ada Boost algorithm underperformed. When the models were trained using the NSL-KDD dataset, the Ada Boost algorithm yielded low scores regardless of the selected optimization method. This underperformance by the Ada Boost algorithm may have been due to the dataset's size or possibly the selection of particular features interacting with this particular algorithm.

Phase Two

Phase Two produced the most surprising results of this research. Despite the largely successful, optimized, and trained models, 92 of the 105 total models failed to predict malicious traffic in unseen data accurately. Only 13 of the 105 models successfully predicted traffic. However, 12 of the 13 models were previously trained using the CICIDS 2017 dataset, meaning that they were making predictions based on previously trained and observed data. The only model that successfully predicted malicious traffic within truly unseen data was the model trained under the KDD 99 dataset while using the precision optimization method and Bagging Classifier algorithm.

The failure of the 92 models may be due to several reasons. One primary reason may be due to overfitting or underfitting of the models to their respective datasets. Overfitting and underfitting are commonly encountered challenges in machine learning and occur when a model is excessively or insufficiently trained using a particular dataset, respectively. Another potential reason for these failed models may be the selection of mapped features between datasets and Zeek log fields. One of the fields or features included in each of the datasets was the IP address field. IP addresses were included features while training the models because some of the selected algorithms may have utilized the frequency of communication between IP addresses and used this to identify malicious traffic.

Regarding the algorithms themselves, some of these models may have failed simply due to the nature of the selected algorithm. These algorithms' effectiveness may dramatically

change depending on the dataset's size or the selection of particular features. As an example, the Stochastic Gradient Descent algorithm underperformed since the very beginning of this research. Another factor that likely played a role in the failure of the 92 models is that the trained models needed to drop unseen variables or terms to ingest new data and make a prediction. The dropping of new variables likely had a significant impact while making predictions, which is another known problem with standard machine learning.

Despite only 13 models accurately predicting new data, this research phase still met one of this research's primary goals. This phase acted as a valid proof of concept that an organization could set up an automated process to continually produce or refine models based on IDS logs to identify malicious traffic. This research's particular setup may not have included inline machine learning computations to deem individual packets as malicious or innocuous as they pass through the wire. However, this research proved that there could be a dedicated and offloaded machine learning IDS device operating off a span port that watches network traffic and flags for anomalies. The Python scripts created for this research could easily be modified to operate in a corporate environment and a more automated fashion. Additionally, the scripts could be further configured such that if malicious traffic is identified, an alert could be sent to a network administrator or security analyst to investigate the traffic further.

Research Questions and Hypotheses Review

Research Question 1

How does the selection of a single network intrusion dataset impact machine learning models' outcomes and performance when trained using multiple machine learning algorithms and optimization methods?

Hypothesis 1

Not all machine learning models will achieve high accuracy when trained using any network intrusion dataset.

Research Question and Hypothesis 1 Analysis

This first hypothesis can be accepted based on the performance of the trained models from Phase One. By grouping each set of models by individual dataset (seen in the tables below), the data indicate that none of the groupings of models could achieve high accuracy, specified as 95% or higher, across all algorithms and optimization methods. The KDD 99 dataset managed to reach the highest number of successful models, with 18 out of 21 models (85.71%) yielding the desired accuracy threshold. The NSL-KDD and CIDDS-001 datasets yielded 12 successful models (57.14%), and the UNSW-NB15 dataset yielded 15 successful models (71.43%). The CICIDS 2017 dataset yielded the lowest number of successful models, with 9 out of 21 models (42.86%) reaching the threshold. These results also show that the major limiting factor for success was the machine learning algorithms. The Stochastic Gradient Descent algorithm seemed always to fail to reach the desired accuracy threshold across each dataset, even when optimized for accuracy.

All Trained Models – Grouped by Dataset

Algorithm	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
Naïve Bayes	Accuracy	0.97303	0.96586	0.94853	0.95689	0.19895
Naïve Bayes	Precision	0.97303	0.96586	0.94853	0.95689	0.19037
Naïve Bayes	F1 Score	0.97303	0.96586	0.94853	0.95689	0.15929
Decision Tree	Accuracy	0.99816	0.996	0.99824	0.99712	0.36601
Decision Tree	Precision	0.99812	0.99598	0.99814	0.99706	0.32635
Decision Tree	F1 Score	0.99816	0.99602	0.99822	0.99712	0.20516
Random Forest	Accuracy	0.99819	0.99603	0.99831	0.99716	2.472
Random Forest	Precision	0.99818	0.99602	0.9983	0.99715	2.49991
Random Forest	F1 Score	0.99818	0.99602	0.9983	0.99715	1.78166
Ada Boost	Accuracy	0.99794	0.99554	0.99804	0.99678	949.45873
Ada Boost	Precision	0.99794	0.99554	0.99804	0.99678	966.79733
Ada Boost	F1 Score	0.99794	0.99554	0.99804	0.99678	943.96914
Bagging Classifier	Accuracy	0.99816	0.99592	0.99831	0.99711	33.93054
Bagging Classifier	Precision	0.99816	0.99598	0.99829	0.99713	70.83681
Bagging Classifier	F1 Score	0.99814	0.99589	0.9983	0.99709	1.05218
Logistic Regression	Accuracy	0.96855	0.96973	0.93048	0.94854	0.43249
Logistic Regression	Precision	0.96855	0.96973	0.93048	0.94854	0.35421
Logistic Regression	F1 Score	0.96855	0.96973	0.93048	0.94854	0.42434
Stochastic Gradient Descent	Accuracy	0.80108	0.64298	0.50011	0.44508	0.67686
Stochastic Gradient Descent	Precision	0.80108	0.80108	1	0.88956	0.19012
Stochastic Gradient Descent	F1 Score	0.80108	0.80108	1	0.88956	0.18991

Table 30: KDD 99 Models

Algorithm	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
Naïve Bayes	Accuracy	0.89173	0.89252	0.89246	0.89173	0.00726
Naïve Bayes	Precision	0.89173	0.89252	0.89246	0.89173	0.00478
Naïve Bayes	F1 Score	0.89173	0.89252	0.89246	0.89173	0.00491
Decision Tree	Accuracy	0.98613	0.98608	0.98616	0.98612	0.01268
Decision Tree	Precision	0.98741	0.98739	0.98741	0.9874	0.00858
Decision Tree	F1 Score	0.98647	0.98643	0.98649	0.98646	0.01002
Random Forest	Accuracy	0.98828	0.98831	0.98824	0.98827	3.8012
Random Forest	Precision	0.98815	0.98817	0.98811	0.98814	0.05314
Random Forest	F1 Score	0.98822	0.98825	0.98817	0.98821	0.81297
Ada Boost	Accuracy	0.98155	0.98154	0.98153	0.98154	15.7271
Ada Boost	Precision	0.98155	0.98154	0.98153	0.98154	15.61126
Ada Boost	F1 Score	0.98155	0.98154	0.98153	0.98154	15.61979
Bagging Classifier	Accuracy	0.98734	0.98742	0.98725	0.98733	0.58591
Bagging Classifier	Precision	0.98734	0.98742	0.98725	0.98733	0.87895
Bagging Classifier	F1 Score	0.98734	0.98742	0.98725	0.98733	0.63036
Logistic Regression	Accuracy	0.90648	0.90636	0.90653	0.90643	0.01096
Logistic Regression	Precision	0.90648	0.90636	0.90653	0.90643	0.00951
Logistic Regression	F1 Score	0.90648	0.90636	0.90653	0.90643	0.00933
Stochastic Gradient Descent	Accuracy	0.51387	0.51387	1	0.67888	0.00446
Stochastic Gradient Descent	Precision	0.51387	0.51387	1	0.67888	0.00392
Stochastic Gradient Descent	F1 Score	0.54612	0.71213	0.55799	0.45311	0.00573

Table 31: KDD-NSL Models

Algorithm	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
Naïve Bayes	Accuracy	0.89173	0.89252	0.89246	0.89173	0.00726
Naïve Bayes	Precision	0.89173	0.89252	0.89246	0.89173	0.00478
Naïve Bayes	F1 Score	0.89173	0.89252	0.89246	0.89173	0.00491
Decision Tree	Accuracy	0.98613	0.98608	0.98616	0.98612	0.01268
Decision Tree	Precision	0.98741	0.98739	0.98741	0.9874	0.00858
Decision Tree	F1 Score	0.98647	0.98643	0.98649	0.98646	0.01002
Random Forest	Accuracy	0.98828	0.98831	0.98824	0.98827	3.8012
Random Forest	Precision	0.98815	0.98817	0.98811	0.98814	0.05314
Random Forest	F1 Score	0.98822	0.98825	0.98817	0.98821	0.81297
Ada Boost	Accuracy	0.98155	0.98154	0.98153	0.98154	15.7271
Ada Boost	Precision	0.98155	0.98154	0.98153	0.98154	15.61126
Ada Boost	F1 Score	0.98155	0.98154	0.98153	0.98154	15.61979
Bagging Classifier	Accuracy	0.98734	0.98742	0.98725	0.98733	0.58591
Bagging Classifier	Precision	0.98734	0.98742	0.98725	0.98733	0.87895
Bagging Classifier	F1 Score	0.98734	0.98742	0.98725	0.98733	0.63036
Logistic Regression	Accuracy	0.90648	0.90636	0.90653	0.90643	0.01096
Logistic Regression	Precision	0.90648	0.90636	0.90653	0.90643	0.00951
Logistic Regression	F1 Score	0.90648	0.90636	0.90653	0.90643	0.00933
Stochastic Gradient Descent	Accuracy	0.51387	0.51387	1	0.67888	0.00446
Stochastic Gradient Descent	Precision	0.51387	0.51387	1	0.67888	0.00392
Stochastic Gradient Descent	F1 Score	0.54612	0.71213	0.55799	0.45311	0.00573

Table 32: NSL-KDD Models

Algorithm	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
Naïve Bayes	Accuracy	0.89293	0.78412	0.64256	0.68054	0.08435
Naïve Bayes	Precision	0.89293	0.78412	0.64256	0.68054	0.08946
Naïve Bayes	F1 Score	0.89293	0.78412	0.64256	0.68054	0.08273
Decision Tree	Accuracy	0.9917	0.9808	0.98154	0.98117	0.37921
Decision Tree	Precision	0.99187	0.98118	0.98195	0.98156	0.29828
Decision Tree	F1 Score	0.99126	0.97967	0.98066	0.98016	0.25876
Random Forest	Accuracy	0.99306	0.98372	0.98481	0.98426	1.49134
Random Forest	Precision	0.99303	0.98386	0.98449	0.98418	12.24554
Random Forest	F1 Score	0.99305	0.98386	0.98459	0.98422	1.38001
Ada Boost	Accuracy	0.98971	0.97442	0.97912	0.97675	466.55119
Ada Boost	Precision	0.98989	0.97491	0.97942	0.97715	778.32787
Ada Boost	F1 Score	0.98971	0.97442	0.97912	0.97675	480.33834
Bagging Classifier	Accuracy	0.99317	0.98403	0.98496	0.98449	19.65784
Bagging Classifier	Precision	0.97446	0.98575	0.89864	0.93638	1.08145
Bagging Classifier	F1 Score	0.99318	0.98397	0.98507	0.98452	54.13052
Logistic Regression	Accuracy	0.97419	0.94521	0.93655	0.94082	0.35886
Logistic Regression	Precision	0.97419	0.94521	0.93655	0.94082	0.35494
Logistic Regression	F1 Score	0.97419	0.94521	0.93655	0.94082	0.52767
Stochastic Gradient Descent	Accuracy	0.87405	0.87405	1	0.93279	0.08537
Stochastic Gradient Descent	Precision	0.87405	0.87405	1	0.93279	0.08048
Stochastic Gradient Descent	F1 Score	0.87405	0.87405	1	0.93279	0.09553

Table 33: UNSW-NB15 Models

Algorithm	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
Naïve Bayes	Accuracy	0.85553	0.81139	0.68245	0.71751	0.09344
Naïve Bayes	Precision	0.85553	0.81139	0.68245	0.71751	0.09795
Naïve Bayes	F1 Score	0.85553	0.81139	0.68245	0.71751	0.08759
Decision Tree	Accuracy	0.99318	0.98845	0.99007	0.98926	0.42589
Decision Tree	Precision	0.99309	0.98826	0.98997	0.98911	0.47588
Decision Tree	F1 Score	0.99318	0.98842	0.9901	0.98926	0.41032
Random Forest	Accuracy	0.99415	0.99032	0.99124	0.99078	20.96378
Random Forest	Precision	0.99415	0.99033	0.99122	0.99077	20.17999
Random Forest	F1 Score	0.99416	0.99029	0.99132	0.9908	18.56984
Ada Boost	Accuracy	0.92856	0.89923	0.86886	0.88291	723.04974
Ada Boost	Precision	0.92856	0.89923	0.86886	0.88291	734.92824
Ada Boost	F1 Score	0.92856	0.89923	0.86886	0.88291	735.96754
Bagging Classifier	Accuracy	0.9944	0.99115	0.99117	0.99116	176.10695
Bagging Classifier	Precision	0.9944	0.99113	0.99121	0.99117	171.72592
Bagging Classifier	F1 Score	0.9944	0.99116	0.99119	0.99117	107.24517
Logistic Regression	Accuracy	0.84409	0.77625	0.6702	0.70009	0.19176
Logistic Regression	Precision	0.84409	0.77625	0.6702	0.70009	0.16185
Logistic Regression	F1 Score	0.84409	0.77625	0.6702	0.70009	0.18086
Stochastic Gradient Descent	Accuracy	0.80256	0.80256	1	0.89047	0.12037
Stochastic Gradient Descent	Precision	0.80256	0.80256	1	0.89047	0.27912
Stochastic Gradient Descent	F1 Score	0.80256	0.80256	1	0.89047	0.08245

Table 34: CICIDS 2017 Models

Algorithm	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
Naïve Bayes	Accuracy	0.90131	0.74686	0.88045	0.79145	0.16439
Naïve Bayes	Precision	0.90131	0.74686	0.88045	0.79145	0.15782
Naïve Bayes	F1 Score	0.90131	0.74686	0.88045	0.79145	0.13131
Decision Tree	Accuracy	0.99951	0.99877	0.99855	0.99866	0.50578
Decision Tree	Precision	0.99948	0.99869	0.99847	0.99858	0.37718
Decision Tree	F1 Score	0.9995	0.99873	0.99853	0.99863	0.31852
Random Forest	Accuracy	0.99963	0.99908	0.99893	0.999	14.92514
Random Forest	Precision	0.99963	0.99908	0.99893	0.999	40.93319
Random Forest	F1 Score	0.99963	0.99908	0.99893	0.999	39.92271
Ada Boost	Accuracy	0.99852	0.99571	0.99627	0.99599	1203.30446
Ada Boost	Precision	0.99852	0.99571	0.99627	0.99599	1116.44332
Ada Boost	F1 Score	0.99852	0.99569	0.99627	0.99598	1051.30573
Bagging Classifier	Accuracy	0.99959	0.99904	0.99872	0.99888	22.60481
Bagging Classifier	Precision	0.99956	0.99909	0.99851	0.9988	15.54522
Bagging Classifier	F1 Score	0.99959	0.99903	0.99877	0.9989	22.53338
Logistic Regression	Accuracy	0.95024	0.91176	0.79929	0.84416	0.3716
Logistic Regression	Precision	0.95024	0.91176	0.79929	0.84416	0.35591
Logistic Regression	F1 Score	0.95024	0.91176	0.79929	0.84416	0.36495
Stochastic Gradient Descent	Accuracy	0.89727	0.89727	1	0.94585	0.14127
Stochastic Gradient Descent	Precision	0.89727	0.89727	1	0.94585	0.13498
Stochastic Gradient Descent	F1 Score	0.89727	0.89727	1	0.94585	0.13185

Table 35: CICIDS-001 Models

Research Question 2

How does the selection of a single machine learning algorithm impact machine learning models' outcomes and performance when trained using multiple network intrusion datasets and optimization methods?

Hypothesis 2

Not all machine learning models will achieve high accuracy when trained using any machine learning algorithm.

Research Question and Hypothesis 2 Analysis

The trained models' yielded metrics from Phase One indicate that this second hypothesis can also be accepted. When grouping each set of the models by individual machine learning algorithm (seen in the tables below), the results show that only some of the sets of models achieved the desired 95% accuracy scores across all datasets and optimization methods. Interestingly enough, the Decision Tree, Random Forest, and Bagging Classifier algorithms all yielded 15 out of 15 models that successfully reached the 95% accuracy threshold across all datasets and optimization methods. Another interesting result of these tests found that the Stochastic Gradient Descent algorithm failed to produce any successful models to reach the desired accuracy score; however, it should also be noted that this algorithm yielded the best recall metrics, with 13 of the 15 models reaching 100% recall scores. While the Stochastic Gradient Descent algorithm yielded lower accuracy scores, it did achieve perfect recall for some of its models, which corresponds to no false negative results. The Ada Boost algorithm yielded 12 successful models (80%), the Logistic Regression algorithm yielded nine successful models (60%), and the Naïve Bayes algorithm yielded three successful models (20%). These results indicate that the major limiting factor for these models was the selection of datasets. The optimization technique selected did not appear to impact the evaluation metrics of the models significantly.

All Trained Models – Grouped by Algorithm

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.97303	0.96586	0.94853	0.95689	0.19895
KDD 99	Precision	0.97303	0.96586	0.94853	0.95689	0.19037
KDD 99	F1 Score	0.97303	0.96586	0.94853	0.95689	0.15929
NSL-KDD	Accuracy	0.89173	0.89252	0.89246	0.89173	0.00726
NSL-KDD	Precision	0.89173	0.89252	0.89246	0.89173	0.00478
NSL-KDD	F1 Score	0.89173	0.89252	0.89246	0.89173	0.00491
UNSW-NB15	Accuracy	0.89293	0.78412	0.64256	0.68054	0.08435
UNSW-NB15	Precision	0.89293	0.78412	0.64256	0.68054	0.08946
UNSW-NB15	F1 Score	0.89293	0.78412	0.64256	0.68054	0.08273
CICIDS 2017	Accuracy	0.85553	0.81139	0.68245	0.71751	0.09344
CICIDS 2017	Precision	0.85553	0.81139	0.68245	0.71751	0.09795
CICIDS 2017	F1 Score	0.85553	0.81139	0.68245	0.71751	0.08759
CIDDS-001	Accuracy	0.90131	0.74686	0.88045	0.79145	0.16439
CIDDS-001	Precision	0.90131	0.74686	0.88045	0.79145	0.15782
CIDDS-001	F1 Score	0.90131	0.74686	0.88045	0.79145	0.13131

Table 36: Naïve Bayes Models

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.99816	0.996	0.99824	0.99712	0.36601
KDD 99	Precision	0.99812	0.99598	0.99814	0.99706	0.32635
KDD 99	F1 Score	0.99816	0.99602	0.99822	0.99712	0.20516
NSL-KDD	Accuracy	0.98613	0.98608	0.98616	0.98612	0.01268
NSL-KDD	Precision	0.98741	0.98739	0.98741	0.9874	0.00858
NSL-KDD	F1 Score	0.98647	0.98643	0.98649	0.98646	0.01002
UNSW-NB15	Accuracy	0.9917	0.9808	0.98154	0.98117	0.37921
UNSW-NB15	Precision	0.99187	0.98118	0.98195	0.98156	0.29828
UNSW-NB15	F1 Score	0.99126	0.97967	0.98066	0.98016	0.25876
CICIDS 2017	Accuracy	0.99318	0.98845	0.99007	0.98926	0.42589
CICIDS 2017	Precision	0.99309	0.98826	0.98997	0.98911	0.47588
CICIDS 2017	F1 Score	0.99318	0.98842	0.9901	0.98926	0.41032
CIDDS-001	Accuracy	0.99951	0.99877	0.99855	0.99866	0.50578
CIDDS-001	Precision	0.99948	0.99869	0.99847	0.99858	0.37718
CIDDS-001	F1 Score	0.9995	0.99873	0.99853	0.99863	0.31852

Table 37: Decision Tree Models

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.99819	0.99603	0.99831	0.99716	2.472
KDD 99	Precision	0.99818	0.99602	0.9983	0.99715	2.49991
KDD 99	F1 Score	0.99818	0.99602	0.9983	0.99715	1.78166
NSL-KDD	Accuracy	0.98828	0.98831	0.98824	0.98827	3.8012
NSL-KDD	Precision	0.98815	0.98817	0.98811	0.98814	0.05314
NSL-KDD	F1 Score	0.98822	0.98825	0.98817	0.98821	0.81297
UNSW-NB15	Accuracy	0.99306	0.98372	0.98481	0.98426	1.49134
UNSW-NB15	Precision	0.99303	0.98386	0.98449	0.98418	12.24554
UNSW-NB15	F1 Score	0.99305	0.98386	0.98459	0.98422	1.38001
CICIDS 2017	Accuracy	0.99415	0.99032	0.99124	0.99078	20.96378
CICIDS 2017	Precision	0.99415	0.99033	0.99122	0.99077	20.17999
CICIDS 2017	F1 Score	0.99416	0.99029	0.99132	0.9908	18.56984
CIDDS-001	Accuracy	0.99963	0.99908	0.99893	0.999	14.92514
CIDDS-001	Precision	0.99963	0.99908	0.99893	0.999	40.93319
CIDDS-001	F1 Score	0.99963	0.99908	0.99893	0.999	39.92271

Table 38: Random Forest Models

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.99794	0.99554	0.99804	0.99678	949.45873
KDD 99	Precision	0.99794	0.99554	0.99804	0.99678	966.79733
KDD 99	F1 Score	0.99794	0.99554	0.99804	0.99678	943.96914
NSL-KDD	Accuracy	0.98155	0.98154	0.98153	0.98154	15.7271
NSL-KDD	Precision	0.98155	0.98154	0.98153	0.98154	15.61126
NSL-KDD	F1 Score	0.98155	0.98154	0.98153	0.98154	15.61979
UNSW-NB15	Accuracy	0.98971	0.97442	0.97912	0.97675	466.55119
UNSW-NB15	Precision	0.98989	0.97491	0.97942	0.97715	778.32787
UNSW-NB15	F1 Score	0.98971	0.97442	0.97912	0.97675	480.33834
CICIDS 2017	Accuracy	0.92856	0.89923	0.86886	0.88291	723.04974
CICIDS 2017	Precision	0.92856	0.89923	0.86886	0.88291	734.92824
CICIDS 2017	F1 Score	0.92856	0.89923	0.86886	0.88291	735.96754
CIDDS-001	Accuracy	0.99852	0.99571	0.99627	0.99599	1203.3045
CIDDS-001	Precision	0.99852	0.99571	0.99627	0.99599	1116.4433
CIDDS-001	F1 Score	0.99852	0.99569	0.99627	0.99598	1051.3057

Table 39: Ada Boost Models

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.99816	0.99592	0.99831	0.99711	33.93054
KDD 99	Precision	0.99816	0.99598	0.99829	0.99713	70.83681
KDD 99	F1 Score	0.99814	0.99589	0.9983	0.99709	1.05218
NSL-KDD	Accuracy	0.98734	0.98742	0.98725	0.98733	0.58591
NSL-KDD	Precision	0.98734	0.98742	0.98725	0.98733	0.87895
NSL-KDD	F1 Score	0.98734	0.98742	0.98725	0.98733	0.63036
UNSW-NB15	Accuracy	0.99317	0.98403	0.98496	0.98449	19.65784
UNSW-NB15	Precision	0.97446	0.98575	0.89864	0.93638	1.08145
UNSW-NB15	F1 Score	0.99318	0.98397	0.98507	0.98452	54.13052
CICIDS 2017	Accuracy	0.9944	0.99115	0.99117	0.99116	176.10695
CICIDS 2017	Precision	0.9944	0.99113	0.99121	0.99117	171.72592
CICIDS 2017	F1 Score	0.9944	0.99116	0.99119	0.99117	107.24517
CIDDS-001	Accuracy	0.99959	0.99904	0.99872	0.99888	22.60481
CIDDS-001	Precision	0.99956	0.99909	0.99851	0.9988	15.54522
CIDDS-001	F1 Score	0.99959	0.99903	0.99877	0.9989	22.53338

Table 40: Bagging Classifier Models

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.96855	0.96973	0.93048	0.94854	0.43249
KDD 99	Precision	0.96855	0.96973	0.93048	0.94854	0.35421
KDD 99	F1 Score	0.96855	0.96973	0.93048	0.94854	0.42434
NSL-KDD	Accuracy	0.90648	0.90636	0.90653	0.90643	0.01096
NSL-KDD	Precision	0.90648	0.90636	0.90653	0.90643	0.00951
NSL-KDD	F1 Score	0.90648	0.90636	0.90653	0.90643	0.00933
UNSW-NB15	Accuracy	0.97419	0.94521	0.93655	0.94082	0.35886
UNSW-NB15	Precision	0.97419	0.94521	0.93655	0.94082	0.35494
UNSW-NB15	F1 Score	0.97419	0.94521	0.93655	0.94082	0.52767
CICIDS 2017	Accuracy	0.84409	0.77625	0.6702	0.70009	0.19176
CICIDS 2017	Precision	0.84409	0.77625	0.6702	0.70009	0.16185
CICIDS 2017	F1 Score	0.84409	0.77625	0.6702	0.70009	0.18086
CIDDS-001	Accuracy	0.95024	0.91176	0.79929	0.84416	0.3716
CIDDS-001	Precision	0.95024	0.91176	0.79929	0.84416	0.35591
CIDDS-001	F1 Score	0.95024	0.91176	0.79929	0.84416	0.36495

Table 41: Logistic Regression Models

Dataset	Scoring Method	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Accuracy	0.80108	0.64298	0.50011	0.44508	0.67686
KDD 99	Precision	0.80108	0.80108	1	0.88956	0.19012
KDD 99	F1 Score	0.80108	0.80108	1	0.88956	0.18991
NSL-KDD	Accuracy	0.51387	0.51387	1	0.67888	0.00446
NSL-KDD	Precision	0.51387	0.51387	1	0.67888	0.00392
NSL-KDD	F1 Score	0.54612	0.71213	0.55799	0.45311	0.00573
UNSW-NB15	Accuracy	0.87405	0.87405	1	0.93279	0.08537
UNSW-NB15	Precision	0.87405	0.87405	1	0.93279	0.08048
UNSW-NB15	F1 Score	0.87405	0.87405	1	0.93279	0.09553
CICIDS 2017	Accuracy	0.80256	0.80256	1	0.89047	0.12037
CICIDS 2017	Precision	0.80256	0.80256	1	0.89047	0.27912
CICIDS 2017	F1 Score	0.80256	0.80256	1	0.89047	0.08245
CIDDS-001	Accuracy	0.89727	0.89727	1	0.94585	0.14127
CIDDS-001	Precision	0.89727	0.89727	1	0.94585	0.13498
CIDDS-001	F1 Score	0.89727	0.89727	1	0.94585	0.13185

Table 42: Stochastic Gradient Descent Models

Research Question 3

How does the selection of a single optimization method impact machine learning models' outcomes and performance when trained using multiple network intrusion datasets and machine learning algorithms?

Hypothesis 3

Not all machine learning models will achieve high respective performance metrics when trained using any optimization method.

Research Question and Hypothesis 3 Analysis

After reviewing the data from Phase One, this hypothesis can be accepted. It should be noted that this is the only hypothesis in this study that compares multiple evaluation metrics, namely accuracy, precision, and F1 score, based on the respective optimization technique. The threshold for success for this hypothesis is a respective metric score of 95% or higher. When evaluating this hypothesis, the models can be grouped by the selected optimization method (seen in the tables below). The models optimized for accuracy yielded 23 out of 35 successful models (65.71%) that achieved a 95% or higher accuracy score. The models optimized for precision yielded 21 successful models (60%) that achieved a 95% or higher precision score. The models optimized for the F1 score yielded 20 successful models (57.14%) that achieved a 95% or higher F1 score. The only consistent parameter for failure across these models appears to be the models trained using the Stochastic Gradient Descent algorithm, which never achieved the 95% threshold across any of the desired evaluation metrics. Lastly, these results indicate that it may have been beneficial to exaggerate the optimization methods to yield more variant models. This optimization variance could be achieved by the following: add in additional parameters, increase the number of values tested for each parameter, or further exaggerate the parameter values used for optimization. However, it should be noted that increasing the number of tested parameter values during this optimization process would exponentially increase the amount of time for optimization

All Trained Models – Grouped by Optimization Method

Dataset	Algorithm	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Naïve Bayes	0.97303	0.96586	0.94853	0.95689	0.19895
KDD 99	Decision Tree	0.99816	0.996	0.99824	0.99712	0.36601
KDD 99	Random Forest	0.99819	0.99603	0.99831	0.99716	2.472
KDD 99	Ada Boost	0.99794	0.99554	0.99804	0.99678	949.45873
KDD 99	Bagging Classifier	0.99816	0.99592	0.99831	0.99711	33.93054
KDD 99	Logistic Regression	0.96855	0.96973	0.93048	0.94854	0.43249
KDD 99	Stochastic Gradient Descent	0.80108	0.64298	0.50011	0.44508	0.67686
NSL-KDD	Naïve Bayes	0.89173	0.89252	0.89246	0.89173	0.00726
NSL-KDD	Decision Tree	0.98613	0.98608	0.98616	0.98612	0.01268
NSL-KDD	Random Forest	0.98828	0.98831	0.98824	0.98827	3.8012
NSL-KDD	Ada Boost	0.98155	0.98154	0.98153	0.98154	15.7271
NSL-KDD	Bagging Classifier	0.98734	0.98742	0.98725	0.98733	0.58591
NSL-KDD	Logistic Regression	0.90648	0.90636	0.90653	0.90643	0.01096
NSL-KDD	Stochastic Gradient Descent	0.51387	0.51387	1	0.67888	0.00446
UNSW-NB15	Naïve Bayes	0.89293	0.78412	0.64256	0.68054	0.08435
UNSW-NB15	Decision Tree	0.9917	0.9808	0.98154	0.98117	0.37921
UNSW-NB15	Random Forest	0.99306	0.98372	0.98481	0.98426	1.49134
UNSW-NB15	Ada Boost	0.98971	0.97442	0.97912	0.97675	466.55119
UNSW-NB15	Bagging Classifier	0.99317	0.98403	0.98496	0.98449	19.65784
UNSW-NB15	Logistic Regression	0.97419	0.94521	0.93655	0.94082	0.35886
UNSW-NB15	Stochastic Gradient Descent	0.87405	0.87405	1	0.93279	0.08537
CICIDS 2017	Naïve Bayes	0.85553	0.81139	0.68245	0.71751	0.09344
CICIDS 2017	Decision Tree	0.99318	0.98845	0.99007	0.98926	0.42589
CICIDS 2017	Random Forest	0.99415	0.99032	0.99124	0.99078	20.96378
CICIDS 2017	Ada Boost	0.92856	0.89923	0.86886	0.88291	723.04974
CICIDS 2017	Bagging Classifier	0.9944	0.99115	0.99117	0.99116	176.10695
CICIDS 2017	Logistic Regression	0.84409	0.77625	0.6702	0.70009	0.19176
CICIDS 2017	Stochastic Gradient Descent	0.80256	0.80256	1	0.89047	0.12037
CIDDS-001	Naïve Bayes	0.90131	0.74686	0.88045	0.79145	0.16439
CIDDS-001	Decision Tree	0.99951	0.99877	0.99855	0.99866	0.50578
CIDDS-001	Random Forest	0.99963	0.99908	0.99893	0.999	14.92514
CIDDS-001	Ada Boost	0.99852	0.99571	0.99627	0.99599	1203.30446
CIDDS-001	Bagging Classifier	0.99959	0.99904	0.99872	0.99888	22.60481
CIDDS-001	Logistic Regression	0.95024	0.91176	0.79929	0.84416	0.3716
CIDDS-001	Stochastic Gradient Descent	0.89727	0.89727	1	0.94585	0.14127

Table 43: Accuracy Models

Dataset	Algorithm	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Naïve Bayes	0.97303	0.96586	0.94853	0.95689	0.19037
KDD 99	Decision Tree	0.99812	0.99598	0.99814	0.99706	0.32635
KDD 99	Random Forest	0.99818	0.99602	0.9983	0.99715	2.49991
KDD 99	Ada Boost	0.99794	0.99554	0.99804	0.99678	966.79733
KDD 99	Bagging Classifier	0.99816	0.99598	0.99829	0.99713	70.83681
KDD 99	Logistic Regression	0.96855	0.96973	0.93048	0.94854	0.35421
KDD 99	Stochastic Gradient Descent	0.80108	0.80108	1	0.88956	0.19012
NSL-KDD	Naïve Bayes	0.89173	0.89252	0.89246	0.89173	0.00478
NSL-KDD	Decision Tree	0.98741	0.98739	0.98741	0.9874	0.00858
NSL-KDD	Random Forest	0.98815	0.98817	0.98811	0.98814	0.05314
NSL-KDD	Ada Boost	0.98155	0.98154	0.98153	0.98154	15.61126
NSL-KDD	Bagging Classifier	0.98734	0.98742	0.98725	0.98733	0.87895
NSL-KDD	Logistic Regression	0.90648	0.90636	0.90653	0.90643	0.00951
NSL-KDD	Stochastic Gradient Descent	0.51387	0.51387	1	0.67888	0.00392
UNSW-NB15	Naïve Bayes	0.89293	0.78412	0.64256	0.68054	0.08946
UNSW-NB15	Decision Tree	0.99187	0.98118	0.98195	0.98156	0.29828
UNSW-NB15	Random Forest	0.99303	0.98386	0.98449	0.98418	12.24554
UNSW-NB15	Ada Boost	0.98989	0.97491	0.97942	0.97715	778.32787
UNSW-NB15	Bagging Classifier	0.97446	0.98575	0.89864	0.93638	1.08145
UNSW-NB15	Logistic Regression	0.97419	0.94521	0.93655	0.94082	0.35494
UNSW-NB15	Stochastic Gradient Descent	0.87405	0.87405	1	0.93279	0.08048
CICIDS 2017	Naïve Bayes	0.85553	0.81139	0.68245	0.71751	0.09795
CICIDS 2017	Decision Tree	0.99309	0.98826	0.98997	0.98911	0.47588
CICIDS 2017	Random Forest	0.99415	0.99033	0.99122	0.99077	20.17999
CICIDS 2017	Ada Boost	0.92856	0.89923	0.86886	0.88291	734.92824
CICIDS 2017	Bagging Classifier	0.9944	0.99113	0.99121	0.99117	171.72592
CICIDS 2017	Logistic Regression	0.84409	0.77625	0.6702	0.70009	0.16185
CICIDS 2017	Stochastic Gradient Descent	0.80256	0.80256	1	0.89047	0.27912
CIDDS-001	Naïve Bayes	0.90131	0.74686	0.88045	0.79145	0.15782
CIDDS-001	Decision Tree	0.99948	0.99869	0.99847	0.99858	0.37718
CIDDS-001	Random Forest	0.99963	0.99908	0.99893	0.999	40.93319
CIDDS-001	Ada Boost	0.99852	0.99571	0.99627	0.99599	1116.44332
CIDDS-001	Bagging Classifier	0.99956	0.99909	0.99851	0.9988	15.54522
CIDDS-001	Logistic Regression	0.95024	0.91176	0.79929	0.84416	0.35591
CIDDS-001	Stochastic Gradient Descent	0.89727	0.89727	1	0.94585	0.13498

Table 44: Precision Models

Dataset	Algorithm	Accuracy	Precision	Recall	F1 Score	Train Time
KDD 99	Naïve Bayes	0.97303	0.96586	0.94853	0.95689	0.15929
KDD 99	Decision Tree	0.99816	0.99602	0.99822	0.99712	0.20516
KDD 99	Random Forest	0.99818	0.99602	0.9983	0.99715	1.78166
KDD 99	Ada Boost	0.99794	0.99554	0.99804	0.99678	943.96914
KDD 99	Bagging Classifier	0.99814	0.99589	0.9983	0.99709	1.05218
KDD 99	Logistic Regression	0.96855	0.96973	0.93048	0.94854	0.42434
KDD 99	Stochastic Gradient Descent	0.80108	0.80108	1	0.88956	0.18991
NSL-KDD	Naïve Bayes	0.89173	0.89252	0.89246	0.89173	0.00491
NSL-KDD	Decision Tree	0.98647	0.98643	0.98649	0.98646	0.01002
NSL-KDD	Random Forest	0.98822	0.98825	0.98817	0.98821	0.81297
NSL-KDD	Ada Boost	0.98155	0.98154	0.98153	0.98154	15.61979
NSL-KDD	Bagging Classifier	0.98734	0.98742	0.98725	0.98733	0.63036
NSL-KDD	Logistic Regression	0.90648	0.90636	0.90653	0.90643	0.00933
NSL-KDD	Stochastic Gradient Descent	0.54612	0.71213	0.55799	0.45311	0.00573
UNSW-NB15	Naïve Bayes	0.89293	0.78412	0.64256	0.68054	0.08273
UNSW-NB15	Decision Tree	0.99126	0.97967	0.98066	0.98016	0.25876
UNSW-NB15	Random Forest	0.99305	0.98386	0.98459	0.98422	1.38001
UNSW-NB15	Ada Boost	0.98971	0.97442	0.97912	0.97675	480.33834
UNSW-NB15	Bagging Classifier	0.99318	0.98397	0.98507	0.98452	54.13052
UNSW-NB15	Logistic Regression	0.97419	0.94521	0.93655	0.94082	0.52767
UNSW-NB15	Stochastic Gradient Descent	0.87405	0.87405	1	0.93279	0.09553
CICIDS 2017	Naïve Bayes	0.85553	0.81139	0.68245	0.71751	0.08759
CICIDS 2017	Decision Tree	0.99318	0.98842	0.9901	0.98926	0.41032
CICIDS 2017	Random Forest	0.99416	0.99029	0.99132	0.9908	18.56984
CICIDS 2017	Ada Boost	0.92856	0.89923	0.86886	0.88291	735.96754
CICIDS 2017	Bagging Classifier	0.9944	0.99116	0.99119	0.99117	107.24517
CICIDS 2017	Logistic Regression	0.84409	0.77625	0.6702	0.70009	0.18086
CICIDS 2017	Stochastic Gradient Descent	0.80256	0.80256	1	0.89047	0.08245
CIDDS-001	Naïve Bayes	0.90131	0.74686	0.88045	0.79145	0.13131
CIDDS-001	Decision Tree	0.9995	0.99873	0.99853	0.99863	0.31852
CIDDS-001	Random Forest	0.99963	0.99908	0.99893	0.999	39.92271
CIDDS-001	Ada Boost	0.99852	0.99569	0.99627	0.99598	1051.30573
CIDDS-001	Bagging Classifier	0.99959	0.99903	0.99877	0.9989	22.53338
CIDDS-001	Logistic Regression	0.95024	0.91176	0.79929	0.84416	0.36495
CIDDS-001	Stochastic Gradient Descent	0.89727	0.89727	1	0.94585	0.13185

Table 45: F1 Score Models

Research Question 4

How does the performance of theoretical machine learning models change when tested in an applied environment?

Hypothesis 4

The accuracy of theoretical machine learning models will perform significantly worse in an applied environment.

Research Question and Hypothesis 4 Analysis

This fourth hypothesis can also be accepted as accurate. Of the 105 total trained models, only 13 of those models were within the acceptable +/- 10% threshold of the target true value for detecting malicious traffic. These 13 successful models and their results can be seen in the table below. Additionally, 12 of the 13 successful models were trained using the same CICIDS 2017 dataset, meaning that those models evaluated previously observed data and did not even filter through truly unseen or new network traffic. Additionally, of the total 21 models trained using the CICIDS 2017 dataset, 9 of those 21 models (42.86%) failed to accurately detect malicious traffic despite being previously trained using the exact dataset, just in a different form. In the end, only a single model was able to flag malicious traffic on completely unseen data accurately. These experiments' results show that 92 of the 105 total models performed worse in an applied environment than a conceptual environment, meaning that 87.62% of the total models failed to identify malicious traffic correctly. Additionally, removing the set of models trained using the CICIDS 2017 dataset shows 83 of the 84 models (98.81%) failed to detect traffic on truly unseen network traffic accurately. The only successful applied model utilized the KDD 99 dataset, Bagging Classifier algorithm, and precision optimization. The other failed models indicate that the success of this individual model may be largely coincidental. Excluding the non-CICIDS 2017 models, the results show that none of the other models trained using the KDD 99 dataset, the Bagging Classifier algorithm, or the precision optimization method yielded successfully applied models. Reviewing all these failed models indicates a strong need to utilize a variety of algorithms, datasets, and optimization methods during model development to yield an ideal model and effectively solve the particular problem at hand.

Successfully Applied Models

Dataset	Model	Score Method	Malicious	Normal	Total	Flagged Traffic
KDD 99	Bagging Classifier	Precision	426829	1692378	2119207	0.20141
CICIDS 2017	Naïve Bayes	Accuracy	531212	1587995	2119207	0.25067
CICIDS 2017	Decision Tree	Accuracy	236062	1883145	2119207	0.11139
CICIDS 2017	Random Forest	Accuracy	360301	1758906	2119207	0.17002
CICIDS 2017	Bagging Classifier	Accuracy	369295	1749912	2119207	0.17426
CICIDS 2017	Naïve Bayes	Precision	531212	1587995	2119207	0.25067
CICIDS 2017	Decision Tree	Precision	415476	1703731	2119207	0.19605
CICIDS 2017	Random Forest	Precision	361007	1758200	2119207	0.17035
CICIDS 2017	Bagging Classifier	Precision	368940	1750267	2119207	0.17409
CICIDS 2017	Naïve Bayes	F1 Score	531212	1587995	2119207	0.25067
CICIDS 2017	Decision Tree	F1 Score	400550	1718657	2119207	0.18901
CICIDS 2017	Random Forest	F1 Score	360171	1759036	2119207	0.16996
CICIDS 2017	Bagging Classifier	F1 Score	368985	1750222	2119207	0.17411

Table 46: Successful Models in Applied IDS Environment

Proposed Research Review

Successful Proposal Research Tasks

Overall, the primary goals of Phase One and Phase Two of this research were successfully met. In Phase One, this research successfully optimized and trained machine learning models trained on various datasets, algorithms, and scoring methods. Additionally, Phase Two provided a valuable proof of concept that displayed the possibility of creating an automated pipeline to apply machine learning models to review IDS logs and network traffic. The second phase displayed successful imports of trained machine learning models and the utilization of them to predict malicious traffic in unseen data, although not as accurately as desired.

This research study successfully answered each of the research questions and conclusively accept each of the hypotheses. These results showcase the need for utilizing a comprehensive combination of algorithms, datasets, and optimization techniques when training machine learning models. They also display which algorithms, datasets, and optimization techniques yield models with the highest evaluation metrics in accuracy,

precision, recall, and F1 score. Lastly, this study's results strongly indicate that theoretical machine learning models may perform substantially worse when applied in a real-world environment.

Failed or Altered Proposal Research Tasks

There were a few deviations from the original proposal of this research. Initially, this research attempted to alter the preprocessors of an IDS detection engine. This preprocessor modification or creation did not occur within this research because, after investigating the possibility, it appears that each preprocessor is protocol-specific and reviews particular packet fields. IDS preprocessors are not intended for intense computation, let alone machine learning. Also, depending on the programming language used to build the IDS, it may have been necessary to manually code the machine learning algorithms since there may not have been a publicly available library, like scikit-learn. Due to these reasons, going down the preprocessor creation route would have been too time and effort-intensive to complete within the proposed timeline. After investigating the feasibility of IDS preprocessor manipulation, it appeared there were two other potential avenues to complete the goals of this research. The second option involved creating an IDS plugin using the programming language of the particular IDS. Due to similar concerns regarding preprocessor creation and the potential need to code machine learning algorithms, this was not the desired route due to this proposed research's limited time. Finally, the third and final option involved a series of tools that could, in theory, be easily set up as a pipeline. This option involved feeding raw network traffic into an IDS, taking the logs and feeding them into Python scripts to produce machine learning models, and then re-ingesting those trained models to predict unseen traffic. This final option seemed to be the most practical option for this research since it still met the objectives while still utilizing practical methods and tools.

The selection of machine learning algorithms also dramatically throughout the research. As previously discussed in Chapter 3, numerous algorithms were tested and dropped from this research. The algorithms tested and dropped include Support Vector Machine, K Nearest Neighbor, Isolation Forest, Novelty Detection, and Outlier Detection. The primary reason these algorithms were dropped is due to their high computation and resource

requirements. When testing different algorithms, if a single model took longer than three days to produce or errored out, it was dropped.

Another alteration to the proposed research was the inclusion of the IRG'16 dataset. This dataset is massive as it is intended to replicate ISP network traffic. This IRG'16 dataset incorporated numerous 10+ GB PCAP files. The merging, ingestion, and sampling of this dataset would have significantly impacted the resource requirements and timeline of this research.

The proposed evaluation metrics also changed throughout this research. A few metrics were dropped for this research, namely False Alarm Rate, Specificity, and Negative Predictive Value. While it would have been possible to calculate these manually, the scikit-learn module did not include an easy method or function to obtain these metrics. Additionally, while reviewing the literature, the most commonly used model evaluation metrics included the ones used for this research - accuracy, precision, recall, and F1 score.

One last alteration to the proposed research tasks was comparing machine learning models to the standard IDS anomaly detection engine. The original plan was to set up the Zeek IDS in anomaly detection mode, train the system using known "good" traffic, and test how effectively the trained IDS handles unseen network traffic. This IDS anomaly detection will act as a baseline to identify if machine learning models perform better than the standard IDS anomaly detection engine. However, this IDS anomaly detection test did not occur in the actual research. The CICIDS 2017 dataset was selected as the prediction and validation dataset for comparing the trained models in an applied environment and observing their detection of malicious traffic. The CICIDS 2017 dataset is publicly available online and includes both labeled CSV and raw PCAP files. The raw PCAPs could easily be fed into the Zeek IDS for anomaly detection training. However, the CICIDS 2017 dataset does not perform proper grouping of known "good" or known "bad" network traffic in the PCAP files. Therefore, unfortunately, the CICIDS 2017 PCAPs could not be split to properly train and test the Zeek IDS.

Conclusions

Some significant conclusions were identified as a result of this research. First, this study indicates a strong need to train ML models using various algorithms and datasets. A researcher cannot simply select any algorithm or dataset and expect them always to generate high-performing models. Second, optimizing models at least once is still highly recommended. However, there may not be a need to test and compare multiple optimization methods. Throughout the literature review, many of the researchers typically only optimized for accuracy; the results of this research show that this may be a good enough practice since the changing of optimization methods did not always yield significantly different models. Lastly, this research showcases that theoretical ML models will most likely perform worse when applied in an implemented environment.

Suggested Alterations

There were certain significant limitations identified after completing and reviewing this research. Particular design decisions should be considered if this research were to be modified and redone. The utilization of purely machine learning algorithms is a downside of this research. These algorithms only operate on previously trained values and must ignore/drop new values. Machine learning algorithms utilize previously seen values to make predictions on newly ingested data, which could easily lead to inaccuracies. Deep learning seems to alleviate this concern and is currently a significant field of study that may be applied to this type of research.

Additionally, the source and destination IP addresses used for training may have negatively impacted the models. As explained earlier, these fields or features were included because specific machine learning algorithms will analyze and incorporate the frequency of communication between devices to make a final categorization decision. Also related to the features, this research only utilized a handful of features available between datasets and matched them as best as possible to the Zeek connection logs. This feature mapping led to inconsistent feature selection between datasets. Specific datasets had six features selected, whereas others had ten features mapped to Zeek fields.

Also, it should be noted that the computational resources for this research could have been better. While this research was conducted using DSU's IA Lab online virtual

environment. Five machines with 100 GB of RAM and 24 processing cores were set up and utilized for this research. However, no graphics cards for machine learning computation were included in this environment. The optimization of machine learning algorithms is a major field of research and, after researching some of these published papers, the application of graphics cards for model development could dramatically increase the speed of production. This lack of graphics cards and low computation speed may have affected the selection of machine learning models for this research since there was a three-day threshold before dropping an algorithm for this research.

It was stated earlier while reviewing this study's research questions and hypotheses, but it would have been quite beneficial for this study to increase variance between trained models by further expanding the optimization process. Adding in additional parameters, increasing the number of values tested for each parameter, and further exaggerating the parameter values used for optimization would have led to better optimized and variant models that utilize different parameter values. The different optimization techniques did appear to optimize 76 of the 105 models before model production successfully. However, many of the optimized models ended up using the same parameters, thereby reducing variance between models and their metric scores. The optimization process for this study took multiple days to complete. Adding in additional parameters and parameter values would have exponentially increased the optimization process due to the nature of the GridSearchCV function. However, this would have proved beneficial for this study if it yielded highly variant models that utilized uniquely optimized parameters.

Lastly, after further researching network patterns and trends, it has been found that the large majority of modern network attacks are identified in the payload of a packet and thereby require deep packet inspection. The exclusion of deep packet inspection (DPI) is a considerable limitation of this research since the models exclusively focused on the packets' metadata fields' size and contents. Network attacks conducted within the payload are application-layer attacks and highly unique to the network's particular hosts. However, there are reasons to include or exclude application-layer data in datasets for model development. In academia, many of these network intrusion datasets are developed to be generalized enough for other researchers to utilize. Application-layer data is highly unique network traffic that other researchers may not be interested in using, so dataset developers may not wish to

include this type of specialized traffic in their published dataset intended for academia. However, the inclusion of DPI and application-layer data should absolutely be included for model development within an applied enterprise network environment to train the model to scan through this application data properly. Therefore, it makes sense for application-layer data to be included when training ML models for applied network environments, but less so for theoretical, proof-of-concept research in academia.

Recommendations for Enterprise Networks

The results of this research identify some essential suggestions for corporate networks with enterprise environments that implement an IDS with machine learning capabilities. This study shows that a model trained off a different dataset and then applied to new network traffic will likely fail in terms of accuracy. Each network, especially enterprise networks, is incredibly unique and has different expected device communication times, observed protocols, open ports, and network segmentation. Despite published datasets attempting to be generalized for others to utilize for research, these datasets are still too specialized in terms of their available features, proportion and content of normal traffic, proportion and content of malicious traffic, the total size of a dataset, collection data type (packet-based, flow-based, hybrid), and their collection point on the network. Due to the extreme uniqueness of both enterprise networks and published datasets, it is recommended for corporate environments to utilize their own “known good” network traffic when training machine learning models. Analysts could then extract the simulated attacks from public datasets and test the trained models to identify specific simulated attacks. Additionally, even with these trained models, it is highly suggested that analysts should continue to monitor and refine the model and manually investigate flagged attacks. Over time, these models will become highly tuned to the uniqueness of the corporate environment and should be able to identify malicious traffic effectively. Additionally, with the customization of models, it should be possible to train the models to implement deep packet inspection features that will scan for attacks within the packet payload.

Future Work

The results of this research yielded some exciting and unsuspecting results. Future research in this combined field of network security and artificial intelligence may consider the following suggestions to expand this research. This research would have benefitted from live network polling or sampling to produce continuous Zeek logs. With that in mind, an actual pipeline configuration of scripts or other dedicated software that automatically transfers data between the IDS and Python scripts would benefit this research. Additionally, the Python scripts in this research could have been combined into a single script workflow that could operate continuously through scheduled tasking and network polling.

Additionally, the inclusion of deep learning techniques will be advantageous for expanding this type of research. Deep learning models are harder to train due to their complexity and time requirements; however, they are considered much more accurate and better prepared to predict attacks on unseen data. There are already numerous security researchers in industry and academia looking into the application of deep learning models.

Lastly, the application of machine learning could also be incorporated into other security intelligence applications and services. Rather than merely analyzing and correlating potential threats at the network level via an IDS device, several other security products could utilize machine learning to identify threats. For example, Security Information and Event Management (SIEM) devices can act as a centralized logging system that can parse through logs sent from numerous logging systems and individual host events across an entire network. A SIEM with machine learning or deep learning capabilities may effectively identify new threats by correlating every log and digital system's events across an entire organization. Additionally, machine learning could be applied to endpoint detection to categorize specific processes, dynamic-link libraries (DDLs), connections, ports, or events as normal or anomalous behaviors of an individual system.

REFERENCES

REFERENCES

Aburomman, A. A., & Reaz, M. B. I. (2017). A survey of intrusion detection systems based on ensemble and hybrid classifiers. *Computers & Security, 65*, 135-152. doi:10.1016/j.cose.2016.11.004

Ahmad, I., Basheri, M., Iqbal, M. J., & Rahim, A. (2018). Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection. *IEEE Access, 6*, 33789-33795. doi:10.1109/ACCESS.2018.2841987

Aijaz, L., Aslam, B., & Khalid, U. (2015). Security operations center — A need for an academic environment. In (pp. 1-7): IEEE.

Akashdeep, I., Manzoor, N., & Kumar, N. (2017). A feature reduced intrusion detection system using ANN classifier. *Expert Systems with Applications, 88*, 249-257. doi:10.1016/j.eswa.2017.07.005

Al-Jarrah, O. Y., Siddiqui, A., Elsalamouny, M., Yoo, P. D., Muhaidat, S., & Kim, K. (2014). Machine-Learning-Based Feature Selection Techniques for Large-Scale Network Intrusion Detection. In (Vol. 30-, pp. 177-181).

Albon, C. (2018). *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning*: O'Reilly Media, Inc.

Ameisen, E. (2020). *Building Machine Learning Powered Applications: Going from Idea to Product*: O'Reilly Media.

Anaconda. (2021). Retrieved from https://www.anaconda.com/

Anson, S. (2020). *Applied Incident Response*: Wiley.

assafmo. (2020). joincap. Retrieved from https://github.com/assafmo/joincap

Bejtlich, R. (2013). *The Practice of Network Security Monitoring: Understanding Incident Detection and Response*: No Starch Press.

Bhuyan, M. H., Bhattacharyya, D. K., & Kalita, J. K. (2014). Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys & Tutorials, 16*(1), 303-336. doi:10.1109/SURV.2013.052213.00046

Bonaccorso, G. (2020). *Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work, 2nd Edition*: Packt Publishing.

Buczak, A. L., & Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials, 18*(2), 1153-1176. doi:10.1109/COMST.2015.2494502

Chiba, Z. (2019). Intelligent approach to build a Deep Neural Network based IDS for cloud environment using combination of machine learning algorithms. *Computers & Security, 86*, 291-318. doi:10.1016/j.cose.2019.06.013

Chio, C., & Freeman, D. (2018). *Machine Learning and Security: Protecting Systems with Data and Algorithms*: O'Reilly Media.

Chollet, F. (2017). *Deep Learning with Python*: Manning Publications Co.

Choudhury, S., & Bhowal, A. (2015). Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. In (pp. 89-95).

Collins, M. (2014). *Network Security Through Data Analysis*: Shroff Publishers & Distr.

Creech, G., & Jiankun, H. (2013). Generation of a new IDS test dataset: Time to retire the KDD collection. In (pp. 4487-4492): IEEE.

Dangi, B., Gamet, J., Kulm, A., Nelson, T., O'Brien, A., & Pauli, W. E. (2020). Alert Prioritization and Strengthening: Towards an Industry Standard Priority Scoring System for IDS Analysts Using Open Source Tools and Models of Machine Learning. *South Dakota law review, 65*(3), 556.

Deyang, Z., & Dedong, Z. (2011). The Analysis of Event Correlation in Security Operations Center. In (Vol. 2, pp. 1214-1216): IEEE.

Ertam, F., Kilinçer, L. F., & Yaman, O. (2017). Intrusion detection in computer networks via machine learning algorithms. In (pp. 1-4).

Garreta, R., Moncecchi, G., Hauck, T., & Hackeling, G. (2017). *scikit-learn : Machine Learning Simplified: Implement scikit-learn into every step of the data science pipeline*: Packt Publishing.

Gron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*: O'Reilly Media, Inc.

Haider, W. (2017). Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. *Journal of Network & Computer Applications, 87*, 185-193. doi:10.1016/j.jnca.2017.03.018

Halder, S., & Ozdemir, S. (2018). *Hands-On Machine Learning for Cybersecurity: Safeguard your system by making your machines intelligent using the Python ecosystem*: Packt Publishing.

Johansen, G. (2020). *Digital Forensics and Incident Response: Incident response techniques and procedures to respond to modern cyber threats, 2nd Edition*: Packt Publishing.

Kumar, R. (2019). *Machine Learning Quick Reference: Quick and essential machine learning hacks for training smart data models*: Packt Publishing.

Lee, W. M. (2019). *Python Machine Learning*: Wiley.

Liu, H., & Lang, B. (2019). Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey. *applied sciences*.

Maniriho, P., & Ahmad, T. (2018). Analyzing the Performance of Machine Learning Algorithms in Anomaly Network Intrusion Detection Systems. In (pp. 1-6).

Maseer, Z. K., Yusof, R., Bahaman, N., Mostafa, S. A., & Foozy, C. F. M. (2021). Benchmarking of Machine Learning for Anomaly-Based Intrusion Detection Systems in the CICIDS2017 Dataset. *Access, 9*, 1-1. doi:10.1109/ACCESS.2021.3056614

McHugh, J. (2000). Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Trans. Inf. Syst. Secur., 3*(4), 262–294. doi:10.1145/382912.382923

McHugh, J., Christie, A., & Allen, J. (2000). Defending Yourself: The Role of Intrusion Detection Systems. *IEEE Software, 17*(5), 42-51. doi:10.1109/52.877859

Mehmood, T., & Rais, H. B. (2016). Machine learning algorithms in context of intrusion detection. In (pp. 369-373).

mergecap: Merging multiple capture files into one. (2021). Retrieved from https://www.wireshark.org/docs/wsug_html_chunked/AppToolsmergecap.html

Nassar, M., El-Bahnasawy, N. A., Ahmed, H.-D. H., Saleeb, A. A., & El-Samie, F. E. A. (2019). Network Intrusion Detection, Literature Review and Some Techniques Comparision. In (pp. 62-71): IEEE.

O'Reilly. (2021). Retrieved from https://www.oreilly.com/

parsezeeklogs 2.0.1. (2019). Retrieved from https://pypi.org/project/parsezeeklogs/

Patterson, J., & Gibson, A. (2017). *Deep Learning: A Practitioner's Approach*: O'Reilly Media.

Phadke, A., Kulkarni, M., Bhawalkar, P., & Bhattad, R. (2019). A Review of Machine Learning Methodologies for Network Intrusion Detection. In (pp. 272-275).

Python. (2021). Retrieved from https://www.python.org/

Rahat, F., & Ahsan, S. N. (2015). Comparative study of machine learning techniques for pre-processing of network intrusion data. In (pp. 46-51).

Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. *Computers & Security, 86*, 147-167. doi:10.1016/j.cose.2019.06.005

Rodda, S., & Erothi, U. S. R. (2016). Class imbalance problem in the Network Intrusion Detection Systems. In (pp. 2685-2688).

Sarkar, D., & Natarajan, V. (2019). *Ensemble Machine Learning Cookbook*: Packt Publishing.

Schinagl, S., Schoon, K., & Paans, R. (2015). A Framework for Designing a Security Operations Centre (SOC). In (pp. 2253-2262): IEEE.

scikit-learn. (2020). Retrieved from https://scikit-learn.org/stable/index.html

Shashank, K., & Balachandra, M. (2018). Review on Network Intrusion Detection Techniques using Machine Learning. In (pp. 104-109).

Sopan, A., Berninger, M., Mulakaluri, M., & Katakam, R. (2018). Building a Machine Learning Model for the SOC, by the Input from the SOC, and Analyzing it for the SOC. In (pp. 1-8): IEEE.

Spyder. (2020). Retrieved from https://www.spyder-ide.org/

Tavallaee, M., Bagheri, E., Wei, L., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In (pp. 1-6).

Thapa, N., Liu, Z., Kc, D. B., Gokaraju, B., & Roy, K. (2020). Comparison of Machine Learning and Deep Learning Models for Network Intrusion Detection Systems. *Future internet, 12*(10), 1. doi:10.3390/fi12100167

Thaseen, S., & Kumar, C. A. (2013). An analysis of supervised tree based classifiers for intrusion detection system. In (pp. 294-299).

Tsai, C.-F., Hsu, Y.-F., Lin, C.-Y., & Lin, W.-Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications, 36*(10), 11994-12000. doi:10.1016/j.eswa.2009.05.029

Ubuntu. (2021). Retrieved from https://ubuntu.com/

Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep Learning Approach for Intelligent Intrusion Detection System. *Access, 7*, 41525-41550. doi:10.1109/ACCESS.2019.2895334

Yihunie, F., Abdelfattah, E., & Regmi, A. (2019). Applying Machine Learning to Anomaly-Based Intrusion Detection Systems. In (pp. 1-5): IEEE.

Yu-Xin, M. (2011). The practice on using machine learning for network anomaly intrusion detection. In (Vol. 2, pp. 576-581).

zeek. (2020). Retrieved from https://zeek.org/

APPENDICES

APPENDIX A: PHASE ONE PYTHON SCRIPT

```
'''
Jonah Baron
PhD Cyber Operations
Dakota State University
MLNIDS - Phase One
'''

import os
import sys
import glob
import time
import numpy as np
import pandas as pd
import warnings
import csv
import bisect

import smtplib
import email.message
import email.utils

import joblib
from joblib import parallel_backend, Parallel

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, Normalizer,
LabelEncoder

from sklearn.feature_selection import SelectPercentile, f_classif, RFE, SelectKBest, VarianceThreshold
from sklearn.decomposition import PCA

from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier, LocalOutlierFactor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, BaggingClassifier, IsolationForest
from sklearn.svm import SVC, LinearSVC, OneClassSVM
from sklearn.linear_model import LogisticRegression, SGDClassifier, RidgeClassifier

from sklearn.metrics import confusion_matrix, multilabel_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_auc_score,
classification_report

hostname = "Windows"
#hostname = 'Ubuntu-1-KDD99'
#hostname = 'Ubuntu-2-NSLKDD'
#hostname = 'Ubuntu-3-UNSW'
#hostname = 'Ubuntu-4-CICIDS'
#hostname = 'Ubuntu-5-CIDDS'

def loadKDD99(datasetName):
    #http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
    print ('Loading dataset:', datasetName)

    col_names = ['duration','protocol_type','service','flag','src_bytes',
        'dst_bytes','land','wrong_fragment','urgent','hot','num_failed_logins',
        'logged_in','num_compromised','root_shell','su_attempted','num_root',
        'num_file_creations','num_shells','num_access_files','num_outbound_cmds',
        'is_host_login','is_guest_login','count','srv_count','serror_rate',
```

```
            'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate',
            'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count',
            'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate',
            'dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate',
            'dst_host_rerror_rate','dst_host_srv_rerror_rate','label']

        filename = os.path.join('KDD99', 'kddcup.data.csv')
        trainset = pd.read_csv(filename, header=None, names = col_names)
        filename = os.path.join('KDD99', 'corrected.csv')
        testset = pd.read_csv(filename, header=None, names = col_names)
        fullset = pd.concat([trainset, testset], ignore_index=True)
        del trainset, testset

        print ('Successfully read in dataset')
        print (fullset.head())
        print ('Fullset shape:', fullset.shape)

        #LABEL EXTRACTION AND GROUPING
        fullset = fullset.fillna(value='-')
        print (fullset['label'].value_counts())
        Y_labels = fullset['label'].copy()
        Y_labels[Y_labels != 'normal.'] = 'malicious'
        Y_labels[Y_labels == 'normal.'] = 'normal'
        print ('Successfully grouped malicious features')
        print (Y_labels.value_counts())

        #FEATURE RENAMING AND DROPPING
        filtered = ['duration', 'protocol_type', 'service',
                    'flag', 'src_bytes', 'dst_bytes']
        fullset = fullset.filter(filtered)
        fullset = fullset.rename(columns={'duration':'duration', 'protocol_type':'proto',
                                          'service':'service', 'flag':'conn_state',
                                          'src_bytes':'orig_ip_bytes', 'dst_bytes':'resp_ip_bytes'})
        zeekpartial = ['duration', 'proto', 'service',
                       'conn_state', 'orig_ip_bytes', 'resp_ip_bytes']
        fullset = fullset.reindex(columns=zeekpartial)
        #fullset = fullset.drop(['label'], axis=1)

        #ENCODING, SCALING, SUBSET CREATION, AND FEATURE REDUCTION
        X_train, X_test, Y_train, Y_test = transformBase(fullset, Y_labels, datasetName)

        return X_train, X_test, Y_train, Y_test

    def loadNSLKDD(datasetName):
        #http://205.174.165.80/CICDataset/NSL-KDD/Dataset/
        #https://www.unb.ca/cic/datasets/index.html
        print ('Loading dataset:', datasetName)

        col_names = ['duration','protocol_type','service','flag','src_bytes',
            'dst_bytes','land','wrong_fragment','urgent','hot','num_failed_logins',
            'logged_in','num_compromised','root_shell','su_attempted','num_root',
            'num_file_creations','num_shells','num_access_files','num_outbound_cmds',
            'is_host_login','is_guest_login','count','srv_count','serror_rate',
            'srv_serror_rate','rerror_rate','srv_rerror_rate','same_srv_rate',
            'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count',
            'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate',
            'dst_host_srv_diff_host_rate','dst_host_serror_rate','dst_host_srv_serror_rate',
            'dst_host_rerror_rate','dst_host_srv_rerror_rate','label','unknown']

        filename = os.path.join('NSL-KDD', 'KDDTrain+.txt')
        trainset = pd.read_csv(filename, header=None, names = col_names)
        filename = os.path.join('NSL-KDD', 'KDDTest+.txt')
        testset = pd.read_csv(filename, header=None, names = col_names)
        fullset = pd.concat([trainset, testset], ignore_index=True)
        del trainset, testset

        print ('Successfully read in dataset')
        print (fullset.head())
        print ('Fullset shape:', fullset.shape)

        #LABEL EXTRACTION AND GROUPING
        fullset = fullset.fillna(value='-')
        print (fullset['label'].value_counts())
```

```
    Y_labels = fullset['label'].copy()
    Y_labels[Y_labels != 'normal'] = 'malicious'
    print ('Successfully grouped malicious features')
    print (Y_labels.value_counts())

    #FEATURE RENAMING AND DROPPING
    filtered = ['duration', 'protocol_type', 'service',
                'flag', 'src_bytes', 'dst_bytes']
    fullset = fullset.filter(filtered)
    fullset = fullset.rename(columns={'duration':'duration', 'protocol_type':'proto',
                                      'service':'service', 'flag':'conn_state',
                                      'src_bytes':'orig_ip_bytes', 'dst_bytes':'resp_ip_bytes'})
    zeekpartial = ['duration', 'proto', 'service',
                   'conn_state', 'orig_ip_bytes', 'resp_ip_bytes']
    fullset = fullset.reindex(columns=zeekpartial)
    #fullset = fullset.drop(['label', 'unknown'], axis=1)

    #ENCODING, SCALING, SUBSET CREATION, AND FEATURE REDUCTION
    X_train, X_test, Y_train, Y_test = transformBase(fullset, Y_labels, datasetName)

    return X_train, X_test, Y_train, Y_test

def loadUNSWNB15(datasetName):
    #https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/
    print ('Loading dataset:', datasetName)

    col_names = ['srcip', 'sport', 'dstip', 'dsport', 'proto', 'state',
                 'dur', 'sbytes', 'dbytes', 'sttl', 'dttl', 'sloss', 'dloss',
                 'service', 'Sload', 'Dload', 'Spkts', 'Dpkts', 'swin', 'dwin',
                 'stcpb', 'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'res_bdy_len',
                 'Sjit', 'Djit', 'Stime', 'Ltime', 'Sintpkt', 'Dintpkt', 'tcprtt', 'synack',
                 'ackdat', 'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd',
                 'is_ftp_login', 'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm',
                 'ct_src_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',
                 'attack_cat', 'Label']

    path = os.path.join('UNSW-NB15', 'Full', '')
    fnames = glob.glob(path + '*.csv')

    fcontent = []
    for fname in fnames:
        fcontent.append(pd.read_csv(fname, header=None, names=col_names, dtype='str'))
    fullset = pd.concat(fcontent, ignore_index=True)
    del fcontent

    print ('Successfully read in dataset')
    print (fullset.head())
    print ('Fullset shape:', fullset.shape)

    #LABEL EXTRACTION AND GROUPING
    fullset = fullset.fillna(value='-')
    print (fullset['Label'].value_counts())
    Y_labels = fullset['Label'].copy()
    Y_labels[Y_labels != '0'] = 'malicious'
    Y_labels[Y_labels == '0'] = 'normal'
    print ('Successfully grouped malicious features')
    print (Y_labels.value_counts())

    #FEATURE RENAMING AND DROPPING
    filtered = ['srcip', 'sport', 'dstip', 'dsport',
                'proto', 'service', 'dur', 'sbytes',
                'dbytes', 'Spkts', 'Dpkts']
    fullset = fullset.filter(filtered)
    fullset = fullset.rename(columns={'srcip':'id.orig_h', 'sport':'id.orig_p',
                                      'dstip':'id.resp_h', 'dsport':'id.resp_p',
                                      'dur':'duration', 'sbytes':'orig_ip_bytes', 'dbytes':'resp_ip_bytes',
                                      'Spkts':'orig_pkts', 'Dpkts':'resp_pkts'})
    zeekpartial = ['id.orig_h', 'id.orig_p', 'id.resp_h', 'id.resp_p',
                   'proto', 'service', 'duration', 'orig_pkts',
                   'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes']
    fullset = fullset.reindex(columns=zeekpartial)
    #fullset = fullset.drop(['attack_cat', 'label'], axis=1)
```

```
    #ENCODING, SCALING, SUBSET CREATION, AND FEATURE REDUCTION
    X_train, X_test, Y_train, Y_test = transformBase(fullset, Y_labels, datasetName)

    return X_train, X_test, Y_train, Y_test

def loadCICIDS2017(datasetName):
    #http://205.174.165.80/CICDataset/CIC-IDS-2017/Dataset/
    #https://www.unb.ca/cic/datasets/index.html
    print ('Loading dataset:', datasetName)

    path = os.path.join('CICIDS2017', 'MachineLearningCSV', 'MachineLearningCVE', '')
    fnames = glob.glob(path + '*.csv')

    fcontent = []
    for fname in fnames:
        fcontent.append(pd.read_csv(fname))
    fullset = pd.concat(fcontent, ignore_index=True)
    fullset.columns = fullset.columns.str.strip()

    print ('Successfully read in dataset')
    print (fullset.head())
    print ('Fullset shape:', fullset.shape)

    #LABEL EXTRACTION AND GROUPING
    fullset = fullset.fillna(value='-')
    print (fullset['Label'].value_counts())
    Y_labels = fullset['Label'].copy()
    Y_labels[Y_labels != 'BENIGN'] = 'malicious'
    Y_labels[Y_labels == 'BENIGN'] = 'normal'
    print ('Successfully grouped malicious features')
    print (Y_labels.value_counts())

    #FEATURE RENAMING AND DROPPING
    filtered = ['Destination Port', 'Flow Duration', 'Total Fwd Packets',
                'Total Backward Packets', 'Total Length of Fwd Packets', 'Total Length of Bwd Packets']
    fullset = fullset.filter(filtered)
    fullset = fullset.rename(columns={'Destination Port':'id.resp_p', 'Flow Duration':'duration',
                                      'Total Fwd Packets':'orig_pkts', 'Total Backward Packets':'resp_pkts',
                                      'Total Length of Fwd Packets':'orig_ip_bytes', 'Total Length of Bwd
Packets':'resp_ip_bytes'})
    zeekpartial = ['id.resp_p', 'duration', 'orig_pkts',
                   'resp_pkts', 'orig_ip_bytes', 'resp_ip_bytes']
    fullset = fullset.reindex(columns=zeekpartial)
    #fullset = fullset.drop(['Label'], axis=1)

    #ENCODING, SCALING, SUBSET CREATION, AND FEATURE REDUCTION
    X_train, X_test, Y_train, Y_test = transformBase(fullset, Y_labels, datasetName)

    return X_train, X_test, Y_train, Y_test

def loadCIDDS001(datasetName):
    #https://www.hs-coburg.de/forschung/forschungsprojekte-oeffentlich/informationstechnologie/cidds-coburg-
intrusion-detection-data-sets.html
    print ('Loading dataset:', datasetName)

    filtered_cols = ['Duration','Proto','Src IP Addr','Src Pt','Dst IP Addr','Dst Pt',
                     'Packets', #'Bytes' not included due to naming convention like '10 M'
                     'Flows','Flags','Tos','class']

    path = os.path.join('CIDDS-001', 'WISENT-CIDDS-001', 'CIDDS-001', 'traffic', 'OpenStack', '')
    fnames = glob.glob(path + '*.csv')

    fcontent = []
    for fname in fnames:
        fcontent.append(pd.read_csv(fname, usecols = filtered_cols))
    fullset = pd.concat(fcontent, ignore_index=True)

    print ('Successfully read in dataset')
    print (fullset.head())
    print ('Fullset shape:', fullset.shape)

    #LABEL EXTRACTION AND GROUPING
    fullset = fullset.fillna(value='-')
```

```
    print (fullset['class'].value_counts())
    Y_labels = fullset['class'].copy()
    Y_labels[Y_labels != 'normal'] = 'malicious'
    print ('Successfully grouped malicious features')
    print (Y_labels.value_counts())

    #FEATURE RENAMING AND DROPPING
    filtered = ['Duration', 'Proto', 'Src IP Addr', 'Src Pt',
                'Dst IP Addr', 'Dst Pt', 'Packets']
    fullset = fullset.filter(filtered)
    fullset = fullset.rename(columns={'Duration':'duration', 'Proto':'proto',
                                      'Src IP Addr':'id.orig_h', 'Src Pt':'id.orig_p',
                                      'Dst IP Addr':'id.resp_h', 'Dst Pt':'id.resp_p',
                                      'Packets':'orig_pkts'})
    zeekpartial = ['duration', 'proto', 'id.orig_h', 'id.orig_p',
                   'id.resp_h', 'id.resp_p', 'orig_pkts']
    fullset = fullset.reindex(columns=zeekpartial)
    #fullset = fullset.drop(['class'], axis=1)

    #ENCODING, SCALING, SUBSET CREATION, AND FEATURE REDUCTION
    X_train, X_test, Y_train, Y_test = transformBase(fullset, Y_labels, datasetName)

    #Utilize only 10% of full dataset (contained in X_test/Y_test) - Unique to CIDDS-001
    X_train, X_test, Y_train, Y_test = train_test_split(X_test, Y_test, test_size = .10, random_state = 0)

    return X_train, X_test, Y_train, Y_test

def transformBase(fullset, Y_labels, datasetName):
    pathname = saveConfig(datasetName)

    #DATATYPE CONVERSION TO STR/OBJECT
    #print (fullset.dtypes)
    col_names = list(fullset)
    for col in col_names:
        fullset[col] = fullset[col].astype('str')
    #print (fullset.dtypes)
    print ('Successfully converted dataframe column datatypes')
    print (fullset.head())

    #UNIQUE VALUES AND ENCODING
    unique = fullset.values.ravel()
    save = pathname + '_unique.joblib'
    joblib.dump(unique, save)

    enc = LabelEncoder()
    enc.fit(np.unique(fullset.values))
    save = pathname + '_encoder.joblib'
    joblib.dump(enc, save)
    fullset = fullset.apply(enc.transform)
    print ('Successfully encoded data')
    print (fullset.head())

    #SCALING
    scaler = StandardScaler()
    scaler.fit(fullset)
    save = pathname + '_scaler.joblib'
    joblib.dump(scaler, save)
    fullset = scaler.transform(fullset)
    print ('Successfully scaled data')

    #SUBSET CREATION
    X_train, X_test, Y_train, Y_test = train_test_split(fullset, Y_labels, test_size = .10, random_state = 0)
    print ('Successfully created subsets')

    #FEATURE REDUCTION
    print ('X_train shape:', X_train.shape)
    print ('X_test shape:', X_test.shape)

    pca = PCA(n_components='mle', svd_solver='full') #svd_solver='auto'
    pca.fit(X_train)
    save = pathname + '_pca.joblib'
    joblib.dump(pca, save)
    X_train = pd.DataFrame(pca.transform(X_train))
```

```
    X_test = pd.DataFrame(pca.transform(X_test))

    print ('Successfully reduced features')
    print ('X_train shape:', X_train.shape)
    print ('X_test shape:', X_test.shape)

    return X_train, X_test, Y_train, Y_test

def saveConfig(datasetName):
    if datasetName == 'KDD Cup 1999':
        pathname = os.path.join('KDD99', 'Output', 'KDD99')
    if datasetName == 'NSL-KDD':
        pathname = os.path.join('NSL-KDD', 'Output', 'NSLKDD')
    if datasetName == 'UNSW-NB15':
        pathname = os.path.join('UNSW-NB15', 'Output', 'UNSW')
    if datasetName == 'CICIDS 2017':
        pathname = os.path.join('CICIDS2017', 'Output', 'CICIDS')
    if datasetName == 'CIDDS-001':
        pathname = os.path.join('CIDDS-001', 'Output', 'CIDDS')
    #return path, name
    return pathname

def modelOutput(expected, predicted, modelName, datasetName):
    accuracy = accuracy_score(expected, predicted)
    precision = precision_score(expected, predicted , average='macro', labels=np.unique(predicted))
    recall = recall_score(expected, predicted, average='macro', labels=np.unique(predicted))
    f1 = f1_score(expected, predicted , average='macro', labels=np.unique(predicted))
    class_report = classification_report(expected, predicted)
    #returned = confusion_matrix(expected, predicted).ravel()
    #print(cm)

    #print(expected, predicted)
    #print ('Labels not found in expected set:')
    #print (set(expected)-set(predicted))

    print ('========================================')
    print (datasetName, '|', modelName)
    print ('========================================')
    print('Accuracy: %.3f' %accuracy)
    print('Precision: %.3f' %precision)
    print('Recall: %.3f' %recall)
    print('F1 score: %.3f' %f1)
    print('Classification report')
    print(class_report)
    print('**************************')

    return accuracy, precision, recall, f1

def modelProduction(datasetName, X_train, X_test, Y_train, Y_test):
    print ('\nProducing models...\n')

    scoringMethods = ['accuracy', 'precision_macro', 'f1_macro']
    for scoreMethod in scoringMethods:
        clfs = getOptimized(datasetName, scoreMethod)

        pathname = saveConfig(datasetName)
        rows = []

        for clf_name in clfs:
            start = time.time()
            print('Producing',clf_name)
            clf = clfs[clf_name]
            clf = clf.fit(X_train, Y_train)
            predicted = clf.predict(X_test)
            expected = Y_test
            end = time.time()
            trainTime = (end-start)/60
            print ('Model training time: %.3f' %trainTime, 'minutes')

            accuracy, precision, recall, f1 = modelOutput(expected, predicted, clf_name, datasetName)
            rows.append([clf_name, accuracy, precision, recall, f1, trainTime, scoreMethod])

            if clf_name == 'Naive Bayes - Gaussian NB':
```

```
                modelSave = 'NB'
            if clf_name == 'Decision Tree':
                modelSave = 'DT'
            if clf_name == 'Ensemble - Random Forest':
                modelSave = 'RF'
            if clf_name == 'Ensemble - Ada Boost':
                modelSave = 'AB'
            if clf_name == 'Ensemble - Bagging Classifier':
                modelSave = 'BC'
            if clf_name == 'Linear - Logistic Regression':
                modelSave = 'LR'
            if clf_name == 'Linear - Stochastic Gradient Descent':
                modelSave = 'SGD'
            saveName = pathname + '-' + scoreMethod + '-' + modelSave + '.joblib'
            joblib.dump(clf, saveName)
            #input('Press any key to continue...')

        fields = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'Time', 'Score Method', datasetName]
        csvname = pathname + '-Models' + '-' + scoreMethod + '.csv'
        with open(csvname, 'w', newline = '') as csvfile:
            csvwriter = csv.writer(csvfile)
            csvwriter.writerow(fields)
            for row in rows:
                csvwriter.writerow(row)

def getOptimized(datasetName, scoreMethod):
    print ('Obtaining optimized model parameters...')

    #ACCURACY SCORE OPTIMIZATION
    if scoreMethod == 'accuracy':
        if datasetName == 'KDD Cup 1999':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='log2'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='log2', n_estimators=100),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=5000),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'NSL-KDD':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='log2'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='log2', n_estimators=5000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=3000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=1000),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=5, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'UNSW-NB15':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='gini', max_features='sqrt'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='log2', n_estimators=100),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1, n_estimators=3000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=1000),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=100,
penalty='l2'),
                }
        if datasetName == 'CICIDS 2017':
```

```
            clfs = {
                #Updated - 10/14/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='sqrt'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='sqrt', n_estimators=1000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=5,
max_samples=0.5, n_estimators=5000),
                'Linear - Logistic Regression' : LogisticRegression(C=1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=3000,
penalty='l2'),
                }
        if datasetName == 'CIDDS-001':
            clfs = {
                #Updated 10/28/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='sqrt'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='sqrt', n_estimators=1000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=5,
max_samples=0.5, n_estimators=1000),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=5, max_iter=100,
penalty='l2'),
                }

    #PRECISION SCORE OPTIMIZATION
    if scoreMethod == 'precision_macro':
        if datasetName == 'KDD Cup 1999':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='auto'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='log2', n_estimators=100),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=5,
max_samples=0.5, n_estimators=5000),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'NSL-KDD':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='gini', max_features='auto'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='sqrt', n_estimators=100),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=3000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=3000),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=5, max_iter=3000,
penalty='l2'),
                }
        if datasetName == 'UNSW-NB15':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='sqrt'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='sqrt', n_estimators=1000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=1,
max_samples=0.5, n_estimators=100),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=10, max_iter=100,
penalty='l2'),
                }
        if datasetName == 'CICIDS 2017':
```

```
            clfs = {
                #Updated - 10/14/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='auto'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='sqrt', n_estimators=1000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=5,
max_samples=0.5, n_estimators=5000),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'CIDDS-001':
            clfs = {
                #Updated 10/28/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='auto'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='log2', n_estimators=3000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=1000),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=5, max_iter=100,
penalty='l2'),
                }

    #F1 SCORE OPTIMIZATION
    if scoreMethod == 'f1_macro':
        if datasetName == 'KDD Cup 1999':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='auto'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='sqrt', n_estimators=100),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=100),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'NSL-KDD':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='log2'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='gini',
max_features='log2', n_estimators=1000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=3000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=1000),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=5, max_iter=5000,
penalty='l2'),
                }
        if datasetName == 'UNSW-NB15':
            clfs = {
                #Updated - 10/13/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='gini', max_features='log2'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='auto', n_estimators=100),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1, n_estimators=3000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=0.5,
max_samples=0.5, n_estimators=3000),
                'Linear - Logistic Regression' : LogisticRegression(C=0.1, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'CICIDS 2017':
```

```
            clfs = {
                #Updated - 10/14/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='log2'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='auto', n_estimators=1000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=5,
max_samples=0.5, n_estimators=3000),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=15, max_iter=1000,
penalty='l2'),
                }
        if datasetName == 'CIDDS-001':
            clfs = {
                #Updated 10/28/2020
                'Naive Bayes - Gaussian NB' : GaussianNB(),
                'Decision Tree' : DecisionTreeClassifier(criterion='entropy', max_features='log2'),
                'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1, criterion='entropy',
max_features='auto', n_estimators=3000),
                'Ensemble - Ada Boost' : AdaBoostClassifier(learning_rate=1.5, n_estimators=5000),
                'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1, max_features=5,
max_samples=0.5, n_estimators=1000),
                'Linear - Logistic Regression' : LogisticRegression(C=10, max_iter=2500, penalty='l2'),
                'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1, alpha=5, max_iter=100,
penalty='l2'),
                }

    return clfs

def optimizeModels(datasetName, X_test, Y_test):
    print ('\nOptimizing models...\n')

    rows = []
    scoringMethods = ['accuracy', 'precision_macro', 'f1_macro']
    clfs = {
        #'Naive Bayes - Gaussian NB' : GaussianNB(),
        'Decision Tree' : DecisionTreeClassifier(),
        'Ensemble - Random Forest' : RandomForestClassifier(n_jobs=-1),
        'Ensemble - Ada Boost' : AdaBoostClassifier(),
        'Ensemble - Bagging Classifier' : BaggingClassifier(n_jobs=-1),
        'Linear - Logistic Regression' : LogisticRegression(),
        'Linear - Stochastic Gradient Descent' : SGDClassifier(n_jobs=-1),
        }
    for scoreMethod in scoringMethods:
        for clf_name in clfs:
            print('Tuning', clf_name, 'for ideal', scoreMethod, 'score')
            start = time.time()
            if clf_name == 'Decision Tree':
                param_grid = {'criterion' : ['gini', 'entropy'],
                              'max_features' : ['auto', 'sqrt', 'log2']
                              }
            if clf_name == 'Ensemble - Random Forest':
                param_grid = {'n_estimators' : [100, 1000, 3000, 5000],
                              'criterion' : ['gini', 'entropy'],
                              'max_features' : ['auto', 'sqrt', 'log2']
                              }
            if clf_name == 'Ensemble - Ada Boost':
                param_grid = {'n_estimators' : [100, 1000, 3000, 5000],
                              'learning_rate' : [0.5, 1, 1.5]
                              }
            if clf_name == 'Ensemble - Bagging Classifier':
                param_grid = {'n_estimators' : [100, 1000, 3000, 5000],
                              'max_features' : [0.5, 1, 5],
                              'max_samples' : [0.1, 0.5, 1]
                              }
            if clf_name == 'Linear - Logistic Regression':
                param_grid = {'penalty' : ['l2', 'l1', 'elasticnet'],
                              'C' : [0.1, 1, 10],
                              'max_iter' : [2500, 5000, 7500, 10000]
                              }
            if clf_name == 'Linear - Stochastic Gradient Descent':
                param_grid = {'alpha' : [5, 10, 15],
```

```
                            'penalty' : ['l2', 'l1', 'elasticnet'],
                            'max_iter' : [100, 1000, 3000, 5000]
                            }
            clf = GridSearchCV(clfs[clf_name], param_grid, pre_dispatch=3, n_jobs=-1, verbose=5, cv=5,
scoring=scoreMethod)
            #with parallel_backend('threading'):
            try:
                clf = clf.fit(X_test, Y_test)
            except:
                subject = hostname + ' - Python Script Crash'
                message = 'Python script crash for %s' %datasetName
                sendEmail(subject, message)
            end = time.time()
            trainTime = (end-start)/60

            rows.append([scoreMethod, clf_name, clf.best_params_, clf.best_score_, trainTime])
            print ('=====================================')
            print (datasetName)
            print (scoreMethod)
            print (clf_name)
            print ('Model training time: %.3f' %trainTime, 'minutes')
            print ('Best parameters:', clf.best_params_)
            print ('Best score: %.5f' %clf.best_score_)
            print ('=====================================')
            print (clf.cv_results_)
            #modelOutput(expected, predicted, clf_name, datasetName)
            #input('Press any key to continue...')

    pathname = saveConfig(datasetName)
    csvname = pathname + '-ModelParams.csv'
    fields = ['Score Method', 'Model', 'Best Parameters', 'Best Score', 'Time', datasetName]
    with open(csvname, 'w', newline = '') as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow(fields)
        for row in rows:
            csvwriter.writerow(row)

def sendEmail(subject, message):
    from_email = #source email
    password = #source email password credential
    to_email = #destination alterate email
    msg = '\r\n'.join([
        'To: %s' %to_email,
        'From: %s' %from_email,
        'Subject: %s' %subject,
        '', message])

    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.ehlo()
    server.starttls()
    server.ehlo()
    server.login(from_email, password)
    server.sendmail(from_email, [to_email], msg)
    server.quit()

def main():
    scriptStart = time.time()
    warnings.filterwarnings('always')
    #sys.stdout = open('output.txt', 'w')
    #sys.stderr = open('output.txt', 'w')
    #pd.set_option('display.max_rows', 300)
    #Parallel(n_jobs=-1)

    for dataset in range(5):
        if dataset == 0:
            datasetName = 'KDD Cup 1999'
            X_train, X_test, Y_train, Y_test = loadKDD99(datasetName)
        if dataset == 1:
            datasetName = 'NSL-KDD'
            X_train, X_test, Y_train, Y_test = loadNSLKDD(datasetName)
        if dataset == 2:
            datasetName = 'UNSW-NB15'
            X_train, X_test, Y_train, Y_test = loadUNSWNB15(datasetName)
```

```
        if dataset == 3:
            datasetName = 'CICIDS 2017'
            X_train, X_test, Y_train, Y_test = loadCICIDS2017(datasetName)
        if dataset == 4:
           datasetName = 'CIDDS-001'
           X_train, X_test, Y_train, Y_test = loadCIDDS001(datasetName)

        print ('X_train shape:', X_train.shape)
        print ('X_test shape:', X_test.shape)

        #optimizeModels(datasetName, X_test, Y_test)
        #modelProduction(datasetName, X_train, X_test, Y_train, Y_test)

    scriptFinish = time.time()
    elapsedTime = scriptFinish-scriptStart

    subject = hostname + ' - Script Completed'
    message = 'Elapsed runtime: %s' %elapsedTime
    sendEmail(subject, message)

if __name__ == '__main__':
    main()
```

APPENDIX B: PHASE TWO PYTHON SCRIPT

```
'''
Jonah Baron
PhD Cyber Operations
Dakota State University
MLNIDS - Phase Two
'''

import os
import sys
import glob
import time
import numpy as np
import pandas as pd
import warnings
import csv
import bisect

import smtplib
import email.message
import email.utils

import joblib
from joblib import parallel_backend, Parallel

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsScaler, RobustScaler, Normalizer,
LabelEncoder

from sklearn.feature_selection import SelectPercentile, f_classif, RFE, SelectKBest, VarianceThreshold
from sklearn.decomposition import PCA

from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier, LocalOutlierFactor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, BaggingClassifier, IsolationForest
from sklearn.svm import SVC, LinearSVC, OneClassSVM
from sklearn.linear_model import LogisticRegression, SGDClassifier, RidgeClassifier

from sklearn.metrics import confusion_matrix, multilabel_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, roc_auc_score,
classification_report

from parsezeeklogs import ParseZeekLogs #custom downloaded PyPI package

hostname = "Windows"
#hostname = 'Ubuntu-1-KDD99'
#hostname = 'Ubuntu-2-NSLKDD'
#hostname = 'Ubuntu-3-UNSW'
#hostname = 'Ubuntu-4-CICIDS'
#hostname = 'Ubuntu-5-CIDDS'

def loadZeekLogs(datasetName):
    zeekFile = os.path.join('Zeek', 'Final', 'CICIDS2017', 'conn.log')
    zeekLogs = ParseZeekLogs(zeekFile, output_format='csv', safe_headers=False)
    print(zeekLogs.get_fields())

    predictionFile = os.path.join('Zeek', 'predictionset.csv')
    with open(predictionFile,'w') as outfile:
        outfile.write(zeekLogs.get_fields() + '\n')
        for log in zeekLogs:
            if log is not None:
                outfile.write(log + '\n')

    zeekfull = ['ts', 'uid', 'id.orig_h', 'id.orig_p', 'id.resp_h', 'id.resp_p',
```

```
                'proto', 'service', 'duration', 'orig_bytes', 'resp_bytes', 'conn_state',
                'local_orig', 'local_resp', 'missed_bytes', 'history', 'orig_pkts',
                'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes', 'tunnel_parents']

    if datasetName == 'KDD Cup 1999':
        zeekpartial = ['duration', 'proto', 'service',
                'conn_state', 'orig_ip_bytes', 'resp_ip_bytes']
    if datasetName == 'NSL-KDD':
        zeekpartial = ['duration', 'proto', 'service',
                'conn_state', 'orig_ip_bytes', 'resp_ip_bytes']
    if datasetName == 'UNSW-NB15':
        zeekpartial = ['id.orig_h', 'id.orig_p', 'id.resp_h', 'id.resp_p',
                'proto', 'service', 'duration', 'orig_pkts',
                'orig_ip_bytes', 'resp_pkts', 'resp_ip_bytes']
    if datasetName == 'CICIDS 2017':
        zeekpartial = ['id.resp_p', 'duration', 'orig_pkts',
                'resp_pkts', 'orig_ip_bytes', 'resp_ip_bytes']
    if datasetName == 'CIDDS-001':
       zeekpartial = ['duration', 'proto', 'id.orig_h', 'id.orig_p',
                'id.resp_h', 'id.resp_p', 'orig_pkts']

    predictionset = pd.read_csv(predictionFile, usecols=zeekpartial)

    print ('Successfully read in prediction dataset')
    print (predictionset.head())
    print ('Predictionset shape:', predictionset.shape)

    return predictionset

def transformPrediction(predictionset, datasetName):
    pathname = saveConfig(datasetName)

    enc = joblib.load(pathname + '_encoder.joblib')
    scaler = joblib.load(pathname + '_scaler.joblib')
    pca = joblib.load(pathname + '_pca.joblib')
    unique = joblib.load(pathname + '_unique.joblib')

    #Transformations - datatype conversions, encoding, scaling, subset creation, feature reduction
    print (enc)
    print (scaler)
    print (pca)
    print (unique)

    #ENSURE '-' IS SEEN IN ENCODER SCHEMA
    enc_classes = enc.classes_.tolist()
    bisect.insort_left(enc_classes, '-')
    enc.classes_ = enc_classes

    #DATATYPE CONVERSION TO STR/OBJECT
    predictionset = predictionset.fillna(value='-')
    #print (fullset.dtypes)
    col_names = list(predictionset)
    for col in col_names:
        predictionset[col] = predictionset[col].astype('str')
    #print (fullset.dtypes)
    print ('Successfully converted dataframe column datatypes')
    print (predictionset.head())

    #ENCODING
    predictionvalues = list(np.unique(predictionset.values.ravel()))
    trainedvalues = list(np.unique(unique.tolist()))

    remove = []
    for value in predictionvalues:
        if value not in trainedvalues:
            remove.append(value)
    predictionset.replace(to_replace=remove, value='-', inplace=True)

    predictionset = predictionset.apply(enc.transform)
    print ('Successfully encoded data')
    print (predictionset.head())

    #SCALING
```

```
    predictionset = scaler.transform(predictionset)
    print ('Successfully scaled data')

    #FEATURE REDUCTION
    predictionset = pca.transform(predictionset)
    print ('Successfully reduced features')
    print ('Prediction set shape:', predictionset.shape)


    scoringMethods = ['accuracy', 'precision_macro', 'f1_macro']
    clfs = ['NB', 'DT', 'RF', 'AB', 'BC', 'LR', 'SGD']
    rows = []

    for scoreMethod in scoringMethods:
        for clfName in clfs:
            clfFile = '-' + scoreMethod + '-' + clfName + '.joblib'
            clf = joblib.load(pathname + clfFile)
            print (clf)
            #print (clf.feature_importances_)

            prediction = clf.predict(predictionset)
            unique, counts = np.unique(prediction, return_counts=True)
            #print (unique, counts)
            print (np.asarray((unique, counts)).T)
            if len(unique) == 1:
                if unique[0] == 'normal':
                    flagged = 0
                    rows.append([clfName, scoreMethod, 0, counts[0], counts[0], flagged])
                if unique[0] == 'malicious':
                    flagged = 1
                    rows.append([clfName, scoreMethod, counts[0], 0, counts[0], flagged])
            if len(unique) == 2:
                flagged = counts[0]/(counts[0]+counts[1])
                rows.append([clfName, scoreMethod, counts[0], counts[1], (counts[0] + counts[1]), flagged])
            print ('Flagged traffic: ', flagged)

    fields = ['Model', 'Score Method', 'Malicious', 'Normal', 'Total', 'Flagged Traffic', datasetName]
    csvname = pathname + '-Predictions.csv'
    with open(csvname, 'w', newline = '') as csvfile:
        csvwriter = csv.writer(csvfile)
        csvwriter.writerow(fields)
        for row in rows:
            csvwriter.writerow(row)

def saveConfig(datasetName):
    if datasetName == 'KDD Cup 1999':
        pathname = os.path.join('KDD99', 'Output', 'KDD99')
    if datasetName == 'NSL-KDD':
        pathname = os.path.join('NSL-KDD', 'Output', 'NSLKDD')
    if datasetName == 'UNSW-NB15':
        pathname = os.path.join('UNSW-NB15', 'Output', 'UNSW')
    if datasetName == 'CICIDS 2017':
        pathname = os.path.join('CICIDS2017', 'Output', 'CICIDS')
    if datasetName == 'CIDDS-001':
        pathname = os.path.join('CIDDS-001', 'Output', 'CIDDS')

    return pathname

def sendEmail(subject, message):
    from_email = #source email
    password = #source email password credential
    to_email = #destination alter email
    msg = '\r\n'.join([
        'To: %s' %to_email,
        'From: %s' %from_email,
        'Subject: %s' %subject,
        '', message])

    server = smtplib.SMTP('smtp.gmail.com', 587)
    server.ehlo()
    server.starttls()
    server.ehlo()
    server.login(from_email, password)
```

```
    server.sendmail(from_email, [to_email], msg)
    server.quit()

def main():
    warnings.filterwarnings('always')
    scriptStart = time.time()

    for dataset in range(5):
        if dataset == 0:
            datasetName = 'KDD Cup 1999'
        if dataset == 1:
            datasetName = 'NSL-KDD'
        if dataset == 2:
            datasetName = 'UNSW-NB15'
        if dataset == 3:
            datasetName = 'CICIDS 2017'
        if dataset == 4:
           datasetName = 'CIDDS-001'

        #LOAD ZEEK CONN LOG AND PREDICT
        predictionset = loadZeekLogs(datasetName)
        transformPrediction(predictionset, datasetName)

    scriptFinish = time.time()
    elapsedTime = scriptFinish-scriptStart

    subject = hostname + ' - Script Completed'
    message = 'Elapsed runtime: %s' %elapsedTime
    sendEmail(subject, message)

if __name__ == '__main__':
    main()
```

APPENDIX C: LIST OF ZEEK CONNECTION LOG FIELDS AND DATASET FEATURES

- Zeek Connection Log - ts, uid, id, proto, service, duration, orig_bytes, resp_bytes, conn_state, local_orig, local_resp, missed_bytes, history, orig_pkts, orig_ip_bytes, resp_pkts, resp_ip_bytes, tunnel_parents, orig_l2_addr, resp_l2_addr, vlan, inner_vlan, speculative_service
- KDD 99 Dataset - duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate
- NSL-KDD Dataset - duration, protocol_type, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate,
- UNSW-NB15 Dataset - srcip, sport, dstip, dsport, proto, state, dur, sbytes, dbytes, sttl, dttl, sloss, dloss, service, Sload, Dload, Spkts, Dpkts, swin, dwin, stcpb, dtcpb, smeansz, dmeansz, trans_depth, res_bdy_len, Sjit, Djit, Stime, Ltime, Sintpkt, Dintpkt, tcprtt, synack, ackdat, is_sm_ips_ports, ct_state_ttl, ct_flw_http_mthd,

is_ftp_login, ct_ftp_cmd, ct_srv_src, ct_srv_dst, ct_dst_ltm, ct_src_ ltm, ct_src_dport_ltm, ct_dst_sport_ltm, ct_dst_src_ltm, attack_cat, Label

- CICIDS 2017 Dataset - DestinationPort, FlowDuration, TotalFwdPackets, TotalBackwardPackets, TotalLengthofFwdPackets, TotalLengthofBwdPackets, FwdPacketLengthMax, FwdPacketLengthMin, FwdPacketLengthMean, FwdPacketLengthStd, BwdPacketLengthMax, BwdPacketLengthMin, BwdPacketLengthMean, BwdPacketLengthStd, FlowBytes/s, FlowPackets/s, FlowIATMean, FlowIATStd, FlowIATMax, FlowIATMin, FwdIATTotal, FwdIATMean, FwdIATStd, FwdIATMax, FwdIATMin, BwdIATTotal, BwdIATMean, BwdIATStd, BwdIATMax, BwdIATMin, FwdPSHFlags, BwdPSHFlags, FwdURGFlags, BwdURGFlags, FwdHeaderLength, BwdHeaderLength, FwdPackets/s, BwdPackets/s, MinPacketLength, MaxPacketLength, PacketLengthMean, PacketLengthStd, PacketLengthVariance, FINFlagCount, SYNFlagCount, RSTFlagCount, PSHFlagCount, ACKFlagCount, URGFlagCount, CWEFlagCount, ECEFlagCount, Down/UpRatio, AveragePacketSize, AvgFwdSegmentSize, AvgBwdSegmentSize, FwdHeaderLength, FwdAvgBytes/Bulk, FwdAvgPackets/Bulk, FwdAvgBulkRate, BwdAvgBytes/Bulk, BwdAvgPackets/Bulk, BwdAvgBulkRate, SubflowFwdPackets, SubflowFwdBytes, SubflowBwdPackets, SubflowBwdBytes, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, ActiveMean, ActiveStd, ActiveMax, ActiveMin, IdleMean, IdleStd, IdleMax, IdleMin, Label,
- CIDDS-001 Dataset - Date first seen, Duration, Proto, Src IP Addr, Src Pt, Dst IP Addr, Dst Pt, Packets, Bytes, Flows, Flags, Tos, class, attackType, attackID, attackDescription