PER-INSTANCE SELECTION OF MACHINE LEARNING CLASSIFIERS FOR IDS AND IPS

A thesis submitted in partial fulfillment of the requirements of the University of East London for Degree of Professional Doctorate in Data Science

By Nasser Mohdhyder J B Al-khuzaei

University of East London

September 7, 2023

Abstract

Generally, malicious attacks on a network or server can be detected and counteracted using various techniques. The intrusion detection systems (IDS) and intrusion prevention systems (IPS) are two of the most common application systems in detecting and preventing cyber threats. Despite the ability of each of these systems to help organizations overcome various types of threats to their networks, additional decisions are required to ensure that they operate effectively. Even IDS and IPS remain vulnerable to conditions that render them less efficient and incapable of meeting the required operational targets. Consequently, it is imperative that organizations make decisions and take actions that tend to optimize the efficiency with which the cybersecurity applications operate.

Most organizations have IT infrastructure nowadays, and they differ in their requirements and sizes, but there is a common problem that is managing the flood of alerts coming from the IDS(Simone, 2009). The IDS creates a huge number of alerts. Not all the threats detected are true, but it means that the IDS has found a matching signature or pattern. These types of alarms are considered false positives and a result of misclassification. They can be a real pain for organizations to determine if these alarms are actionable or not. Because of the issues with the current IDS, there is a need for continued research to solve the classification issues, and for that, a per-instance multi-classifier is proposed.

This research will discuss the importance of researching a new algorithm that is a portfolio of multiple classifiers for intrusion detection systems in the cyber-security space. There is already much research in this field, and many classifiers have been proposed, but the fact there is no single classifier that can cover all threats with high accuracy. The intention is to have a portfolio of classifiers. Each classifier will be tested and trained on the dataset. The idea of having multiple classifiers that each classifier can complement and contribute to the classification. A Master classifier will determine the fitness of each classifier, depending on the presented instance, and all the fit classifiers will contribute to the classification by voting. The vote will determine if the instance is benign or an anomaly, and if it is an anomaly, it will determine the type of attack.

Keywords:

Multiclassification, Hybrid Classifiers, Intrusion Detection System (IDS), CSE-CIC-IDS2018, Machine Learning (ML), Deep Learning (DL), Artificial Neural Networks (ANN), Random Forest (RF), Convolutional Neural Networks (CNN), Network Traffic Analysis, Cybersecurity, Anomaly Detection, Model Accuracy, False Alarm Rate, Detection Rate, F1 Score, Model Optimization, Dataset Analysis

Contents

List of Tables

List of Figures

Acknowledgment

Most importantly, I give thanks to Almighty God for all his blessings and for giving me the strength and patience to complete this work.

I want to express my sincere gratitude to Dr. Yang Lee for his continues support during my doctorate studies. Dr Yang is a great example of professionalism and dedication. His feedback was constructive as it had wisdom and insights, which waived obstacles and improved the quality of my research. I am grateful to have Dr Yang as my supervisor for his compassion and thoughtful for his students.

I also want to thank my mother, my brother (Ibrahim), and my wife for their unconditional love, patience, and understanding. Their support and patience during my studies helped me go through all the challenges. Finally, I want to dedicate this work to my father, in loving memory

Chapter 1 Introduction To Thesis and Cybersecurity

1.1 Chapter Introduction

Due to the rapid technological evolution that has been witnessed in the world today, most activities in sectors such as business, healthcare, education, and entertainment are widely controlled by various forms of information technology. Such developments have also resulted in a situation whereby almost the entire world has become dependent on information technology. In each of the mentioned sectors, information technology offers additional quality by enhancing accuracy, speed, and efficiency, among other essential attributes.

Business and manufacturers are two of the sectors in which most aspects of technology are employed. Some of the key applications include automated production, especially in industries where large-scale production is required (KAREHKA, 2012). This aspect of technology is common in car manufacturing industries, chemical factories, food manufacturing and packaging, and heavy metal industries, among other similar sectors. The automation of the manufacturing process reduces the time taken in the production process while increasing both efficiency and accuracy. Other areas of application within the business and industry sectors include inventory management, supply chain management, and information management. Certain organizations also use techniques such as RFID time attendance or biometric systems to monitor their employees more effectively.

Technology has also boosted communication both locally and across borders, especially with the introduction of digital methods of exchanging information between different groups of audiences. Communication has been enhanced by the introduction of media such as video conferencing, mobile technologies, and emails (KAREHKA, 2012). Social media also promote interactions among people who are characterized by substantial geographical isolation. In education, the learning process has become even easier with the introduction of online libraries, simulated learning aids, and various forms of e-learning. On the other hand, bankers benefit from automatic teller machines, e-banking applications, and other emerging resources like cryptocurrencies (KAREHKA, 2012). Generally, technology has become an integral part of global society, a condition that necessitates high levels of cybersecurity to ensure that no costly damages are encountered while using such systems.

Th computer network has evolved from a simple communication medium between two systems into a very complex network architecture. Computer network evolution includes the internet, which introduced easy communication between different geographies, and the concept of public and private networks. These networks can carry and control critical information like power grids, stock markets, and the military. The network became heterogeneous, and the general approach is to have IT elements share the same resource pools (Storage, Memory, Compute, Network). This will lead to critical mission systems sharing some resources with non-critical systems. For example, you may have a power grid system that may share resources with an email system. That will put the power grid system at risk of being attacked because it shares resources with the email system and is not properly guarded. This

risk is very critical and can't be overlooked. Cybersecurity is mandatory to protect these systems. There are various cyber threats and different entities that might have the intention to do such acts (Political agendas or activists). One of the main threats that are very difficult to mitigate is (Distributed Denial of service/ Denial of service) DDOS/DOS attacks because the operator can't distinguish legitimate connections from attack connections. Also, the volume of attempted connections will be too huge to evaluate in time. There exist industrial solutions in the market for enterprises, but as long as the threats are evolving, Cybersecurity solutions need to evolve.

This research will focus on the detection of Cyber Security threats using machine learning. Looking after these two fields (Machine Learning and Cyber security), I will do interdisciplinary research where a portion of the research will focus on Cyber Security, and the other portion will be on machine learning.

1.1.1 Problem statement

The problem statement of the research is the following.

"Is it possible to increase the performance of IDS in precision and accuracy using per-instance selection of classifiers from a portfolio of classifiers?"

Information technology has intruded into many fields, and there is a very high dependency on technology for many processes these days. Connectivity is becoming increasingly necessary with the rise of smart cities and the internet of things. According to IDC, it is estimated that there will be 41.6 billion connected devices, generating 79.4 Zettabytes in 2025("The Global DataSphere & Its Enterprise Impact | IDC Blog," n.d.). With this amount of data trafficking, it's getting more challenging to analyze and identify threats.

At the same time, with the amount of generated data, Artificial intelligence flourished. Artificial intelligence can have real-world applications when provided with sufficient data. AI and machine learning can be beneficial in developing more robust Intrusion Detection systems (IDS). And indeed, many IDS depend on machine learning to identify anomalies.

Many models have already been developed for IDS, but not all models perform well for all types of network attacks. It's necessary to have an accurate system to act correctly according to the alarm. It's a real pain for SOC (security operation centers) to deal with false-positive alarms, as these alarms can come in millions. They have to investigate the alarms to determine if it's a genuine threat or benign traffic. A single model might not have the capability to classify some types of attacks or can misclassify regular traffic as a threat.

1.1.2 Contribution to Knowledge

My study bridges the gap and tackles the lack of the following:

- The ability to create a (portfolio classifier) with no budget with precision and accuracy relevant to my thesis
- per instance selection of classifier, where only selected classifiers can vote in each instance, so there will be a different set of classifiers to vote on for every threat.
- Modularity of the Model, where additional classifiers can be plugged in to enhance performance.

1.1.3 Aim

This research aims to investigate the application of multiple algorithms on security threat detection systems. Generally, different classifiers are characterized by both negative and positive characteristics. The study intends to determine the benefits that may be associated with the act of using multiple classifiers in a single system. The rationale behind this argument is that the weaknesses of given classifiers can be supplemented by the strengths of others. As a result of such a relationship, it is hypothesized that combining numerous classifiers of different types helps to establish a more effective hybrid compared to each of the individual constituents. However, the study approaches the idea from a unique perspective, which involves the selection of one classifier for each instance. This idea was inspired by SatZailla, which won the SAT competition multiple times("SATzilla: Portfolio-based algorithm selection for SAT," 2017). The proposed research procedure can be summarized in the following points.

- Use a simulated dataset that reflects modern cybersecurity threats.
- Build a portfolio of multiple classifiers.
- Create a Master Classifier that will select a classifier from the created portfolio based on the instance (connection) features.
- Classifier selection will be done for each instance.
- Test the classifier on the publicly available datasets for benchmarking with other classifiers.

The objectives for each chapter:

- Chapter1: Objectives and aims
- Chapter 2: Literature review for Cybersecurity and survey for for related research papers
- Chapter 3: Survey available datasets for research and define known problems in these datasets
- Chapter 4: define the projected design, and which framework will be adopted. Also, explain the proposed method.
- Chapter 5: Survey, Test and select the tools that will be used in this research
- Chapter 6: Data exploration using the standard techniques in data science to understand the dataset.
- Chapter 7: Build Multiple models along with the master classifier on a subsample
- Chapter 8: Build Multiple models along with the master classifier on complete dataset
- Chapter 9: Evaluate the results by benchmarking with different research papers that uses the same dataset and multi-classification (Not binary)
- Chapter 10: Discuss the outcomes and verify if the contribution of knowledge is achieved.

1.1.4 Study Hypothesis

Even though the answers to the study questions will only be established after performing the research, certain predictions can be established based on theoretical inferences and information gathered from existing studies. However, such predictions are limited due to a lack of adequate information and theoretical

frameworks on the proposed model. The proposed hypotheses for this research are as follows:

- The instantaneous selection of classifiers substantially improves the efficiency with which threats are detected in a server or network.
- Using the algorithmic approach to the selection of classifiers is more appropriate as it encourages the detailed consideration of all the performance factors of each individual classifier.
- The proposed model offers higher classifier performance outcomes than existing models.

1.1.5 Motivation

It's common to use IDS/IPS (intrusion detection system/Introduction prevention system). The IDS analyzes the traffic and identifies anomalies, triggering an alert. The IDS uses a model trained in historical traffic with all possible malicious and benign traffic scenarios. These models can report false alarms based on the quality of the data provided and the type of the model used.

Some models perform very well with some classes, and on the other hand, they misclassify very severely with other classes. A model can have a very high accuracy rate in detecting anomalies, but at the same time, it provides a lot of false-positive reports. Data traffic is not consistent, and it changes depending on the activity and the type of malicious attack. For example, network activity increases in the early morning when all people start their computers and check their emails. While most of these activities disappear at night, a model might not be able to distinguish morning

activity from a DDOS attack. At the same time, hackers know how to hide their activity in the network. For example, a normal port scan will be immediately detected, but hackers can do a slow scan that might not be detected by the IDS. For this reason, a portfolio of multiple classifiers might assist in increasing the accuracy, wherein, for each flow, a model will determine the most suitable classifier for that type of threat.

1.1.6 Scope of the research:

This research aims to increase the accuracy of the IDS model by using an algorithm to select a classifier for each flow that is most suitable for classification. To achieve this objective, we need data capture of the network, which can be achieved by having the PCAP file (Packet capture)to be used to train and validate the model. The data should have benign flow and anomalies to represent a more realistic network flow.

As discussed earlier, a portfolio of multiple models will be constructed. In order to determine if a selected model can fit in the portfolio, we have to have some metrics and criteria to calculate the fitness of these models. Each Model will be evaluated for their accuracy and precision and then benchmarked among each other. Then, we can validate if a model is suitable to be used for IDS in the first place. The evaluation will undertake three steps. The first step will investigate the model precision for each class. The second step will evaluate its ability to perform well with multiclassification. Finally, we will test if it can handle big data with reasonable time and compute resources. One of the essential parts of the research is developing a technique to select the suitable model for each flow in the network traffic. To develop this technique, we can use the evaluation of multiple models and build a model that will interface with the others as a selector.

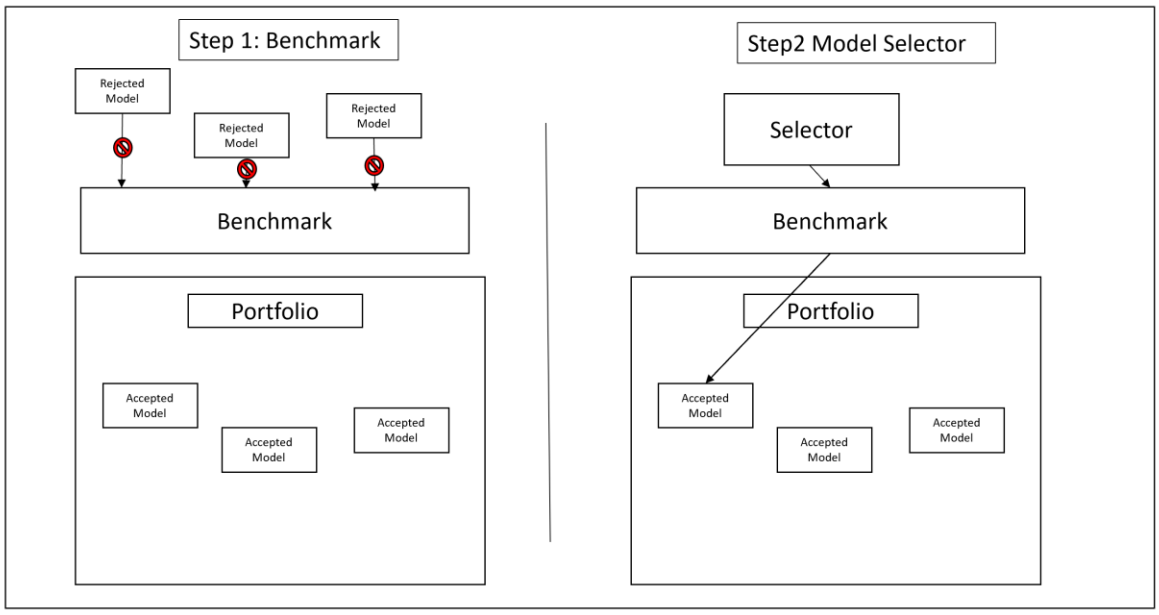


Figure 1 Benchmark and selector

1.1.7 The Rational of the Study

This study is expected to produce results that will contribute to the war against cybercrime in various contexts in which network-based technologies are employed. Classifiers are renowned for their role in enhancing the security capabilities of intrusion detection systems. These benefits can be analyzed from two principal perspectives. These perspectives are based on the fact that there are specific assumptions regarding classifier dependencies, different classifiers have varying outputs, and the idea that the process of selecting classifiers depends on neural networks, special mathematical functions, or algorithms. Based on these parameters, the benefits that come with a specially designed method of selection can be viewed from the perspectives of either classifier selection or classifier fusion(Ludmila I. Kuncheva et al., 2001). Thus, the rationale of this study can be described on the basis of the benefits associated with each of these approaches.

The first aspect of interpretation is the classifier selection, which forms the main core of this proposal. Under this criterion, each of the individual classifiers is assumed to be a special kind of expert. The expertise associated with each ‘expert' is believed to be useful in specific feature spaces. In the event that a feature vector represented as $x \in R^p$ is availed for a given classification, the process of assigning the class label to x is done in such a way that the highest credit is given to the specific classifier that is responsible for the vicinity of x (Ludmila I. Kuncheva et al., 2001). The process can be achieved through the nomination of either a single or multiple classifiers depending on the purpose for which the entire process is conducted(Subbulakshmi and Afroze, 2013). On the other hand, classifier fusion operates under the assumption that all classifiers are trained over the entire feature space, a condition that makes them more competitive than complementary(Saleem Malik Raja and Jeya Kumar, 2014). In such a situation, the selection process is considered to be more complex than in the case of a typical classifier selection process. Each of the available alternatives is selected mainly on the basis of the benefits or performance advantages that it introduces. Since it becomes relatively challenging to establish the best approach to the selection and combination of various types of classifiers, mathematical functions or algorithms are always used to accomplish this task.

The methods discussed above result in the establishment of a system consisting of a hybrid of critically selected classifiers. Generally, multiple classifiers are characterized by impressive outputs and performance efficiencies. It is evident that the entire performance of an intrusion detection system depends upon the types of classifiers that are selected in a given instance. Selecting high-efficiency classifiers would always result in generally high efficiencies, while poor selection will be

characterized by undesirable outputs. This project intends to solve classifier selection challenges by establishing a reliable algorithm for undertaking the process with the least errors. It intends to improve the effectiveness of intrusion detection systems by making them more reliable.

The benefits discussed above can be expressed in simple terms, which mainly revolve around improving cybersecurity in different contexts. Cybercrime has already been identified as one of the main challenges facing the implementation of technology in education, banking, construction, healthcare, and several other sectors across the globe. Research indicates that poorly developed intrusion detection systems and intrusion prevention systems are often vulnerable to false alarms. There are cases in which these resources fail to detect any threatening alarms within the servers or networks in which they are installed. In such a situation, they may report the absence of security threats even in cases where networks or servers are subjected to serious security compromises. The effective selection of classifiers helps to boost the overall efficiencies with which such systems work and make them more reliable. Thus, this study focuses primarily on improving the performance characteristics of intrusion detection systems to make them more effective. It tries to ensure that only the best-performing components of each security application are employed in the war against network-to-server intrusion. Overall, the outcomes of the study are expected to be of significant resourcefulness not only to researchers in the same field but also to the digital world at large.

This thesis has a novel approach for multi-classifier classification for Cybersecurity. This novel approach could help to develop IDS systems and improve the identification and classification of cybersecurity threats. There exist systems that can

perform the IDS tasks (classification), but the development and improvement of this novel approach could reduce the rate of errors, which are very costly for any entity or organization.

1.1.8 Thesis Structure

The structure of the remainder of the thesis will be as follows:

Cybersecurity:

This chapter will introduce the reader to the general concepts of cybersecurity in order to allow him/her to grasp the meaning of IDS and its function. The chapter will go through the history and evolution of the cyber security threats. In addition, we will review the tools and approaches that mitigate and defend systems from Cybersecurity threats.

Literature review and Systematic search:

In this part, we will view a systematic search for research that relates to this research. Different approaches will be reviewed that are either hybrid, pre-processing or Deep Neural. For each type of these approaches, we will discuss different papers that cover them.

Dataset Survey:

In this part, we will discuss the Datasets that are used in IDS. A survey will be reviewed on these datasets, which cover the distribution of how much each dataset is used in research. In addition, some problems of these datasets will be discussed that can affect the research in IDS.

Methodology:

This chapter will have the methodology that we will use and how we have created a method that is inspired by a prober framework within the Data mining community. By the end, we will have a detailed explanation of the proposed model.

Investigation and Selection of Software Packages:

In this part, we will have a small survey on the tools that are involved in Artificial intelligence and Machine learning. We will view the tool in two aspects. These aspects are cloud and local computing. In each aspect, we will determine if the available tools are fit for this research.

Dataset and Data Analysis:

Data analysis is an integral part of this research. In order to build a model, we need to understand the data and have proper insight into its variables and a summery of it. We will present the distribution of the variables and the problems that the dataset has. Finally, we will show how we have pre-processed that dataset.

Building models using Sub-sample:

In this part, we built different models using a sub-sample from the dataset. Depending on the results of these models. We build a portfolio of models that will contribute to the proposed portfolio model. At the end of the chapter, we have built the portfolio classifier and made a comparison with the individual classifiers that have participated in the portfolio.

Building Models using the complete Dataset:

In this part, we are repeating the process as in the “Building Models using Sub-sample,” but we have built the model without sub-sampling and using the complete dataset (except for duplication and removing NA). The dataset was only split between training and testing without any up sampling and subsampling. At the end of this chapter, we benchmark the results between the Full dataset results and the sub-sampling for all the models.

Benchmarking to other tests and studies:

We have benchmarked the proposed portfolio classifier to other research and studies. In this benchmark, we tried to make a fair comparison, but most of the studies do a binary classification, and few do multi-classification. We have built tables that compare Accuracy, Precision, F1 Score, and Recall.

Discussion and Conclusion:

This chapter will have the discussion and conclusion by summarizing the thesis and showing if the research goals have been fulfilled. Also, it will show major findings, impact on the industry, limitations, and future work.

1.2 Chapter Conclusion

In conclusion, this chapter laid the groundwork for the research by highlighting the growing dependency on technology and the corresponding rise in cyber threats. The background discussion included the types and roles of IDS, along with their limitations in practical environments. The issues identified, such as high false positive rates and inability to generalize across various traffic types, justified the need for a more flexible and intelligent solution. The proposed idea of using a classifier portfolio, capable of making per-instance decisions, was introduced as a

response to these gaps. Furthermore, the scope, aims, and hypothesis of the study were clearly defined, aligning the technical challenges with the intended solution. The content of this chapter ensures that the reader understands the relevance and significance of the problem and how it will be approached throughout the rest of the thesis. The next chapter will examine existing literature to further support the design and novelty of the proposed model.

Chapter 2 Literature Review by Systematic Search

2.1 Chapter Introduction

This chapter will give an in-depth understanding of the importance of cybersecurity and the impact of any cyber threat. There will be an explanation of the different types of IDS, which are HIDS and NIDS. Also, it will view the different approaches for the IDS to identify the threats with its' advantages and disadvantages. Finally, it will view the different DDOS attack techniques along with the major DDOS attacks that happened in chronological order.

2.2 Cybersecurity

To understand Cyber Security, we need to look into its definition. According to Merriam-webster, Cyber Security is the measure taken to protect a system against an attack and unauthorized access(“Cybersecurity Definition & Meaning - Merriam-Webster,” n.d.). The systems that can be targeted for cybersecurity can be any form of information technology resource. These resources can be either Compute, Network, or applications. Also, cybersecurity covers the integrity, availability, and confidentiality of the data. There are enormous types of security defense tools and techniques used by many originations, and each tool targets a different component of the system to protect. For example, a generic firewall will protect the overall internal IT infrastructure from any external network security threats, while WAF (web application firewall) is application-specific and will protect applications such as Email systems. These roles are unique and can’t be interchanged. Cybersecurity defenses don’t always depend on physical or software modules. Some policies can protect the system, like encryption, authentication, privileges, and segregation (logical or physical). On the other hand, we can have modules that can be hardware or software like firewall, IDS/IPS, security gateway…etc. As technology evolves, with systems being updated every while and new components being introduced, it becomes more difficult to cope with this rapid change and make sure that the system is hardened and safe from cyberthreats. Attackers can exploit and find ways into the systems without being noticed or detected since the people who manage these systems might not be aware of loopholes in the new systems and updates.

Intrusion detection systems have been used in various networks to boost the network security. Their roles are restricted primarily to the detection of any threats to which a server may be subjected. By definition, intrusion detection systems (IDS) refer to systems that are charged with the responsibility of monitoring network traffic(Margaret, 2018). Their activities are also helpful to security management in individual servers. These systems are always on the lookout for any suspicious activities within the servers or network in which they are installed. Therefore, IDSs are responsible not only for the detection of suspicious activities in networks and servers but also for reporting such issues when discovered. Even though their main roles are the detection and reporting of malicious activities, intrusion detection systems may be designed with special features that enable them to take the required courses of action in cases where suspicious activities are detected. Some of such responses include preventative measures like the obscuration of the traffic sent from the detected malicious IP addresses. The operations of IDSs are never completely efficient as they may also be subjected to various forms of interference, such as false security alerts within the networks or servers guided by them. Thus, it is imperative that companies or network administrators perform adequate fine-tuning to their intrusion detection systems before installing them(Margaret, 2018). Fine-tuning, essentially, involves the proper configuration of intrusion detection systems to familiarize them with the server's or the network's normal traffic. This way, it becomes easy for such systems to differentiate between normal activities within the traffic and malicious events within the networks in which they are installed(Margaret, 2018). Therefore, the installation of intrusion detection systems is not efficient enough to guarantee absolute security in a given server or network.

There are four principal categories of intrusion detection systems; they include network intrusion detection systems (NIDS), host intrusion detection systems (HIDS) (Khor et al., 2010), signature-based intrusion detection systems (SIDS), and anomaly-based intrusion detection systems (Saleem Malik Raja and Jeya Kumar, 2014). NIDS are installed at specific points within a network to monitor traffic from various sources within the network. HIDS are often deployed on all computers and other devices in a network and usually have direct access to an enterprise's internal networks and the internet(Tanmoy and Niva, 2017). The main advantages of HIDS over NIDS include the ability to detect malicious actions originating from both the organization and infected hosts (Tanmoy and Niva, 2017). Signature-based intrusion detection systems mainly detect threats by comparing the characteristics of data traffic within a network and comparing them to the properties of known threats. On the other hand, anomaly-based intrusion detection systems often determine the anomalies within the general traffic of a network by comparing them against an established baseline. Thus, the rationale behind the operation of such systems is the ability to report any changes in the network baseline. Such changes are automatically associated with malicious intrusion.

The installation of intrusion detection systems in devices, servers, or networks comes with several benefits that help to enhance cybersecurity. The first and the most essential benefit of these systems is the ability to provide organizations and other forms of network managers with information on the security statuses of their networks or servers. Generally, intrusion and detection systems can be used to not only detect the presence of malicious activities but also to analyze and categorize such incidents. The results of such analyses are useful to organizations as they

dictate the types of actions to be taken by the affected organizations for optimal security. Thus, they act as both network guards and essential factors of decision-making processes aimed at the improvement of network security.

Opposite to the notion that network security systems are caused by data traffic from external sources, there are cases in which such issues originate from within the individual servers connected to a given network. Thus, it is imperative that the system bugs that originate from within these devices are also detected. Intrusion detection systems, especially the HIDS, are renowned for their ability to detect malicious activities of traffic within individual servers. This property enables organizations to identify and remedy faulty network configurations and assess such systems for future risks. By providing organizations with adequate information on regulatory condition, IDSs offers more unobstructed visibility across the networks of such organizations, making it easier for them to conform to the security regulations(DOUGLAS et al., 2015). The final benefit of such systems comes in the form of their ability to boost organizational security response systems. Due to their ability to detect hosts and devices within a network, intrusion detection systems are used for data inspection within network packets and the identification of the operating systems of the services employed. The utilization of IDS to gather such information is considered more efficient as it operates better than manual census (FSabahi and AMovaghar, 2008). Generally, the installation of intrusion detection systems in a network comes with more benefits than just the detection of intrusion; it also enables organizations to effectively control their networks and servers while conforming to the security legislation.

2.2.1 IDS (Intrusion Detection System)

The Intrusion detection system is a system that analyzes and observes the network activity to find intrusions. Intrusion is any attempt that may affect the integrity, availability, and confidentiality of a system or network. Generally, there are two approaches to collecting data for the intrusion detection system. The data can be either collected from the network or the Hosts(Sazzadul Hoque, 2012).

There are different approaches that allow the IDS to detect malicious attacks, which are (anomaly, misuse)

2.2.2 NIDS (Network Intrusion Detection System):

With the increase of intercommunications in either Local area networks or wide area networks, the cyber security threats have increased on the network side, mainly because there is more exposure to the internet and more visible attack points. For this reason, NIDs were introduced to close this gap and secure the systems. NIDS scans network traffic and analyzes any abnormality on the network level. Network traffic can be local between hosts in the local network or communication that leaves the local network to WAN (wide area Network) or the internet. Usually, the networks are not unified, even in a single organization. An organization can have completely separated networks or, logically, separated by VLAN or subnet. It might require installing a firewall/IDS/IPS in each network or segment. Even in the same network segment, multiple IDSs with different roles might be required as there might be various systems and applications.

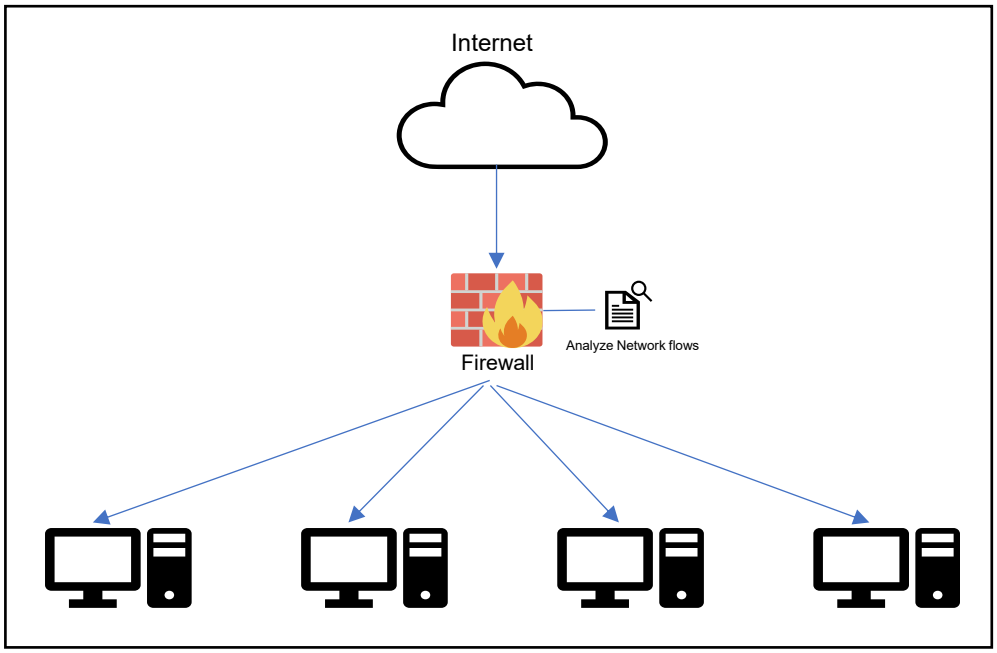


Figure 2 NIDS

2.2.3 HIDS (Host-Based Intrusion Detection System)

The HIDS evaluates the host activity, primarily the logs generated from the operating system or the applications. These logs contain information related to computing utilization, network, memory, health, etc. Usually, the monitoring happens by installing an agent in the host, which will collect the required information and pass it to a centralized server. In a virtual environment or cloud, it can be done on the hypervisor level, where there is no need to install an agent, and it does not require resources on the targeted virtual machine.

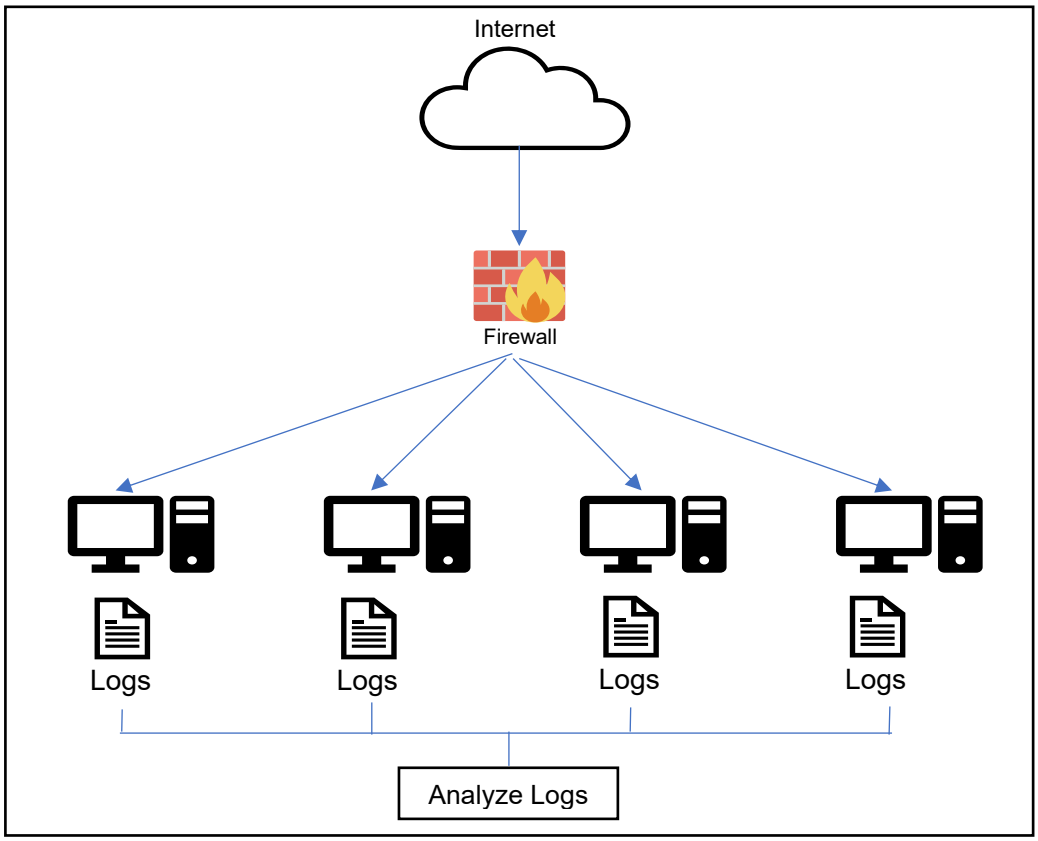


Figure 3 HIDS

2.2.4 IDS approaches

There are different approaches that allow the IDS to detect malicious attacks (anomaly, misuse), which will be discussed as follows.

2.2.4.1 Misuse:

These types of IDS depend on signatures to identify benign flows in the network. After observing the network activity, a knowledge base can be constructed. Depending on this knowledge base, the IDS can trigger an alarm after observing the signatures. Signatures are unique by nature, and the issue with this approach is that the network is dynamic and changing. If the behavior of the system changes due to user activity or an update, the IDS will trigger false alarms. Even on the malicious

side, if there is a small change in the malicious attack or a variation, it will have a unique identity. The IDS might not be able to identify it and will miss a legitimate alarm.

Advantage:

One of the main advantages of the Misuse is that the models are built on known intrusive malicious signatures. Since these signatures are well-defined, the administrator of the systems can easily relate to any alert that exists in the knowledge base. Also, this approach can immediately start protecting your network since the signatures are already installed in the IDS(“Intrusion Detection Systems > Triggering Mechanisms | Cisco Press,” n.d.).

Disadvantages:

- Maintaining the signature database for all types of cyber threats is a very difficult task.
- It is necessary to update the database very frequently to keep the IDS up to date.
- Misuse might have trust issues, as these signatures are usually provided by vendors and suppliers that offer the IDS modules. The vendors can prepare signatures that can be intentionally bypassed without triggering any alarm.(“Intrusion Detection Systems > Triggering Mechanisms | Cisco Press,” n.d.)

2.2.4.2 Anomaly

This approach depends on having a dataset that captures network traffic. This dataset contains variables or features that represent each flow in the network communications. The variables and features are reflections of the network activity, so when observed and analyzed, the IDS can distinguish the benign flow from the malicious flows. This can be done by understanding how far these features deviate from the benign features.

There are multiple approaches that can achieve anomaly detection, which are supervised and unsupervised classification.

Advantages

- Since every network is unique, anomaly systems take advantage of that as it is trained on the target network. Attackers will have difficulty imitating users as the anomaly system keeps user-profile records, and each user in the network is unique, and any deviation from these profiles will trigger an alarm.
- Anomaly is not based on specific traffic that represents known intrusive activity (as in a signature-based IDS). An anomaly detection system can detect zero-day attacks, as the system generates an alarm because it deviates from normal activity, not from a handcrafted malicious signature database.(“Intrusion Detection Systems > Triggering Mechanisms | Cisco Press,” n.d.)

Disadvantages

- The System must be trained on the target network, and during the training time, the network can be targeted for attacks.
- Since the training will be specific to the target network, the administrators will face the complex task of associating events with alarms and triggers. This task can't be recycled as it is very specific to the target network.
- There are no guarantees that the training will actually result in a capable system to detect malicious events. The only way to validate is to simulate attacks and study the triggered events.
- If the attacks have a very similar pattern to the user profile, maybe the system will not detect these attacks.

2.2.4.3 Supervised Models:

The supervised models require input and output data. The input data represents the features and variables. The output data represents the class or type of input data. For the supervised model to work for IDS, the data acquired needs to be labeled for each flow. This might cause an issue for the development of this type of model because these labels are usually handcrafted. Another issue is determining the dimensionality of the features and how much is enough of a subset while maintaining accuracy. The data needs to be split into two parts: one part is training, and the other is validation. There are many types of supervised models with many variations, like support vector machines, decision trees, random forests, and neural networks.

2.2.4.4 Unsupervised:

Unsupervised learning does not require labels (output). These models capture patterns from the features. The model can be provided with a numerical target. From these patterns, the model can make clusters or groups where we can distinguish the benign group from the malicious groups. Examples of unsupervised algorithms are k-nearest neighbors, k-means clustering, and hierarchical clustering.

2.2.5 DDoS attacks (Distributed Denial-of-Service Attack):

Denial of service and distributed denial of service (DoS/DDoS) remain one of the cybersecurity issues that are persistent and keep happening throughout the years. One of the main features of the Internet is the openness and connectivity of diverse networks together. At the same time, the DOS/DDOS takes advantage of the internet diversity and connectivity. The advantage is that the internet is not centralized and distributed all around the world, with many networks that are connected together. Since there happens to be connections from anywhere, the DDOS can orchestrate a DDOS attack from many networks, and it will be difficult to validate all of these connections. In this part, there will be a discussion on DDOS and how it works, along with some history of DDOS.

In order to proceed, we need to understand the concept of DDOS. The DDOS is an orchestrated attempt between multiple computers to initiate an attack at the same time targeting a server or service. This attempt will consume the resources of the service, and when the server is short of resources (compute/memory/Storage), legitimate connections from normal users will not be granted.

To summarize, DDoS is a collection of multiple interconnected devices that can be computers, smartphones, or IoT devices. These devices are infected and can be controlled, which allows 3rd party to control these devices. Once a large number of network devices have been in control, they can be used to orchestrate a DDoS attack. The infected devices are usually called zombies in computer science terminology(Kamboj et al., 2017). Cybercriminals exchange these zombies and can offer them for rent to facilitate the attacks. Zombies can also be called bots. There are different models to make bots, and the first method is the client-server model. All the infected bots are connected to a central server that controls them all. The second approach is to have a peer-to-peer model. These bots don't connect directly to a central server. But they connect with each other, and the control of these bots can be done by digital signatures. The peer-to-peer model is the latest way for the botnets because it is difficult to trace, as each bot is connected to another bot instead of the central server. The way these bots can communicate with each other is by initiating a connection to a random IP address until the other bots reply.

Why Multiple Machines:

A single machine can only generate a small number of requests and bandwidth, but multiple machines can make much more. While having multiple machines, it will be much harder to shut down connections as there will be many connections established at the same time, and it will be stealthier. A simple filter of IP address will not work, and if the service provider considers adding more resources to accommodate the attack, the attacker will simply add more bots until the service fails and the attack succeeds. The scale of the DDoS attacks has advanced in the past years, and the volume of attacks has reached 3.47 terabytes per second in

2022(“Microsoft fends off record-breaking 3.47Tbps DDoS attack | Ars Technica,” n.d.). Some examples of DDoS attacks are as follows:

Yo-Yo attack:

One of the recent types of DDoS attacks is the Yo-Yo attack. The Yo-Yo attacks take advantage of the auto-scaling of the Virtual machines in the cloud. The process of this type of attack is to overwork the virtual machine and invoke autoscaling. The primary intention of the Yo-Yo attack is to cause economic damage to the organization as the organization will allocate more unnecessary resources to keep the virtual Machine running. At the same time, the Yo-Yo attack can also cause performance damage because, during the resource allocation in the autoscaling process, the Virtual Machine will face performance degradation(David and Barr, 2021).

Advanced persistent DoS:

Most DDoS attacks don’t last for a long period as the victim can mitigate and find a solution for the attack. Meanwhile, the persistent attack can last as long as a month. The attackers will have multiple targets and switch between these targets, so the victim can’t mitigate and have the time to resolve these attacks, while at the same time, they keep concentrating on the main target server.

Major Attacks in History:

The table below has the chronological order of the DDoS attack and the increased complexity of the attacks as time passes. The table was retrieved from (İlker and Richard, 2020)

Table 1 Chronological evolution of DDoS attacks(İlker and Richard, 2020)

Date	Description
Pre-1989	Non-computer DOS using sabotage and sit-ins
1989	AIDS ransomware.
1995	German government blocks access to sexual material.
1995	Strano Network DDoS protests French nuclear weapons tests.
1996	Panix ISP in New York disabled by SYN flood attack.
1997	Electronic Disturbance Theater (EDT) uses Floodnet to protest Mexican government attacks on Mayan anarchists.
1998	US DoD DDoS attack on EDT during Ars Technica festival.
1998	LOpht testify to Congress that total Internet disruption is easy.
1999	Electro-hippies use EDT Floodnet to attack WTO.
1999_2000	Trinoo, TFN, TFN2K and Stacheldraht available online.
2000	Mafiaboy takes down Yahoo, Amazon, Dell, ebay, CNN, etc.
2001	Code Red worm DDoS of WhiteHouse.gov.
2001	After Hainan incident, Chinese group launches DDoS on US military sites.
2001	German protesters use Floodnet to attack Lufthansa.
2003	Blaster worm SYN flood of Microsoft update servers.
2003	Blaster worm during blackout of US power grid.
2005	Gpcoder ransomware.
2007	Russian population launches Cyberwar with DDoS on Estonia.
2007	Pro-Putin botnets launch DDoS attacks.
2008	Chinese DDoS attacks on CNN.
2008	DDoS attacks on Georgia sites while Russian military attacks.
2008	Myanmar state uses DDoS to silence dissident Voice of Burma.
2008	DDoS attacks on RFE/RL Tajik, Farsi, Russian, etc. services.

2008	Ukraine attacked by unidentified anti-NATO sources.
2009	Hamas and Israel launch DDoS attacks on each other.
2009	USA and South Korean sites get DDoS attacks.
2009	Kyrgyzstan, Kazakhstan and Iran DDoS silences dissent.
2010	Vietnamese protest of Chinese mining gets DDoS.
2010	Anonymous titstorm attack on RIAA and MPAA.
2010	Anonymous Operation Payback DDoS on payment sites.
2010	Arab Spring leads to Internet blackouts.
2011	Telecomix anti-Internet blackout actions.
2013	Spamhaus receives massive DDoS of 300 Gbps.
2013	Cryptolocker ransomware spread by botnets.
2014	Ransomware (Cryptorbit, Locky, Petya) starts using bitcoin.
2014- 2015	Lizard Squad stresser 579 Gbps DDoS of gaming industry.
2015	DDoS for hire botnets: 25 wired, 8 mobile.
2015	Large parts of the power grid in Ukraine disabled.
2016	Black Lives Matter receives flooding and slow loris attacks.

This chapter will go through a systematic search. The systematic search will consist of all research that relates to IDS that uses machine learning and Artificial intelligence. A table of the systematic search will be reviewed with all the researched regardless of the type of the dataset and the approach used. A detailed review of some research will be done into three categories, which are Hybrid Models, Pre-processing, and Deep Learning.

The search is primarily done in IEEE, Elsevier, and ACM. The search was broadened by using the library site and Google Scholar. Search Keys that were used in the systematic search (Multiclass, multi classifier) with the assessment terms (classification, model, algorithm, machine learning, AI) and then industry terms (IDS, Cybersecurity, DDoS) and other related terms (KDDCUP, CIC-IDS-2018).

We have used different search key strings that are as follows:

Table 2 Systematic Search Key Strings

step 1		step 2		step 3		step 4
Multiclass	AND/OR	classification	AND/OR	IDS	AND/OR	KDDCUP
multi classifier		model		Cybersecurity		CIC-IDS-2018
		algorithm		DDoS		
		machine learning				
		AI				

We have used different permutations and concatenation of the above table. The search period varied based on the time point of the research, as there were some updates during the research period. The presented research was done on points of time; the first one focused on research done between 2020 and 2021. The second

search was focused on the time period (2020 to 2023). The criteria for the selected papers are published, and it can be peer-reviewed a conference, or a survey.

2.3 Literature Review and Systematic Content-Analysis

2.3.1 Search (2020 to 2021)

Table 3 systematic search

Reference	Title	Year	Type	Author	Dataset	Methodology	Results
(Arivardhini et al., 2020)	A Hybrid Classifier Approach for Network Intrusion Detection	2020	Conference	Arivardhini, S Alamelu, L M Deepika, S	NSL KDD data	Support Vector Machine (SVM), decision tree (J48), and Naive Bayes (NB) Use a Majority Voting scheme	No Results were shown
(Chen et al., 2020)	A Novel Preprocessing Methodology for DNN-Based Intrusion Detection	2020	Conference	Chen, Peng Guo, Yunfei Zhang, Jianpeng Wang, Yawen Hu, Hongchao	KDDCup'99	DNN Deep Neural Network Pre-processing : 1- Numeralization 2- Transformation: 3- Numeralization(Min-Max)	improve accuracy, recall, and F1 score, with no significant degradation in precision
(Bharati and Tamane, 2020)	NIDS-Network Intrusion Detection System Based on Deep and Machine Learning Frameworks with CICIDS2018 using Cloud Computing	2020	Conference	Bharati, M P Tamane, S	CIC-IDS-2018	Extra-Tree GBoost Tree Random Forest Tree MLP	The results compare different Models and their results.
(Haghighat and Li, 2021)	Intrusion detection system using voting-based neural network	2021	Journal Article	Haghighat, MH Li, J	KDDCUP'99 CTU-13	novel voting-based deep learning framework - Deep Neural Network - Recurrent Neural Network (RNN),	The voting system has shown better results, where it reduces false alarms.

						- Convolutional Neural Network - Boltzmann Machine - Stacked Auto-Encoder	
(Syarif et al., 2020)	Feature Selection Algorithm For Intrusion Detection Using Cuckoo Search Algorithm	2020	Conference	Syarif, I Afandi, R F Astika Saputra, F	Botnet ISCX 2017 KDDCUP'99 NSL-KDDCUP	Cuckoo Search (CS) as feature selection and compare with GA and PSO using Decision Tree	The CS has reduced most of the features, but it only has shown better performance in ISCX 2017, while PSO had better accuracy in NSL KDDcup99 and NSL-KDDCUP.
(Atefi et al., 2020)	A Hybrid Anomaly Classification with Deep Learning (DL) and Binary Algorithms (BA) as Optimizer in the Intrusion Detection System (IDS)	2020	Conference	Atefi, K Hashim, H Khodadadi, T	"CICIDS 2017	A hybrid Anomaly Classification of IDS with Deep Learning (DL) and Binary Algorithms (BA) as	The author did not explain how the hybrid models work, but he compared different hybrid couples and showed that DNN+BGSA has increased accuracy.
(Kishore and Chauhan, 2020)	Evaluation of Deep Neural Networks for Advanced Intrusion Detection Systems	2020	Conference	Kishore, R Chauhan, A	KDDCUP'99'	DNN and compared it to linear regression, Naive Bayes, K nearest neighbor, Decision tree, and Adaboost.	In this paper, it was shown that SNN performs better than other Models, but The Author did not show a confusion matrix where we can compare FP, TP, TN, and FN.
(Abdul Lateef et al., 2020)	Hybrid Intrusion Detection System Based on Deep Learning	2020	Conference	Abdul Lateef, A A Faraj Al-Janabi, S T Al-Khateeb, B	KDD'99	binary class IDS based on RNNs The Crow Swarm Optimization (CSO) algorithm has been used to reduce the dataset features.	The author has made multiple trails for feature reduction using CSO and trained on RNN with each trail. It was shown that only three features are sufficient to make a classification, but the paper did not show any

							results that identify false alarms.

2.3.2 Search (2020 to 2023)

This Systematic search was focused on the CIC-IDS-2018 dataset and models that used multiclassification with this dataset for IDS, and the search period was from 2020 to 2023.

Table 4 Systematic Search Table (2020-2023)

Row Labels	(Kabir et al.,	(Seth et al.,	(Hua, 2020)	(Dini et al.,	(Handika et	(Fitni and	(Alkanjr and	(Shahbanday	(Das et al.,	(Yoo et al.,	(Siddiqi and	(Chimphlee et	(Hagar et al.,
Benign	0.99	0.92	x	x	x	x	x	x	x	x	x	0.99	1.00
Bot	1.00	1.00	x	x	x	x	x	x	x	x	x	1.00	0.83
Brute Force -Web	x	x	x	x	x	x	x	x	x	x	x	0.00	0.75
Brute Force -XSS	x	x	x	x	x	x	x	x	x	x	x	0.00	1.00
BruteForce	1.00	0.99	x	x	x	x	x	x	x	x	x	x	x
DDoS	0.99	1.00	x	x	x	x	x	x	x	x	x	x	x
DDOS -HOIC	x	x	x	x	x	x	x	x	x	x	x	1.00	1.00
DDOS -LOIC-UDP	x	x	x	x	x	x	x	x	x	x	x	0.72	1.00
DDoS attacks-LOIC-HTTP	x	x	x	x	x	x	x	x	x	x	x	0.99	1.00
DoS	0.99	0.98	x	x	x	x	x	x	x	x	x	x	x
DoS attacks-GoldenEye	x	x	x	x	x	x	x	x	x	x	x	0.99	1.00
DoS attacks-Hulk	x	x	x	x	x	x	x	x	x	x	x	0.96	1.00
DoS attacks-SlowHTTPTest	x	x	x	x	x	x	x	x	x	x	x	0.75	1.00
DoS attacks-Slowloris	x	x	x	x	x	x	x	x	x	x	x	0.95	1.00
FTP-BruteForce	x	x	x	x	x	x	x	x	x	x	x	0.71	0.88
Infiltration	0.96	0.97	x	x	x	x	x	x	x	x	x	0.44	1.00
Overall	x	x	0.98	94.39	96.00	98.80	1.00	0.78	99.00	0.89	0.98	x	x
SQL Injection	0.87	x	x	x	x	x	x	x	x	x	x	0.00	0.92

SSH-Bruteforce	x	x	x	x	x	x	x	x	x	x	x	0.99	1.00
Web attacks	x	1.00	x	x	x	x	x	x	x	x	x	x	x

2.4 Hybrid Models:

Many researchers used the approach of having multiple algorithms, but they have taken different techniques to make them interact or merge the results. The following will discuss different strategies used in multiple papers mentioned in the table above.

2.4.1 A Hybrid Classifier Approach for Network Intrusion Detection

We have the first review for the paper “A Hybrid Classifier Approach for Network Intrusion Detection” (Arivardhini et al., 2020). The author has proposed to use multiple classifiers. Each classifier will classify the flow to determine if the flow is considered an anomaly or benign. We can see the model proposed per the below algorithm.

algorithm 1 A Hybrid Classifier Approach for Network Intrusion Detection(Arivardhini et al., 2020)

Procedure model ( )

Input = NSL KDD data set

Reduce 'n' features to 'm’ based on number of proposed filters.

Use Majority Voting scheme

Deploy a hybrid model consisting of J48, SVM, Naive Bayes.

Propose the model ‘M’ for every feature F n

Provide F n to J48, SVM, Naive Bayes using NSL KDD data set.

Calculate

- A1 = J48 model accuracy
- A2 = SVM model accuracy
- A3 = Naive Bayes model accuracy

- E = Ensemble Representing J48, SVM, Naive Bayes

Compare the accuracy of A1, A2, A3, E

Select the model which has the highest accuracy M = E

The proposed model will select the correct model based on accuracy, and the author claims that the voting system has improved performance. Unfortunately, no results were presented in the paper. It is a reasonable approach, but there are different issues, which are as follows:

1- The voting system is based on the model's accuracy, but the author did not mention different accuracy for different classes.
2- How will the model be able to produce accuracy for a single new instance? It seems that the hybrid model will vote for a single model for the whole Dataset.
3- There are no results to evaluate or to compare.

2.4.2 Intrusion detection system using voting-based neural network

The paper(Haghighat and Li, 2021) below has a similar approach to the previous one. The author proposes to have multiple classifiers, and they all will be trained and validated. Then, the author proposed to have a mechanism to choose algorithms that are eligible to enter the voting process. The voting process goes as follows:

- All trained models will have an uncertainty factor on the output layer. This factor can be used as a measure to let the model participate in the voting.
- All model’s uncertainty will be sorted, and a threshold will be used to eliminate unfitted models.

Then, voting will be used to determine the results, as shown in the diagram below.

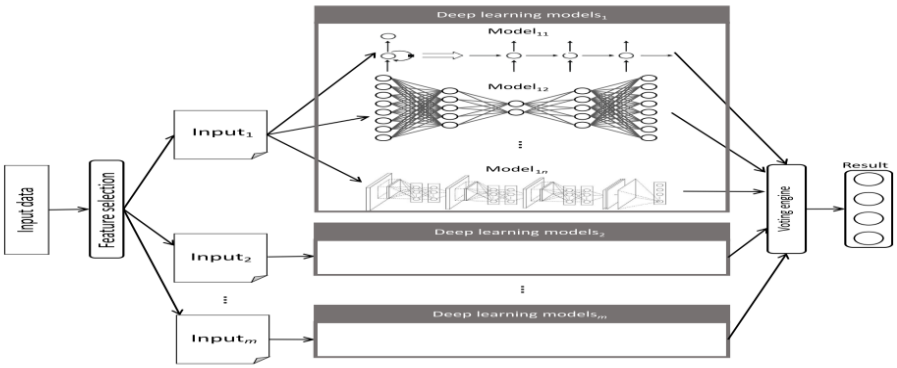


Figure 4 Intrusion detection system using a voting-based neural network (Haghighat and Li, 2021)

This is a summary of the paper, and there are more details in the algorithm. But in general, the same procedure was used in two datasets, which are NSL-KDD and CTU-13.

The results presented in the paper were in two forms (binary and multiple classes). In general, the results in the binary were much better than the multiple classes, and it was expected. As per (Personnaz et al., 1990), using binary states instead of multilayer networks is much more powerful, and using binary states can have better results and less training time. The result is presented in the paper for NSL-KDD.

Binary Results:

Table 5 Binary Results for (Intrusion Detection System using Voting-based Neural Network)(Haghighat and Li, 2021)

		Predicted		
		Normal	Malicious	Total
Actual	Normal	301 031	203	301 234
	Malicious	488	188 121	188 609
	Total	301 519	188 324	489 843

Multiclass:

Table 6 Multiclass Results for (Intrusion Detection System using Voting-based Neural Network)(Haghighat and Li, 2021)

		Predicted					
		Normal	DoS	R2L	U2R	Probing	Total
Actual	Normal	277 269	219	20 608	0	0	298 096
	DoS	490	188 107	5	7	0	188 609
	R2L	0	62	3060	0	0	3122
	U2R	0	1	0	0	0	1
	Probing	0	14	1	0	0	15
	Total	277 759	188 403	23 674	7	0	489 843

False Negative/Positive Rates:

Table 7 False Negative/Positive Rates for (Intrusion Detection System using a Voting-based Neural Network)(Haghighat and Li, 2021)

	FPR	FNR	Accuracy	Precision	Recall	F Score
Binary	0.0011	0.0016	0.9986	0.9993	0.9984	0.9989
Multiclass	0.0982	0.0021	0.9563	0.9302	0.9979	0.9628

2.4.3 A Hybrid Anomaly Classification with Deep Learning (DL) and Binary Algorithms (BA) as Optimizer in the Intrusion Detection System (IDS)

This paper(Atefi et al., 2020) below proposes to use DNN with Binary algorithms for IDS using the CICIDS 2017 dataset. This proposal uses DNN as the primary classifier and then uses the Binary Bat algorithm (BBA), Binary Genetic Algorithm (BGA), and Binary Gravitational Search Algorithm as optimizers. The author has tested DNN with every mentioned Binary algorithm independently. DNN was coupled either with BBA, BGA, or BGSA. The way each test for each couple was conducted as illustrated below:

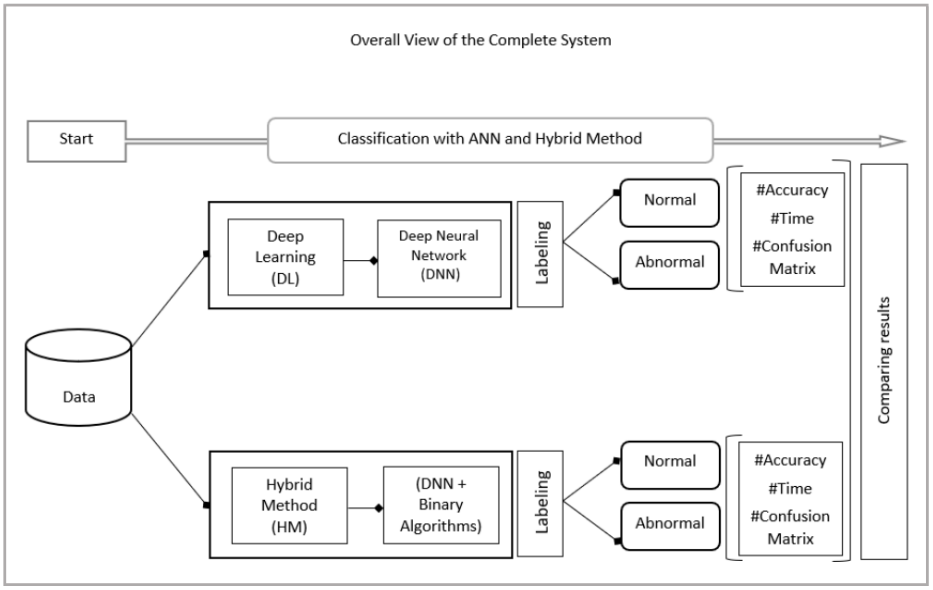


Figure 5 overall view of the complete system(Atefi et al., 2020)

As seen in the diagram above. There are two models presented. One of them is DDN, and the other is one of the Binary algorithms. Both will process the data and classification, and based on the accuracy, Time, and Confusion matrix, the result will be computed from the output of both models.

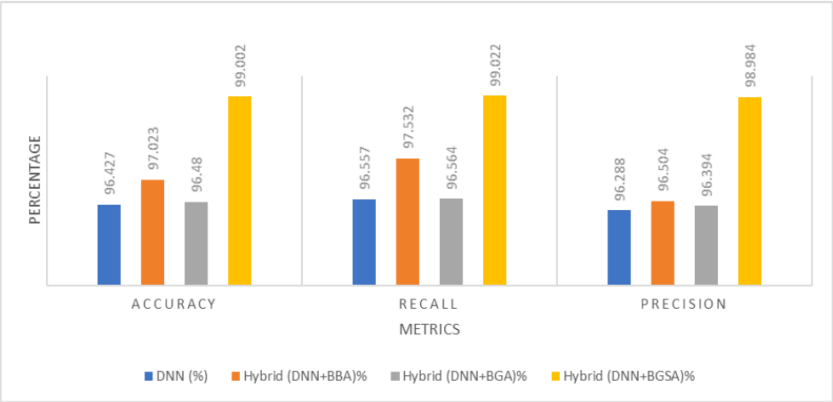


Figure 6 Comparative Results(Atefi et al., 2020)

It is clearly visible from the graph that the Model DNN+BGSA has performed significantly better than the rest of the models in accuracy, recall, and precision.

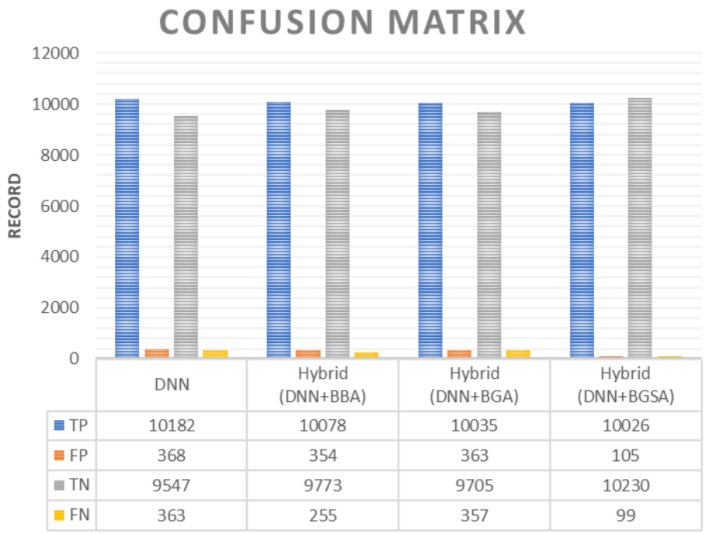


Figure 7 Confusion Matrix(Atefi et al., 2020)

From the confusion matrix, again, we can see that DNN+BGSA performed better than the rest of the models in all categories (TP, FP, TN, and FN)

The results look promising, but the author did not show how exactly the results of the DDN and binary algorithms will be joined for the final results. There is no discussion of whether the results will be based on the voting system or explicitly use

the accuracy and confusion matrix to get the results for classification. Also, this seems to work on offline data, where you can generate and model and select the binary model with a confusion matrix and accuracy. How will this work on a live system where the IDS receives network flows almost every second?

2.5 Pre-processing and feature reduction techniques:

This part will focus on papers that have an emphasis on pre-processing and feature reduction. This part is mandatory for research as most of the Dataset related to IDS has a very high dimensionality. For example, the KDDCUP has 42 features, while the CICIDS2018 has 80 features. Computing that many features in any model may cause a lot of issues. Most of the IDS datasets are considered big data, and the computing power required to create a model is massive. Some algorithms refuse to take Dataset with very high dimensionality, such as the default random forest package in RStudio. In a much worse scenario, the algorithm will start, and after a very long time of waiting, it will fail because of limited resources. This will definitely cause issues with many algorithms and compute resources. Finally, the imbalance of the Dataset. The datasets will definitely be imbalanced, and some pre-processing is required in order for the algorithms to output reasonable results.

2.5.1 Hybrid Intrusion Detection System Based on Deep Learning

The paper(Abdul Lateef et al., 2020) below proposed a model that utilizes an algorithm called CSO (CROW SWARM OPTIMIZATION ALGORITHM), which will act to reduce the number of features in the Dataset. Then, RNN will be used to calculate the accuracy to either accept these features or not. In this research, the author has opted to use the popular Dataset for IDS, which is KDDCUP. The model proposed can be well presented in the diagram below.

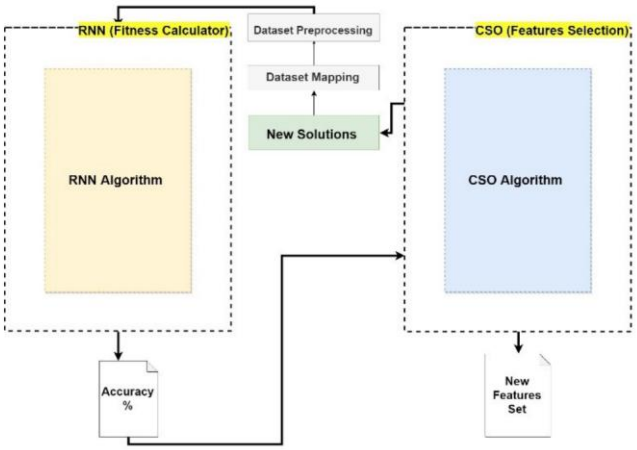


Figure 8 Overall view of the model process(Abdul Lateef et al., 2020)

As we can see, the model will have the following steps based on the diagram above.

1- New Dataset as input
2- Mapping and pre-processing
3- RNN will compute the Dataset and generate mode features accuracy as output
4- CSO will output the current features and halt, or it will propose a new feature set that will loop back to step 1 (new Dataset input)

The author has tested this method with 20 trials, and the optimal number of features selected was 3 with a CSO accuracy of 96.2720% and (RNN-selected features accuracy) of 98.34%. Luckily, the author shared a comparison table using a similar approach for feature reduction with different algorithms.

Table 8 Accuracy for each model(Abdul Lateef et al., 2020)

Method Name	Number of Features	Accuracy
PSO + AUC	12	94.49
SSO + RS	6	93.60
CFA+ DT	10	92.05
ACO	8	98.90
ACO + SVM	14	98.00
PSO + RF	6	98.00
ABC + KNN	7	98.90
IBWOA	5	97.89
Firefly + BN	10	99.95
FGLCC + CFA + DT	10	95.03
CSO +RNN (this paper)	3	98.34

It is true that the author was able to reduce the number of features to as low as three features, but the aim is to reduce the features while maintaining accuracy. Also, the author has only shown the general accuracy of the model. In reality, the datasets will have mostly benign flows, and if the model predicts all the benign flows correctly, it will definitely have a high accuracy. But that does suffice the aim of IDS, where it has detected flows with anomalies, and naturally, they have a small percentage of the overall Dataset. A confusion matrix will be more suitable for evaluating and finding the rate of error for each class of anomaly.

2.5.2 A Novel Preprocessing Methodology for DNN-Based Intrusion Detection

In the paper(Chen et al., 2020) below, the author proposes some pre-processing techniques that can enhance the performance of DNN models on the KDDCup dataset. The author claims that these techniques increase the F1, accuracy, and recall in his/her experiments.

The pre-processing methods used in this paper are as follows:

- Enumeration:

  There are some features in the KDDCUP dataset that are factors or symbolic. These features, such as (service, protocol, and type) may cause issues in DNN models and need to be modified. The author suggests having these features mapped to numerical values.

- Transformation:

  There are some features when compared to each other; we can see that there is a significant gap between them. For example, comparing duration [0, 58329] and src_bytes [0, 1379963888], we can notice the gap between these two features. The solution is to manipulate the values mathematically using $y = \lg(x + 1)$

- Normalization:

  Finally, the Min-Max method is used to normalize the Dataset.

The author did the experiment on multiple algorithms, and he/she has benchmarked the performance between raw Dataset and pre-processed Dataset for KDDCUP

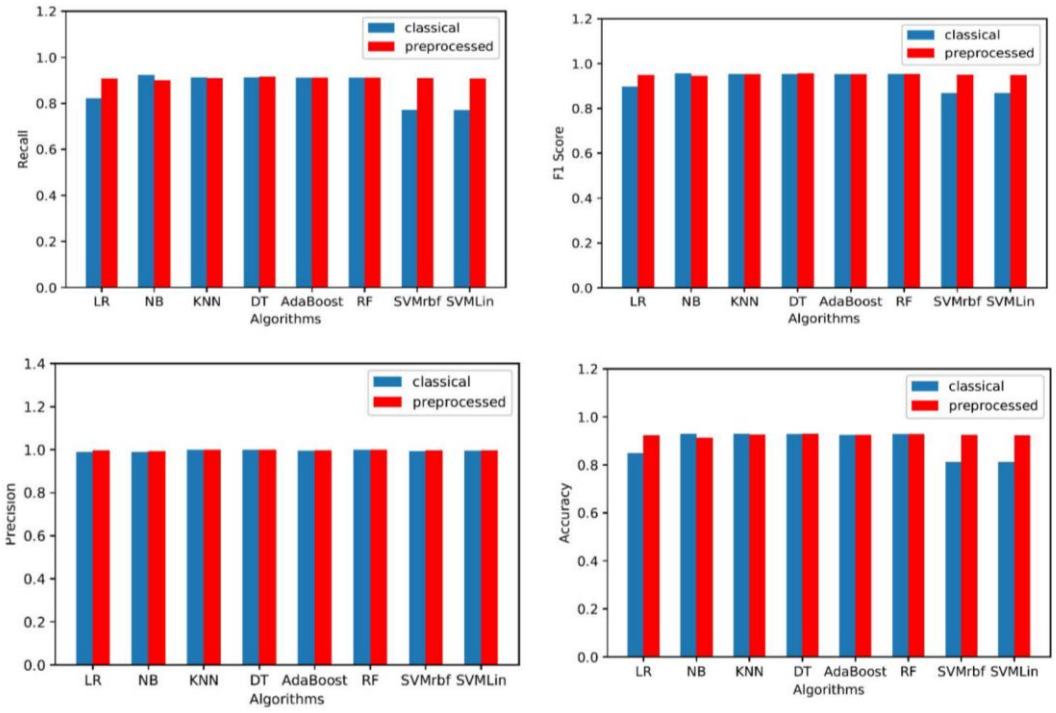


Figure 9 F1, Accuracy, Precision, and Recall(Chen et al., 2020)

As we can see from the graph, the preprocessing has increased the performance in recall, F1, and accuracy. The issue is that the author has calculated the general accuracy of the model but did not show the accuracy for each class, and then we can decide if it has made any enhancements in FP or FN.

Also, the enumeration of factor or symbolic data might cause issues . Some data, if enumerated, may cause wrong assumptions. For example, if we assume TCP=1, UDP=2, then we can make a comparison (1<2). But we can't say that TCP<UDP. I suppose these features might require different ways to be modified.

2.5.3 Feature Selection Algorithm For Intrusion Detection Using Cuckoo Search Algorithm

In the paper(Syarif et al., 2020) below, the author aims at feature reduction to facilitate building models with a very high dimensionality. Datasets with a very high dimensionality might be very difficult to process and may consume a lot of time and process power to model. The idea is to select features that have more importance to make a decision and remove features that are irrelevant.

To achieve this task, the author suggested using a coco search algorithm and building a decision tree model. In this paper, multiple tests were made with different datasets and different feature selection algorithms.

Table 9 Accuracy before and after reduction(Syarif et al., 2020)

Name of Dataset	Number of Features				Training Time				Accuracy			
	Full Data	GA	PSO	CS	Full Data	GA	PSO	CS	Full Data	GA	PSO	CS
Botnet ISCX 2017	79	15	21	11	138s	80s	98 s	19s	99,98%	99,85%	99,85%	99,98%
KDDCup '99	41	17	19	13	524 s	200s	198 s	106 s	99,96%	99,93%	99,95%	99,94%
NSL-KDD	41	15	12	9	227 s	63s	62 s	34 s	99,78%	99,69%	99,79%	99,60%

From the table above, we can observe the CS algorithm is able to reduce more features than GA and PSO. For that reason, building a decision with CS reduction took much less time to train. On the other hand, the CS was only able to perform better in Botnet ISCX Dataset in accuracy than the other algorithm, while PSO exceeded the other in accuracy with KDDCup and NSL-KDD datasets.

It is true the proposed features reduction algorithm (CS) was able to outperform the others, but the performance in accuracy did not meet expectations. The main target of the IDS is to provide true alarms to facilitate operations in IT departments and not to be overwhelmed with false alarms that have to be investigated.

2.6 Deep Neural Networks

This part will focus on research that only covers deep learning techniques on IDS. There are multiple papers that have shown interest in this type of algorithm.

2.6.1 Evaluation of Deep Neural Networks for Advanced Intrusion Detection Systems

In the paper(Kishore and Chauhan, 2020) below, the author tried to test DNN with KDDCUP for IDS. Multiple DNN models were produced, and they were compared with other models, which are:

1- Linear Regression
2- Naïve Bayes
3- KNN
4- Decision Trees
5- AdaBoost
6- Random Forest
7- SVM-Rbf
8- SVM-Linear

A benchmark comparison between these models so we can compare.

Table 10 Benchmark table with Accuracy, precision, Recall, and F1(Kishore and Chauhan, 2020)

Algorithms	Accuracy	Precision	Recall	F1-Score
Lin-Regression	0.849	0.988	0.822	0.896
Naive Bayes	0.928	0.987	0.924	0.953
KNN	0.928	0.997	0.914	0.953
Decision Tree	0.927	0.998	0.911	0.952
Adaboost	0.926	0.996	0.912	0.950
Random Forest	0.927	0.998	0.911	0.951
SVM-Rbf	0.811	0.922	0.772	0.868
SVM-Linear	0.813	0.993	0.771	0.867
*DNN-1	0.928	0.997	0.914	0.953
DNN-2	0.929	0.996	0.913	0.955
DNN-3	0.930	0.996	0.916	0.956
DNN-4	0.928	0.998	0.914	0.953
DNN-5	0.927	0.999	0.912	0.954

As we can observe, there are five tests on DNN (1-5), and we can see that DNN-3 achieved the best accuracy. DNN-5 gets the best precision, Naïve Bayes the best recall, and DNN-2 the best F1. From a personal perspective, these results do not show any real advantage of DNN compared to the other algorithms because the increased performance is scattered between different models in DNN. Also, the comparison is not sufficient and not deep enough to clearly state that DNN models perform better than the others.

2.6.2 NIDS-Network Intrusion Detection System Based on Deep and Machine Learning Frameworks with CICIDS2018 using Cloud Computing

The paper(Bharati and Tamane, 2020) below compares different algorithms using the CICIDS2018 Dataset. The author's focus is that the Dataset is modern compared to the KDDCUP dataset, and it contains the latest and cutting-edge threats. For this purpose, the author made multiple tests with different algorithms, which are:

- Extra-Tree
- GBoost Tree

- Random Forest Tree
- MLP

For these tests with different algorithms coupled with the CICIDS2018 Dataset, a comparison table is produced below:

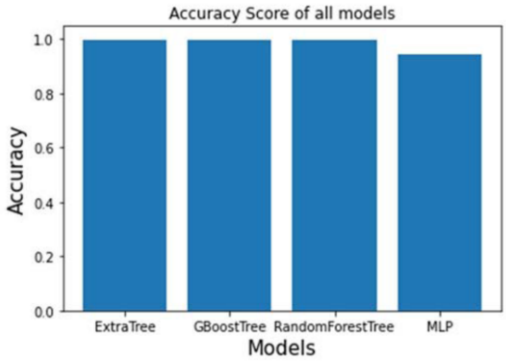


Figure 10 Accuracy for the models(Bharati and Tamane, 2020)

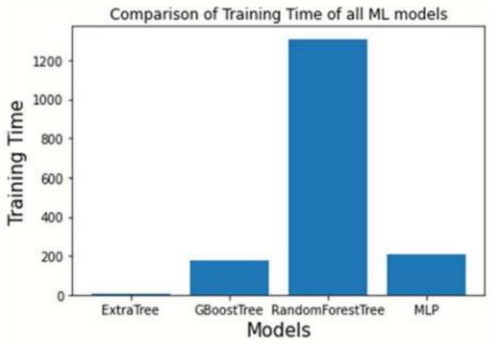


Figure 11 Training Time(Bharati and Tamane, 2020)

From the results above, we can see the models have a very comparable accuracy while only the MLP has slightly less accuracy. On the other hand, the random forest was the slowest to train, and it took much more time. There are many factors to consider in the training time for each model, such as optimization for big data, parallelism (multi-threading, clustering), and time and space complexity. There are different implementations in different environments, which will definitely affect the training time.

2.7 Discussion

This chapter provided a comprehensive review of the key studies and techniques used in the design and implementation of intrusion detection systems using machine learning. Throughout the review, it became evident that while many approaches have reported good results in specific use cases, most models failed to perform consistently across different traffic types and datasets. A significant observation was the dominance of binary classification methods in past research, where the focus is often limited to distinguishing between benign and malicious traffic without further granularity. This approach lacks the depth needed to deal with real-world scenarios, which require detection at the level of specific attack types. The review also noted that hybrid and ensemble techniques often achieved higher performance, but these were rarely modular or adaptive. These insights further support the idea of adopting a per-instance model selection strategy, where each instance is dynamically evaluated to determine the most effective classifier based on its specific characteristics.

2.8 Chapter Conclusion

After surveying multiple recent papers, we can see some trends or popular techniques that can be found in most of them. Even though the KDDCUP is considered old and does not represent current cybersecurity threats, it is still used in recent research for IDS and test models. The reason could be that many researchers were already using the Dataset, and new studies needed to compare and benchmark with older research. Another reason is that the Dataset is already converted to flows and ready for machine learning algorithms. Another trend that we can see is the use of deep learning algorithms, especially DNN. Many researchers

introduced DNN for IDS with different flavors. Many of these papers have shown increased performance in accuracy and precision. Dealing with a very high dimensionality dataset could be an obstacle to performing the tests and proceeding with research. For that reason, some researchers suggested dimensionality reduction with different approaches and algorithms that can ease and facilitate machine learning. Finally, we have the hybrid models, where multiple models will work together to classify a flow as a benign or a threat. In this research, the adopted technique would be a “hybrid classifier”. It is inspired by “A Hybrid Classifier Approach for Network Intrusion Detection” (Arivardhini et al., 2020) The voting system will be used after the selection of the classifiers that will be determined by the Master classifier. In addition, the voting criteria may be amended and not necessarily follows the same condition.

Chapter 3 Dataset Scoping:

3.1 Chapter Introduction

This chapter will have a survey about the datasets that are being used in the research field. It will show how that many of the datasets are derived from the KDDCUP dataset. Even though it's widely used, the chapter will show the shortcomings of the dataset and why it should not be used for IDS. The chapter will have some studies that show the inconsistencies and the other issues with the dataset. In the end, the chapter will have the reasoning for choosing CSE-CIC-IDS2018. More details about CSE-CIC-IDS2018 are discussed in Chapter 6. The further details cover the features, type of attacks, and statistical analysis of the dataset.

3.2 Sample of major Datasets

Identifying Cybersecurity threats in the network traffic has become more and more complex as the cyber attacks have become more advanced. Data has become a tool that facilitates cyber threats, and the number of attacks has increased in recent years. Many organizations have an Intrusion Detection System(IDS), but the data that powers the IDS are outdated and can't keep up with the rapid change in cyber attacks. In this section, we will discuss the current datasets that are used in research and the problems that may occur with these datasets. Then, we will view the limitations of the classes that are present in these datasets. The nature of these datasets is that it has a huge gap in classes (attack types) and imbalance, which may cause a problem in machine learning classification and detection.

In 1999, the KDDCUP99("KDD Cup 1999 Data," n.d.) was introduced, and since that time, it has been considered the de facto IDS machine learning dataset, and it was mainly used for anomaly detection. The data is based on the DARPA98 dataset(Lippmann et al., 2000), and it was prepared by Stolfo(Stolfo et al., 2000). The data represents seven weeks of network traffic with 5 million records. The dataset contains 41 features for each record, and every record is labeled either benign or part of the following groups:

1) Denial of Service Attack (DoS)
2) User to Root Attack (U2R)
3) Remote to Local Attack (R2L)
4) Probing Attack

KDDCUP has some problems, and these problems are the following(Tavallaee et al., 2015):

- The data was synthesized to preserve privacy
- Traffic collectors were overloaded, and there were packet drops
- No exact definition of the attacks

The CSE-CIC-IDS2018 Dataset was chosen because the data is recent, and there are many different cybersecurity attacks that are considered modern and have a true representation of modern networks. The CSE-CIC-IDS2018 data is a result of simulated attacks that took several days and are captured as PCAPs. Then, the data is converted to CSV format with labels. In addition, the CIC-IDS-2018 has a wider range of attacks than the KDDcup, which is limited. These attacks cover various forms, such as denial of service, distributed denial of service, Brute force, Botnet, Web attack, XSS, Infiltration, and SQL injection(Thakkar and Lohiya, 2020). These attacks are captured from real network traffic with 80 features.

3.3 Dataset Problem

The problem with the current research is that most of them are dependent on the kddcup dataset. According to many critiques, the dataset is outdated and has many inconsistencies and redundancies that can skew the results. Sabhnani suggests that the dataset has deficiencies and limitations and should not be used in machine learning (Sabhnani and Serpen, 2004). Hindi, who conducted a survey on many types of research done on IDS from 2008 to 2018, found that most of the research depended on the KDD CUP dataset or forked versions(Hindy et al., 2018).

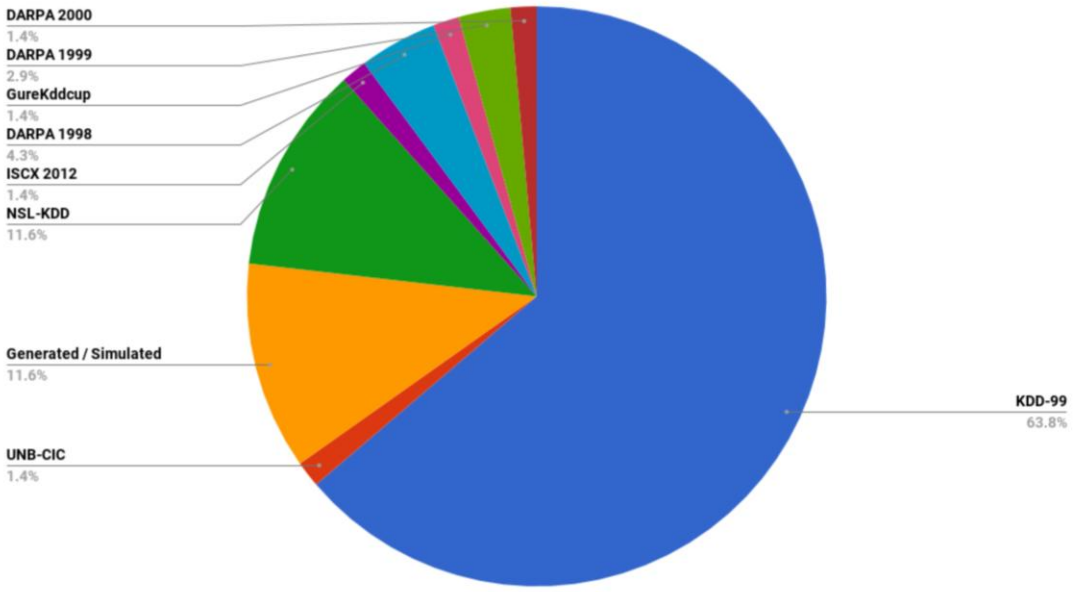


Figure 12 Dataset Distribution (From Hindi)

A similar survey was conducted by myself for 2019-2020, and I found out that the same dataset is heavily used (table systematic search).

3.4 Limited Classes

As a result of using the same dataset by many researchers, most of the classifications are done on the same group of threats (Dos, R2L, U2R, Probe). These groups of threats do not reflect current network cybersecurity threats. For illustration, see the pie chart below provided by (Hindy et al., 2018).

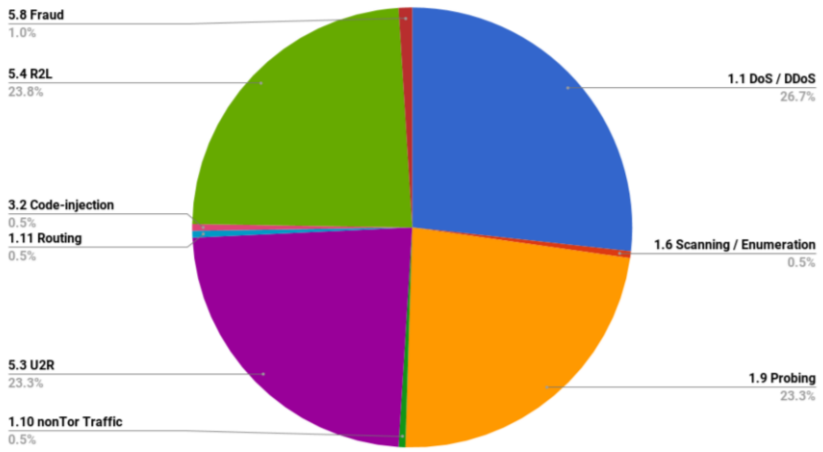


Figure 13 Covered Attacks from 2008 to 2018

3.5 Inconsistent accuracy for different classes

Another reason to establish this research is that most of the researchers use similar schema, and most of them fail to present the accuracy of the classifier for each type of attack (Dos, R2L, U2R, Probe). Some of the researchers have shown the results for each class of intrusion, but the results show the contrast of accuracies of each class. A survey was done by Urvashi where the researcher made a comparison between 20 classifiers based on True-positive and false-positive(Urvashi and Jain, 2015). As seen in the table below, most of the classifiers fail to achieve consistent results for all categories.

Table 11 Classifiers Comparison from Urvashi

Seq.	Classifier	Metric	DoS	Probe	U2R	R2L	Training Set Size
1	K-Means (Qiang W.V.,2004)	TP	87.6	97.3	29.8	6.4	2,776
		FP	2.6	0.4	0.4	0.1	
2	NEA (Maheshkumar S., 2002)	TP	96.7	72.4	22.3	7.8	1,074,991
		FP	0.8	0.2	0.1	0.6	

Seq.	Classifier	Metric	DoS	Probe	U2R	R2L	Training Set Size
3	FCC (Qiang W.V.,2004)	TP	91.6	77.8	12.7	27.8	2,776
		FP	**0.03**	0.023	0.13	0	
4	ID3 (Amanpreet C., 2011)	TP	74.4	57.14	20	6.25	145,586
		FP	1.71	2.5	3.1	1.1	
5	J48 (Huy A.N., 2008)	TP	96.8	75.2	12.2	0.1	49,596
		FP	1	0.2	0.1	0.5	
6	PART (Mohammed M.M., 2009)	TP	97.0	80.8	1.8	4.6	444,458
		FP	0.7	0.3	0.5	0.01	
7	NBTree (Huy A.N., 2008)	TP	97.4	73.3	1.2	0.1	49,596
		FP	1.2	1.1	0.1	0.5	
8	SVM (Huy A.N., 2008)	TP	96.8	70.1	15.7	2.2	49,596
		FP	1.11	0.5	**0.01**	**0**	
9	Fuzzy logic (Shanmugaradtru R, 2011)	TP	94.8	**98.4**	69.6	92.1	54,226
		FP	5.5	1.8	6.7	10.7	
10	naïve Bayes (Huy A.N.,2008)	TP	79.2	94.8	12.2	0.1	49,596
		FP	1.7	13.3	0.9	0.3	
11	BayesNet (Huy A.N., 2008)	TP	94.6	83.8	30.3	5.2	49,596
		FP	0.2	0.13	0.3	0.6	
12	Decision Table (Yeung d.Y., 2002)	TP	97.0	57.6	32.8	0.3	15,919
		FP	10.7	0.4	0.3	0.1	
13	Random Forest classifier (Yeung D.Y., 2002)	TP	**99.2**	98.2	86.2	54.0	15,919
		FP	0.05	**0.01**	0.02	0.09	
14	Jrip (Huy A.N., 2008)	TP	97.4	83.8	12.8	0.1	49,596
		FP	0.3	0.1	0.1	0.4	
15	OneR (Huy A.N.,2008)	TP	94.2	12.9	10.7	10.7	49,596
		FP	6.8	0.1	2	0.1	
16	MLP (Huy A.N.,2008)	TP	96.9	74.3	20.1	0.3	49,596
		FP	1.4	0.1	0.1	0.5	
17	SOM (Huy A.N.,2008)	TP	96.4	74.3	13.3	0.1	49,596
		FP	0.8	0.3	0.1	0.4	
18	GAU (Maheshkumar S., 2002)	TP	82.4	90.2	22.8	9.6	1,074,991
		FP	0.9	11.3	0.05	0.1	
19	MARS (Sriniras M.,2002)	TP	94.7	92.32	**99.7**	**99.5**	11,982

Seq.	Classifier	Metric	DoS	Probe	U2R	R2L	Training Set Size
		FP	8.9	12.2	22.4	17.9	
20	Apriori (Mohammed M.M.,2009)	TP	87.9	76.23	12.3	30.6	444,458
		FP	0.67	1.7	8.9	23.8	

As seen in the table above, most of the classifiers fail to predict U2R and R2L accurately. Even if some of them have high accuracy in all classes, they produce very high False-Positive rates. As an example, for that scenario, the MARS classifier has 99.7 for U2R, but the false positive is 22.4. Not being able to predict all classes accurately or having high false positives can lead to unreliable results.

For these reasons, research and investigation should be conducted. A dataset needs to be created that represents a modern network with modern cybersecurity threats because of the time restrictions and the tools that regenerate the flows that are the same as the datasets presented. We had to move this target as part of the future work. There was some work already conducted for this part of the research, but it was not complete. Reference to the work can be found in the appendix, titled (Building PCAP)

3.6 Discussion

This chapter reviewed several commonly used datasets in intrusion detection system (IDS) research, ultimately focusing on CIC-IDS2018 due to its richness and alignment with modern network threats. Despite being a popular choice in recent studies, the dataset presents several challenges that could affect model performance if left unaddressed. These include class imbalance, especially with underrepresented attack types, redundant or noisy features, and inconsistent distribution of labels. Such issues can cause biased learning and misleading evaluation results, especially in multiclass scenarios. The discussion in this chapter

emphasized the importance of understanding these structural limitations early in the process so they can be accounted for during preprocessing and model design. The choice to use CIC-IDS2018 was not without compromise, but the availability of multiclass labels and updated attack simulations made it the most suitable option for the goals of this research. The insights gained here guide the technical decisions applied in later chapters.

3.7 Chapter Conclusion:

This chapter presented a detailed analysis of datasets relevant to IDS research, emphasizing the importance of choosing one that aligns with the objectives of multiclass classification and model flexibility. After assessing the features and limitations of multiple sources, CIC-IDS2018 was selected due to its broad representation of attack types and traffic diversity. However, several issues, such as the presence of class imbalance, redundant fields, and inconsistency in some labels, were identified. These factors are expected to influence training effectiveness and classifier accuracy, particularly in identifying minority class attacks. Acknowledging these limitations early allows the research to proactively apply targeted preprocessing techniques such as feature selection, normalization, and sampling strategies. Ultimately, the dataset chosen provides a strong yet realistic foundation on which to evaluate the proposed classifier portfolio model. The next chapter introduces the methodological framework that builds on this dataset and supports the core structure of the research design.

Chapter 4 Research Methodologies

4.1 Chapter Introduction

will cover two main aspects of the research design. The first aspect is the general design, where the project is inspired by the CRISP-DM framework. In this part, the main focus is to have a structured approach to build and deploy a model. It covers Data understanding, preparation, modeling, Evaluation, and deployment. Then, we propose an adaptation of the framework that will be used to develop the AI models. The second part of the chapter will describe the proposed classifier. This part will go into detail on how to construct the portfolio of classifiers along with the master classifier and then how the classification will work. A complete step-by-step process is presented to clarify how it would work.

4.2 Proposed Design

In this chapter, we will discuss the research methodologies and approaches. The research approach is mainly inspired by the CRISP-DM approach. CRISP-DM, which stands for (Cross Industry Standard Process for Data Mining) is considered the de facto for data science standardization. Most of the steps taken in this research are based on this approach to achieve the aims and targets of the research.

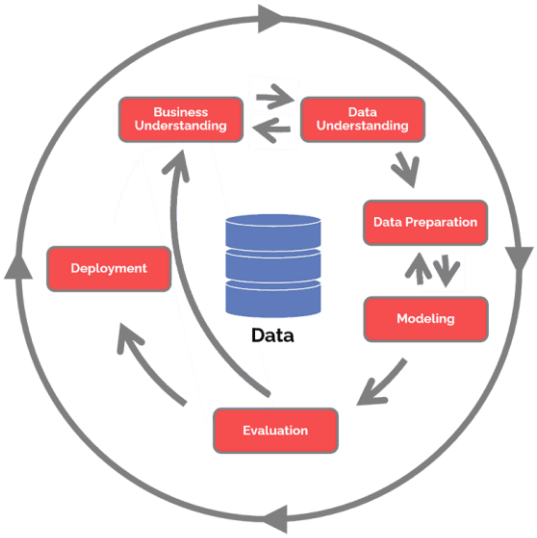


Figure 14 CRISP-DM retrieved from the official CRISP-DM website("What is CRISP DM? - Data Science Process Alliance," n.d.)

The research will adopt the steps shown in the diagram above that are part of the CRSIP-DM. As it is clearly visible in the diagram, there are multiple steps in circular motion that can ensure continuous improvements. The steps start with understanding the business, Data Understanding, Data Preparation, Modeling, and Evaluation, and finally it ends with Deployment.("What is CRISP DM? - Data Science Process Alliance," n.d.)

The research will go through multiple stages. The first stage is to evaluate different multiclass classification algorithms. The evaluation will go through the performance

and accuracy of the classifiers and determine if the classifier is fit to be included in the research. As most of the classifiers are already established and can perform well in the domain of IT security, we might have to make further inspections since the nature of the data requires further input and validation. The overview of the data will have a very high percentage and occurrences of benign connections. As a result of that huge imbalance, any classifier will be able to have a very high general accuracy in binary classification. In this case, we will investigate the accuracy of each type of attack as the classification in minority data is more important because the minority is the attack class. In addition, another metric that will be considered is the TF/FT/TT/FF matrix so we can have a better understanding of each classification type.

The second step is to evaluate different methods to have different classifiers work together for classification. Based on the survey done in the previous chapter, there are some researchers that have made this approach either by voting or merging. In this research, the result of each classifier will be encoded, and the dataset will be rebuilt for classification and to find the best classifier for each connection.

We will follow the below logical flow chart that's inspired by the "Cross Industry Standard Process for Data Mining."

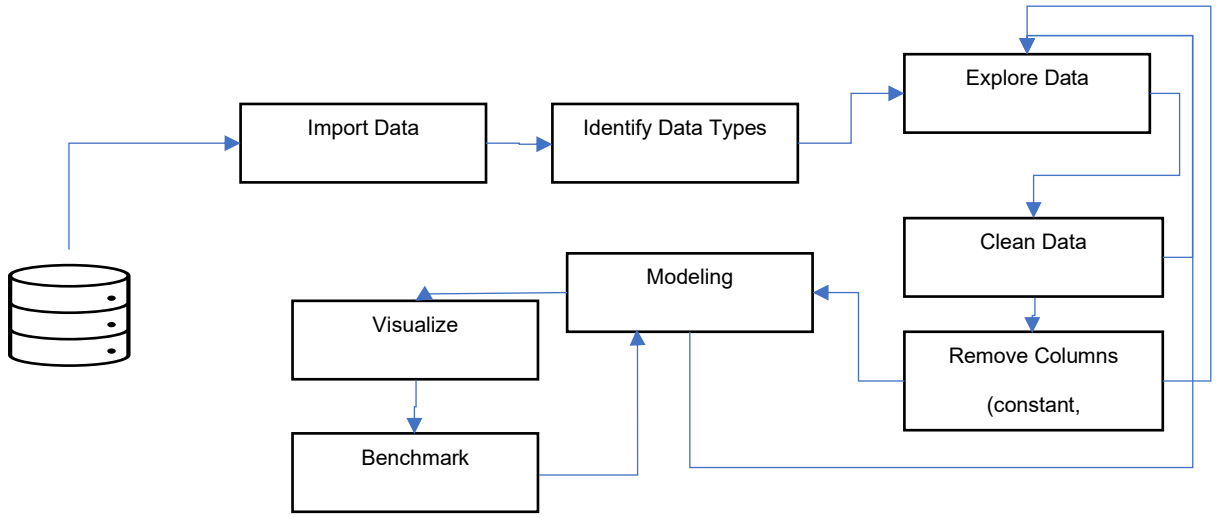


Figure 15 Research Methodology

The first phase of the approach is to get the Data. The usual data format for this type of problem is PCAP. SolarWinds defined the PCAP as “Packet capture is a networking practice involving the interception of data packets traveling over a network. Once the packets are captured, they can be stored by IT teams for further analysis. The inspection of these packets allows IT teams to identify issues and solve network problems affecting daily operations.”(“What Is Packet Capture (PCAP)? - IT Glossary | SolarWinds,” n.d.). PCAPs are all the communication in the network, either from the end device, servers, or network devices. The PCAP can easily be converted to CSV (comma-separated vectors). While it’s easy to get the PCAP, it’s difficult to get a PCAP that has genuine connections that are threats. Another issue with this type of data is that it needs to be labeled. Labeling millions of records is a very difficult task, and even if the process is automated, the automation is prone to errors and can mislabel the records. The reasonable approach is to find readymade data for cybersecurity with labels. A survey about the datasets is already done in the Dataset chapter.

Importing the data in the prober system is a challenge because of the size of the data and the high dimensionality (number of features or rows). We will go through different methods like sampling and testing tools that can handle this size of data. Since sampling theoreticality could work to import a sub-sample that can represent the complete dataset, but in reality, the data is very sensitive, and all records must be imported. For this reason, in this research, we have tested different tools to check if they can handle the data with different tools.

After importing the data to the desired tool, it will be explored and checked to see if the data is clean. Cleaning the data will include deleting rows that have any missing information or labels. In addition, the process will check if there are any unnecessary columns to be removed for different reasons. For example, some columns could be removed because they have constant values, or the data presented is irrelevant to the classification and might affect the model building. Another step in cleaning the data is the redundancy of the rows. Repeated records should be removed from the dataset to reduce the size and proper build for models. Once the data is cleaned up, it will be ready to be used to construct a model for classification. Different models will be built and evaluated. Finally, after classification and prediction, we will visualize the result for benchmarking and validation. The process will be repeated until satisfactory results are acquired.

The idea is to have multiple classifiers work together to determine the best classier in order to have the highest accuracy possible. As discussed in the literature review, there are some approaches that use a voting system, while others use fusion. Some papers propose a generic approach that will determine the classifier that will work

with all the data; on the other hand, some researchers use cluster and group data to determine the classifier that fits each group. In our case, a master classifier will select a classifier for each instance or flow. For this part, we propose a novel approach to build up a master model with the following process.

First, we determine some classifiers that can work with this type of dataset. The classifier should be capable of handling big data, high dimensionality, and multiclass classification. In addition to these conditions, the duration to process these data and build a model should be in a reasonable time. To build these models, we depend on software and packages that are specialized in Machine learning and Artificial intelligence. In the selection of software packages chapter, we will show the experience with multiple tools and libraries that are potentially suitable for this purpose, and we have determined that h2o in Rstudio would be the suitable tool to process the data and build models. Within h2o, there are multiple models that can be used, but we have restricted the use of Distributed Random Forest, Naïve bays, Deep Learning, GLM, and GBM; the reason for this restriction is that only these models can support multiclass classification.

4.3 Proposed Method:

The idea of the proposed method is to have a portfolio of multiple classifiers to detect or classify threats in IT networks. Different classifiers behave differently, and no single classifier can have 100% accuracy. For this reason, will have multiple classifiers that will be trained. These classifiers will be evaluated to determine if they can be part of the portfolio and benchmarked. Then, we will use a Master classifier to classify which classifier is more suitable for each instance or connection flow. The

whole process is done in three phases, which are the offline phase, the classification of a classifier, and finally, the classification.

4.3.1 Offline phase:

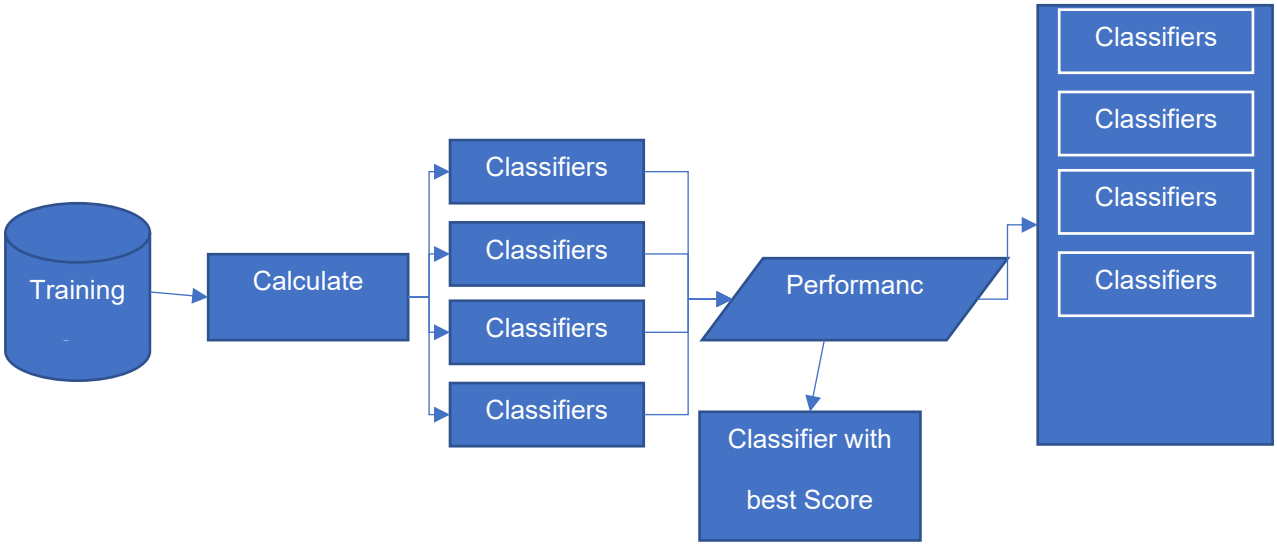


Figure 16 Offline phase in the proposed method

In the offline phase, once the dataset is ready and cleaned from any missing data and the columns have an insignificant effect on the prediction removed, the process will start to build and train multiple classification models. After that, each model will be evaluated and benchmarked based on the general accuracy. If the model has an accuracy that is within the threshold, then it will be part of the portfolio of classifiers. On the contrary, if the model is below the threshold, it will be discarded and will not join the portfolio. During the benchmarking process, the top classifier will be determined as a backup classifier.

4.3.2 Encoding for the master classifier:

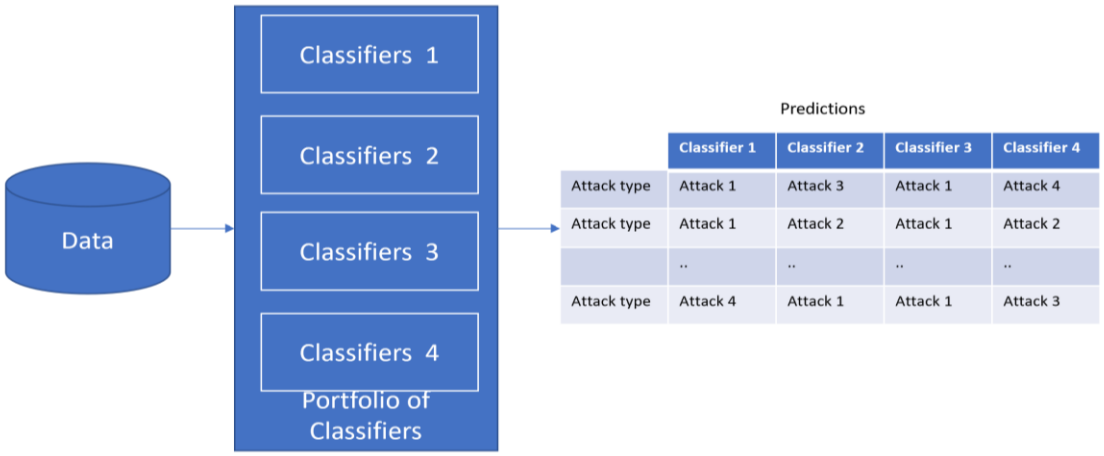


Figure 17 get all classifications from all models.

Encoding for the master classifier will be done as described in the above diagram. First, the dataset will be prepared, where we will get the prediction and classification from all classifiers in the portfolio. Each classifier might have a different prediction or matching prediction to other classifiers. This process is still offline, and it's part of the training, so the classification doesn't have to be in parallel. The process will be sequential, where the classification will start from the first classifier and finish the whole process, then it will proceed to the next classifier. Doing this step in a sequential manner can save computing power and memory resources, as each model can consume all the resources that we have in the research. Once all the classifications from all classifiers are done, the results will be aggregated, as shown in the diagram above.

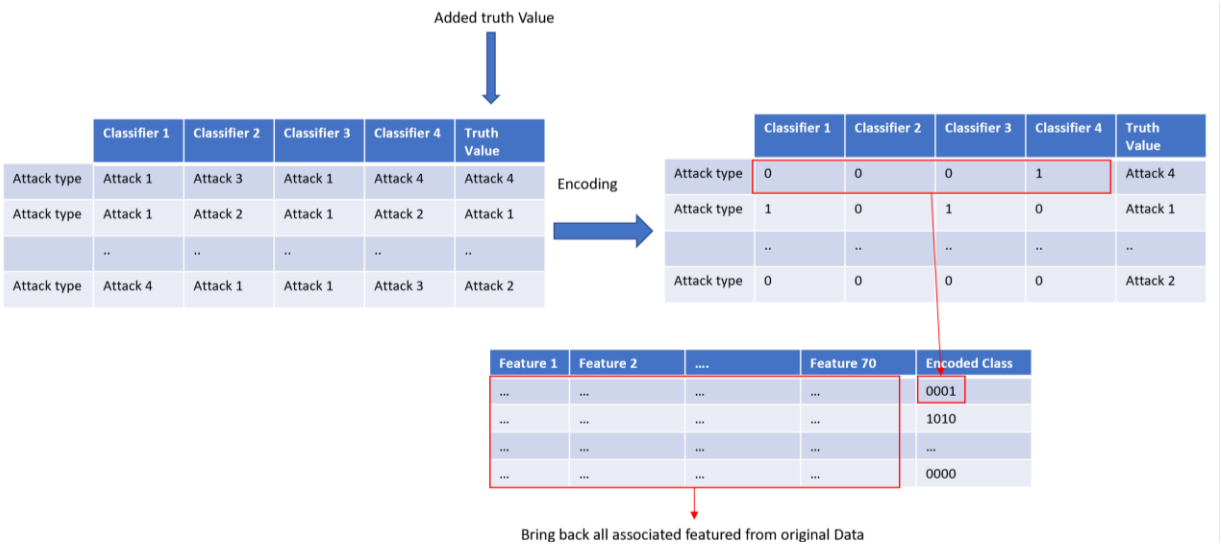


Figure 18 Preparing encoded dataset for the master Model.

The next step will start with adding the truth value from the original dataset to the aggregated data. Then, the encoding process will start with every Classifier prediction being compared to the truth value, and if the prediction matches the truth value, then the prediction will be changed to 1. If the prediction does not match, then the value will be changed to 0. The binary values will be concatenated and merged into a single binary value, as shown in the table in the diagram above. Finally, all the columns that represent the features in the original data will be added back. The dataset will be ready for the master classifier for training, as shown in the diagram below.

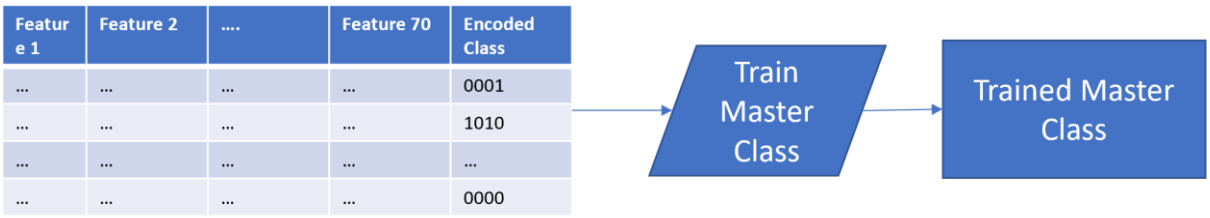


Figure 19 Encoded Dataset ready for Training.

4.3.3 Online prediction:

After the training is done, the master model is ready for prediction. The original dataset will be used as an input for the master classifier. The master classifier will output a binary code, as seen in the diagram below. Each digit in the binary code represents a classifier in the portfolio, and if the digit is 1, we can use the classifier represented in that digit. If all digits have zero value, then we will depend on the backup classifier.

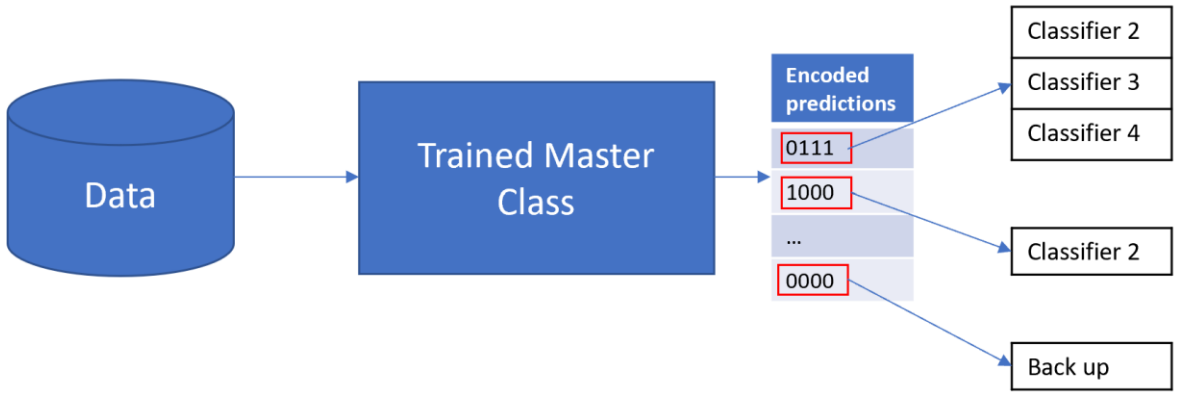


Figure 20 Classification for classifiers

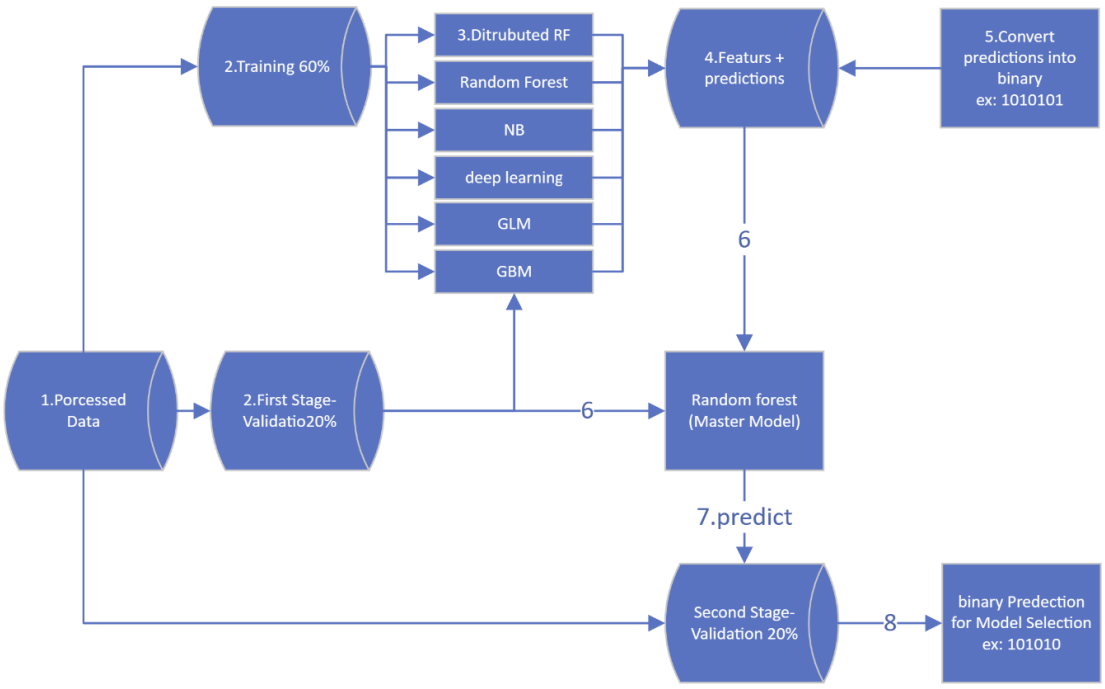


Figure 21 Overall Methodology

In order to understand the flow of the process, the following will explain every step as enumerated in the graph.

1- The data will be split into three parts.
    a. Training 60%
    b. First stage-Validation 20%
    c. Second Stage-Validation 20%
2- Five models are built using the training data and first-stage validation data, which are distributed Random forest, Deep learning, Naïve Bayese, Random Forest, GLM, and GBM.
3- The validation predictions will be aggregated along with features it will look as below.

…Features…	Class(Original)	DRF	DL	RF	GLM	GBM
…	Benign	Infiltration	Benign	Benign	Brute Force -Web	Benign

4- The labels will converted into binary values in two steps

Step 1

…Features…	Class(Original)	DRF	DL	RF	GLM	GBM
…	Benign	0	1	1	0	1

Step 2

…Features…	Class
…	01101

5- In this step, a Random Forest Model, which is considered the master classifier, is built based on the binary class generated.

6- The model will predict the phase 2 validation data using the same set of features, except the class is changed to the binary code. Example below:

…Features…	Class
…	10011

7- The selection phase will be done as below:

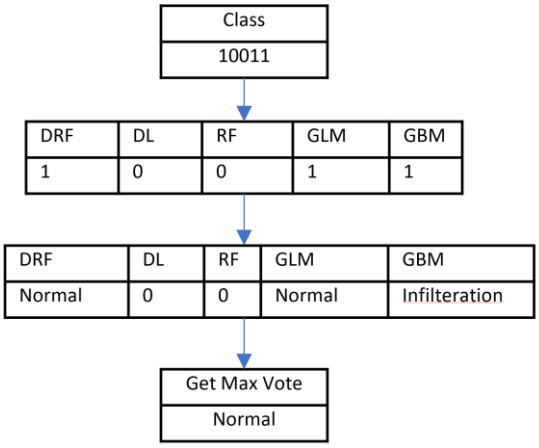


Figure 22 Overall view of the Portfolio result decodes and final result.

4.4 Discussion

This chapter detailed the methodological framework used throughout the research, with a specific emphasis on how machine learning models are developed and evaluated in the context of intrusion detection. The use of the CRISP-DM model structure provided a logical and systematic path from problem definition to evaluation. The design of a per-instance model selection approach addresses one of the key limitations identified in earlier chapters, namely, that a single classifier often fails to generalize across different traffic patterns. The dynamic portfolio strategy allows the system to apply the most suitable classifier based on the nature of each instance, improving classification accuracy and adaptability. The discussion also highlighted some of the practical constraints in methodology, such as the need for computational efficiency, cross-validation strategies, and maintaining a balanced evaluation approach for multiclass problems. These factors were carefully accounted for to ensure that the chosen methods align with both academic rigor and real-world practicality.

4.5 Chapter Conclusion

In conclusion, this chapter established the methodological backbone of the study, clearly outlining how the research was designed, executed, and validated. By adopting a structured and iterative approach to data mining and model evaluation, the methodology ensures that the research remains replicable and logically sound. The chapter explained how CRISP-DM principles were adapted to suit the needs of the intrusion detection use case, while also introducing the concept of per-instance classifier selection. This framework is intended to offer both flexibility and precision, reflecting the core research aim of developing a responsive and effective detection

system. The processes described in this chapter, from dataset preparation to portfolio design, provide the foundation upon which the classifier models are built and tested in the following chapters. The next step in the research is to describe the technical environment and toolset used to support this methodology.

Chapter 5 Investigation and selection of software packages

5.1 Chapter Introduction

In Chapter 5, we survey different tools that serve AI and machine learning. In order to have the right tools, we need to make the survey and understand the tools. Having the right software and services would facilitate the progress of the research. In addition to the software and services, we will compare local compute and cloud services. The selection criteria for the tools would take into consideration the capability of the tool, ability to handle the dataset, ease of use, and finally, cost.

In this chapter, we will discuss different tools and software that could be used in machine learning which are related to this research. First, we will see the evolution of these tools and how they have evolved over the years with many introductions of machine learning tools. As we will see, the machine learning tool took advantage of different architectures. The first architecture in this chapter is the traditional local compute, and then the GPU compute that can speed up the computation process. Finally, we have the cloud, which provides virtually unlimited resources that can accommodate any need for any machine learning project. Overly, the chapter will go as follows: chronological order of machine learning software and testing software packages (local and cloud), and finally, the selected list of tools and software.

5.2 Chronological order of libraries in Machine learning and Artificial intelligence

In this part, we will go over the libraries in chronological order from the past 30 years.

Table 12 History of Machine learning tools and software

Machine learning Tool	Brief description
R language	R was developed in 1993, and the framework for R is still under active development. Since R language is an open-source project, there are many packages and integrations that are developed by different entities that keep this language and its development tools alive and active. R language was initially targeting statistical computing and analysis, but now R language covers wider domains such as data mining, bioinformatics, and machine learning. The R framework has its official package manager called CRAN, and it can use external sources using R tools, which allows R language to have the latest algorithms and functions available for the user(“R: The R Project for Statistical Computing,” n.d.).
MLC++	From the name of the library, we can know the library was developed in C++. MLC++ was developed at Sanford University in 1994 and then maintained by Silicon Graphics IC (Silicon Graphics was bought by Hewlett Packard). This library mainly focuses on pattern recognition, data mining, and statistical analysis. The development of these libraries seems to be halted since there is no reference for the library on the Hewlett Packard website(“MLC++, A Machine Learning Library in C++,” n.d.).
OpenCV	OpenCV was released in 2000. This library was developed by Intel, and its focus is Real-time computer vision. This library allows image recognition, tracking, and object identification. The library has interfaces with many languages and is widely used(“OpenCV - Open Computer Vision Library,” n.d.).
scikit-learn	The introduction of scikit-learn was a major milestone. The library is open-source, and it is a product of Google Code Summer Camp. The

	library was written in Python using existing library's like NumPy, and matplotlib. scikit-learn is in continuous development and has a great number of functions in machine learning("scikit-learn: machine learning in Python — scikit-learn 1.3.0 documentation," n.d.).
Weka	Weka was developed in 1993 at the University of Waikato, New Zealand. Initially, the library was built in C++, but then it was rebuilt in Java. The tool can do pre-processing, clustering, classification, and regression. In addition, deep learning was later introduced. The tool is intuitive and has a graphical user interface for ease of use("Weka 3 - Data Mining with Open Source Machine Learning Software in Java," n.d.).
RapidMiner	RapidMiner was known as Yale and was released in 2001. The application can provide a wide spectrum of machine learning and data analysis applications. The tool does provide a graphical user interface for ease of use and processing("RapidMiner | Amplify the Impact of Your People, Expertise & Data," n.d.).
Spark MLlib	Spark MLlib is part of the spark project from Apache, which was introduced in 2015. The project depends on the Spark core, and some of the main advantages are its ability to operate in distributed systems, and it is memory-based rather than disk-based. These two features can make this library operate much faster than any other library because it can be installed on many commodity hardware to achieve great performance that is parallel to enterprise hardware("MLlib | Apache Spark," n.d.).
Torch	Torch is a machine-learning language that provides a scientific computing framework. The language is open-source, and it is based on Lua. The language provided functions that are written in C language and can be invoked by LuaJIT. In recent years, the development has moved to Python, and now it is called Pytorch ("PyTorch," n.d.).
Caffe	Caffe (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework originally developed at the University of California,

	Berkeley, in 2017. It is open source, under a BSD license. It is written in C++ with a Python interface. Caffe is being used in academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia(“Caffe | Deep Learning Framework,” n.d.).
TensorFlow	TensorFlow is an open-source project that was introduced in 2015. The tool has great support from the open-source community, which gives it flexibility in its ecosystem. TensorFlow allowed scientists and researchers to develop state-of-the-art Machine learning models and build applications that are supported by Artificial Intelligence. The framework has the latest functions and algorithms in Machine Learning, and it does support the latest technologies like GPU processing. Finally, TensorFlow is compatible with Python, C++, and many other languages.(“TensorFlow,” n.d.)
H2O	H2O is an open-source project that was introduced in 2011. One of the main features of this library is that it has linear scalability, and it does operate in memory rather than on disk. These two features make it robust and much faster than other tools. The library supports most of the Machine learning algorithms, and it keeps updates coming very frequently.(“H2O.ai | The fastest, most accurate AI Cloud Platform,” n.d.)

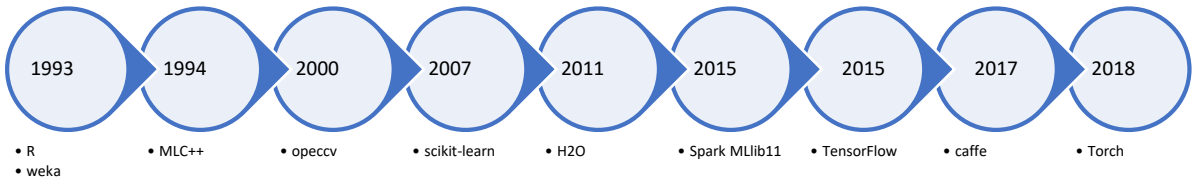


Figure 23 Chronological order of Machine Learning Tools

5.3 Testing Software Packages

The initial preferred approach is to use local resources (compute) for machine learning and building multiple models. The benefit of having the models built locally is that it will eliminate the requirement to transmit the data from local to cloud and vis versa. The second benefit is that the cloud solutions for machine learning will require the reconstruction of the script and the study of new tools with a steep learning curve. Studying will be required to adapt to any public cloud solution for machine learning. On the other hand, the problem with local resources is the limitation of the resources. The size of the datasets is huge and doesn't fit into memory. Also, the compute time is long, which will slow the flow and speed of the development of the models. Any minor modification in the model will require multiple hours to rebuild.

In contrast, we have cloud solutions that are elastic and have the capability to virtually accommodate any workload required. The initial impression of cloud computing was that it would be cost-effective to build the models, but after investigating the pricing for cloud computing, it was clear that the cloud is much more expensive than local resources. Using Microsoft Azure Calculator ("Pricing Calculator | Microsoft Azure," n.d.) the average monthly cost for the resource (D64d v5) with the following specifications (64 vCPUs, 256 GB RAM) is US$3,106.88 monthly. This cost will come to around 37,282.56 USD annually. These values are way beyond the budget, and even if we consider that these resources will be shut down periodically, the value will remain high. Another factor that needs to be considered in cloud adoption is the cost of the bandwidth. The bandwidth cost will be high since the dataset

size is huge and will require a lot of bandwidth for transit between local and cloud. The cloud solution is convenient as a solution, but financially, it does not fit all scenarios and requirements. It is difficult to make a direct comparison in pricing between cloud and on-premise solutions, but even the intention to acquire these specifications on-premise will be difficult as these requirements are enterprise and can't be acquired by an average consumer.

5.3.1 Local Compute

The initial preferred approach is to use local resources (compute) for machine learning and building multiple models. Below are some of the tested packages and tools.

- TensorFlow
  - TensorFlow is dependent on other libraries (Cuda) and specific drivers in order to function. Multiple attempts were made in order to get the TensorFlow to work in the local system. The problem with this library is that it requires a specific version of Cuda and NVidia drivers in order to function. Also, the documentation for these libraries is weak and not centralized. It does not work out of the box and requires a lot of work to have a functioning system. Any minor update will cause the system to crash. Another issue is that the library uses GPU to process the data, and if the loaded data exceeds the memory in the GPU, the performance will degrade drastically. Finally, for the

reasons mentioned above, we have opted not to use TensorFlow.

- Python with Scikit-learn
  - Using Scikit-learn would require using Python. In order to load the data, Pandas libraries need to be called to load the data frame. The Data frame does not fit into memory, and Panda does handle large data very well. As a workaround, chunks were used to resolve the issue, but the performance was very slow to build models with this method.
- Standard RStudio packages
  - RStudio with Standard libraries was used in the initial tests of the project. The data was loaded without any issues. RStudio was capable of managing memory and utilizing storage paging and did all of the management without any intervention. However, the issue relies on standard packages that have a lot of limitations. For example, Neuralnet did not utilize all CPU resources and was prohibitively slow. Another example is the standard random forest from CRAN. The Standard Random Forest has a limit on the number of levels of the factors. And the data types have some factors that exceed these levels. At the same time, the library is also not exhausting the CPU resources, and the process time is very slow.
- RapidMiner

- RapidMiner was tested on loading the dataset. The Dataset was loaded, but after importing the data, the application was barely usable and not responsive. Also, the application crashes randomly and unpredictably. Another issue is that the user does not have the freedom to manipulate the data within the application. For example, converting the data from Decimal to binary. For these reasons, RapidMiner was excluded from the project.

- Weka
  - Weka had very similar issues to RapidMiner. The application becomes unresponsive and crashes unpredictably. If the data loads, Weka will take unreasonable time to process the data and build models.

- H2O package in RStudio
  - H2O.ai was used in the project in RStudio. It has good integration with RStudio, and the data can be moved between H2o Data and RStudio easily without any dependency on any additional libraries. The data was easily loaded and processed without error and crashing. H2O is able to fully utilize the CPU, and the memory is fully optimized, where it does not fully occupy the memory when the data is loaded and the models are built. The h2O can use the CUDA libraries to utilize the GPU, but it was not used as the official website lacks documentation for the

integration, and a previous attempt with TensorFlow showed that if the data does not fit in the GPU memory, the performance will get dramatically worse. In addition, h2o can run as a service where the h2o can run on its own, and the data can be viewed in the web browser, build models, and generate plots and graphs. Once a model is built, it's easy to move this model regardless of the h2o version and system that is being used since the h2o can export the models in POJO format, which facilitates moving the models between different systems and versions.

5.3.2 Cloud Compute:

- There are multiple public cloud services, and they offer machine learning models as a service. Most of these services are dependent on the compute resource, and the service comes prebuilt with the required machine learning libraries. It will be fair to compare the computing price only, as the libraries and tools can be installed by the user. For comparison, the following specifications were targeted.
  - 64 compute cores. (Regardless of generation)
  - 256 Gigabytes of Memory
  - Ubuntu operating system
  - The Server can be either dedicated or shared
  - The resource will be on demand, and the selection will not consider reservation discount as the reservation will require at

least 1-year reservation commitment. For that reason, we will only consider the pay-as-you-use model.

- Azure
  - Using Microsoft Azure Calculator (Pricing Calculator | Microsoft Azure), the average monthly for the resource (E64a v4) with the following specs (64 vCPUs, 256 GB RAM) is 3,270.40 monthly. Annually, 39,244.8 USD West US
- Amazon
  - Amazon was checked for the computing price. For a compute instance that matches the specifications that we have determined, we have chosen m6g.16xlarge. This instance will cost 2354.69 USD monthly and 28,256.26 USD annually. East US Ohio
- Huawei
  - 1,710.28 USD and annually 20,523.36USD instance m6.16xlarge.8 CN-southwest Guiyang1

Table 13 Comparison between different cloud providers

Specifications	Cloud Provider	Amazon	Azure	Huawei
vCPU 64	Location	East US Ohio	West US	CN-southwest Guiyang1
	Instance type	m6g.16xlarge	E64a v4	m6.16xlarge.8

Memory 562	Monthly	2354.69 USD	3,270.40 USD	1,710.28 USD
	Yearly	28,256.26 USD	39,244.8 USD	20,523.36 USD

5.3.3 Tools used in the research:

After reviewing and investigating many tools for the research, it was determined to use the following tools:

- Local Compute:
    - Rstudio
    - H2o.AI
- Cloud
    - Huawei Cloud
    - Regular Virtual Machine, with specifications mentioned before.
    - Use Rstudio
    - H2o.AI

The reason to use Regular virtual machines instead of readymade Machine learning tools is to preserve the code that was developed in local computing and avoid redeveloping the same script. At the same time, any saved data frame and datatypes can be preserved and transitioned between local and cloud.

5.4 Discussion

This chapter reviewed the software tools and platforms used throughout the research, with a focus on selecting components that support both model

development and performance evaluation. The investigation covered a range of environments, including both local machines and cloud-based platforms, to determine the most suitable setup for training and testing complex models. Several trade-offs were noted during this process. For instance, while local tools provided more direct control and simplicity for debugging, cloud platforms offered better scalability and compute capacity for training deep learning models. The selection of tools like RStudio, Python, and specific machine learning libraries was made based on their compatibility with the chosen methodology and dataset. This discussion also emphasized the importance of a modular and reproducible environment, one that could support different classifiers under a unified framework. The software architecture was structured in a way that enables consistent preprocessing, training, validation, and result interpretation across all model types.

5.5 Chapter Conclusion

In conclusion, this chapter outlined the process of evaluating and selecting the software tools required to implement the proposed methodology. The research demanded a flexible environment capable of supporting both traditional machine learning models and more complex ensemble approaches. By assessing different platforms, IDEs, and library options, the final selection was made with an emphasis on computational efficiency, ease of integration, and reproducibility. Both local and cloud-based setups were utilized at different phases of experimentation, each playing a role in balancing control with processing power. The tools chosen allow for seamless model deployment, testing, and visualization, all of which are critical in executing the methodology described in the previous chapter. With the software environment established, the thesis now moves forward to data exploration, where

the structure and behavior of the selected dataset are analyzed in preparation for model building.

Chapter 6 Data Exploration

6.1 Chapter Introduction

In this research, we have determined to use the CSE-CIC-IDS2018 data. As discussed in the Dataset chapter, the KDDCUP dataset and all of its derivatives will be excluded as the data is old, and there are many recommendations in the data science community not to use the KDDcup dataset as it has a lot of inconsistency, and it's not a true representation of the modern network in the industry, and the type of threats included in the dataset are old and not modern.

The CSE-CIC-IDS2018 Dataset was chosen because the data is recent, and there are many different cybersecurity attacks that are considered modern and have a true representation of modern networks.

In this chapter, we will explore the CSE-CIC-IDS2018 dataset. first, we will view how the dataset is generated and labeled. A discussion was presented about some issues related to the dataset, such as size and consistency. Finally, after reviewing the feature, a statistical exploration is done that will show distribution trends that show patterns for selected attacks.

6.2 Construction of the dataset

The CSE-CIC-IDS2018 data is a result of simulated attacks that took several days and are captured as PCAPs. Then, the data is converted to CSV format with labels.

The advantage of using the CSE-CIC-IDS2018 data is that the data is reproducible with the same features. The author of the data demonstrated how the data is generated and captured. In addition, the author made the tools available to convert the PCAP to flows with all the features. Simply, any person can regenerate the data with the same format easily with the tool available with any new attack. The only problem that can be faced to reproduce the data is labeling the flows with the right information. Even if we consider the timestamps of the attacks, we might mislabel the benign flows with the attacks. Below is the summary of the attacks done on the data, along with the duration from the author's website.

Table 14: attacks durations in CSE-CIC-IDS2018 from CSE-CIC website (Canadian Institute for Cybersecurity, 2018)

Attack	**Tools**	**Duration**	**Attacker**	**Victim**
Bruteforce attack	FTP – Patator SSH – Patator	One day	Kali linux	Ubuntu 16.4 (Web Server)
DoS attack	Hulk, GoldenEye, Slowloris, Slowhttptest	One day	Kali linux	Ubuntu 16.4 (Apache)
DoS attack	Heartleech	One day	Kali linux	Ubuntu 12.04 (Open SSL)
Web attack	• Damn Vulnerable Web App (DVWA) • In-house selenium framework (XSS and Brute-force)	Two days	Kali linux	Ubuntu 16.4 (Web Server)

Infiltration attack	• First level: Dropbox download in a windows machine • Second Level: Nmap and portscan	Two days	Kali linux	Windows Vista and Macintosh
Botnet attack	• Ares (developed by Python): remote shell, file upload/download, capturing • screenshots and key logging	One day	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)
DDoS+PortScan	Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests	Two days	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)

6.3 Challenges:

There were different challenges in the acquired data from CSE-CIC-IDS2018. The challenges start from getting the data from the Amazon servers, and there are issues with the consistency of the data. There are millions of data records, and it's difficult to scan through the data manually and find the errors in each record. In this research, we have faced two major challenges with the CSE-CIC-IDS2018 data, which are the size and the consistency.

1- Data Size:

One of the main challenges in acquiring the data was size. The author made the data available through Amazon bucket storage. Downloading the data was by parts for each day, and the downloaded data included CSV files. The CSV files were already labeled and ready for machine learning. In addition to that, the Data included the TCP dumps, which were huge and very slow to download. The process has faced many interruptions and disconnections. Even after downloading the data, we have faced data corruption that needs to be mitigated by redownloading the same files again.

2- Data Consistency:

The CSE-CIC-IDS2018 data has a lot of issues related to format and consistency. Finding these errors was a pure trial and error process. The CSE-CIC-IDS2018 data comes in multiple files, and each file represents one day of data collection. The data needs to be merged for exploration and machine learning. Upon merging these files, many issues occurred, which related to mismatching the number of columns and repetitive headers.

a. Mismatching number of columns:

As mentioned, each file represents a day of collected data. When we attempted to merge all files together, we received an error that there was a mismatching number of columns. We had to view different files randomly to find out that some files had extra columns that consisted of (TimeStamp, src IP, src port, and dest IP). We had to drop these extra columns from the files that had it so all the files were uniform and consistent.

b. Repeated header:

This error was discovered when some columns were supposed to have a datatype that is number/integer, but the RStudio identified it as text/string. We had to run a script that would go through each line of the CSV file to check if it was a number or not. Then we found the first line that is not a number was actually a repeated header of the column names. After inspecting the files, we found that some files have the header repeated randomly in different parts of the file.

6.4 List of features in the CSE-CIC-IDS2018 Dataset:

The below list is the extracted features from the PCAP files. The extracted features were captured using the CICflowmeter tool that was developed by the author of the dataset. These features represent some characteristics of the network flows. Each group of flows can be viewed either as benign or attacks, and the way to distinguish any of them is by the list of features presented. We can see that some features represent time, and some of them represent size and number while others flag. Some features can be derived from others, and some of them are tightly related. This can be explained since some features represent the size of packets (max, min, average, and std) for the forward direction, and we can see that these features have a tight relationship. It's difficult to review every feature in the dataset, as the main focus of this research is the classification rather than network study. For further information, a person can review these features from the author's site(“IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB,” n.d.). The list of features provided below is presented exactly as written by the author.

fl_dur : Flow duration

tot_fw_pk : Total packets in the forward direction

tot_bw_pk : Total packets in the backward direction

tot_l_fw_pkt : Total size of packet in forward direction

fw_pkt_l_max : Maximum size of packet in forward direction

fw_pkt_l_min : Minimum size of packet in forward direction

fw_pkt_l_avg : Average size of packet in forward direction

fw_pkt_l_std : Standard deviation size of packet in forward direction

Bw_pkt_l_max : Maximum size of packet in backward direction

Bw_pkt_l_min : Minimum size of packet in backward direction

Bw_pkt_l_avg : Mean size of packet in backward direction

Bw_pkt_l_std : Standard deviation size of packet in backward direction

fl_byt_s : flow byte rate that is number of packets transferred per second

fl_pkt_s : flow packets rate that is number of packets transferred per second

fl_iat_avg : Average time between two flows

fl_iat_std : Standard deviation time two flows

fl_iat_max : Maximum time between two flows

fl_iat_min : Minimum time between two flows

fw_iat_tot : Total time between two packets sent in the forward direction

fw_iat_avg : Mean time between two packets sent in the forward direction

fw_iat_std : Standard deviation time between two packets sent in the forward direction

fw_iat_max : Maximum time between two packets sent in the forward direction

fw_iat_min : Minimum time between two packets sent in the forward direction

bw_iat_tot : Total time between two packets sent in the backward direction

bw_iat_avg : Mean time between two packets sent in the backward direction

bw_iat_std : Standard deviation time between two packets sent in the backward direction

bw_iat_max : Maximum time between two packets sent in the backward direction

bw_iat_min : Minimum time between two packets sent in the backward direction

fw_psh_flag : Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)

bw_psh_flag : Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)

fw_urg_flag : Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)

bw_urg_flag : Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)

fw_hdr_len : Total bytes used for headers in the forward direction

bw_hdr_len : Total bytes used for headers in the forward direction

fw_pkt_s : Number of forward packets per second

bw_pkt_s : Number of backward packets per second

pkt_len_min : Minimum length of a flow

pkt_len_max : Maximum length of a flow

pkt_len_avg : Mean length of a flow

pkt_len_std : Standard deviation length of a flow

pkt_len_va : Minimum inter-arrival time of packet

fin_cnt : Number of packets with FIN

syn_cnt : Number of packets with SYN

rst_cnt : Number of packets with RST

pst_cnt : Number of packets with PUSH

ack_cnt : Number of packets with ACK

urg_cnt : Number of packets with URG

cwe_cnt : Number of packets with CWE

ece_cnt : Number of packets with ECE

down_up_ratio : Download and upload ratio

pkt_size_avg : Average size of packet

fw_seg_avg : Average size observed in the forward direction

bw_seg_avg : Average size observed in the backward direction

fw_byt_blk_avg : Average number of bytes bulk rate in the forward direction

fw_pkt_blk_avg : Average number of packets bulk rate in the forward direction

fw_blk_rate_avg : Average number of bulk rate in the forward direction

bw_byt_blk_avg : Average number of bytes bulk rate in the backward direction

bw_pkt_blk_avg : Average number of packets bulk rate in the backward direction

bw_blk_rate_avg : Average number of bulk rate in the backward direction

subfl_fw_pk : The average number of packets in a sub flow in the forward direction

subfl_fw_byt : The average number of bytes in a sub flow in the forward direction

subfl_bw_pkt : The average number of packets in a sub flow in the backward direction

subfl_bw_byt : The average number of bytes in a sub flow in the backward direction

fw_win_byt : Number of bytes sent in initial window in the forward direction

bw_win_byt : # of bytes sent in initial window in the backward direction

Fw_act_pkt : # of packets with at least 1 byte of TCP data payload in the forward direction

fw_seg_min : Minimum segment size observed in the forward direction

atv_avg : Mean time a flow was active before becoming idle

atv_std : Standard deviation time a flow was active before becoming idle

atv_max : Maximum time a flow was active before becoming idle

atv_min : Minimum time a flow was active before becoming idle

idl_avg : Mean time a flow was idle before becoming active

idl_std : Standard deviation time a flow was idle before becoming active

idl_max : Maximum time a flow was idle before becoming active

idl_min : Minimum time a flow was idle before becoming active

6.5 Dataset Exploration

The original dataset contains around 16 million records with very high dimensionality, so it will be hard to manipulate and process as it is. We seek to remove duplicate records from the dataset to facilitate the process, as it requires more time, memory, and computing power. After deduplicating the data and removing consistent columns, the dataset summary is as follows in the table below. We can notice that most of the data consists of benign records and data flows that make up approximately 88% of the dataset. The rest of the data consists of all the other data which are the target for classification. Some of the attacks have a higher percentage of records, and that's understandable because of their nature. For example, we have

the sum of all DDoS attack types that have a high percentage in the dataset, which is around ~6.7%. This is normal as the DDoS attacks will require a large number of connections that are initiated from many different sources to a targeted server. These huge numbers of connections will exhaust the server and will start dropping legitimate connections and will not be able to provide service.

Table 15 Distribution of Classes in CSE-CIC-IDS2018

names	Count	Percentage
Benign	10210250	88.30831
Bot	144535	1.25008
Brute Force -Web	553	0.00478
Brute Force -XSS	228	0.00197
DDOS attack-HOIC	198861	1.71995
DDOS attack-LOIC-UDP	1730	0.01496
DDoS attacks-LOIC-HTTP	575364	4.97632
DoS attacks-GoldenEye	41406	0.35812
DoS attacks-Hulk	145199	1.25582
DoS attacks-SlowHTTPTest	55	0.00048
DoS attacks-Slowloris	9908	0.08569
FTP-BruteForce	53	0.00046
Infilteration	139775	1.20891
SQL Injection	84	0.00073
SSH-Bruteforce	94048	0.81342
Total	11562049	100.00000

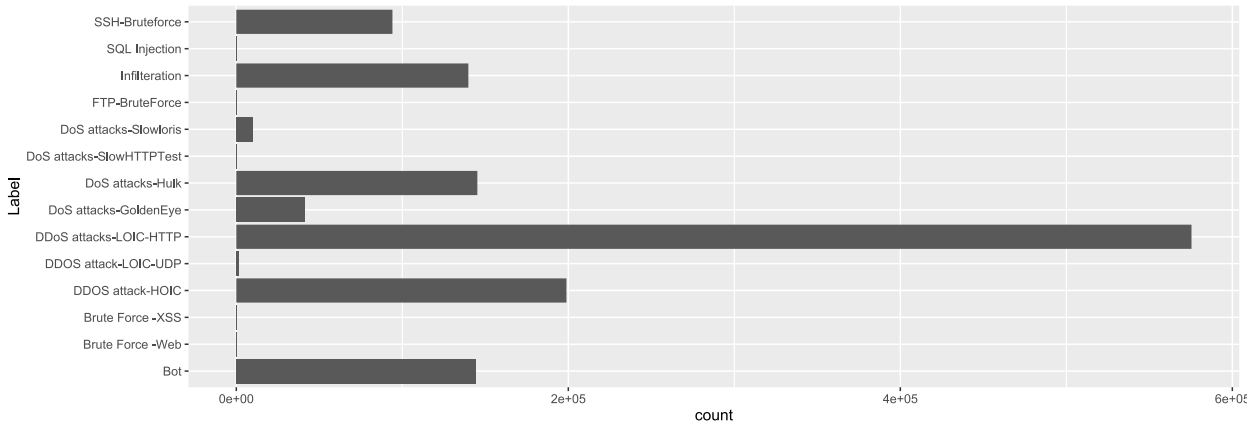


Figure 24 Distribution of attacks after removing Benign

benign was removed from the chart so the attack distribution could be more visible. As we have seen in the summary table above, the benign records are around ~88%, so we will have difficulty viewing the attacks in the graph. As discussed before, we can visibly see that most of the remaining data is DDOS attacks, which is around ~6.7%, and then the rest of the attacks would be around ~4.9%.

6.5.1 Initial observations:

- The dataset size and the number of features (80 features) are huge for any manual process and highly intensive for computation.
- Not all features are worth analyzing to find any pattern to assist in any classification and making the decision.
- The distribution of classes is not balanced and might cause a high rate of False-Positive or False-Negative results.

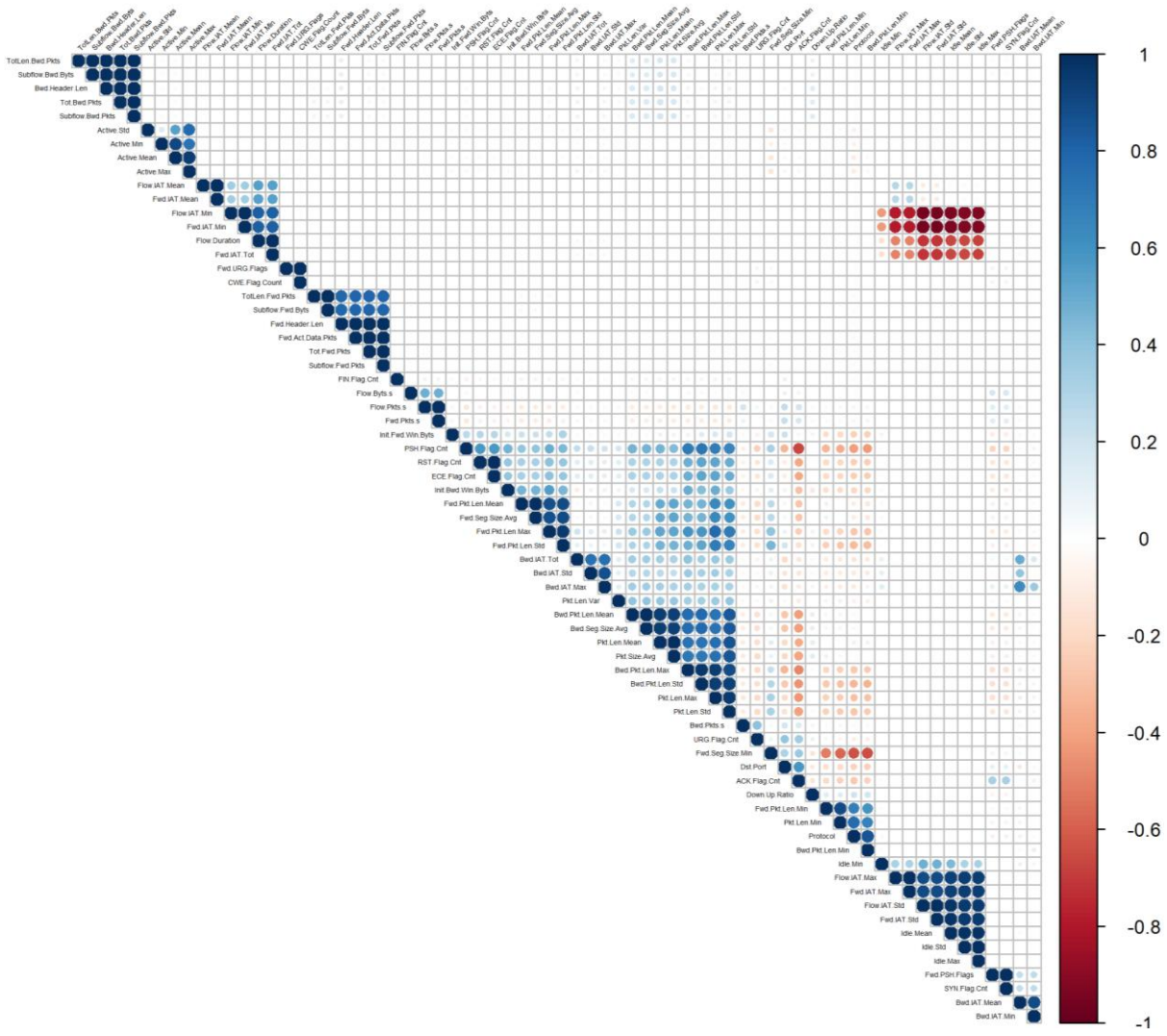


Figure 25 Correlation Matrix for CSE-CIC-IDS2018

If we have a look at the correlation matrix of the features in the chart and the table below, We can see many relations between the features, and it is expected, as some features are just derived from other features like (Min-Max-std-Mean). The case can be seen in Idle (Idl.min, Idle.max, Idle.std, and Idle.Mean). These types of features

will have a correlation. All features are either linked to the duration, time, size, or length of different elements within a flow.

In the table below, we only show variables that have a correlation that is more than 0.9 and below -0.90. As we can observe from the table, many of the variables have a correlation with some of their derivatives (Maximum, Minimum, Standard deviation, or Mean).

Table 16 Correlation Coefficient Table

Row	Column	Correlation Coefficient
Tot.Bwd.Pkts	TotLen.Bwd.Pkts	0.993591
Fwd.Pkt.Len.Max	Fwd.Pkt.Len.Mean	0.879368
Fwd.Pkt.Len.Max	Fwd.Pkt.Len.Std	0.954824
Fwd.Pkt.Len.Mean	Fwd.Pkt.Len.Std	0.890569
Bwd.Header.Len	Subflow.Bwd.Pkts	0.997801
Bwd.Pkt.Len.Max	Bwd.Pkt.Len.Std	0.96619
Flow.IAT.Std	Flow.IAT.Max	0.895744
Flow.Duration	Flow.IAT.Min	0.814468
Flow.IAT.Std	Flow.IAT.Min	-0.95762
Bwd.Header.Len	Subflow.Bwd.Byts	0.996151
Flow.IAT.Min	Fwd.IAT.Tot	0.814475
Flow.IAT.Mean	Fwd.IAT.Mean	0.999964
Flow.IAT.Std	Fwd.IAT.Std	0.999981
Flow.IAT.Max	Fwd.IAT.Std	0.895752
Flow.IAT.Min	Fwd.IAT.Std	-0.95761
Flow.IAT.Std	Fwd.IAT.Max	0.895737
Flow.IAT.Max	Fwd.IAT.Max	0.999994
Fwd.IAT.Std	Fwd.IAT.Max	0.895755
Bwd.Pkt.Len.Mean	Pkt.Size.Avg	0.938971
Flow.IAT.Std	Fwd.IAT.Min	-0.95761
Flow.IAT.Min	Fwd.IAT.Min	0.999996
Fwd.IAT.Tot	Fwd.IAT.Min	0.814487
Fwd.IAT.Std	Fwd.IAT.Min	-0.9576

Bwd.IAT.Std	Bwd.IAT.Max	0.875919
Bwd.IAT.Mean	Bwd.IAT.Min	0.89453
Bwd.Pkt.Len.Mean	Bwd.Seg.Size.Avg	1
Flow.Duration	Fwd.IAT.Tot	0.999986
Flow.Duration	Fwd.IAT.Min	0.81448
Flow.Pkts.s	Fwd.Pkts.s	0.994678
Fwd.Pkt.Len.Min	Pkt.Len.Min	0.896394
Bwd.Pkt.Len.Max	Pkt.Len.Max	0.949866
Bwd.Pkt.Len.Std	Pkt.Len.Max	0.944087
Bwd.Pkt.Len.Mean	Pkt.Len.Mean	0.943686
Bwd.Pkt.Len.Max	Pkt.Len.Std	0.899934
Bwd.Pkt.Len.Mean	Pkt.Len.Std	0.849163
Bwd.Pkt.Len.Std	Pkt.Len.Std	0.937577
Pkt.Len.Max	Pkt.Len.Std	0.958434
Pkt.Len.Mean	Pkt.Len.Std	0.861471
Flow.IAT.Max	Idle.Mean	0.930967
Flow.IAT.Max	Idle.Std	0.940928
RST.Flag.Cnt	ECE.Flag.Cnt	0.999987
Flow.IAT.Max	Idle.Max	0.948219
Flow.IAT.Min	Idle.Mean	-0.93713
Flow.IAT.Min	Idle.Std	-0.92362
Flow.IAT.Min	Idle.Max	-0.92779
Flow.IAT.Std	Idle.Mean	0.974618
Flow.IAT.Std	Idle.Std	0.956771
Flow.IAT.Std	Idle.Max	0.956541
Fwd.Header.Len	Subflow.Fwd.Pkts	0.995575
Fwd.Header.Len	Fwd.Act.Data.Pkts	0.991472
Pkt.Size.Avg	Bwd.Seg.Size.Avg	0.938971
Tot.Fwd.Pkts	Subflow.Fwd.Pkts	1
Fwd.IAT.Max	Idle.Mean	0.930953
Fwd.IAT.Max	Idle.Std	0.940928
Tot.Bwd.Pkts	Subflow.Bwd.Pkts	1
TotLen.Bwd.Pkts	Subflow.Bwd.Pkts	0.993591
Fwd.IAT.Max	Idle.Max	0.948214
Tot.Bwd.Pkts	Subflow.Bwd.Byts	0.993591

Fwd.IAT.Min	Idle.Mean	-0.93711
Fwd.IAT.Min	Idle.Std	-0.92362
Subflow.Bwd.Pkts	Subflow.Bwd.Byts	0.993591
Fwd.IAT.Min	Idle.Max	-0.92778
Fwd.IAT.Std	Idle.Mean	0.974652
Fwd.IAT.Std	Idle.Std	0.956757
Active.Mean	Active.Max	0.947795
Active.Mean	Active.Min	0.907754
Fwd.IAT.Std	Idle.Max	0.956546
Fwd.Pkt.Len.Max	Fwd.Seg.Size.Avg	0.879368
Fwd.Pkt.Len.Mean	Fwd.Seg.Size.Avg	1
Fwd.Pkt.Len.Std	Fwd.Seg.Size.Avg	0.890569
Fwd.PSH.Flags	SYN.Flag.Cnt	1
Fwd.URG.Flags	CWE.Flag.Count	1
Pkt.Len.Mean	Pkt.Size.Avg	0.995625
Pkt.Len.Mean	Bwd.Seg.Size.Avg	0.943686
Pkt.Len.Std	Pkt.Size.Avg	0.853663
Pkt.Len.Std	Bwd.Seg.Size.Avg	0.849163
Protocol	Bwd.Pkt.Len.Min	0.851049
Subflow.Fwd.Pkts	Fwd.Act.Data.Pkts	0.999189
Idle.Mean	Idle.Std	0.980718
Tot.Bwd.Pkts	Bwd.Header.Len	0.997801
Tot.Fwd.Pkts	Fwd.Header.Len	0.995575
Tot.Fwd.Pkts	Fwd.Act.Data.Pkts	0.999189
TotLen.Bwd.Pkts	Bwd.Header.Len	0.996149
TotLen.Bwd.Pkts	Subflow.Bwd.Byts	1
TotLen.Fwd.Pkts	Subflow.Fwd.Byts	1
Idle.Mean	Idle.Max	0.981832
Idle.Std	Idle.Max	0.992293

In order to reduce the list and have fewer parameters to investigate, we removed rows that have variable name similarity of more than 70%. The below table shows the output result.

Table 17 Reduced Correlation Table

Row	Column	Correlation Coefficient
Bwd.Header.Len	Subflow.Bwd.Pkts	0.997801
Bwd.Header.Len	Subflow.Bwd.Byts	0.996151
Bwd.Pkt.Len.Mean	Pkt.Size.Avg	0.938971
Bwd.Pkt.Len.Mean	Bwd.Seg.Size.Avg	1
Flow.Duration	Fwd.IAT.Tot	0.999986
Flow.Duration	Fwd.IAT.Min	0.81448
Flow.IAT.Max	Idle.Mean	0.930967
Flow.IAT.Max	Idle.Std	0.940928
Flow.IAT.Max	Idle.Max	0.948219
Flow.IAT.Min	Idle.Mean	-0.93713
Flow.IAT.Min	Idle.Std	-0.92362
Flow.IAT.Min	Idle.Max	-0.92779
Flow.IAT.Std	Idle.Mean	0.974618
Flow.IAT.Std	Idle.Std	0.956771
Flow.IAT.Std	Idle.Max	0.956541
Fwd.Header.Len	Subflow.Fwd.Pkts	0.995575
Fwd.Header.Len	Fwd.Act.Data.Pkts	0.991472
Fwd.IAT.Max	Idle.Mean	0.930953
Fwd.IAT.Max	Idle.Std	0.940928
Fwd.IAT.Max	Idle.Max	0.948214
Fwd.IAT.Min	Idle.Mean	-0.93711
Fwd.IAT.Min	Idle.Std	-0.92362
Fwd.IAT.Min	Idle.Max	-0.92778
Fwd.IAT.Std	Idle.Mean	0.974652
Fwd.IAT.Std	Idle.Std	0.956757
Fwd.IAT.Std	Idle.Max	0.956546
Fwd.Pkt.Len.Max	Fwd.Seg.Size.Avg	0.879368
Fwd.Pkt.Len.Mean	Fwd.Seg.Size.Avg	1
Fwd.Pkt.Len.Std	Fwd.Seg.Size.Avg	0.890569
Fwd.PSH.Flags	SYN.Flag.Cnt	1
Fwd.URG.Flags	CWE.Flag.Count	1
Pkt.Len.Mean	Pkt.Size.Avg	0.995625
Pkt.Len.Mean	Bwd.Seg.Size.Avg	0.943686
Pkt.Len.Std	Pkt.Size.Avg	0.853663
Pkt.Len.Std	Bwd.Seg.Size.Avg	0.849163
Protocol	Bwd.Pkt.Len.Min	0.851049
Subflow.Fwd.Pkts	Fwd.Act.Data.Pkts	0.999189
Tot.Bwd.Pkts	Bwd.Header.Len	0.997801
Tot.Fwd.Pkts	Fwd.Header.Len	0.995575
Tot.Fwd.Pkts	Fwd.Act.Data.Pkts	0.999189
TotLen.Bwd.Pkts	Bwd.Header.Len	0.996149
TotLen.Bwd.Pkts	Subflow.Bwd.Byts	1
TotLen.Fwd.Pkts	Subflow.Fwd.Byts	1

From the table above, we will inspect and plot some of the related columns from the dataset. As we can see, the correlations had been coupled into groups so we could have a better understanding of the relations. At first, we inspect (Bwd.Header.Lenm and Subflow.Bwd.Pkts).

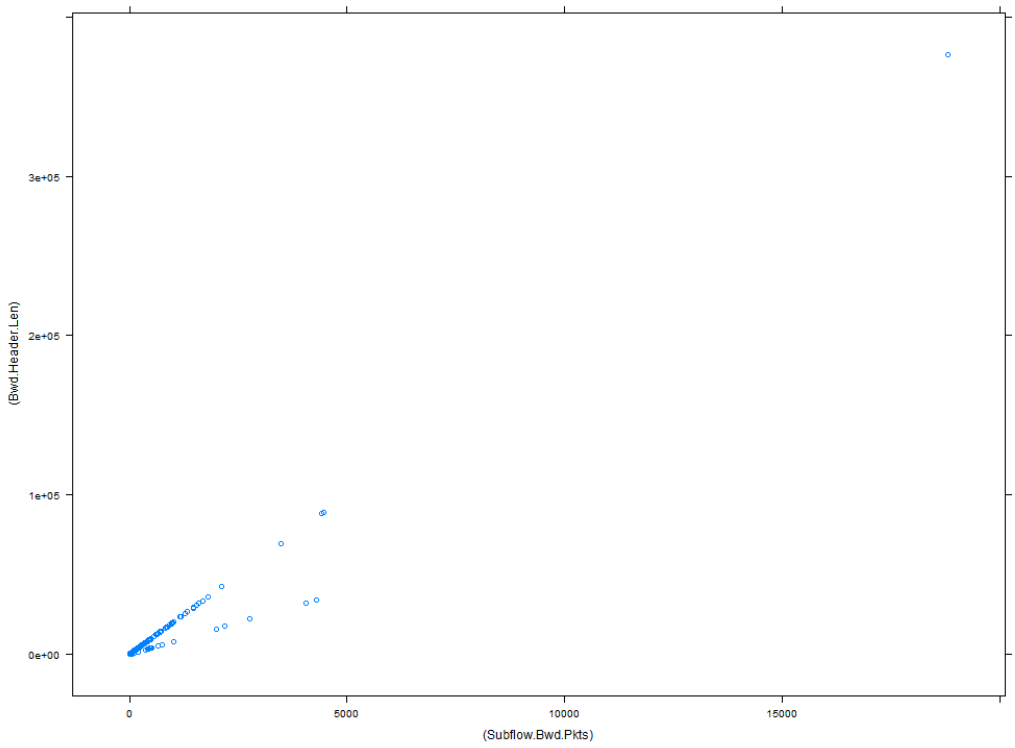


Figure 26 plot for Bwd.Header.Lenm vs Subflow.Bwd.Pkts

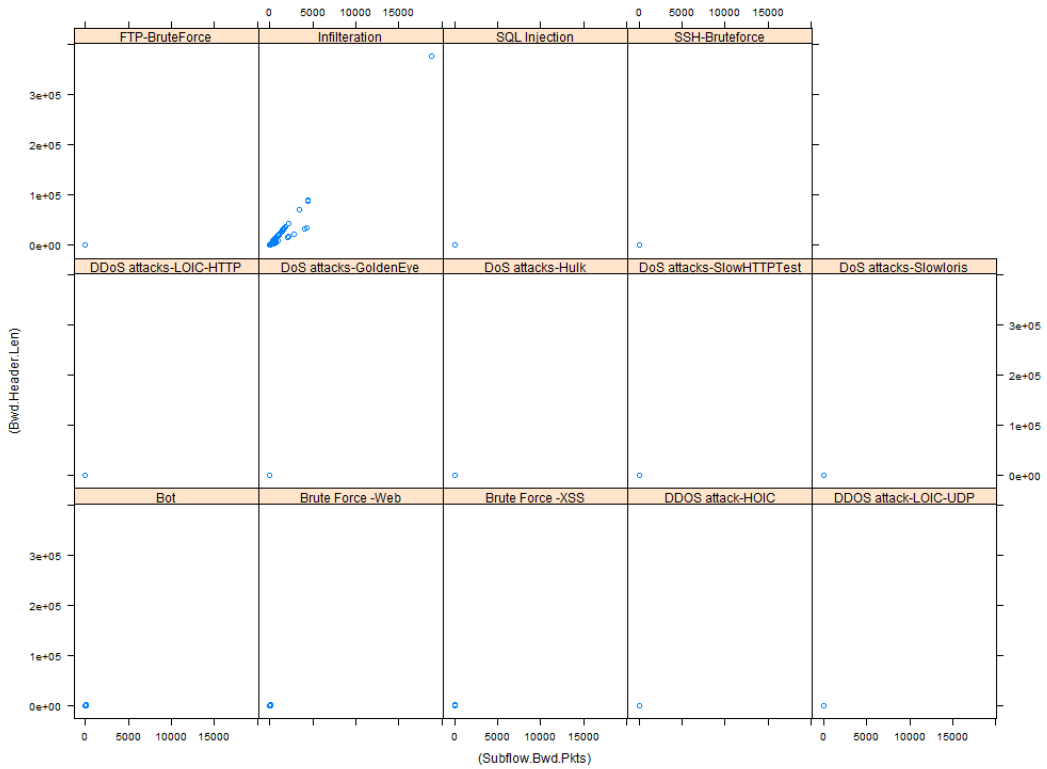


Figure 27 (Bwd.Header.Lenm vs Subflow.Bwd.Pkts) for each class

From the plot above, we can't have a lot of information or observation since most of the data density is on a very small spot on the plot. In order to have a better view of the plot, we removed the outliers so we can see if there is any trend or pattern.

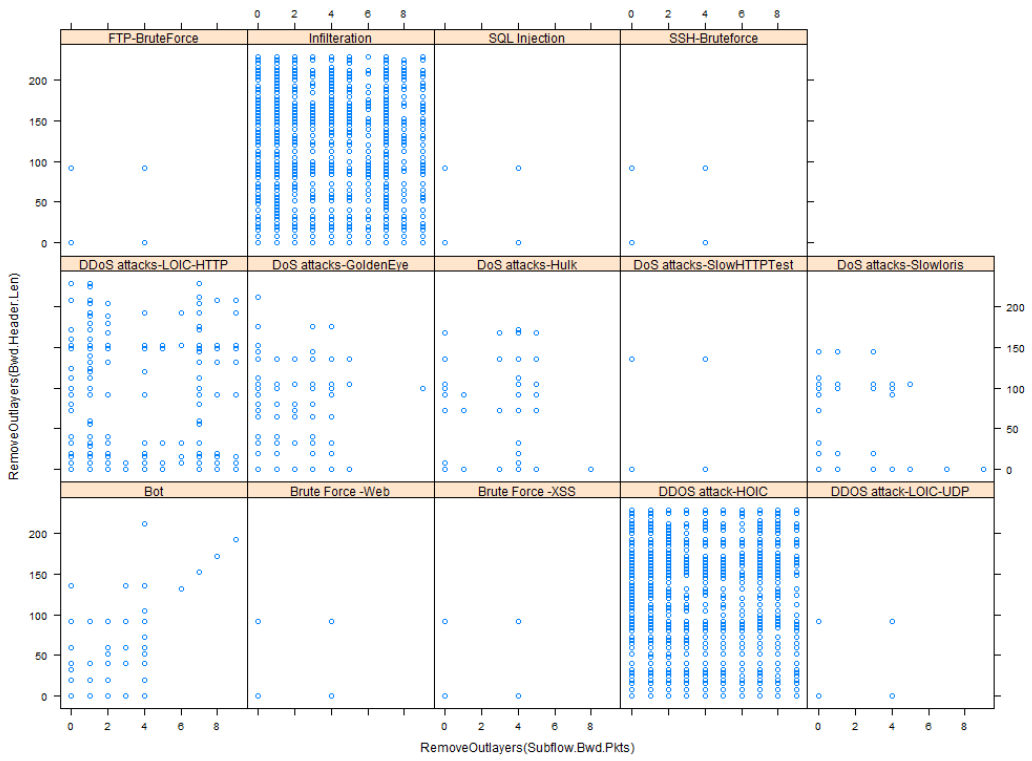


Figure 28 (Bwd.Header.Lenm vs Subflow.Bwd.Pkts) for each class after removing outliers

From the above plot, we can have a lot of observations for these two features. We can summarize them with the following: FTP- Bruteforce, SQL injection, SSH-Bruteforce, DoS attacks slowHTTPtest, Bruteforce -Web, BruteForce -XSS, and DDoS attack-LOIC-UDP are all restricted in four points, and if any data are in these points is suspected to be part of one of these attacks. The same is true for the other attacks; we can see some patterns and concentration of points in the plots that can help as indicators of attacks.

We have created other samples of relations that can be viewed for observation.

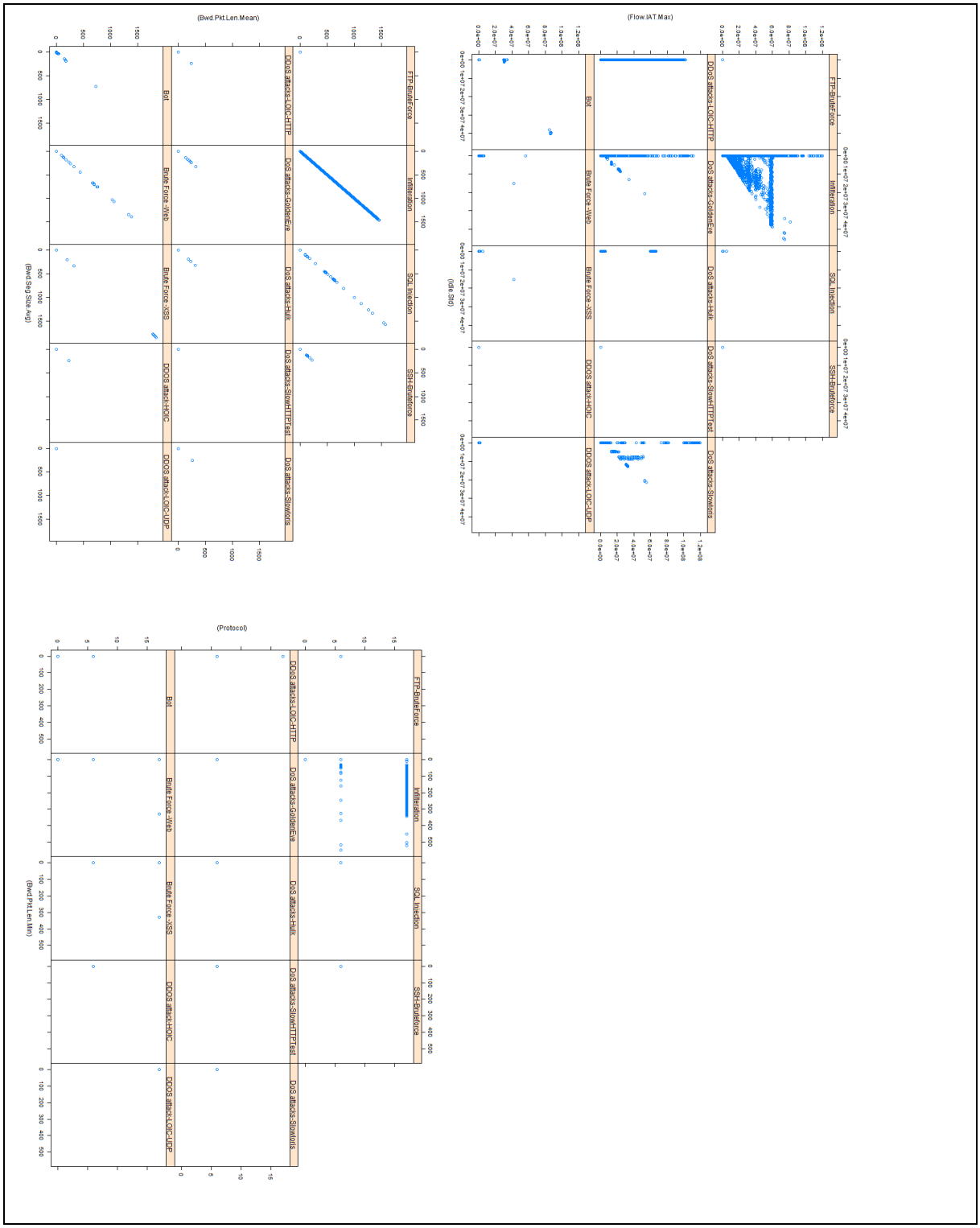


Figure 29 Sample plots

6.6 Discussion

The data exploration process was an essential phase in understanding the structure and behavior of the CIC-IDS2018 dataset. One of the most prominent issues identified was the significant class imbalance, where certain types of attacks, such as “Infiltration” and “Heartbleed,” were represented by only a few instances compared to much larger classes like “BENIGN” or “DoS Hulk.” This imbalance can lead to biased model training, where the classifier becomes more accurate at detecting frequent attack types but fails to properly recognize rare or emerging threats. On the other hand, the correlation matrix revealed that some features exhibited strong linear relationships, indicating potential redundancy. These relationships suggest that not all features are equally valuable for model training, and that dimensionality reduction techniques, such as feature selection or PCA, could improve overall performance and reduce overfitting. Additionally, the feature value distributions varied significantly across attributes, which supports the decision to normalize the dataset to bring features to a similar scale. Another key observation was that certain classes shared similar patterns in feature space, which could increase the risk of misclassification. These overlaps suggest the need for more flexible and instance-aware classifiers. Overall, the insights gained from this exploration directly informed the preprocessing strategies and helped shape the model-building steps in the upcoming chapters.

6.7 Chapter Conclusion

In conclusion, the data exploration process helped uncover essential characteristics of the CIC-IDS2018 dataset that directly influence how the classification models should be developed and evaluated. The analysis revealed several key issues, most

notably, the imbalance in class representation, where some attack types were underrepresented to the point that a default classifier might easily overlook them. This highlights the need for sampling or balancing techniques during preprocessing to avoid model bias. Additionally, the correlation analysis showed that many features are highly related, which means that some attributes could be removed without compromising the integrity of the dataset. This opens the door for dimensionality reduction techniques to simplify the model and improve training speed. Feature distribution plots also indicated the need for normalization to ensure that classifiers treat all attributes fairly. Beyond the technical findings, the exploration helped shape the direction of the modeling strategy by pointing out which features and issues to prioritize. It confirmed that CIC-IDS2018, while not perfect, is still highly suitable for building a robust multiclass intrusion detection system. The insights gathered here serve as a bridge into the model development phase, where a sub-sampled version of the dataset will be used to begin training and evaluating classification models.

Chapter 7 Building Models using Sub-Sample:

7.1 Chapter Introduction

In this chapter, we will go through building and testing different models. This process will have the model being built with the training and validation data. An additional test will be conducted using the test data portion of the dataset. Overall, this chapter will go through the following: fixing the data balance in the Dataset (CSE-CIC-IDS2018) by getting subsamples and upsampling. Then, we will build models and get their performance per class (Gradient Boosting Machine, Generalized Linear Models, Deep Learning Model, Random Forest, Distributed Random Forest). Finally, we will review some failed attempts and review the result.

7.2 Justification for using Sub-sampling

Due to the limitation of the local compute, memory, and fast storage, the logical approach is to use sub-sample of the dataset. In order to read the dataset and build multiple models on a standard workstation would not be feasible as the models would also require a large amount of memory. In addition the dataset has a very high dimensionality and the models that need to be built require a very high compute power. In the end, it was decided to test and start with sub-sample.ts.

7.3 Data Balance

During the investigation and exploration of the dataset. The presented data has major issues. The first issue is the imbalance of the classes, and the second issue is the size of the data. We needed to make sure that Data manipulation does not affect the accuracy of the actual test. So, the manipulations were done only on the training subset. The process was done with the following steps:

1- 60% was taken from the data. Because there is a huge gap between each class, we wanted to guarantee that each class is presented in the training, so we took 60% from each class instead of a random sample.

Table 18 60% sample from each class

Benign	Bot	Brute Force -Web
6126150	86721	331
Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP
136	119316	1038
DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk
345218	24843	87119
DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce
33	5944	31
Infilteration	SQL Injection	SSH-Bruteforce
83865	50	56428

2- There is a big gap between Benign and the rest of the classes, so we had to down-sample it.

Table 19 Down-sampling benign Class

Benign	Bot	Brute Force -Web
126150	86721	331
Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP
136	119316	1038
DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk
345218	24843	87119
DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce
33	5944	31
Infilteration	SQL Injection	SSH-Bruteforce
83865	50	56428

3- For the rest of the classes, we had to up-sample it. And then, we took a subset for each (10%)

Table 20 up sampling

Benign	Bot	Brute Force -Web	Brute Force -XSS
34521	34521	34521	34521
DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye
34521	34521	34521	34521
DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce
34521	34521	34521	34521
Infilteration	SQL Injection	SSH-Bruteforce	
34521	34521	34521	

Using the balanced data, we have tested it by building a model, which is a Random forest from Ranger. This package is part of the CRAN Package manager from RStudio. The results are as follows.

Table 21 Overall Accuracy

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue	McnemarPValue
0.94167415	0.93750802	0.94103238	0.94231097	0.06666667	0.00000000	NaN

Table 22 Ranger Training results per class

Class	precision	recall	f1
Benign	0.97023	0.941253	0.955522
Bot	0.99971	0.999942	0.999826
Brute Force -Web	0.996511	0.97642	0.986364
Brute Force -XSS	0.999768	1	0.999884
DDOS attack-HOIC	0.999768	0.999942	0.999855
DDOS attack-LOIC-UDP	0.999103	1	0.999551
DDoS attacks-LOIC-HTTP	0.99942	0.999073	0.999247
DoS attacks-GoldenEye	1	1	1
DoS attacks-Hulk	1	1	1
DoS attacks-SlowHTTPTest	0.585971	0.8184	0.682952
DoS attacks-Slowloris	1	1	1
FTP-BruteForce	0.699069	0.421859	0.526186
Infilteration	0.944691	0.96828	0.95634
SQL Injection	0.976742	1	0.988234
SSH-Bruteforce	1	0.999942	0.999971

From the multiclass results and overall results, we can see that the trained model outputs have great results where each class has a precision of either one or approaching one. We can proceed using the data for testing/validation.

The model reports the important variables. We have kept the important variables and removed the least important ones. The remaining used variables are the following.

Table 23 Remaining Features

Dst.Port	Flow.Duration	Tot.Fwd.Pkts	TotLen.Fwd.Pkts	Fwd.Pkt.Len.Max	Fwd.Pkt.Len.Mean
Fwd.Pkt.Len.Std	Flow.Byts.s	Flow.Pkts.s	Flow.IAT.Mean	Flow.IAT.Std	Flow.IAT.Max
Flow.IAT.Min	Fwd.IAT.Mean	Fwd.IAT.Std	Bwd.IAT.Min	Fwd.Header.Len	Fwd.Pkts.s
Bwd.Pkts.s	Fwd.Seg.Size.Avg	Subflow.Fwd.Pkts	Subflow.Fwd.Byts	Init.Fwd.Win.Byts	Fwd.Act.Data.Pkts
Fwd.Seg.Size.Min					

7.4 Test and validation with different classification models:

There were multiple models that we built in order to create a portfolio of models. The aim is to have a model that supports multiclassification. Some models may support multiclassification but in different frameworks or scripting languages. This research target is to validate the proposed methodology and algorithm using multiple classifiers for multi-classification. We opted to use tools that have support for multi-classification out of the box, where there is the least modification and alteration needed to build the model. Our search has resulted in the following Models that support both RStudio and H2O:

- Random Forest H2O
- Distributed Random forest from Ranger
- Deep Learning
- GLM
- GBM
- Naïve Bays (Failed attempt presented in the appendix)
- Support Vector Machine ((Failed attempt presented in the appendix)
- KNN (Failed attempt presented in the appendix)

7.4.1 Gradient Boosting Machine (GBM)

A gradient gradient-boosting machine is used for Classification. The function uses a forward learning ensemble method. Prediction quality increases through refined approximations. H2O's GBM will build regression trees with all features in a distributed manner. These regression trees would be built in parallel.

Model Parameters:

Table 24 GBM Model Parameters

Parameter	***Value***	***Description***
model_id	GBMModel	Destination id for this model; auto-generated if not specified.
nfolds	5	Number of folds for K-fold cross-validation (0 to disable or >= 2).
keep_cross_validation_predictions	true	Whether to keep the predictions of the cross-validation models.
fold_assignment	Random	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems.
response_column	Class	Response variable column.
ignored_columns		Names of columns to ignore for training.
r2_stopping	1.7976931348623157e+308	r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this
stopping_metric		Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anonomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client.
seed	1111	Seed for pseudo random number generator (if applicable)
distribution	multinomial	Distribution function

histogram_type	UniformAdaptive	What type of histogram to use for finding optimal split points
max_abs_leafnode_pred	1.7976931348623157e+308	Maximum absolute value of a leaf node prediction
categorical_encoding	Enum	Encoding scheme for categorical features

Building the GBM Matrix requires a Training set and validation set, and the framework will output a confusion matrix for each. We will examine both below.

Training Confusion Matrix:

Table 25 Training Confusion Matrix

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	25564	0	428	49	1	0	27	1	0	1	4	0	8417	29	0	0.2595	8,957 / 34,521	0.72
Bot	1	34519	0	0	0	0	0	0	0	0	0	0	1	0	0	0.0001	2 / 34,521	1
Brute Force -Web	0	0	33611	101	0	0	0	0	0	0	0	0	0	809	0	0.0264	910 / 34,521	0.95
Brute Force -XSS	0	0	1043	33231	0	0	0	0	0	0	0	0	0	247	0	0.0374	1,290 / 34,521	0.99
DDOS attack-HOIC	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	0	0	0 / 34,521	1
DDOS attack-LOIC-UDP	0	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	0	0 / 34,521	1
DDoS attacks-LOIC-HTTP	0	0	44	2	0	43	34429	0	0	0	0	0	0	3	0	0.0027	92 / 34,521	1
DoS attacks-GoldenEye	2	0	0	0	0	0	1	34510	0	0	8	0	0	0	0	0.0003	11 / 34,521	1
DoS attacks-Hulk	0	0	0	0	0	0	0	21	34500	0	0	0	0	0	0	0.0006	21 / 34,521	1
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	17826	0	16695	0	0	0	0.4836	16,695 / 34,521	1
DoS attacks-Slowloris	0	0	6	0	0	0	3	9	0	0	34503	0	0	0	0	0.0005	18 / 34,521	1
FTP-BruteForce	0	0	0	0	0	0	0	0	0	0	0	34521	0	0	0	0	0 / 34,521	0.67
Infilteration	9725	2	101	67	0	0	7	2	0	2	3	4	24568	40	0	0.2883	9,953 / 34,521	0.74
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	34521	0	0	0 / 34,521	0.97
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	1	3	0	0	34517	0.0001	4 / 34,521	1
Total	35292	34521	35233	33450	34522	34564	34467	34543	34500	17829	34519	51223	32986	35649	34517	0.0733	37,953 / 517,815	
Recall	0.74	1	0.97	0.96	1	1	1	1	1	0.52	1	1	0.71	1	1			

Cross-validation Matrix

Table 26 Cross-validation Matrix

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	25425	0	418	33	7	0	23	2	0	2	6	1	8571	33	0	0.2635	9,096 / 34,521	0.71
Bot	3	34516	0	0	0	0	0	0	0	0	0	0	2	0	0	0.0001	5 / 34,521	1
Brute Force -Web	97	0	33635	125	0	0	0	0	0	0	0	0	0	664	0	0.0257	886 / 34,521	0.97
Brute Force -XSS	0	0	403	33871	0	0	0	0	0	0	0	0	0	247	0	0.0188	650 / 34,521	0.99
DDOS attack-HOIC	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	0	0	0 / 34,521	1
DDOS attack-LOIC-UDP	0	0	0	0	0	34512	9	0	0	0	0	0	0	0	0	0.0003	9 / 34,521	1
DDoS attacks-LOIC-HTTP	3	0	36	1	0	44	34424	0	0	0	0	0	0	13	0	0.0028	97 / 34,521	1
DoS attacks-GoldenEye	0	0	1	0	0	0	1	34500	7	0	12	0	0	0	0	0.0006	21 / 34,521	1
DoS attacks-Hulk	1	0	0	0	0	0	0	26	34494	0	0	0	0	0	0	0.0008	27 / 34,521	1
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	19042	0	15479	0	0	0	0.4484	15,479 / 34,521	0.93
DoS attacks-Slowloris	0	0	6	0	0	0	1	11	0	0	34500	0	0	3	0	0.0006	21 / 34,521	1
FTP-BruteForce	0	0	0	0	0	0	0	0	0	1326	0	33195	0	0	0	0.0384	1,326 / 34,521	0.68
Infilteration	10091	4	126	39	1	0	14	4	0	9	6	4	24179	44	0	0.2996	10,342 / 34,521	0.74
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	34521	0	0	0 / 34,521	0.97
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	1	3	2	0	34515	0.0002	6 / 34,521	1
Total	35620	34520	34625	34069	34529	34556	34472	34543	34501	20379	34525	48682	32754	35525	34515	0.0733	37,965 / 517,815	
Recall	0.74	1	0.97	0.98	1	1	1	1	1	0.55	1	0.96	0.7	1	1			

Based on the precision of both confusion matrices (Training and Validation), we can see that they are almost identical, and there is no change in precision between the training and validation. We can notice that the precision for the benign class is around 0.7, which is considered very low since the dataset has a high constitution of the benign class. It is expected that the model will not perform well, even if the other classes have a high precision. The rate of misclassification on the benign side will be high.

Performance table with test Data:

Table 27 Performance and Overall Accuracy

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.7568898	0.4244448	0.7564380	0.7573412	0.8830848
AccuracyPValue	McnemarPValue			
1.0000000	NaN			

As expected, we discussed in the training and validation confusion matrices that the benign has around 0.7 precision, which will impact the overall performance of the model. We can see the impact in the above matrix, as the accuracy has a value around ~0.756

Performance Per-Class

Their performance per class has differences in the test data compared to training. The model failed to classify BruteForce -Web, BruteForce -XSS, DoS attacks-SlowHTTPtes, FTP-BruteForce, infiltration, and SQL-Injection.

Table 28 Per-Class Performance

Class	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.728964	0.969909	0.994565	0.32147	0.994565	0.728964	0.841299	0.883085	0.643737	0.647255	0.849436139
Class: Bot	0.999654	0.999902	0.992308	0.999996	0.992308	0.999654	0.995967	0.012501	0.012496	0.012593	0.999777982
Class: Brute Force -Web	0.993939	0.989203	0.00436	1	0.00436	0.993939	0.008682	4.76E-05	4.73E-05	0.010844	0.991571021
Class: Brute Force -XSS	0.970588	0.998606	0.013464	0.999999	0.013464	0.970588	0.026559	1.96E-05	1.90E-05	0.001413	0.984596994

Class: DDOS attack-HOIC	1	0.999913	0.995046	1	0.995046	1	0.997517	0.017199	0.017199	0.017285	0.999956438	
Class: DDOS attack-LOIC-UDP	1	0.999937	0.7023	1	0.7023	1	0.825119	0.00015	0.00015	0.000213	0.999968282	
Class: DDoS attacks-LOIC-HTTP	0.996999	0.999148	0.983945	0.999843	0.983945	0.996999	0.990429	0.049763	0.049614	0.050423	0.998073528	
Class: DoS attacks-GoldenEye	0.999597	0.999918	0.977638	0.999999	0.977638	0.999597	0.988496	0.003581	0.00358	0.003661	0.999757642	
Class: DoS attacks-Hulk	0.999242	0.9996	0.99684	0.99999	0.99684	0.999242	0.99804	0.012558	0.012549	0.012588	0.999601058	
Class: DoS attacks-SlowHTTPTest	0.1875	0.999992	0.096774	0.999996	0.096774	0.1875	0.12766	4.61E-06	8.65E-07	8.94E-06	0.593745964	
Class: DoS attacks-Slowloris	0.998991	0.999815	0.822438	0.999999	0.822438	0.998991	0.902157	0.000857	0.000856	0.001041	0.99940281	
Class: FTP-BruteForce	0.866667	0.999978	0.147727	0.999999	0.147727	0.866667	0.252427	4.32E-06	3.75E-06	2.54E-05	0.933322522	
Class: Infilteration	0.703162	0.772155	0.036391	0.995318	0.036391	0.703162	0.0692	0.012089	0.008501	0.233591	0.73765863	
Class: SQL Injection	0.92	0.999086	0.007206	0.999999	0.007206	0.92	0.014299	7.21E-06	6.63E-06	0.00092	0.959543185	
Class: SSH-Bruteforce	0.999716	0.999995	0.999362	0.999998	0.999362	0.999716	0.999539	0.008134	0.008132	0.008137	0.99985561	

Table 29 Confusion Matrix for the Test Data

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2232870	8	0	0	0	0	22	0	1	0	0	0	12172	0	0
Bot	332	43345	0	0	0	0	0	0	0	0	0	0	4	0	0
Brute Force -Web	37063	0	164	0	0	0	267	0	0	0	0	0	120	0	0
Brute Force -XSS	4749	0	0	66	0	0	6	0	0	0	0	0	80	1	0
DDOS attack-HOIC	295	0	0	0	59658	0	0	0	0	0	0	0	2	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	519	220	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	2789	0	0	0	0	0	172091	0	0	0	0	0	19	0	0
DoS attacks-GoldenEye	243	0	0	0	0	0	0	12416	32	0	3	0	6	0	0
DoS attacks-Hulk	136	0	0	0	0	0	0	0	43526	0	0	0	2	0	0

DoS attacks-SlowHTTPTest	24	0	0	0	0	0	0	0	0	3	0	2	2	0	0
DoS attacks-Slowloris	632	0	0	0	0	0	0	4	0	0	2969	0	3	1	1
FTP-BruteForce	58	0	0	0	0	0	0	0	0	13	0	13	3	0	1
Infilteration	780738	7	0	0	0	0	0	0	0	0	0	0	29485	0	6
SQL Injection	3129	0	1	2	0	0	3	0	0	0	0	0	34	23	0
SSH-Bruteforce	17	0	0	0	0	0	0	1	0	0	0	0	0	0	28206

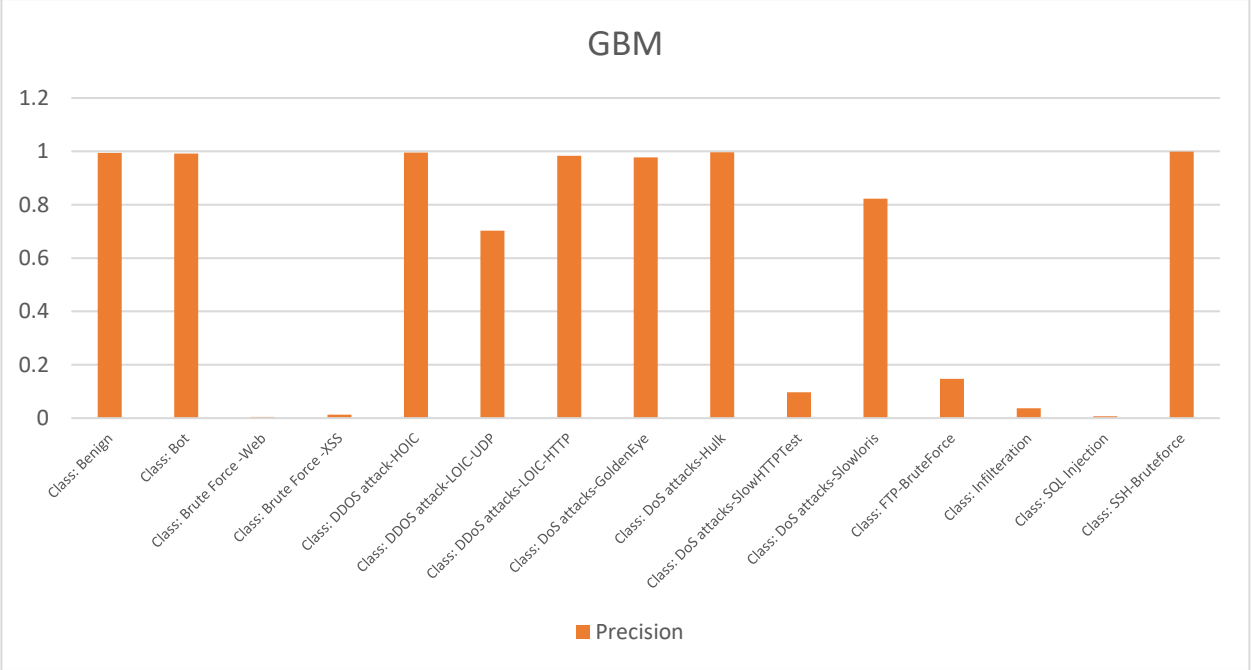


Figure 30 Per-Class precision

Depending on the performance metrics presented, we can note that the GBM Performs well in most of the classes, but there are still some classes that are either unpredicted or have a very low precision, which affects the overall performance of the Model.

7.4.2 Generalized Linear Models (GLM)

The Generalized Linear Model (GLM) from H2o is a regression model that targets exponential, Gaussian, binomial, and gamma distributions. The function can behave differently based on the requirements and type of prediction or classification.

Table 30 Model Parameters for GLM

Parameter	***Value***	***Description***
model_id	GLMModel	Destination id for this model; auto-generated if not specified.
seed	-8071143158777241479	Seed for pseudo random number generator (if applicable)
fold_assignment		Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems.
response_column	Class	Response variable column.
ignored_columns		Names of columns to ignore for training.
family	multinomial	Family. Use binomial for classification with logistic regression, others are for regression problems.
solver	IRLSM	AUTO will set the solver based on given data and the other parameters. IRLSM is fast on on problems with small number of predictors and for lambda-search with L1 penalty, L_BFGS scales better for datasets with many columns.
alpha	0.5	Distribution of regularization between the L1 (Lasso) and L2 (Ridge) penalties. A value of 1 for alpha represents Lasso regression, a value of 0 produces Ridge regression, and anything in between specifies the amount of mixing between the two. Default value of alpha is 0 when SOLVER = 'L-BFGS'; 0.5 otherwise.
lambda	0	Regularization strength
max_iterations	50	Maximum number of iterations
objective_epsilon	0.000001	Converge if objective value changes less than this. Default indicates: If lambda_search is set to True the value of objective_epsilon is set to .0001. If the lambda_search is set to False and lambda is equal to zero, the value of objective_epsilon is set to .000001, for any other value of lambda the default value of objective_epsilon is set to .0001.
gradient_epsilon	0.000001	Converge if objective changes less (using L-infinity norm) than this, ONLY applies to L-BFGS solver. Default indicates: If lambda_search is set to False and lambda is equal to zero, the default value of gradient_epsilon is equal to .000001, otherwise the default value is .0001. If lambda_search is set to True, the conditional values above are 1E-8 and 1E-6 respectively.

link	multinomial	Link function.
lambda_min_ratio	0.0001	Minimum lambda used in lambda search, specified as a ratio of lambda_max (the smallest lambda that drives all coefficients to zero). Default indicates: if the number of observations is greater than the number of variables, then lambda_min_ratio is set to 0.0001; if the number of observations is less than the number of variables, then lambda_min_ratio is set to 0.01.
max_active_predictors	5000	Maximum number of active predictors during computation. Use as a stopping criterion to prevent expensive model building with many predictors. Default indicates: If the IRLSM solver is used, the value of max_active_predictors is set to 5000 otherwise it is set to 100000000.
obj_reg	0.00000193119164180257 4	Likelihood divider in objective value computation, default is 1/nobs

Training Confusion Matrix

Table 31 Confusion Matrix for GLM

	Benign	Bot	Brute Force - Web	Brute Force - XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	14787	503	1178	784	189	0	2629	39	219	2	109	1	13585	485	11	**0.5717**	**19,734 / 34,521**	**0.59**
Bot	1043	32954	3	0	0	0	0	0	8	0	0	0	513	0	0	**0.0454**	**1,567 / 34,521**	**0.97**
Brute Force - Web	1150	0	15050	5778	0	0	1486	0	0	0	0	0	620	10437	0	**0.564**	**19,471 / 34,521**	**0.79**
Brute Force - XSS	0	0	1308	29302	0	0	1691	0	0	0	0	0	237	1983	0	**0.1512**	**5,219 / 34,521**	**0.67**
DDOS attack-HOIC	0	0	0	0	34514	0	0	0	0	0	0	0	4	3	0	**0.002**	**7 / 34,521**	**0.99**
DDOS attack-LOIC-UDP	0	0	0	0	0	34198	0	0	0	0	0	0	323	0	0	**0.0094**	**323 / 34,521**	**1**
DDoS attacks-LOIC-HTTP	0	0	0	2	0	44	34472	0	0	0	0	0	3	0	0	**0.014**	**49 / 34,521**	**0.82**
DoS attacks-GoldenEye	4	0	0	1	0	0	0	34234	197	0	85	0	0	0	0	**0.083**	**287 / 34,521**	**0.98**

DoS attacks-Hulk	0	0	28	1	0	0	0	112	34104	0	0	0	0	276	0	**0.0121**	**417 / 34,521**	**0.97**
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	28231	0	6290	0	0	0	**0.1822**	**6,290 / 34,521**	**0.5**
DoS attacks-Slowloris	0	0	0	46	0	0	0	342	256	28	33685	4	24	136	0	**0.0242**	**836 / 34,521**	**0.99**
FTP-BruteForce	0	0	0	0	0	0	0	0	0	27848	0	6673	0	0	0	**0.8067**	**27,848 / 34,521**	**0.51**
Infiltration	7954	493	639	3028	163	0	889	320	361	19	160	2	20030	440	23	**0.4198**	**14,491 / 34,521**	**0.56**
SQL Injection	0	0	749	4892	0	0	685	0	0	0	0	0	710	27485	0	**0.2038**	**7,036 / 34,521**	**0.67**
SSH-Bruteforce	0	0	0	0	0	0	0	0	60	1	0	3	0	9	34448	**0.0021**	**73 / 34,521**	**1**
Total	**24938**	**33950**	**18955**	**43834**	**34866**	**34242**	**41852**	**35047**	**35205**	**56129**	**34039**	**12973**	**36049**	**41254**	**34482**	**0.2002**	**103,648 / 517,815**	
Recall	**0.43**	**0.95**	**0.44**	**0.85**	**1**	**0.99**	**1**	**0.99**	**0.99**	**0.82**	**0.98**	**0.19**	**0.58**	**0.8**	**1**			

In the training confusion matrix, we can see that benign has a very low precision. This is considered a bad indicator, as it happened in GBM. The high percentage of the benign in the dataset will have an effect on the overall accuracy and precision. The other classes have good overall precision, except for FTP-BruteForce and SQL injection.

Overall Performance

Table 32 Overall Performance for the Model

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.4874235	0.2155967	0.4868973	0.4879497	0.8830848

AccuracyPValue	McnemarPValue
1.0000000	NaN

As expected from the training confusion matrix, the test classification accuracy has an overall accuracy of ~0.48. General overview of the results and inspecting the

precision of each class. We can conclude that the model is not performing very well. In the majority of the classes, the model fails in classification.

Performance Per-Class

Table 33 Performance per-class for GLM

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.42643	0.972663	0.991584	0.18335	0.991584	0.42643	0.596385	0.883085	0.376574	0.37977	0.699546
Class: Bot	0.952975	0.987287	0.486897	0.999397	0.486897	0.952975	0.644503	0.012501	0.011913	0.024467	0.970131
Class: Brute Force -Web	0.442424	0.970766	0.000719	0.999973	0.000719	0.442424	0.001437	4.76E-05	2.10E-05	0.029253	0.706595
Class: Brute Force -XSS	0.955882	0.978725	0.00088	0.999999	0.00088	0.955882	0.001759	1.96E-05	1.87E-05	0.021293	0.967304
Class: DDOS attack-HOIC	0.999883	0.994925	0.775172	0.999998	0.775172	0.999883	0.873304	0.017199	0.017197	0.022185	0.997404
Class: DDOS attack-LOIC-UDP	0.99422	0.999936	0.698241	0.999999	0.698241	0.99422	0.82035	0.00015	0.000149	0.000213	0.997078
Class: DDoS attacks-LOIC-HTTP	0.998499	0.928342	0.421873	0.999915	0.421873	0.998499	0.59314	0.049763	0.049689	0.117781	0.963421
Class: DoS attacks-GoldenEye	0.99211	0.998738	0.738656	0.999972	0.738656	0.99211	0.846825	0.003581	0.003553	0.00481	0.995424
Class: DoS attacks-Hulk	0.988292	0.993791	0.669346	0.99985	0.669346	0.988292	0.798135	0.012558	0.012411	0.018542	0.991041
Class: DoS attacks-SlowHTTPTest	0.75	0.999842	0.021429	0.999999	0.021429	0.75	0.041667	4.61E-06	3.46E-06	0.000161	0.874921
Class: DoS attacks-Slowloris	0.972746	0.996831	0.208405	0.999977	0.208405	0.972746	0.343268	0.000857	0.000833	0.003999	0.984789
Class: FTP-BruteForce	0.4	0.999981	0.082192	0.999997	0.082192	0.4	0.136364	4.32E-06	1.73E-06	2.10E-05	0.69999
Class: Infilteration	0.574215	0.647137	0.019524	0.992013	0.019524	0.574215	0.037765	0.012089	0.006942	0.355539	0.610676
Class: SQL Injection	0.8	0.986482	0.000426	0.999999	0.000426	0.8	0.000852	7.21E-06	5.77E-06	0.013524	0.893241
Class: SSH-Bruteforce	0.997377	0.999669	0.961098	0.999978	0.961098	0.997377	0.978902	0.008134	0.008113	0.008441	0.998523

Table 34 Confusion Matrix for GLM (Test Data)

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	1306186	42979	100553	69966	17129	2	235064	3833	20650	520	10763	61	1208399	45863	1107
Bot	1301	41321	0	0	0	0	0	0	14	0	0	0	724	0	0
Brute Force -Web	5	0	73	32	0	0	4	0	0	0	0	0	1	50	0
Brute Force -XSS	0	0	0	65	0	0	1	0	0	0	0	0	1	1	0
DDOS attack-HOIC	0	0	0	0	59651	0	0	0	0	0	0	0	5	2	0
DDOS attack-LOIC-UDP	0	0	0	0	0	516	0	0	0	0	0	0	3	0	0
DDoS attacks-LOIC-HTTP	0	0	0	25	0	220	172350	0	0	0	0	0	11	3	0
DoS attacks-GoldenEye	1	0	0	0	0	0	0	12323	66	0	31	0	0	0	0
DoS attacks-Hulk	0	0	31	2	0	0	0	116	43049	0	0	0	0	361	0
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	12	0	4	0	0	0
DoS attacks-Slowloris	0	0	0	3	0	0	0	36	25	2	2891	0	2	13	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	9	0	6	0	0	0
Infilteration	9779	566	810	3762	172	1	1116	375	456	16	187	2	24078	580	32
SQL Injection	0	0	1	3	0	0	0	0	0	0	0	0	1	20	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	55	1	0	0	1	17	28140

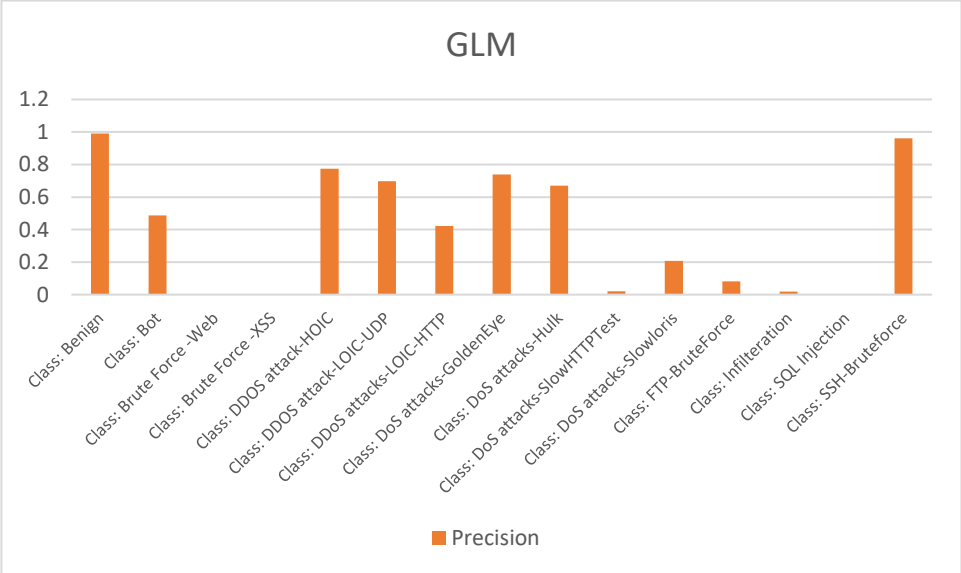


Figure 31 Per-Class Performance for GLM

We can observe that GLM has failed in multiple classes, which are very similar to the GBM model, but with additional classes that have very low precision, which is DoS attacks-Sloworis.

7.4.3 Deep Learning (Neural Networks)

The package used for the Deep neural network is from h2o in Rstudio. This package uses the feedforward method artificial network. The network is capable of having a huge number of hidden layers that have neurons, rectifiers, and activation functions. This function has a lot of features that assist the training of models in deep learning. Some of them are adaptive learning and grid search, which can increase and facilitate a high accuracy with the model predictions and classifications. Finally, the function is highly optimized as it uses multithreading (async), which allows utilization of the system resources. Similar to GBM, in order to build the model, we have to provide Training and validation Data. In building the model, we used the below inputs.

Table 35 Model Parameters for DeepLearning

Parameter	***Value***	***Description***
model_id	NNModel	Destination id for this model; auto-generated if not specified.
nfolds	15	Number of folds for K-fold cross-validation (0 to disable or >= 2).
fold_assignment	Random	Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems.
response_column	Class	Response variable column.
ignored_columns		Names of columns to ignore for training.
score_each_iteration	true	Whether to score during each iteration of model training.
overwrite_with_best_model	false	If enabled, override the final model with the best model found during training.

activation	RectifierWithDropout	Activation function.
hidden	200, 200	Hidden layer sizes (e.g. [100, 100]).
epochs	3.540479626199161	How many times the dataset should be iterated (streamed), can be fractional.
seed	42	Seed for random numbers (affects sampling) - Note: only reproducible when running single threaded.
distribution	multinomial	Distribution function
stopping_rounds	0	Early stopping based on convergence of stopping_metric. Stop if simple moving average of length k of the stopping_metric does not improve for k:=stopping_rounds scoring events (0 to disable)
stopping_metric		Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anonomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client.
export_weights_and_biases	true	Whether to export Neural Network weights and biases to H2O Frames.
categorical_encoding	OneHotInternal	Encoding scheme for categorical features

Training Confusion Matrix:

Table 36 Training Confusion Matrix for Deep Learning

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	560	27	14	39	7	4	49	2	0	0	0	0	0	0	0	**0.2023**	**142 / 702**	**0.54**
Bot	2	647	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.0031**	**2 / 649**	**0.89**
Brute Force -Web	0	0	441	186	0	0	0	0	0	0	0	0	0	0	0	**0.2967**	**186 / 627**	**0.52**
Brute Force -XSS	0	0	14	664	0	0	0	0	0	0	0	0	0	0	0	**0.0206**	**14 / 678**	**0.52**
DDOS attack-HOIC	0	0	0	0	667	0	0	0	0	0	0	0	0	0	0	**0**	**0 / 667**	**0.98**
DDOS attack-LOIC-UDP	1	0	0	0	0	631	0	0	0	0	0	0	0	0	0	**0.0016**	**1 / 632**	**0.98**
DDoS attacks-LOIC-HTTP	0	0	0	4	0	2	664	0	0	0	0	0	0	0	0	**0.009**	**6 / 670**	**0.92**
DoS attacks-GoldenEye	0	0	0	0	0	0	0	653	0	0	1	0	0	0	0	**0.0015**	**1 / 654**	**0.96**
DoS attacks-Hulk	0	0	1	0	0	0	0	3	652	0	0	0	0	0	0	**0.0061**	**4 / 656**	**0.98**
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	655	0	0	0	0	0	**0**	**0 / 655**	**0.5**
DoS attacks-Slowloris	8	0	2	4	0	0	0	13	4	0	628	0	0	0	0	**0.047**	**31 / 659**	**1**
FTP-BruteForce	0	0	0	0	0	0	0	0	0	658	0	0	0	0	0	**1**	**658 / 658**	**NaN**
Infilteration	475	50	4	83	7	8	7	9	7	0	0	0	0	0	1	**1**	**651 / 651**	**NaN**
SQL Injection	0	0	365	301	0	0	0	0	0	0	0	0	0	0	0	**1**	**666 / 666**	**NaN**
SSH-Bruteforce	0	0	0	0	0	0	0	0	3	0	0	0	0	0	681	**0.0044**	**3 / 684**	**1**
Total	**1046**	**724**	**841**	**1281**	**681**	**645**	**720**	**680**	**666**	**1313**	**629**	**0**	**0**	**0**	**682**	**0.2387**	**2,365 / 9,908**	
Recall	**0.81**	**1**	**0.7**	**0.98**	**1**	**1**	**0.99**	**1**	**0.99**	**1**	**0.95**	**0**	**0**	**0**	**1**			

Cross-validation Matrix

Table 37 Cross-validation Matrix for Deep Learning

	Benign	Bot	Brute Force - Web	Brute Force - XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	825646	34870	16112	54141	11156	3430	68167	5877	611	32	763	0	20	0	200	0.1914	195,379 / 1,021,025	0.99
Bot	25	14424	0	0	0	4	0	0	0	0	0	0	0	0	0	0.002	29 / 14,453	0.29
Brute Force - Web	0	0	43	12	0	0	0	0	0	0	0	0	0	0	0	0.2182	Dec-55	0
Brute Force -XSS	0	0	1	21	0	0	0	0	0	0	0	0	0	0	0	0.0455	22-Jan	0
DDOS attack-HOIC	0	0	0	0	19886	0	0	0	0	0	0	0	0	0	0	0	0 / 19,886	0.64
DDOS attack-LOIC-UDP	1	0	0	0	0	172	0	0	0	0	0	0	0	0	0	0.0058	1 / 173	0.04
DDoS attacks-LOIC-HTTP	6	0	0	336	5	79	57110	0	0	0	0	0	0	0	0	0.0074	426 / 57,536	0.45
DoS attacks-GoldenEye	1	0	0	0	0	0	0	4136	0	0	3	0	0	0	0	0.001	4 / 4,140	0.4
DoS attacks-Hulk	0	0	2	1	0	0	0	54	14457	0	5	0	0	0	0	0.0043	62 / 14,519	0.95
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0 / 5	0.1
DoS attacks-Slowloris	15	0	0	4	1	2	0	15	8	0	945	0	0	0	0	0.0455	45 / 990	0.55
FTP-BruteForce	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	1	5-May	NaN
Infilteration	9979	1288	145	1702	76	222	275	179	83	8	11	0	2	0	7	0.9999	13,975 / 13,977	0.09
SQL Injection	0	0	7	1	0	0	0	0	0	0	0	0	0	0	0	1	8-Aug	NaN
SSH-Bruteforce	0	0	0	0	0	2	0	0	13	1	0	0	0	0	9388	0.0017	16 / 9,404	0.98
Total	835673	50582	16310	56218	31124	3911	125552	10261	15172	51	1727	0	22	0	9595	0.1816	209,963 / 1,156,198	
Recall	0.81	1	0.78	0.95	1	0.99	0.99	1	1	1	0.95	0	0	0	1			

The Training and Validation Matrices have different values in precision for each class. Some classes had a very low precision, especially Benign, which has a precision of 0.5, but staringly, it got ~0.9 in the validation matrix. Some classes were not detected at all in both Matrices. Anyway, we will have the model test on the test Data.

Performance table with test Data:

Table 38 Overall Performance

Accuracy	Kappa	AccuracyLower	AccuracyUpper
AccuracyNull			
0.8178779	0.4877315	0.8174714	0.8182840
0.8830848			
AccuracyPValue	McnemarPValue		
1.0000000	NaN		

The overall accuracy of the model is performing well in comparison to the Training metrics. Even though the model has failed in many classes, as we can see in the table below. However, the high accuracy value may be attributed to the high precision of the benign class.

Table 39 Per-Class Accuracy for Deep Learning

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.808039	0.925905	0.988005	0.389719	0.988005	0.808039	0.889006	0.883085	0.713567	0.722229	0.866972
Class: Bot	0.997763	0.968186	0.284188	0.999971	0.284188	0.997763	0.442376	0.012501	0.012473	0.043889	0.982974
Class: Brute Force -Web	0.751515	0.98606	0.002558	0.999988	0.002558	0.751515	0.005099	4.76E-05	3.57E-05	0.013975	0.868788
Class: Brute Force -XSS	0.985294	0.950931	0.000394	1	0.000394	0.985294	0.000787	1.96E-05	1.93E-05	0.049088	0.968112
Class: DDOS attack-HOIC	0.999933	0.990087	0.638373	0.999999	0.638373	0.999933	0.779256	0.017199	0.017198	0.026941	0.99501
Class: DDOS attack-LOIC-UDP	0.996146	0.996709	0.043329	0.999999	0.043329	0.996146	0.083046	0.00015	0.000149	0.00344	0.996427

Class: DDoS attacks-LOIC-HTTP	0.992776	0.937587	0.454451	0.999597	0.454451	0.992776	0.623493	0.049763	0.049404	0.108711	0.965181
Class: DoS attacks-GoldenEye	0.998712	0.99481	0.408826	0.999995	0.408826	0.998712	0.580161	0.003581	0.003576	0.008748	0.996761
Class: DoS attacks-Hulk	0.996602	0.999353	0.951411	0.999957	0.951411	0.996602	0.973483	0.012558	0.012515	0.013155	0.997978
Class: DoS attacks-SlowHTTPTest	1	0.999958	0.1	1	0.1	1	0.181818	4.61E-06	4.61E-06	4.61E-05	0.999979
Class: DoS attacks-Slowloris	0.955249	0.999355	0.559519	0.999962	0.559519	0.955249	0.705692	0.000857	0.000818	0.001463	0.977302
Class: FTP-BruteForce	0	1	NA	0.999996	NA	0	NA	4.32E-06	0	0	0.5
Class: Infilteration	0.000405	0.999986	0.261538	0.987916	0.261538	0.000405	0.0008	0.012089	4.90E-06	1.87E-05	0.500196
Class: SQL Injection	0	1	NA	0.999993	NA	0	NA	7.21E-06	0	0	0.5
Class: SSH-Bruteforce	0.997377	0.999814	0.977796	0.999978	0.977796	0.997377	0.9874	0.008134	0.008113	0.008297	0.998596

Table 40 Confusion Matrix for Deep Learning with Test Data

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2475083	104951	47917	164170	33535	10512	204877	17177	1886	122	2180	0	46	0	619
Bot	81	43263	0	0	0	15	0	1	0	0	0	0	0	0	0
Brute Force -Web	0	0	124	41	0	0	0	0	0	0	0	0	0	0	0
Brute Force -XSS	0	0	1	67	0	0	0	0	0	0	0	0	0	0	0
DDOS attack-HOIC	0	0	2	0	59654	2	0	0	0	0	0	0	0	0	0
DDOS attack-LOIC-UDP	2	0	0	0	0	517	0	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	33	0	0	979	14	221	171362	0	0	0	0	0	0	0	0
DoS attacks-GoldenEye	1	0	0	0	0	0	0	12405	3	0	12	0	0	0	0
DoS attacks-Hulk	2	0	6	13	0	0	0	117	43411	0	10	0	0	0	0
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0
DoS attacks-Slowloris	40	0	6	10	5	0	0	53	17	0	2839	0	2	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0
Infilteration	29889	4020	407	4972	239	658	836	590	245	6	33	0	17	0	20

SQL Injection	0	0	11	14	0	0	0	0	0	0	0	0	0	0	0
SSH-Bruteforce	0	0	0	0	0	7	0	0	66	1	0	0	0	0	28140

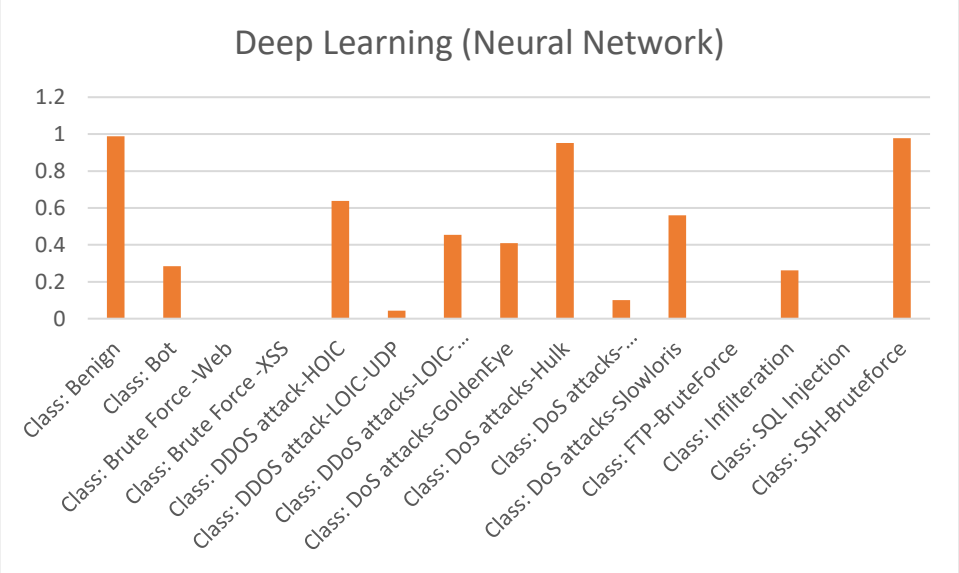


Figure 32 Per-Class Performance for Deep Learning

As seen in the figure above, the deep learning model has failed many classes and could not classify them with 0 precision. Even though the model's overall accuracy is over ~0.8, which means that the model can perform on the majority of the data, it will misclassify minor classes.

7.4.4 Random Forest (Ranger)

Ranger is one of the fastest implementations for RStudio that exists in standard CRAN packages. The Ranger function is extremely fast, and it can handle large data with high dimensionality even in commodity hardware, as the algorithm is highly optimized. The algorithm can have different functions based on the scenario as it supports classification, regression, and prediction.

The model has followed the same procedure as the previous models, and we have used the parameters below to build the model.

Table 41 Parameter Inputs for Random Forest (Ranger)

Type:	Classification
Number of trees:	500
Sample size:	517815
Number of independent variables:	70
Mtry:	8
Target node size:	1
Variable importance mode:	impurity
Splitrule:	gini
OOB prediction error:	7.37 %

Table 42 Confusion Matrix for Random Forest (Ranger) - Training Data

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-Goldeneye	DoS attacks-Hulk	DoS attacks-Slowhttptest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Precision
Benign	333555	0	0	0	0	0	4	0	0	0	0	0	390	0	0	0.99
Bot	0	34521	0	0	0	0	0	0	0	0	0	0	0	0	0	1.00
Brute Force -Web	12	0	34012	0	0	0	0	0	0	0	0	0	1	0	0	1.00
Brute Force -XSS	0	0	0	34274	0	0	0	0	0	0	0	0	0	0	0	1.00
DDOS attack-HOIC	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	0	1.00
DDOS attack-LOIC-UDP	0	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	1.00

DDoS attacks-LOIC-HTTP	2	0	0	0	0	0	34517	0	0	0	0	0	0	0	0	1.00
DoS attacks-GoldenEye	0	0	0	0	0	0	0	34521	0	0	0	0	0	0	0	1.00
DoS attacks-Hulk	0	0	0	0	0	0	0	0	34521	0	0	0	0	0	0	1.00
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	14692	0	0	1	0	0	1.00
DoS attacks-Slowloris	0	0	0	0	0	0	0	0	0	0	34521	0	1	0	0	1.00
FTP-BruteForce	0	0	0	0	0	0	0	0	0	19829	0	34521	0	0	3	0.64
Infilteration	1151	0	0	0	0	0	0	0	0	0	0	0	34128	0	0	0.97
SQL Injection	1	0	509	247	0	0	0	0	0	0	0	0	0	34521	0	0.98
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34518	1.00

On the training set, we can see that the model is very promising, and the per-class precision is around ~1 for almost all the classes except for the FTP-brute force. We need to validate these metrics with actual test data because we might have an overfitting issue with results mostly approaching 1.

Performance table with test Data:

Table 43 Overall Performance for Random Forest (Ranger)

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.7893861	0.4687281	0.7889566	0.7898151	0.8830848
AccuracyPValue	McnemarPValue			
1.0000000	NaN			

With the test data, we have some changes that we can immediately notice. First, the accuracy is around ~0.79., and the majority of the classes have a precision approaching 1, but now we have more classes that have less precision, which are BruteForce-Web, BruteForce-XSS, FTP-BruteForce, and Infiltration. It's important to

note that the results of the training phase should not be used as an indicator, and testing must always be conducted.

Table 44 Per-Class Performance for Random Forest (Ranger)

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.764046	0.981373	0.996783	0.355109	0.996783	0.764046	0.865034	0.883085	0.674718	0.676895	0.872709
Class: Bot	0.999862	0.999941	0.995362	0.999998	0.995362	0.999862	0.997607	0.012501	0.012499	0.012557	0.999901
Class: Brute Force -Web	0.981818	0.999416	0.074074	0.999999	0.074074	0.981818	0.137755	4.76E-05	4.67E-05	0.000631	0.990617
Class: Brute Force -XSS	0.926471	0.999893	0.144828	0.999999	0.144828	0.926471	0.250497	1.96E-05	1.82E-05	0.000125	0.963182
Class: DDOS attack-HOIC	1	0.999807	0.989107	1	0.989107	1	0.994524	0.017199	0.017199	0.017389	0.999904
Class: DDOS attack-LOIC-UDP	1	0.999945	0.732017	1	0.732017	1	0.845277	0.00015	0.00015	0.000204	0.999973
Class: DDoS attacks-LOIC-HTTP	0.998494	0.9998	0.990159	0.999921	0.990159	0.998494	0.994309	0.049763	0.049688	0.050182	0.998987
Class: DoS attacks-GoldenEye	1	0.999964	0.990037	1	0.990037	1	0.994993	0.003581	0.003581	0.003617	0.999982
Class: DoS attacks-Hulk	0.999908	0.999985	0.998808	0.999999	0.998808	0.999908	0.999358	0.012558	0.012557	0.012572	0.999946
Class: DoS attacks-SlowHTTPTest	0.125	0.999998	0.25	0.999996	0.25	0.125	0.166667	4.61E-06	5.77E-07	2.31E-06	0.562499
Class: DoS attacks-Slowloris	1	0.999931	0.925857	1	0.925857	1	0.961501	0.000857	0.000857	0.000925	0.999966
Class: FTP-BruteForce	0.866667	0.999992	0.325	0.999999	0.325	0.866667	0.472727	4.32E-06	3.75E-06	1.15E-05	0.933329
Class: Infilteration	0.821234	0.790726	0.04582	0.997241	0.04582	0.821234	0.086797	0.012089	0.009928	0.216672	0.80598
Class: SQL Injection	0.92	0.999937	0.095436	0.999999	0.095436	0.92	0.172932	7.21E-06	6.63E-06	6.95E-05	0.959969
Class: SSH-Bruteforce	0.999965	0.999987	0.998408	1	0.998408	0.999965	0.999185	0.008134	0.008134	0.008147	0.999976

Table 45 Confusion Matrix for Test Data for Random Forest (Ranger)

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2340331	202	2013	372	653	5	1713	119	51	3	236	10	717110	212	45
Bot	6	43354	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	1	0	162	0	0	0	0	0	0	0	0	0	0	2	0
Brute Force -XSS	1	0	3	63	0	0	0	0	0	0	0	0	0	1	0
DDOS attack-HOIC	0	0	0	0	59658	0	0	0	0	0	0	0	0	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	519	0	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	71	0	1	0	0	185	172349	0	0	0	0	0	3	0	0
DoS attacks-GoldenEye	0	0	0	0	0	0	0	12421	0	0	0	0	0	0	0
DoS attacks-Hulk	0	0	0	0	0	0	0	4	43555	0	0	0	0	0	0
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	2	0	14	0	0	0
DoS attacks-Slowloris	0	0	0	0	0	0	0	0	0	0	2972	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	2	0	13	0	0	0
Infilteration	7475	0	6	0	4	0	0	2	1	1	2	2	34436	3	0
SQL Injection	0	0	2	0	0	0	0	0	0	0	0	0	0	23	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	0	1	0	0	28213

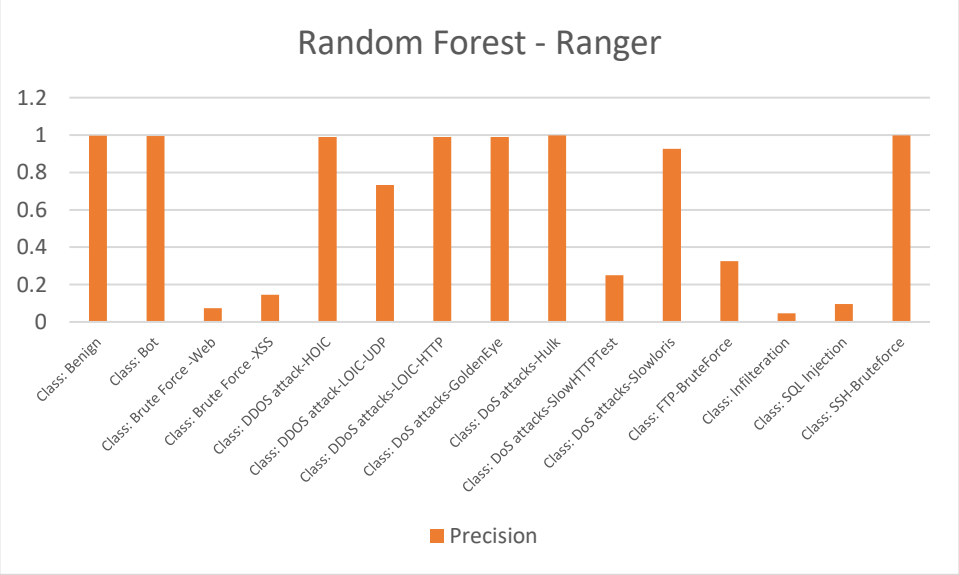


Figure 33 Per-class Performance for Random Forest (Ranger)

As discussed earlier, the Random Forest Model performed very well in most of the classes, but in some classes, the model failed with a very low precision.

7.4.5 Distributed Random Forest (DRF)

One of the powerful classification Models is the distributed Random Forest (DRF). When the function is invoked, the DRF will generate a forest and its primary tasks classification or regression. The more trees that the function generates, the less variance will be present in the results.

Model Parameters:

Table 46 Model Parameters for DRF

Parameter	***Value***	***Description***
model_id	RFModel	Destination id for this model; auto-generated if not specified.
fold_assignment		Cross-validation fold assignment scheme, if fold_column is not specified. The 'Stratified' option will stratify the folds based on the response variable, for classification problems.
response_column	Class	Response variable column.
ignored_columns		Names of columns to ignore for training.
ntrees	100	Number of trees.
max_depth	500	Maximum tree depth (0 for unlimited).
min_rows	10	Fewest allowed (weighted) observations in a leaf.
r2_stopping	1.7976931348623157e+308	r2_stopping is no longer supported and will be ignored if set - please use stopping_rounds, stopping_metric and stopping_tolerance instead. Previous version of H2O would stop making trees when the R^2 metric equals or exceeds this
stopping_metric		Metric to use for early stopping (AUTO: logloss for classification, deviance for regression and anonomaly_score for Isolation Forest). Note that custom and custom_increasing can only be used in GBM and DRF with the Python client.
seed	-5640707449536764199	Seed for pseudo random number generator (if applicable)
binomial_double_trees	true	For binary classification: Build 2x as many trees (one per class) - can lead to higher accuracy.
histogram_type	UniformAdaptive	What type of histogram to use for finding optimal split points

categorical_encoding	Enum	Encoding scheme for categorical features
calibration_frame	ValidateSampel_sid_903c_23	Calibration frame for Platt Scaling
distribution	multinomial	Distribution function

Training Confusion Matrix:

Table 47 Training Confusion Matrix for DRF

	Benign	Bot	Brute Force - Web	Brute Force - XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	25745	2	466	10	9	0	27	0	4	0	9	0	8227	22	0	0.2542	8,776 / 34,521	0.77
Bot	7	34512	0	0	0	0	0	0	0	0	0	0	2	0	0	0.0003	9 / 34,521	1
Brute Force - Web	0	0	33542	195	0	0	0	0	0	0	0	0	0	784	0	0.0284	979 / 34,521	0.95
Brute Force - XSS	0	0	1043	33231	0	0	0	0	0	0	0	0	0	247	0	0.0374	1,290 / 34,521	0.99
DDOS attack-HOIC	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	0	0	0 / 34,521	1
DDOS attack-LOIC-UDP	0	0	0	0	0	34521	0	0	0	0	0	0	0	0	0	0	0 / 34,521	1
DDoS attacks-LOIC-HTTP	15	0	28	1	0	46	34425	0	0	0	0	0	1	5	0	0.0028	96 / 34,521	1
DoS attacks-GoldenEye	0	0	0	1	0	0	0	34500	13	0	7	0	0	0	0	0.0006	21 / 34,521	1
DoS attacks-Hulk	0	0	0	0	0	0	0	22	34499	0	0	0	0	0	0	0.0006	22 / 34,521	1
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	18743	0	15778	0	0	0	0.4571	15,778 / 34,521	0.95
DoS attacks-Slowloris	0	0	1	0	0	0	0	3	0	0	34510	0	6	1	0	0.0003	11 / 34,521	1
FTP-BruteForce	0	0	0	0	0	0	0	0	0	946	0	33575	0	0	0	0.0274	946 / 34,521	0.68
Infilteration	7583	1	218	13	2	4	15	4	0	3	0	0	26663	15	0	0.2276	7,858 / 34,521	0.76
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	34521	0	0	0 / 34,521	0.97
SSH-Bruteforce	0	0	0	0	0	0	0	0	4	0	0	3	0	0	34514	0.0002	7 / 34,521	1
Total	33350	34515	35298	33451	34532	34571	34467	34529	34520	19692	34526	49356	34899	35595	34514	0.0691	35,793 / 517,815	
Recall	0.75	1	0.97	0.96	1	1	1	1	1	0.54	1	0.97	0.77	1	1			

Validation Matrix:

Table 48 Validation Matrix for DRF

	Benign	Bot	Brute Force - Web	Brute Force - XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce	Error	Rate	Precision
Benign	753269	78	12996	222	209	8	1024	80	128	0	477	1	251893	638	2	**0.2622**	**267,756 / 1,021,025**	**1**
Bot	4	14448	0	0	0	0	0	0	0	0	0	0	1	0	0	**0.0003**	**5 / 14,453**	**0.99**
Brute Force - Web	0	0	55	0	0	0	0	0	0	0	0	0	0	0	0	**0**	**0 / 55**	**0**
Brute Force - XSS	0	0	1	20	0	0	0	0	0	0	0	0	0	1	0	**0.0909**	**22-Feb**	**0.08**
DDOS attack-HOIC	0	0	0	0	19886	0	0	0	0	0	0	0	0	0	0	**0**	**0 / 19,886**	**0.99**
DDOS attack-LOIC-UDP	0	0	0	0	0	173	0	0	0	0	0	0	0	0	0	**0**	**0 / 173**	**0.65**
DDoS attacks-LOIC-HTTP	39	0	46	0	0	85	57362	0	0	0	0	0	3	1	0	**0.003**	**174 / 57,536**	**0.98**
DoS attacks-GoldenEye	0	0	0	0	0	0	0	4139	0	0	1	0	0	0	0	**0.0002**	**1 / 4,140**	**0.98**
DoS attacks-Hulk	0	0	0	0	0	0	0	7	14512	0	0	0	0	0	0	**0.0005**	**7 / 14,519**	**0.99**
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	3	0	2	0	0	0	**0.4**	**5-Feb**	**0.5**
DoS attacks-Slowloris	0	0	0	0	0	0	0	0	0	0	989	0	1	0	0	**0.001**	**1 / 990**	**0.67**
FTP-BruteForce	0	0	0	0	0	0	0	0	0	2	0	3	0	0	0	**0.4**	**5-Feb**	**0.33**
Infilteration	3228	1	114	2	3	0	10	5	0	1	1	2	10606	4	0	**0.2412**	**3,371 / 13,977**	**0.04**
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	**0**	**0 / 8**	**0.01**
SSH-Bruteforce	0	0	0	0	0	0	0	0	1	0	0	1	0	0	9402	**0.0002**	**2 / 9,404**	**1**
Total	**756540**	**14527**	**13212**	**244**	**20098**	**266**	**58396**	**4231**	**14641**	**6**	**1468**	**9**	**262504**	**652**	**9404**	**0.2347**	**271,323 / 1,156,198**	
Recall	**0.74**	**1**	**1**	**0.91**	**1**	**1**	**1**	**1**	**1**	**0.6**	**1**	**0.6**	**0.76**	**1**	**1**			

After observing the Training and the validation confusion matrices, we see a lot of contradictions. The first one is that the training matrix has almost all the classes have a precision of almost 1, except for the benign, where the precision is around ~0.77.

In contrast, the validation matrix has benign precision with a value of 1, and the rest of the classes also approach 1, except for DoS attacks-SlowHTTPTest, DoS attacks-Slowloris, FTP-BruteForce, Infiltration, and SQL Injection. The precision of these classes is very low or almost 0.

Overall Performance with Test Data:

After Inspecting the overall accuracy of the mode and looking at the precision for each class, we say that the model is performing well in most of the classes, but in some classes, it failed to predict, and that impacts the overall accuracy, which is ~0.76. If we observe the precision for each class, we can see that the model has the majority of the classes with precision approaching 1, except for Brute Force -Web, Brute Force -XSS, DoS attacks-SlowHTTPTest, DoS attacks-Slowloris, FTP-BruteForce, Infiltration, and SQL Injection. The precision for these classes is low or approaching zero.

Table 49 Overall Accuracy for DRF with Test Data

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.7656437	0.4366761	0.7651976	0.7660894	0.8830848
AccuracyPValue	McnemarPValue			
1.0000000	NaN			

Table 50 Per-Class Performance for DRF with Test Data

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.738044	0.976115	0.995734	0.330358	0.995734	0.738044	0.847739	0.883085	0.651756	0.654548	0.85708
Class: Bot	0.999493	0.999922	0.9939	0.999994	0.9939	0.999493	0.996688	0.012501	0.012494	0.012571	0.999707
Class: Brute Force -Web	0.981818	0.988682	0.00411	0.999999	0.00411	0.981818	0.008186	4.76E-05	4.67E-05	0.011364	0.98525
Class: Brute Force -XSS	0.926471	0.999785	0.077874	0.999999	0.077874	0.926471	0.143672	1.96E-05	1.82E-05	0.000233	0.963128
Class: DDOS attack-HOIC	1	0.99981	0.989271	1	0.989271	1	0.994607	0.017199	0.017199	0.017386	0.999905
Class: DDOS attack-LOIC-UDP	1	0.99992	0.65283	1	0.65283	1	0.789954	0.00015	0.00015	0.000229	0.99996
Class: DDoS attacks-LOIC-HTTP	0.997051	0.999008	0.981359	0.999845	0.981359	0.997051	0.989143	0.049763	0.049616	0.050559	0.99803
Class: DoS attacks-GoldenEye	0.999758	0.999935	0.982204	0.999999	0.982204	0.999758	0.990903	0.003581	0.00358	0.003645	0.999847
Class: DoS attacks-Hulk	0.999656	0.999879	0.990582	0.999996	0.990582	0.999656	0.995098	0.012558	0.012554	0.012673	0.999767
Class: DoS attacks-SlowHTTPTest	0.1875	0.999997	0.230769	0.999996	0.230769	0.1875	0.206897	4.61E-06	8.65E-07	3.75E-06	0.593749
Class: DoS attacks-Slowloris	1	0.999631	0.69913	1	0.69913	1	0.822927	0.000857	0.000857	0.001226	0.999815
Class: FTP-BruteForce	0.866667	0.999992	0.317073	0.999999	0.317073	0.866667	0.464286	4.32E-06	3.75E-06	1.18E-05	0.933329
Class: Infilteration	0.76357	0.779726	0.040693	0.996303	0.040693	0.76357	0.077268	0.012089	0.009231	0.226841	0.771648
Class: SQL Injection	0.92	0.999433	0.011569	0.999999	0.011569	0.92	0.022851	7.21E-06	6.63E-06	0.000573	0.959717
Class: SSH-Bruteforce	0.999575	0.999995	0.999362	0.999997	0.999362	0.999575	0.999468	0.008134	0.008131	0.008136	0.999785

Table 51 Confusion Matrix (DRF) for Test Data

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2260685	21	0	0	0	0	89	0	0	0	0	0	9576	0	0
Bot	263	43338	0	0	0	0	0	0	0	0	0	0	3	0	0
Brute Force -Web	38812	0	162	3	0	0	158	0	0	0	0	0	281	1	0
Brute Force -XSS	740	0	1	63	0	0	0	0	0	0	0	0	4	1	0
DDOS attack-HOIC	644	0	0	0	59658	0	0	0	0	0	0	0	3	0	0
DDOS attack-LOIC-UDP	28	0	0	0	0	519	246	0	0	0	0	0	2	0	0
DDoS attacks-LOIC-HTTP	3250	0	0	0	0	0	172100	0	0	0	0	0	19	0	0
DoS attacks-GoldenEye	204	0	0	0	0	0	0	12418	15	0	0	0	6	0	0
DoS attacks-Hulk	398	0	0	0	0	0	0	3	43544	0	0	0	2	0	11
DoS attacks-SlowHTTPTest	6	0	0	0	0	0	0	0	0	3	0	2	2	0	0
DoS attacks-Slowloris	1279	0	0	0	0	0	0	0	0	0	2972	0	0	0	0
FTP-BruteForce	12	0	0	0	0	0	0	0	0	13	0	13	2	0	1
Infilteration	754799	1	0	0	0	0	6	0	0	0	0	0	32018	0	0
SQL Injection	1937	0	2	2	0	0	10	0	0	0	0	0	14	23	0
SSH-Bruteforce	18	0	0	0	0	0	0	0	0	0	0	0	0	0	28202

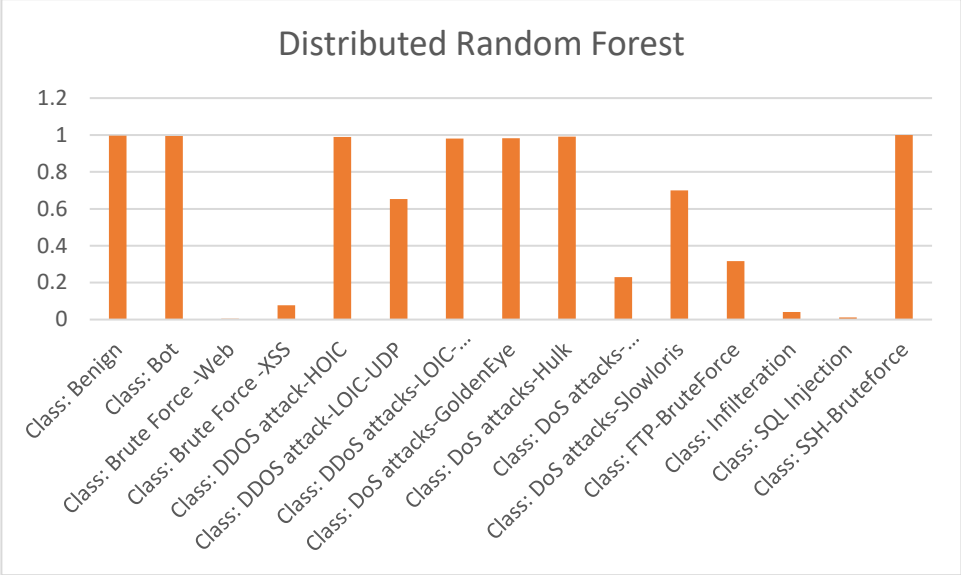


Figure 34 Performance for DRF

As discussed earlier, we can see on the graph above that the Model is performing well except for some classes that were totally misclassified or have a very low precision.

7.4.6 Portfolio Classifier (Random Forest)

As discussed earlier and explained, the portfolio classifier is built using multiple classifiers; we will follow the pseudo-code below for the model build.

7.4.6.1 Build mode:

We will follow the pseudo-code that is below to build the portfolio classifier. For more details, please refer to the source code in the appendix.

```
Load Models (GBM,GLM,DeepLearning,RF,DRF)

Load Training Data ->TD

For each Model:

        Model.predict(TD)->Predictions

Convert Predictions to binary:

        For each prediction:

                If prediction is True:

                        Add Model Digit as 1

                Else

                        Add Model Digit as 0

Add Binary Codes to the TD

Build Model to predict the binary codes->Portfolio Classifier
```

Figure 35 Build Master Model

7.4.6.2 Classify using the portfolio Model.

The same goes for prediction. We will use the following step to retrieve the prediction from the model.

```
Load Models (GBM,GLM,DeepLearning,RF,DRF)

Load Portfolio Classifier

Load test Data->TD

Portfolio Classifier.predict (TD)->Binary Prediction

For each in binary prediction:

        Identify the model for each digit

        If the digit is 1:

                Model.predict(TD)

        Else :

                Ignore Model

Get vote for successful models that can participate in the vote.
```

Figure 36Classification Using Master Model

Please note that these pseudo codes are abstract, and a detailed code can be found in the appendix, along with an explanation of the methodology.

7.4.6.3 Performance

Inspecting the overall accuracy of the portfolio multi-classifier, we can see that there is a slight increase, but the increase is insignificant.

Table 52 Overall Results

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull
0.7977272	0.4796524	0.7973041	0.7981499	0.8830833

AccuracyPValue	McnemarPValue
1.0000000	NaN

Table 53 Per-Class Results

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.773789	0.979249	0.996462	0.364325	0.996462	0.773789	0.871121	0.883083	0.68332	0.685746	0.876519
Class: Bot	0.999815	0.999943	0.995522	0.999998	0.995522	0.999815	0.997664	0.012501	0.012499	0.012555	0.999879
Class: Brute Force -Web	0.981818	0.999269	0.060089	0.999999	0.060089	0.981818	0.113247	4.76E-05	4.67E-05	0.000777	0.990544
Class: Brute Force -XSS	0.926471	0.999855	0.111504	0.999999	0.111504	0.926471	0.199052	1.96E-05	1.82E-05	0.000163	0.963163
Class: DDOS attack-HOIC	1	0.999725	0.98452	1	0.98452	1	0.9922	0.0172	0.0172	0.01747	0.999862
Class: DDOS attack-LOIC-UDP	1	0.999941	0.715862	1	0.715862	1	0.834405	0.00015	0.00015	0.000209	0.99997
Class: DDoS attacks-LOIC-HTTP	0.998308	0.999441	0.989418	0.999911	0.989418	0.998308	0.993843	0.049764	0.04968	0.050211	0.998875
Class: DoS attacks-GoldenEye	1	0.999948	0.985872	1	0.985872	1	0.992886	0.003581	0.003581	0.003632	0.999974
Class: DoS attacks-Hulk	0.999839	0.999985	0.998807	0.999998	0.998807	0.999839	0.999323	0.012558	0.012556	0.012571	0.999912
Class: DoS attacks-SlowHTTPTest	0.1875	0.999994	0.130435	0.999996	0.130435	0.1875	0.153846	4.61E-06	8.65E-07	6.63E-06	0.593747
Class: DoS attacks-Slowloris	1	0.999943	0.937539	1	0.937539	1	0.967763	0.000857	0.000857	0.000914	0.999971
Class: FTP-BruteForce	0.866667	0.999991	0.288889	0.999999	0.288889	0.866667	0.433333	4.32E-06	3.75E-06	1.30E-05	0.933329
Class: Infilteration	0.80052	0.799749	0.046634	0.996957	0.046634	0.80052	0.088135	0.012088	0.009677	0.207507	0.800134
Class: SQL Injection	0.88	0.999922	0.075601	0.999999	0.075601	0.88	0.139241	7.21E-06	6.34E-06	8.39E-05	0.939961
Class: SSH-Bruteforce	0.999716	0.999992	0.999044	0.999998	0.999044	0.999716	0.99938	0.008134	0.008132	0.00814	0.999854

Table 54 Master Classifier - Confusion Matrix

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2370123	7	1	1	0	0	76	0	0	0	0	0	8330	0	0
Bot	195	43352	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	2500	0	162	3	0	0	17	0	0	0	0	0	12	2	0
Brute Force -XSS	499	0	0	63	0	0	0	0	0	0	0	0	2	1	0
DDOS attack-HOIC	933	0	0	0	59658	0	0	0	0	0	0	0	5	0	0
DDOS attack-LOIC-UDP	6	0	0	0	0	519	199	0	0	0	0	0	1	0	0
DDoS attacks-LOIC-HTTP	1842	0	0	0	0	0	172317	0	0	0	0	0	1	0	0
DoS attacks-GoldenEye	169	0	0	0	0	0	0	12421	7	0	0	0	2	0	0
DoS attacks-Hulk	46	0	0	0	0	0	0	0	43552	0	0	0	2	0	4
DoS attacks-SlowHTTPTest	17	0	0	0	0	0	0	0	0	3	0	2	1	0	0
DoS attacks-Slowloris	198	0	0	0	0	0	0	0	0	0	2972	0	0	0	0
FTP-BruteForce	15	0	0	0	0	0	0	0	0	13	0	13	3	0	1
Infilteration	686178	1	0	0	0	0	0	0	0	0	0	0	33565	0	3
SQL Injection	261	0	2	1	0	0	0	0	0	0	0	0	5	22	0
SSH-Bruteforce	27	0	0	0	0	0	0	0	0	0	0	0	0	0	28206

If we view the precision for each class, we can see that the portfolio classifier has most of the classes approaching 1. However, the model still follows the same trend, as there are some classes that have been misclassified, and they match the models in the portfolio.

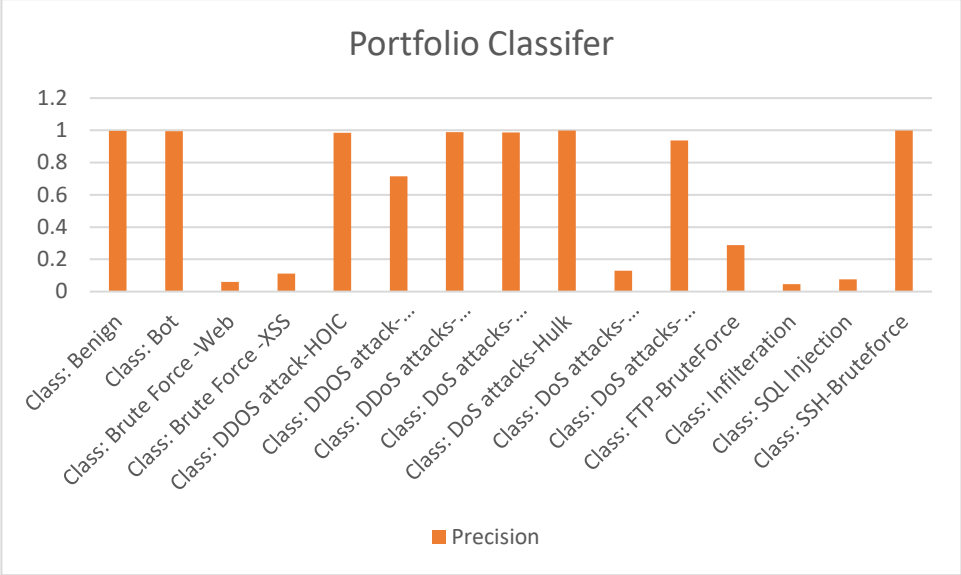


Figure 37Precision for each class (Portfolio Classifier)

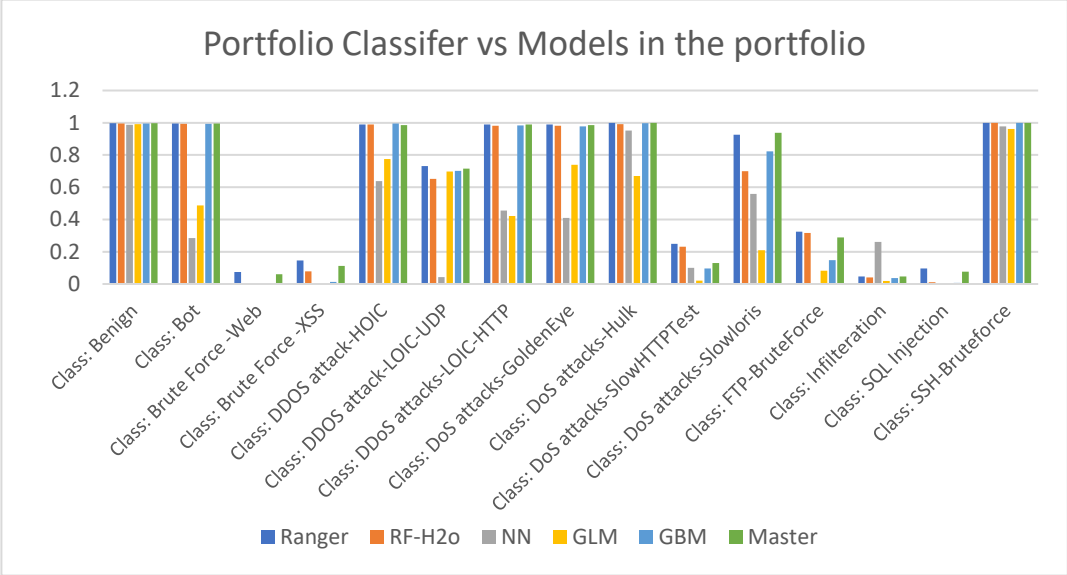


Figure 38 Precision Comparison Between All Models including Portflio Classifier

Comparing the Portfolio classier to the other models that we have built. We note the portfolio classifier outperforms, and in almost every class, the portfolio classifier is either ranked one or very close to 1. There are a few exceptions that we can see in the graph where the portfolio classifier is underperforming.

7.5 Discussion

This chapter introduced the first experimental phase of the research, focusing on training and evaluating models using a sub-sample of the CIC-IDS2018 dataset. The goal was to test the core classifiers independently and assess how well they perform on a controlled, smaller dataset before scaling up. The results demonstrated a clear difference in performance between models, with algorithms such as Random Forest and Deep Learning achieving stronger accuracy and F1 scores, especially in handling multiclass predictions. Meanwhile, models like Generalized Linear Models (GLM) and Gradient Boosting Machine (GBM) showed relatively weaker performance on minority attack types. These results reinforce the idea that no single classifier performs best across all scenarios. Additionally, the per-instance voting mechanism, where multiple models are consulted for classification, showed early signs of improving precision and recall by leveraging each model's strengths. This discussion also highlighted how the sub-sample allowed for fast iteration and early testing of the hybrid model design without requiring full-scale computational resources. The performance variations across attack classes further support the proposed per-instance selection strategy, as it offers flexibility and adaptability when dealing with diverse traffic patterns in real-world networks.

7.6 Chapter Conclusion

In summary, this chapter evaluated the performance of several individual classifiers on a sub-sampled version of the CIC-IDS2018 dataset. The experiments confirmed that different models excel in different areas, with some providing better overall accuracy, while others performed better on underrepresented classes. These results validated one of the key premises of the research: that no single model is sufficient

for optimal detection in a multiclass intrusion environment. The findings also emphasized the importance of model diversity and the value of a dynamic selection strategy, where classifiers can be chosen based on the characteristics of each instance. The sub-sampling approach proved useful for early-stage model comparison, allowing for efficient testing of design assumptions and baseline performance without the overhead of full dataset training. The next chapter will extend this work by applying the same modeling techniques to the complete dataset, offering a more robust evaluation of classifier performance under realistic data conditions.

Chapter 8 Building Models with a Complete Dataset

8.1 Chapter Introduction

After attempting to use a reduced subset of the dataset for training, it was noticed that the results could be unreliable, and many of the models were not able to produce reasonable predictions. The justification for using a subset of the dataset for training was the lack of computing and memory resources to build the required models. In order to eliminate the lack of computing resources, we opted to use cloud services from Huawei. There are AI services available in Amazon, Microsoft Azure.. etc., but we chose to provision a Virtual Machine in the cloud with the same tools that were used in the local computer for the first tests. The reason that we used a virtual machine was to use the same script that was already done on the local machine. In this part, we have done the same process but using the complete dataset for training and testing without getting smaller subsets for training that will affect the performance of the produced models. At first, we have built all models that will participate in the portfolio.

8.2 Gradient Boosting Machine (GBM)

We have built a model using the GBM package from h2o in Rstudio, as done in the previous attempt with the full dataset, and these are the parameters that were used to build this model.

Table 55 Performance for GBM

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.9878105	0.9411469	0.9876833	0.9879367	0.8829398	0.0000000
McnemarPValue					
NaN					

Table 56 Per-Class Performance for GBM

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.999608	0.8992	0.986807	0.996724	0.986807	0.999608	0.993166	0.88294	0.882594	0.894393	0.949404
Class: Bot	0.9984	1	0.999972	0.99998	0.999972	0.9984	0.999186	0.012544	0.012524	0.012524	0.9992
Class: Brute Force -Web	0.405594	1	0.983051	0.999971	0.983051	0.405594	0.574257	4.95E-05	2.01E-05	2.04E-05	0.702797
Class: Brute Force -XSS	0.893617	0.999999	0.933333	0.999998	0.933333	0.893617	0.913043	1.63E-05	1.45E-05	1.56E-05	0.946808
Class: DDOS attack-HOIC	1	1	1	1	1	1	1	0.017081	0.017081	0.017081	1
Class: DDOS attack-LOIC-UDP	0.953437	0.999975	0.858283	0.999993	0.858283	0.953437	0.903361	0.000156	0.000149	0.000173	0.976706
Class: DDoS attacks-LOIC-HTTP	0.99849	0.999985	0.999709	0.999921	0.999709	0.99849	0.999099	0.049942	0.049866	0.049881	0.999237
Class: DoS attacks-GoldenEye	0.999418	0.999998	0.999322	0.999998	0.999322	0.999418	0.99937	0.003569	0.003567	0.003569	0.999708
Class: DoS attacks-Hulk	0.999972	0.999995	0.999642	1	0.999642	0.999972	0.999807	0.012547	0.012547	0.012551	0.999984

Class: DoS attacks-SlowHTTPTest	0.578947	0.999997	0.52381	0.999997	0.52381	0.578947	0.55	6.57E-06	3.81E-06	7.27E-06	0.789472
Class: DoS attacks-Slowloris	0.994045	0.999999	0.999202	0.999995	0.999202	0.994045	0.996617	0.000871	0.000866	0.000867	0.997022
Class: FTP-BruteForce	0.3125	0.999997	0.384615	0.999996	0.384615	0.3125	0.344828	5.54E-06	1.73E-06	4.50E-06	0.656249
Class: Infilteration	0.035185	0.999662	0.56102	0.988304	0.56102	0.035185	0.066217	0.012114	0.000426	0.00076	0.517424
Class: SQL Injection	0.419355	1	0.928571	0.999994	0.928571	0.419355	0.577778	1.07E-05	4.50E-06	4.84E-06	0.709677
Class: SSH-Bruteforce	0.999788	0.999999	0.99983	0.999998	0.99983	0.999788	0.999809	0.008147	0.008145	0.008147	0.999893

Confusion Matrix

Table 57Confusion Matrix For GBM

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2551149	58	80	3	0	1	147	4	0	0	15	0	33783	14	2
Bot	1	36201	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	1	0	58	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -XSS	0	0	0	42	0	0	0	0	0	0	0	0	0	3	0
DDOS attack-HOIC	0	0	0	0	49374	0	0	0	0	0	0	0	0	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	430	71	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	15	0	4	2	0	20	144139	0	0	0	0	0	0	1	0
DoS attacks-GoldenEye	5	0	0	0	0	0	0	10310	1	0	0	0	0	0	1
DoS attacks-Hulk	9	0	0	0	0	0	0	2	36267	0	0	0	0	0	2
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	11	0	10	0	0	0
DoS attacks-Slowloris	2	0	0	0	0	0	0	0	0	0	2504	0	0	0	0

FTP-BruteForce	0	0	0	0	0	0	0	0	0	8	0	5	0	0	0
Infilteration	964	0	0	0	0	0	0	0	0	0	0	0	1232	0	0
SQL Injection	0	0	1	0	0	0	0	0	0	0	0	0	0	13	0
SSH-Bruteforce	3	0	0	0	0	0	0	0	0	0	0	1	0	0	23544

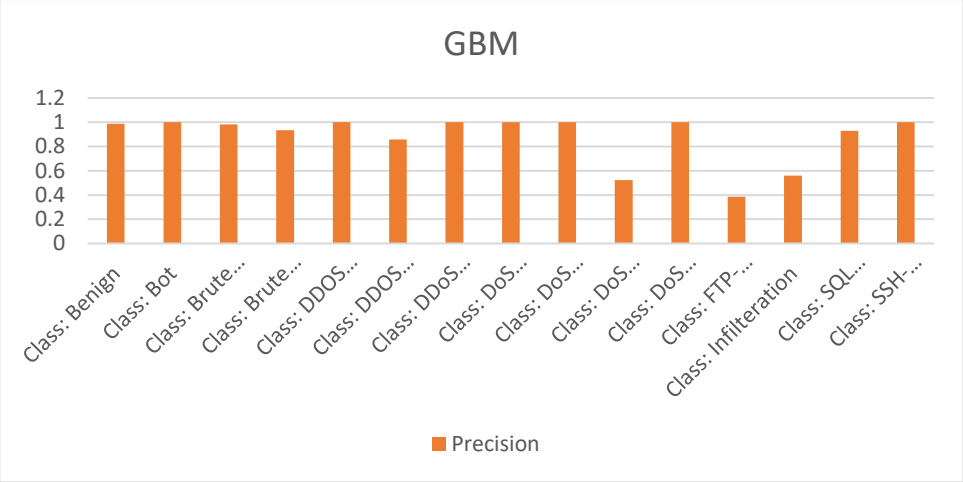


Figure 39 Per-Class precision for GBM

8.3 Generalized Linear Models (GLM)

Overall

Table 58 Overall Performance for GLM

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.9777974	0.8908473	0.9776269	0.9779669	0.8829398	0.0000000

McnemarPValue
NaN

Table 59 Per-Class Performance for GLM

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.997068	0.842743	0.979518	0.974426	0.979518	0.997068	0.988215	0.88294	0.880351	0.898759	0.919905
Class: Bot	0.984804	0.999888	0.991091	0.999807	0.991091	0.984804	0.987937	0.012544	0.012354	0.012465	0.992346
Class: Brute Force -Web	0	0.999997	0	0.999951	0	0	NA	4.95E-05	0	3.11E-06	0.499998
Class: Brute Force -XSS	0.468085	1	1	0.999991	1	0.468085	0.637681	1.63E-05	7.61E-06	7.61E-06	0.734043
Class: DDOS attack-HOIC	0.824402	0.998284	0.893062	0.996952	0.893062	0.824402	0.85736	0.017081	0.014082	0.015768	0.911343
Class: DDOS attack-LOIC-UDP	0.97561	0.999938	0.710824	0.999996	0.710824	0.97561	0.82243	0.000156	0.000152	0.000214	0.987774
Class: DDoS attacks-LOIC-HTTP	0.940162	0.999616	0.992287	0.996863	0.992287	0.940162	0.965521	0.049942	0.046953	0.047318	0.969889
Class: DoS attacks-GoldenEye	0.714424	0.999926	0.971911	0.998978	0.971911	0.714424	0.82351	0.003569	0.00255	0.002623	0.857175
Class: DoS attacks-Hulk	0.997132	0.998837	0.915915	0.999964	0.915915	0.997132	0.9548	0.012547	0.012511	0.01366	0.997985
Class: DoS attacks-SlowHTTPTest	0	1	NA	0.999993	NA	0	NA	6.57E-06	0	0	0.5
Class: DoS attacks-Slowloris	0.703057	0.999984	0.974684	0.999741	0.974684	0.703057	0.816882	0.000871	0.000613	0.000629	0.85152
Class: FTP-BruteForce	0	1	NA	0.999994	NA	0	NA	5.54E-06	0	0	0.5
Class: Infilteration	0.008111	0.999846	0.391724	0.987981	0.391724	0.008111	0.015893	0.012114	9.83E-05	0.000251	0.503978
Class: SQL Injection	0	1	NA	0.999989	NA	0	NA	1.07E-05	0	0	0.5
Class: SSH-Bruteforce	0.997452	0.999822	0.97879	0.999979	0.97879	0.997452	0.988033	0.008147	0.008126	0.008302	0.998637

Confusion Matrix

Table 60 Confusion Matrix for GLM

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2544665	551	142	25	8670	4	8459	17	0	19	729	15	34540	21	18
Bot	314	35708	0	0	0	0	0	0	0	0	0	0	7	0	0
Brute Force -Web	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Brute Force -XSS	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0
DDOS attack-HOIC	4782	0	0	0	40704	0	0	0	0	0	0	0	82	10	0
DDOS attack-LOIC-UDP	0	0	0	0	0	440	179	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	1046	0	0	0	0	5	135719	0	0	0	0	0	4	0	0
DoS attacks-GoldenEye	102	0	0	0	0	0	0	7370	104	0	4	0	3	0	0
DoS attacks-Hulk	271	0	0	0	0	0	0	2914	36164	0	15	0	78	0	42
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DoS attacks-Slowloris	30	0	0	0	0	0	0	15	0	0	1771	0	1	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Infilteration	439	0	0	0	0	2	0	0	0	0	0	0	284	0	0
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SSH-Bruteforce	491	0	1	0	0	0	0	0	0	0	0	1	16	0	23489

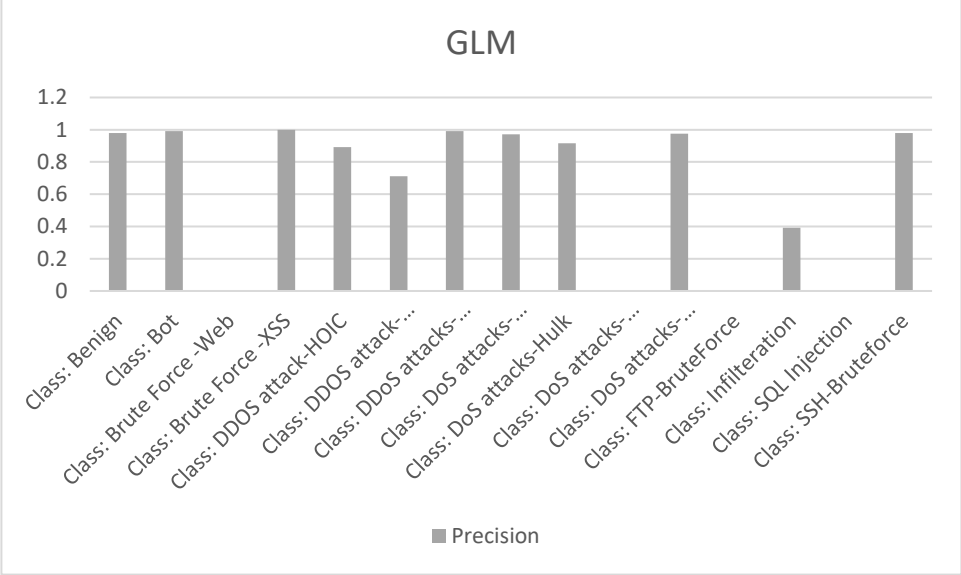


Figure 40 Per-Class Precision for GLM

8.4 Deep Learning (Neural network)

Table 61 Overall Performance for Deep Learning

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.9813317	0.9081043	0.9811750	0.9814874	0.8829398	0.0000000
McnemarPValue					
NaN					

Table 62 Per-Class Performance for Deep Learning

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.99964	0.859666	0.981728	0.996851	0.981728	0.99964	0.990603	0.88294	0.882622	0.899049	0.929653
Class: Bot	0.982625	0.999999	0.999944	0.999779	0.999944	0.982625	0.991209	0.012544	0.012326	0.012327	0.991312
Class: Brute Force -Web	0	1	NA	0.999951	NA	0	NA	4.95E-05	0	0	0.5
Class: Brute Force -XSS	0	1	NA	0.999984	NA	0	NA	1.63E-05	0	0	0.5
Class: DDOS attack-HOIC	0.999696	0.999987	0.999251	0.999995	0.999251	0.999696	0.999474	0.017081	0.017076	0.017089	0.999842
Class: DDOS attack-LOIC-UDP	0	1	NA	0.999844	NA	0	NA	0.000156	0	0	0.5
Class: DDoS attacks-LOIC-HTTP	0.938756	0.999878	0.997534	0.996791	0.997534	0.938756	0.967253	0.049942	0.046883	0.046999	0.969317
Class: DoS attacks-GoldenEye	0.439705	0.999996	0.997581	0.997997	0.997581	0.439705	0.610375	0.003569	0.001569	0.001573	0.719851
Class: DoS attacks-Hulk	0.98285	0.99806	0.86553	0.999782	0.86553	0.98285	0.920467	0.012547	0.012332	0.014248	0.990455
Class: DoS attacks-SlowHTTPTest	0	1	NA	0.999993	NA	0	NA	6.57E-06	0	0	0.5
Class: DoS attacks-Slowloris	0.457721	0.999995	0.988003	0.999527	0.988003	0.457721	0.62561	0.000871	0.000399	0.000404	0.728858
Class: FTP-BruteForce	0	1	NA	0.999994	NA	0	NA	5.54E-06	0	0	0.5
Class: Infilteration	0	0.999867	0	0.987885	0	0	NA	0.012114	0	0.000131	0.499933
Class: SQL Injection	0	1	NA	0.999989	NA	0	NA	1.07E-05	0	0	0.5
Class: SSH-Bruteforce	0.997197	0.999944	0.993233	0.999977	0.993233	0.997197	0.995211	0.008147	0.008124	0.00818	0.998571

Table 63 Confusion Matrix for Deep Learning

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2551230	605	143	47	15	367	8739	542	622	19	1351	16	34923	31	64
Bot	2	35629	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -XSS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DDOS attack-HOIC	36	0	0	0	49359	0	0	0	0	0	0	0	1	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	279	0	0	0	0	0	135516	55	0	0	0	0	1	0	0
DoS attacks-GoldenEye	9	0	0	0	0	0	0	4536	0	0	0	0	2	0	0
DoS attacks-Hulk	267	0	0	0	0	0	0	5169	35646	0	15	0	85	0	2
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DoS attacks-Slowloris	0	0	0	0	0	0	0	14	0	0	1153	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Infilteration	169	25	0	0	0	84	102	0	0	0	0	0	0	0	0
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SSH-Bruteforce	157	0	0	0	0	0	0	0	0	0	0	0	3	0	23483

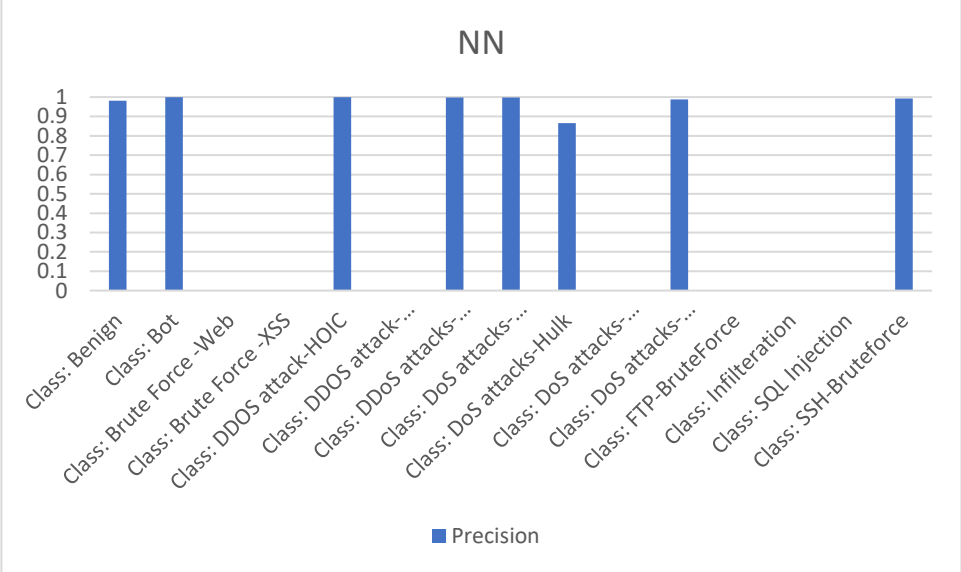


Figure 41 Precision for Deep Learning

8.5 Random Forest (Ranger)

Table 64 Overall Performance for RF (Ranger)

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.9898437	0.9514503	0.9897274	0.9899589	0.8829398	0.0000000
McnemarPValue					
NaN					

Results

Table 65 Per-Class Performance for RF (Ranger)

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.999402	0.918026	0.989242	0.995115	0.989242	0.999402	0.994296	0.88294	0.882412	0.892008	0.958714
Class: Bot	0.993298	1	1	0.999915	1	0.993298	0.996638	0.012544	0.01246	0.01246	0.996649
Class: Brute Force -Web	0.496503	1	1	0.999975	1	0.496503	0.663551	4.95E-05	2.46E-05	2.46E-05	0.748252
Class: Brute Force -XSS	0.723404	1	1	0.999996	1	0.723404	0.839506	1.63E-05	1.18E-05	1.18E-05	0.861702
Class: DDOS attack-HOIC	0.996476	0.999958	0.997567	0.999939	0.997567	0.996476	0.997021	0.017081	0.017021	0.017063	0.998217
Class: DDOS attack-LOIC-UDP	0.973392	0.999976	0.862475	0.999996	0.862475	0.973392	0.914583	0.000156	0.000152	0.000176	0.986684
Class: DDoS attacks-LOIC-HTTP	0.99876	0.999975	0.999515	0.999935	0.999515	0.99876	0.999137	0.049942	0.04988	0.049904	0.999367
Class: DoS attacks-GoldenEye	0.999903	1	1	1	1	0.999903	0.999952	0.003569	0.003569	0.003569	0.999952
Class: DoS attacks-Hulk	1	0.999999	0.999945	1	0.999945	1	0.999972	0.012547	0.012547	0.012548	1
Class: DoS attacks-SlowHTTPTest	0.473684	0.999997	0.5	0.999997	0.5	0.473684	0.486486	6.57E-06	3.11E-06	6.23E-06	0.736841
Class: DoS attacks-Slowloris	1	0.999998	0.998019	1	0.998019	1	0.999009	0.000871	0.000871	0.000873	0.999999
Class: FTP-BruteForce	0.125	1	0.666667	0.999995	0.666667	0.125	0.210526	5.54E-06	6.92E-07	1.04E-06	0.5625
Class: Infilteration	0.225932	0.99953	0.854874	0.990593	0.854874	0.225932	0.357406	0.012114	0.002737	0.003202	0.612731
Class: SQL Injection	0.709677	1	1	0.999997	1	0.709677	0.830189	1.07E-05	7.61E-06	7.61E-06	0.854839
Class: SSH-Bruteforce	0.999958	1	1	1	1	0.999958	0.999979	0.008147	0.008147	0.008147	0.999979

Confusion Matrix

Table 66 Confusion Matrix RF(Ranger)

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2550624	243	72	13	174	0	109	0	0	9	0	5	27103	9	0
Bot	0	36016	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	0	0	71	0	0	0	0	0	0	0	0	0	0	0	0

Brute Force -XSS	0	0	0	34	0	0	0	0	0	0	0	0	0	0	0
DDOS attack-HOIC	119	0	0	0	49200	0	0	0	0	0	0	0	1	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	439	70	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	58	0	0	0	0	12	144178	0	0	0	0	0	0	0	0
DoS attacks-GoldenEye	0	0	0	0	0	0	0	10315	0	0	0	0	0	0	0
DoS attacks-Hulk	0	0	0	0	0	0	0	1	36268	0	0	0	0	0	1
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	9	0	9	0	0	0
DoS attacks-Slowloris	5	0	0	0	0	0	0	0	0	0	2519	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	1	0	2	0	0	0
Infilteration	1343	0	0	0	0	0	0	0	0	0	0	0	7911	0	0
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	22	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23548

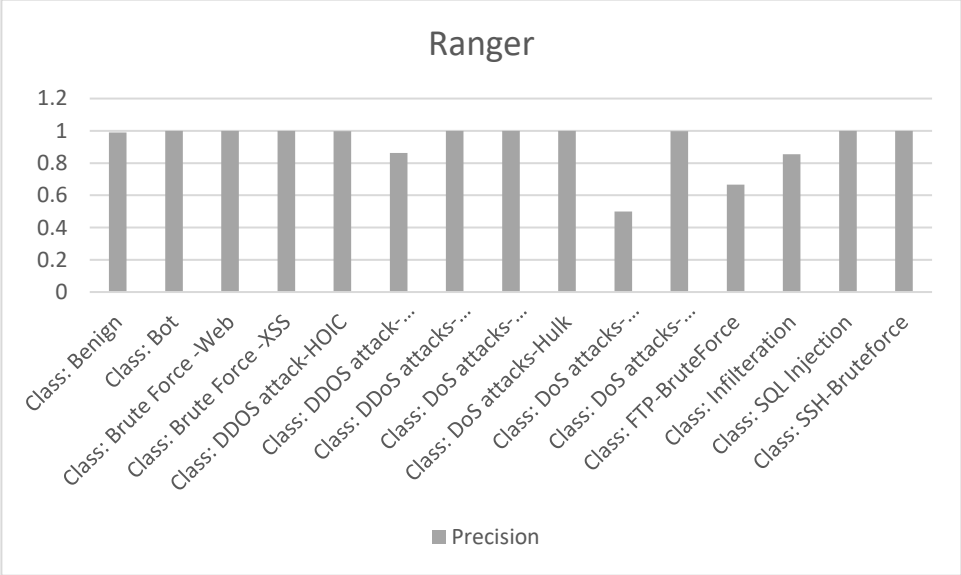


Figure 42 Precision for RF

8.6 Distributed Random Forest (DRF)

Table 67 Overall Performance for DRF

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.9875572	0.9397946	0.9874288	0.9876847	0.8829398	0.0000000

McnemarPValue
NaN

Results

Table 68 Per-Class Performance for DRF

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.999761	0.896041	0.986401	0.997992	0.986401	0.999761	0.993036	0.88294	0.882729	0.894898	0.947901
Class: Bot	0.989079	1	1	0.999861	1	0.989079	0.994509	0.012544	0.012407	0.012407	0.994539
Class: Brute Force -Web	0.426573	1	1	0.999972	1	0.426573	0.598039	4.95E-05	2.11E-05	2.11E-05	0.713287
Class: Brute Force -XSS	0.510638	1	1	0.999992	1	0.510638	0.676056	1.63E-05	8.30E-06	8.30E-06	0.755319
Class: DDOS attack-HOIC	0.999939	1	1	0.999999	1	0.999939	0.999997	0.017081	0.01708	0.01708	0.99997
Class: DDOS attack-LOIC-UDP	0.977827	0.999943	0.726524	0.999997	0.726524	0.977827	0.833648	0.000156	0.000153	0.00021	0.988885
Class: DDoS attacks-LOIC-HTTP	0.998261	0.999798	0.99617	0.999909	0.99617	0.998261	0.997215	0.049942	0.049855	0.050046	0.99903
Class: DoS attacks-GoldenEye	0.999806	1	0.999903	0.999999	0.999903	0.999806	0.999855	0.003569	0.003568	0.003569	0.999903
Class: DoS attacks-Hulk	1	0.999996	0.999724	1	0.999724	1	0.999862	0.012547	0.012547	0.012551	0.999998
Class: DoS attacks-SlowHTTPTest	0	1	NA	0.999993	NA	0	NA	6.57E-06	0	0	0.5
Class: DoS attacks-Slowloris	0.99206	1	1	0.999993	1	0.99206	0.996014	0.000871	0.000865	0.000865	0.99603
Class: FTP-BruteForce	0	1	NA	0.999994	NA	0	NA	5.54E-06	0	0	0.5
Class: Infilteration	0.015336	0.999979	0.901007	0.98807	0.901007	0.015336	0.030159	0.012114	0.000186	0.000206	0.507658
Class: SQL Injection	0	1	NA	0.999989	NA	0	NA	1.07E-05	0	0	0.5
Class: SSH-Bruteforce	0.998938	1	1	0.999991	1	0.998938	0.999469	0.008147	0.008138	0.008138	0.999469

Confusion Matrix

Table 69 Confusion for DRF

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infilteration	SQL Injection	SSH-Bruteforce
Benign	2551539	396	80	23	3	2	85	0	0	19	20	16	34478	31	23
Bot	0	35863	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	0	0	61	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -XSS	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0
DDOS attack-HOIC	0	0	0	0	49371	0	0	0	0	0	0	0	0	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	441	166	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	544	0	2	0	0	8	144106	0	0	0	0	0	0	0	0
DoS attacks-GoldenEye	1	0	0	0	0	0	0	10314	0	0	0	0	0	0	0
DoS attacks-Hulk	6	0	0	0	0	0	0	2	36268	0	0	0	0	0	2
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DoS attacks-Slowloris	0	0	0	0	0	0	0	0	0	0	2499	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Infilteration	59	0	0	0	0	0	0	0	0	0	0	0	537	0	0
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23524

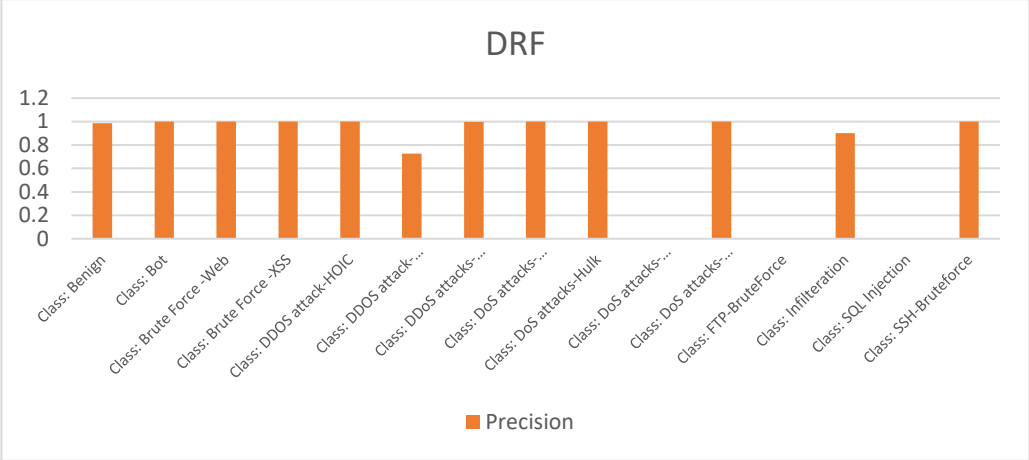


Figure 43 Percision for DRF

8.7 Portfolio Classifier (Random Forest)

Table 70 Overall Performance

Accuracy	Kappa	AccuracyLower	AccuracyUpper	AccuracyNull	AccuracyPValue
0.9899385	0.9519115	0.9898228	0.9900532	0.8829630	0.0000000
McnemarPValue					
NaN					

Results

Table 71 Per-Class Performance

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence	Detection Rate	Detection Prevalence	Balanced Accuracy
Class: Benign	0.999409	0.918791	0.989344	0.995168	0.989344	0.999409	0.994351	0.882963	0.882441	0.891945	0.9591
Class: Bot	0.994815	1	1	0.999934	1	0.994815	0.997401	0.012545	0.01248	0.01248	0.997408
Class: Brute Force -Web	0.496503	1	1	0.999975	1	0.496503	0.663551	4.95E-05	2.46E-05	2.46E-05	0.748252
Class: Brute Force -XSS	0.723404	1	1	0.999996	1	0.723404	0.839506	1.63E-05	1.18E-05	1.18E-05	0.861702
Class: DDOS attack-HOIC	0.999757	1	1	0.999996	1	0.999757	0.999878	0.017082	0.017078	0.017078	0.999878

Class: DDOS attack-LOIC-UDP	0.973392	0.999973	0.84749	0.999996	0.84749	0.973392	0.906089	0.000156	0.000152	0.000179	0.986683
Class: DDoS attacks-LOIC-HTTP	0.99849	0.999987	0.999743	0.999921	0.999743	0.99849	0.999116	0.004943	0.049868	0.049881	0.999238
Class: DoS attacks-GoldenEye	0.999903	1	1	1	1	0.999903	0.999952	0.003569	0.003569	0.003569	0.999952
Class: DoS attacks-Hulk	1	0.999998	0.999862	1	0.999862	1	0.999931	0.012548	0.012548	0.012549	0.999999
Class: DoS attacks-SlowHTTPTest	0.263158	0.999997	0.384615	0.999995	0.384615	0.263158	0.3125	6.57E-06	1.73E-06	4.50E-06	0.631578
Class: DoS attacks-Slowloris	0.999206	1	1	0.999999	1	0.999206	0.999603	0.000871	0.000871	0.000871	0.999603
Class: FTP-BruteForce	0.0625	1	1	0.999995	1	0.0625	0.117647	5.54E-06	3.46E-07	3.46E-07	0.53125
Class: Infilteration	0.226751	0.999481	0.842497	0.990623	0.842497	0.226751	0.35733	0.012087	0.002741	0.003253	0.613116
Class: SQL Injection	0.709677	1	1	0.999997	1	0.709677	0.830189	1.07E-05	7.61E-06	7.61E-06	0.854839
Class: SSH-Bruteforce	0.999915	1	1	0.999999	1	0.999915	0.999958	0.008147	0.008147	0.008147	0.999958

Confusion Matrix

Table 72 Confusion Matrix for Portfolio Classifier

	Benign	Bot	Brute Force -Web	Brute Force -XSS	DDOS attack-HOIC	DDOS attack-LOIC-UDP	DDoS attacks-LOIC-HTTP	DoS attacks-GoldenEye	DoS attacks-Hulk	DoS attacks-SlowHTTPTest	DoS attacks-Slowloris	FTP-BruteForce	Infiltration	SQL Injection	SSH-Bruteforce
Benign	2550624	1888	71	13	12	1	139	0	0	14	2	7	27015	9	1
Bot	0	36071	0	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -Web	0	0	71	0	0	0	0	0	0	0	0	0	0	0	0
Brute Force -XSS	0	0	0	34	0	0	0	0	0	0	0	0	0	0	0

DDOS attack-HOIC	0	0	0	0	49362	0	0	0	0	0	0	0	0	0	0
DDOS attack-LOIC-UDP	0	0	0	0	0	439	79	0	0	0	0	0	0	0	0
DDoS attacks-LOIC-HTTP	25	0	1	0	0	11	144139	0	0	0	0	0	0	0	0
DoS attacks-GoldenEye	0	0	0	0	0	0	0	10315	0	0	0	0	0	0	0
DoS attacks-Hulk	3	0	0	0	0	0	0	1	36268	0	0	0	0	0	1
DoS attacks-SlowHTTPTest	0	0	0	0	0	0	0	0	0	5	0	8	0	0	0
DoS attacks-Slowloris	0	0	0	0	0	0	0	0	0	0	2517	0	0	0	0
FTP-BruteForce	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Infilteration	1481	0	0	0	0	0	0	0	0	0	0	0	7922	0	0
SQL Injection	0	0	0	0	0	0	0	0	0	0	0	0	0	22	0
SSH-Bruteforce	0	0	0	0	0	0	0	0	0	0	0	0	0	0	23547

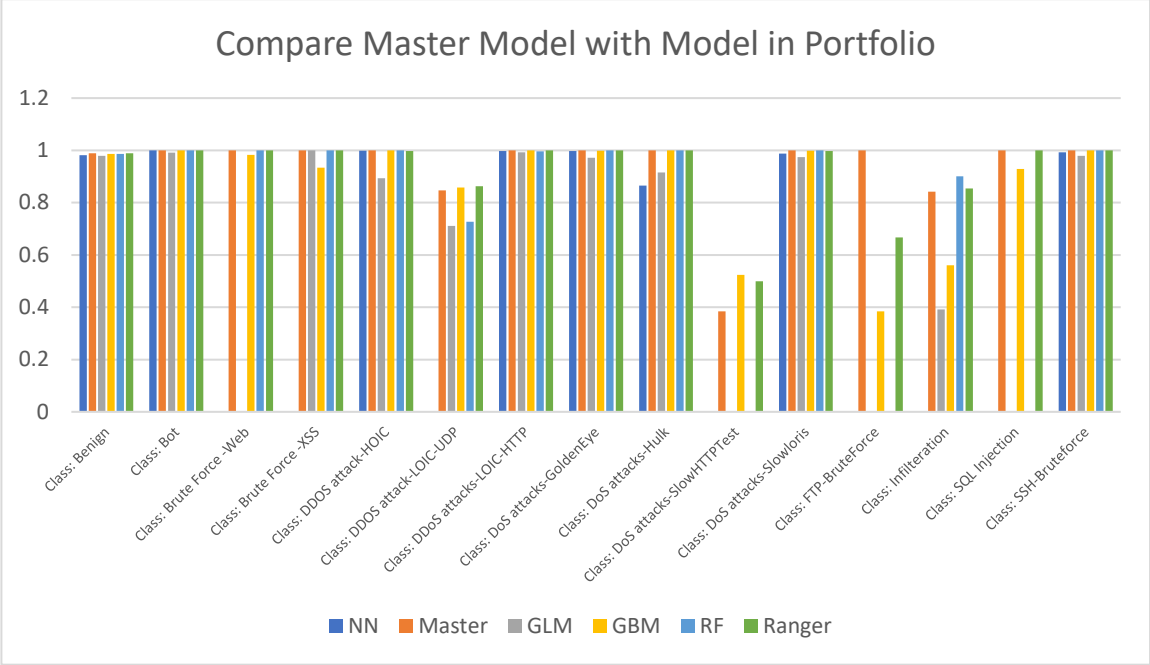


Figure 44 Compare the Master Model with the Models in Portfolio

In the figure above, we can see clearly that the master model outperforms the other models in the portfolio. In almost every case, the master model has better precision. The assumption is that the multi-classifier portfolio works as expected. It seems the Ranger model has more influence on the master model than the others. Between both, Ranger has similar results, but there is a big gap in the FTP-Brute Force Class. It seems that the master model has taken advantage of the multiple classifiers. Further details can be seen in the next graphs.

In the next graphs (Figure 47 Benchmark (Full vs Sub-sample)), we have compared the performance between the models using the complete sample and the subsample. We can see a clear improvement in the performance precision between the two sets of models. Especially in the master model, we can see that the master model using subsample has failed in many classes as many class has precision

close to zero, while the model built with the full dataset is almost 1 in all classes with few exceptions, as seen in the figures.

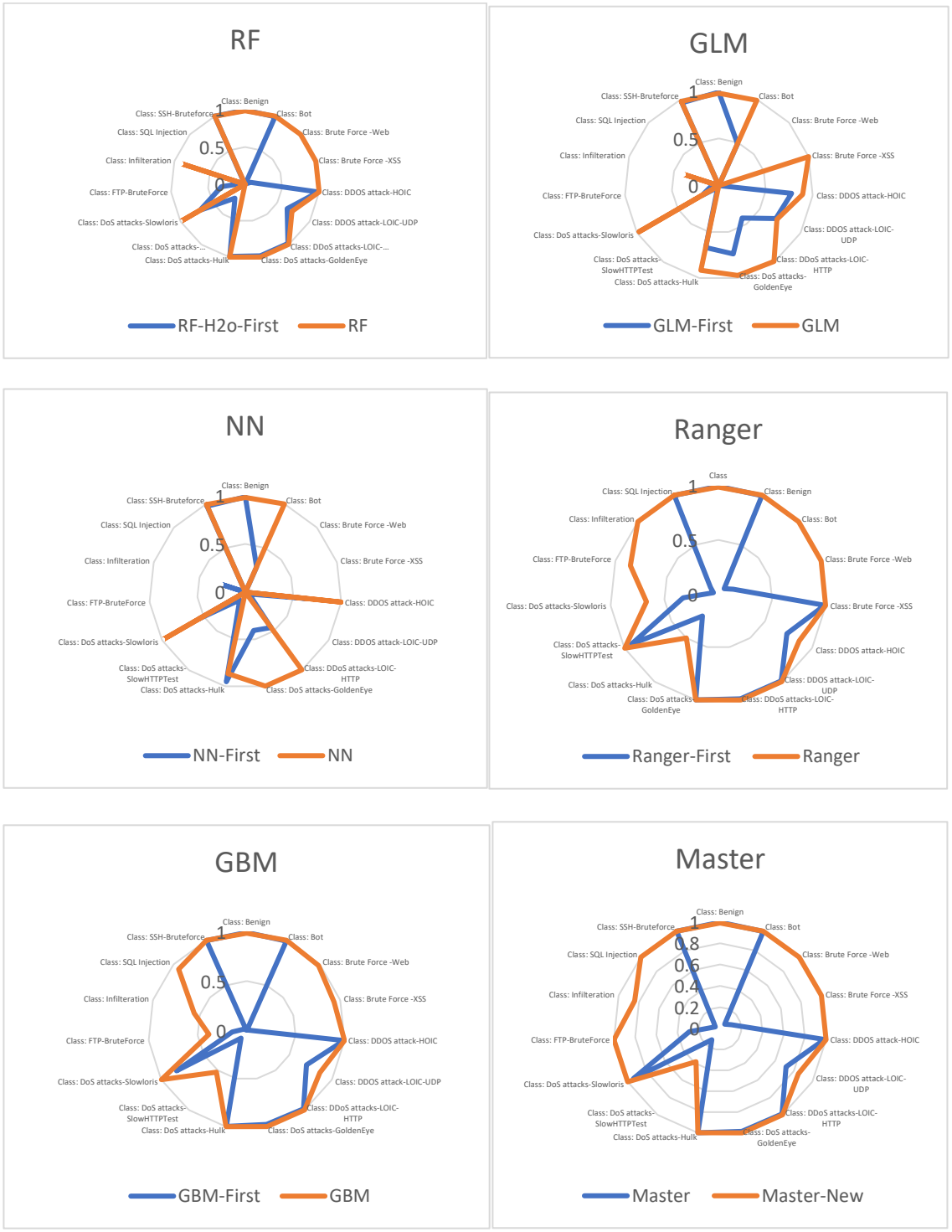


Figure 45 Benchmark (Full vs Sub-sample)

8.8 Discussion

In this chapter, the models introduced earlier were trained and evaluated using the full CIC-IDS2018 dataset to understand how each classifier performs under more realistic and demanding conditions. Compared to the sub-sampled experiments in the previous chapter, the complete dataset introduced new challenges in terms of processing time, class imbalance, and detection complexity. The performance results revealed that while some models maintained their effectiveness, others experienced a drop in accuracy and precision, particularly when dealing with minority attack classes. Random Forest and Deep Learning models continued to perform strongly, particularly in handling the more frequent classes. However, their ability to generalize across rare or less distinguishable attacks remained limited without additional balancing techniques. The per-instance voting classifier again showed improvement over individual models, especially in reducing false negatives and achieving better consistency across all classes. This reinforces the argument that combining models based on instance-specific behavior offers more robust and flexible detection. The discussion also considered the scalability of the approach, as training on the full dataset placed higher demands on computational resources and memory, particularly for deep learning-based architectures.

8.9 Chapter Conclusion

This chapter completed the second stage of experimentation by scaling the classifier models to the full CIC-IDS2018 dataset. The purpose of this step was to observe model behavior under real-world data conditions and validate the earlier results obtained from the sub-sampled dataset. The findings confirmed that while some classifiers performed well in both setups, others struggled with the full dataset's

complexity, highlighting the value of using a diverse set of models rather than relying on a single one. The performance of the per-instance classifier portfolio remained strong and consistent, offering better generalization across both major and minor classes, and handling imbalanced data more effectively than standalone classifiers. These results support the feasibility of deploying a model selection strategy in a practical environment where traffic types vary significantly. With the full dataset results established, the next chapter shifts focus toward comparative benchmarking with existing solutions and further analyzing the evaluation metrics that define the effectiveness of the proposed approach.

Chapter 9 Evaluation and Discussion

9.1 Chapter Introduction

In this Chapter, we will benchmark the results of the proposed Portfolio classifier with other research that was done between 2020 and 2023 on the same dataset, which is CSE-CIC-IDS2018. We will view the benchmark multiple performance metrics, which are precision, F1 Score, and Accuracy. Making the benchmark has some challenges, and the primary challenge is the type of classification. Many researchers do binary classification, and that means the classification will only identify if the presented instance is benign or a threat. Without knowing the type of the threat, it's difficult to mitigate and plan for action for the threat. In contrast, binary classification has shown that they have a very high accuracy and precision. In this part, we have ignored the sample size and any manipulation of the dataset. We only focused on the end results, and the reason is that we can't replicate each research in order to be able to do a direct comparison and benchmarking. It's also worth noting that some researchers have done multiclassification, but they have coupled and merged multiple classes into one. For example, some research papers used the DDoS class, which covers all other DDoS classes such as DDoS attack-HOIC DDoS attack and LOIC-UDP. Doing such a thing will increase classifier performance, but at the same time, the user will lose information to identify which type of DDoS attack he/she is facing.

Through all the benchmarks that are listed below, we can observe that the proposed model performance is either better or similar to the other research. We will see in detail the benchmarks that cover different metrics, which are F1 Score, accuracy, precision, and recall.

We can see below the list of classes that are original in the dataset and the ones that are recreated and coupled.

Label map:

Table 73 List of original Classes and new classes

Class	Original Class
Benign	TRUE
Bot	TRUE
Brute Force -Web	TRUE
Brute Force -XSS	TRUE
DDOS attack-HOIC	TRUE
DDOS attack-LOIC-UDP	TRUE
DDoS attacks-LOIC-HTTP	TRUE
DoS attacks-GoldenEye	TRUE
DoS attacks-Hulk	TRUE
DoS attacks-SlowHTTPTest	TRUE
DoS attacks-Slowloris	TRUE
FTP-BruteForce	TRUE
Infilteration	TRUE
SQL Injection	TRUE
SSH-Bruteforce	TRUE
DDOS	False
DOS	False
BruteForce	False

9.2 Precision Benchmark:

In this part, we will do the benchmark on a precision metric with 13 research papers, as seen in the table below. The first part, which is the most important, is the precision of the benign class. We can see that in this class, most of the proposed models achieved a precision of 0.99, which is similar to the other research. The same scenario for Bot, but Hagar's research got a little lower performance, which is around ~0.83. For the rest of the classes that are original, we note Brute Force-Web, Brute Force-XSS, and SQL injection. They have 0 precision in Chimphlee Research, but the proposed model along with Heger has similar results with small differences. In the proposed model, DoS attacks-SlowHTTPTest had the lowest precision compared to the other research, and it was around ~0.38. There are nine researches included in the benchmark that have an overall precision, and the precision overly ranges between ~0.9 to 1, but some searches were as low as 0.78. It is expected to have an overall precision higher as this approach is dependent on binary classification, and it reduces the amount of information needed to understand the advantages and disadvantages of the model. Finally, the researchers that have done classification on classes that are not original have achieved very precision on classes that they have introduced.

Table 74 Precision Performance Benchmark

Precision	Precision (Proposed Model)	(Kabir et al., 2021)	(Seth et al., 2021)	(Hua, 2020)	(Dini et al., 2022)	(Handika et al., 2022)	(Fitni and Ramli, 2020)	(Alkanjr and Alshammari,	(Shahbandayeva et al., 2022)	(Das et al., 2023)	(Yoo et al., 2021)	(Siddiqi and Pak, 2022)	(Chimphlee et al., 2022)	(Hagar et al., 2022)
Benign	0.99	0.99	0.92	x	x	x	x	x	x	x	x	x	0.99	1.00
Bot	1.00	1.00	1.00	x	x	x	x	x	x	x	x	x	1.00	0.83
Brute Force -Web	1.00	x	x	x	x	x	x	x	x	x	x	x	0.00	0.75
Brute Force -XSS	1.00	x	x	x	x	x	x	x	x	x	x	x	0.00	1.00
BruteForce	x	1.00	0.99	x	x	x	x	x	x	x	x	x	x	x
DDoS	x	0.99	1.00	x	x	x	x	x	x	x	x	x	x	x
DDOS attack-HOIC	1.00	x	x	x	x	x	x	x	x	x	x	x	1.00	1.00
DDOS attack-LOIC-UDP	0.85	x	x	x	x	x	x	x	x	x	x	x	0.72	1.00
DDoS attacks-LOIC-HTTP	1.00	x	x	x	x	x	x	x	x	x	x	x	0.99	1.00
DoS	x	0.99	0.98	x	x	x	x	x	x	x	x	x	x	x
DoS attacks-GoldenEye	1.00	x	x	x	x	x	x	x	x	x	x	x	0.99	1.00
DoS attacks-Hulk	1.00	x	x	x	x	x	x	x	x	x	x	x	0.96	1.00
DoS attacks-SlowHTTPTest	0.38	x	x	x	x	x	x	x	x	x	x	x	0.75	1.00
DoS attacks-Slowloris	1.00	x	x	x	x	x	x	x	x	x	x	x	0.95	1.00
FTP-BruteForce	1.00	x	x	x	x	x	x	x	x	x	x	x	0.71	0.88
Infilteration	0.84	0.96	0.97	x	x	x	x	x	x	x	x	x	0.44	1.00
Overall	x	x	x	0.98	94.39	96.00	98.80	1.00	0.78	99.00	0.89	0.98	x	x
SQL Injection	1.00	0.87	x	x	x	x	x	x	x	x	x	x	0.00	0.92
SSH-Bruteforce	1.00	x	x	x	x	x	x	x	x	x	x	x	0.99	1.00
Web attacks	x	x	1.00	x	x	x	x	x	x	x	x	x	x	x

9.3 Accuracy Benchmark

In this part, there are seven researchers that provided the accuracy and were included in this benchmark. As discussed earlier, each research is done differently, and the comparison could have some difficulties as the metric is not exactly the same. We can directly compare the proposed model with Jiyeon's research as the research provides accuracy for each class. Comparing the accuracies for both models, we can observe that both perform well in each class, but we will view the extreme differences between both. In the FTP-Brute Force and DoS attack HTTPtest, the proposed model accuracies were around 0.53 and 0.63, respectively,

while Jiyeon achieved 0.98 and 1 in the same order. On the other hand, infiltration and SQL injection, the proposed model achieved 0.61 and 0.85 compared to 0.35 and 0.08 in Jiyeon. The overall accuracy in the proposed model is 98.9, and when compared to the other research, we can see that it ranges between 0.83 and 1.

Overall, the accuracy in the proposed model has acceptable results when compared to the other research, either in the multi-Classifiaction or the general accuracy.

Table 75 Accuracy Benchmark

Accuracy	Proposed Model	(Tonni and Mazumder, 2023)	(Hua, 2020)	(Shorubiga and Shyam, 2023)	(Dini et al., 2022)	(Fitni and Ramli, 2020)	(Alkanjr and Alshammari,	(Shahbandayeva et al., 2022)	(Das et al., 2023)	(Yoo et al., 2021)	(Siddiqi and Pak, 2022)	(Jiyeon et al., 2021)
Benign	0.96	x	x	x	x	x	x	x	x	x	x	1
Bot	1	1	x	x	x	x	x	x	x	x	x	1
Brute Force -Web	0.75	x	x	x	x	x	x	x	x	x	x	0.3
Brute Force -XSS	0.86	x	x	x	x	x	x	x	x	x	x	0.65
BruteForce	x	1	x	x	x	x	x	x	x	x	x	x
DDoS	x	x	x	x	x	x	x	x	x	x	x	x
DDOS attack-HOIC	1	x	x	x	x	x	x	x	x	x	x	1
DDOS attack-LOIC-UDP	0.99	x	x	x	x	x	x	x	x	x	x	1
DDoS attacks-LOIC-HTTP	1	x	x	x	x	x	x	x	x	x	x	1
DoS	x	1	x	x	x	x	x	x	x	x	x	x
DoS attacks-GoldenEye	1	x	x	x	x	x	x	x	x	x	x	0.47
DoS attacks-Hulk	1	x	x	x	x	x	x	x	x	x	x	1
DoS attacks-SlowHTTPTest	0.63	x	x	x	x	x	x	x	x	x	x	1
DoS attacks-Slowloris	1	x	x	x	x	x	x	x	x	x	x	0.66
FTP-BruteForce	0.53	x	x	x	x	x	x	x	x	x	x	0.98
Infilteration	0.61	0.86	x	x	x	x	x	x	x	x	x	0.35
SQL Injection	0.85	1	x	x	x	x	x	x	x	x	x	0.08
SSH-Bruteforce	1	x	x	x	x	x	x	x	x	x	x	0.96
Web attacks	x	x	x	x	x	x	x	x	x	x	x	X
Overall	98.9	x	0.98	99.97	94.5	98.8	1	0.83	98	0.89	0.98	x

9.4 F1 Score Benchmark

There are 11 pieces of research included in the F1 score benchmark. In the General view we can observe that the proposed model has acceptable results in all classes

compared to the other models, which have an F1 score per class, except for the BruteForce-Web, FTP-BruteForce, and infiltration, where the results are 0.66, 0.12, and 0.36 respectively compared to results that are around 0.9 for both Brute force-Web and FTP-Brute Force in Di and Handika, except for infiltration where the results were not presented or 0 like chimphlee research.

Table 76 F1 Score Benchmark

F1 Score	Proposed Model	(Tonni and Mazumder, 2023)	(Hua, 2020)	(Di et al., 2022)	(Handika et al., 2022)	(Handika et al., 2022)	(Fitni and Ramli, 2020)	(Alkanjr and Alshammari,	(Shahbandayeva et al., 2022)	(Yoo et al., 2021)	(Siddiqi and Pak, 2022)	(Chimphlee et al., 2022)
Benign	0.99	x	x	x	0.99	x	x	x	x	x	x	0.99
Bot	1.00	1.00	x	95.13	1.00	x	x	x	x	x	x	1.00
Brute Force -Web	0.66	x	x	89.10	0.80	x	x	x	x	x	x	0.00
Brute Force -XSS	0.84	x	x	90.11	0.66	x	x	x	x	x	x	0.00
BruteForce	x	1.00	x	x	x	x	x	x	x	x	x	x
DDoS	x	x	x	x	x	x	x	x	x	x	x	x
DDOS attack-HOIC	1.00	x	x	90.08	1.00	x	x	x	x	x	x	1.00
DDOS attack-LOIC-UDP	0.91	x	x	96.99	1.00	x	x	x	x	x	x	0.84
DDoS attacks-LOIC-HTTP	1.00	x	x	89.32	1.00	x	x	x	x	x	x	0.98
DoS	x	1.00	x	x	x	x	x	x	x	x	x	x
DoS attacks-GoldenEye	1.00	x	x	94.11	1.00	x	x	x	x	x	x	0.71
DoS attacks-Hulk	1.00	x	x	91.91	1.00	x	x	x	x	x	x	0.98
DoS attacks-SlowHTTPTest	0.31	x	x	95.54	x	x	x	x	x	x	x	0.61
DoS attacks-Slowloris	1.00	x	x	90.35	1.00	x	x	x	x	x	x	0.94
FTP-BruteForce	0.12	x	x	92.41	1.00	x	x	x	x	x	x	0.79
Infilteration	0.36	0.93	x	x	x	x	x	x	x	x	x	0.02
Overall	x	x	0.98	x	x	93.00	97.90	1.00	0.78	0.85	0.98	x
SQL Injection	0.83	1.00	x	x	0.71	x	x	x	x	x	x	0.00
SSH-Bruteforce	1.00	x	x	93.00	1.00	x	x	x	x	x	x	1.00
Web attacks	x	x	x	x	x	x	x	x	x	x	x	x

9.5 Recall Benchmark

The Recall Benchmark had 11 research that have presented the recall metric. Only three researchers have included per-class recall values, which are Seth, Chimphlee, and Hagar. When comparing the proposed mode to these three searches, we can

note that Hagar has the best results, which is almost around 1 in all classes, except FTP-Brute Force (~0.7).

Table 77 Recall Benchmark

Recall	Proposed Model	(Seth et al., 2021)	(Hua, 2020)	(Handika et al., 2022)	(Fitni and Ramli, 2020)	(Alkanjr and Alshammari,	(Shahbandayeva et al., 2022)	(Das et al., 2023)	(Yoo et al., 2021)	(Siddiqi and Pak, 2022)	(Chimphlee et al., 2022)	(Hagar et al., 2022)
Benign	1.00	1.00	x	x	x	x	x	x	x	x	1.00	0.95
Bot	0.99	1.00	x	x	x	x	x	x	x	x	1.00	0.90
Brute Force -Web	0.50	x	x	x	x	x	x	x	x	x	0.00	0.99
Brute Force -XSS	0.72	x	x	x	x	x	x	x	x	x	0.00	1.00
BruteForce	x	0.96	x	x	x	x	x	x	x	x	x	x
DDoS	x	1.00	x	x	x	x	x	x	x	x	x	x
DDOS attack-HOIC	1.00	x	x	x	x	x	x	x	x	x	1.00	1.00
DDOS attack-LOIC-UDP	0.97	x	x	x	x	x	x	x	x	x	0.99	1.00
DDoS attacks-LOIC-HTTP	1.00	x	x	x	x	x	x	x	x	x	0.97	1.00
DoS	x	0.99	x	x	x	x	x	x	x	x	x	x
DoS attacks-GoldenEye	1.00	x	x	x	x	x	x	x	x	x	0.55	1.00
DoS attacks-Hulk	1.00	x	x	x	x	x	x	x	x	x	1.00	1.00
DoS attacks-SlowHTTPTest	0.26	x	x	x	x	x	x	x	x	x	0.51	1.00
DoS attacks-Slowloris	1.00	x	x	x	x	x	x	x	x	x	0.94	1.00
FTP-BruteForce	0.06	x	x	x	x	x	x	x	x	x	0.88	0.76
Infilteration	0.23	0.62	x	x	x	x	x	x	x	x	0.01	1.00
Overall	x	x	0.98	91.00	97.10	1.00	0.83	99.00	0.89	0.98	x	x
SQL Injection	0.71	x	x	x	x	x	x	x	x	x	0.00	0.99
SSH-Bruteforce	1.00	x	x	x	x	x	x	x	x	x	1.00	1.00
Web attacks	x	1.00	x	x	x	x	x	x	x	x	x	x

9.6 Discussion

This chapter provided a detailed evaluation of the proposed hybrid model using performance metrics that reflect real-world IDS expectations. The comparison between individual classifiers and the per-instance voting approach demonstrated consistent advantages in precision, recall, and overall classification stability. One of the key findings was the improved handling of minority classes, where most standalone models typically showed poor sensitivity. The master classifier, acting as

a rule-based guide, was effective in filtering out less-relevant models and contributed to the overall efficiency of the portfolio. Furthermore, the combination of the master model and portfolio strategy resulted in fewer false positives compared to baseline models, which is a critical factor in operational environments. These results highlight the benefit of dynamically selecting classifiers based on instance-level behavior, especially when traffic is highly diverse. The discussion also touched on benchmark comparisons with related studies, showing that the proposed method is not only competitive but also introduces a flexible architecture that can adapt to different detection requirements. This adaptability positions the model as a practical candidate for deployment in layered security systems.

9.7 Chapter Conclusion

In conclusion, this chapter validated the performance of the proposed intrusion detection system by comparing it against traditional models and existing solutions in the literature. The per-instance classifier portfolio consistently demonstrated better balance between detection rates and false alarms across all tested scenarios. Its effectiveness in detecting minority class attacks and reducing false positives supports the original research hypothesis and confirms the value of integrating model diversity. Additionally, the evaluation confirmed that combining a rule-based master classifier with dynamic portfolio voting results in a more adaptable and accurate detection process. The benchmark comparisons further established that this research offers improvements over many standard classification strategies used in IDS today. These conclusions mark the completion of the experimental phase of the thesis. The final chapter will summarize the research contributions, discuss the implications of the findings, and outline potential directions for future work.

Chapter 10 Conclusion

This chapter will have the discussion and conclusion by summarizing the thesis and showing if the research goals have been fulfilled. Also, it will show major findings, impact on the Industry, limitations, and future work.

In the introduction, we have 3 points that we consider contributions to knowledge. We will review each and validate each one of them as follows.

- The ability to create a (portfolio classifier) with no budget with precision and accuracy relevant to my thesis
    - The Portfolio Classifier (hybrid Classifier) is created in this research, and as shown in the benchmark, the performance and accuracy are comparable to other researches that use the same dataset and do multiclassification.
- per instance selection of classifier, where only selected classifiers can vote in each instance, so there will be a different set of classifiers to vote on for every threat.
    - We have trained multiple models on a dataset that was captured from network traffic. The models were used to be used in the portfolio. A master classifier was built based on the classifiers that are part of the portfolio. The master classifier is able to select the classifiers that can participate in the voting to classify the instance.
- Modularity of the Model, where additional classifiers can be plugged in to enhance performance

- As we have progressed in the research. We were able to add and remove models from the portfolio. Adding more classifiers is feasible and potentially can increase the performance and accuracy..

In this thesis, we have covered the general concepts of cybersecurity in order to allow the reader to comprehend the meaning of IDS and its function. We have covered the history and evolution of cyber security threats. In addition, we reviewed the tools and approaches that mitigate and defend systems from Cybersecurity threats. We have made data exploration and analysis of the dataset. Via this process, we have faced some challenges, but through the summaries and observing the plots and distribution of the classes, we were able to understand some characteristics of the Dataset CIC-IDS-2018.

Our proposed model is a portfolio of classifiers, where a classifier will be chosen for each network flow to assist in the classification of either a benign or a threat. For this reason, we have built multiple models that support multiclassification to construct the portfolio. We have made different attempts in this thesis. The first attempt involved using a subsample of the data. The results of this attempt were not satisfying when compared to published research papers. Then, we made a second attempt where we used the complete dataset, except for duplicates and NAs. In the second attempt, the results were much better than the first attempt, and a benchmark between the first and second attempts to compare both results. When both results were compared, we noticed a significant increase in performance for the second attempt.

We have made a similar comparison to other papers that have used the same dataset and used multiclassification. The benchmark showed that the proposed model either has a similar or exceeds the other research results (in most cases). The benchmarks used to compare other research results were precision, accuracy, F1 score, and recall. Not all research presented the complete data or used the same criteria, as some of the research clubbed some classes into categories, so the comparison and benchmarking would not have a complete picture. Overall, we could see potential in the proposed model when compared to others, and there is room for a lot of improvements and enhancements.

Using multiple classifiers can increase the performance as some classifiers work better with different types of classes in the same dataset. There are algorithms that use this approach that are either dependent on voting or fusion. In the proposed model, we classify the classifiers that will participate in the vote. Not all classifiers fit for each instance or flow in the dataset. In the proposed model, for each flow, we can have a different set of classifiers that can participate in the vote. This approach has increased the performance of the overall model when compared to each individual model in the portfolio. This approach can assist in cybersecurity as it can be easily expanded and enhanced by simply adding more models to the portfolio, which will increase performance and accuracy. With increased accuracy, organizations and industries will have fewer false alarms and a higher detection rate for anomalies, which will help Security operation centers (SOC) operators handle fewer floods of false alarms and focus on real threats.

The proposed model is modular in its nature. There are a lot of ways to enhance and have better performance. The simple way to enhance the model is by adding more

models to the portfolio; this way, the master classifier will have a wider selection of models that can participate in the voting. The second approach is to use multiple datasets for training or develop a tool that can capture the data for training. Another enhancement that can be done is by using multi-step classification. We can use a binary classifier that will determine if the flow is benign or an anomaly. If the flow anomaly, then we used the proposed model.

References:

Abdul Lateef, A.A., Faraj Al-Janabi, S.T., Al-Khateeb, B., 2020. Hybrid Intrusion Detection System Based on Deep Learning, in: 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI). pp. 1–5. https://doi.org/10.1109/ICDABI51230.2020.9325669

Alkanjr, B., Alshammari, T., 2023. IoBT Intrusion Detection System using Machine Learning. 2023 IEEE 13th Annual Computing and Communication Workshop and Conference, CCWC 2023 886–892. https://doi.org/10.1109/CCWC57344.2023.10099340

Arivardhini, S., Alamelu, L.M., Deepika, S., 2020. A Hybrid Classifier Approach for Network Intrusion Detection, in: 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS). pp. 824–827. https://doi.org/10.1109/ICACCS48705.2020.9074216

Atefi, K., Hashim, H., Khodadadi, T., 2020. A Hybrid Anomaly Classification with Deep Learning (DL) and Binary Algorithms (BA) as Optimizer in the Intrusion Detection System (IDS), in: 2020 16th IEEE International Colloquium on Signal Processing Its Applications (CSPA). pp. 29–34. https://doi.org/10.1109/CSPA48992.2020.9068725

Bharati, M.P., Tamane, S., 2020. NIDS-Network Intrusion Detection System Based on Deep and Machine Learning Frameworks with CICIDS2018 using Cloud Computing, in: 2020 International Conference on Smart Innovations in Design,

Environment, Management, Planning and Computing (ICSIDEMPC). pp. 27–30. https://doi.org/10.1109/ICSIDEMPC49020.2020.9299584

Caffe | Deep Learning Framework [WWW Document], n.d. URL https://caffe.berkeleyvision.org/ (accessed 8.16.23).

Canadian Institute for Cybersecurity, 2018. CSE-CIC-IDS2018 on AWS [WWW Document]. URL https://www.unb.ca/cic/datasets/ids-2018.html (accessed 9.28.19).

Chen, P., Guo, Y., Zhang, J., Wang, Y., Hu, H., 2020. A Novel Preprocessing Methodology for DNN-Based Intrusion Detection, in: 2020 IEEE 6th International Conference on Computer and Communications, ICCC 2020. pp. 2059–2064. https://doi.org/10.1109/ICCC51575.2020.9345300

Chimphlee, W., Chimphlee, S., Professor of Data Science, A., 2022. Network Intrusion Detector using Multilayer Perceptron (MLP) Approach. Turkish Journal of Computer and Mathematics Education 13, 488–499.

Cybersecurity Definition & Meaning - Merriam-Webster [WWW Document], n.d. URL https://www.merriam-webster.com/dictionary/cybersecurity (accessed 8.14.23).

Das, P., Illa, M., Pokhariyal, R., Latoria, A., Hemlata, Saini, D.K.J.B., 2023. Role of Neural Network, Fuzzy, and IoT in Integrating Artificial Intelligence as a Cyber Security System. Proceedings of the 2023 2nd International Conference on Electronics and Renewable Systems, ICEARS 2023 652–658. https://doi.org/10.1109/ICEARS56392.2023.10084988

David, R. Ben, Barr, A.B., 2021. Kubernetes Autoscaling: YoYo Attack Vulnerability and Mitigation. International Conference on Cloud Computing and Services Science, CLOSER - Proceedings 2021-April, 34–44. https://doi.org/10.5220/0010397900340044

Di, T., Wu, Y., Li, W., 2022. Deep Security Analysis Model for Smart Grid. 2022 IEEE 10th International Conference on Information, Communication and Networks, ICICN 2022 276–280. https://doi.org/10.1109/ICICN56848.2022.10006496

Dini, P., Begni, A., Ciavarella, S., De Paoli, E., Fiorelli, G., Silvestro, C., Saponara, S., 2022. Design and Testing Novel One-Class Classifier Based on Polynomial Interpolation with Application to Networking Security. IEEE Access 10, 67910–67924. https://doi.org/10.1109/ACCESS.2022.3186026

DOUGLAS, B., BILL, S., TIANQIU, W., 2015. A Survey of Intrusion Detection Systems. nternational Journal of Computer Applications.

Fitni, Q.R.S., Ramli, K., 2020. Implementation of ensemble learning and feature selection for performance improvements in anomaly-based intrusion detection systems. Proceedings - 2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology, IAICT 2020 118–124. https://doi.org/10.1109/IAICT50021.2020.9172014

FSabahi, AMovaghar, 2008. Intrusion detection: A survey. Proc. - The 3rd Int. Conf. Systems and Networks Communications, ICSNC 2008 - Includes I-CENTRIC 2008: Int. Conf. Advances in Human-Oriented and Personalized Mechanisms, Technologies, and Services 23–26. https://doi.org/10.1109/ICSNC.2008.44

H2O.ai | The fastest, most accurate AI Cloud Platform [WWW Document], n.d. URL https://h2o.ai/ (accessed 8.16.23).

Hagar, A., Gawali, B.W., Sciences, C., Technology, I., 2022. Deep Learning for Improving Attack Detection System Using CSE-CICIDS2018 20, 3064–3074. https://doi.org/10.14704/nq.2022.20.7.NQ33385

Haghighat, M.H., Li, J., 2021. Intrusion detection system using voting-based neural network. Tsinghua Sci Technol 26, 484–495. https://doi.org/10.26599/TST.2020.9010022

Handika, V., Istiyanto, J.E., Ashari, A., Purnama, S.R., Rochman, S., Dharmawan, A., 2022. Feature Representation for Network Intrusion Detection System Trough Embedding Neural Network. Proceeding of the International Conference on Computer Engineering, Network and Intelligent Multimedia, CENIM 2022 349–352. https://doi.org/10.1109/CENIM56801.2022.10037425

Hindy, H., Brosset, D., Bayne, E., Seeam, A., Tachtatzis, C., Atkinson, R., Bellekens, X., 2018. A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets 1.

Hua, Y., 2020. An Efficient Traffic Classification Scheme Using Embedded Feature Selection and LightGBM. 2020 Information Communication Technologies Conference, ICTC 2020 125–130. https://doi.org/10.1109/ICTC49638.2020.9123302

IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB [WWW Document], n.d. URL https://www.unb.ca/cic/datasets/ids-2018.html (accessed 8.16.23).

İlker, Ö., Richard, B., 2020. Distributed Denial of Service Attacks Real-world Detection and Mitigation. CRC, Oxon.

Intrusion Detection Systems > Triggering Mechanisms | Cisco Press [WWW Document], n.d. URL https://www.ciscopress.com/articles/article.asp?p=25334 (accessed 8.14.23).

Jiyeon, K., Yulim, S., Choi, E., 2021. An Intrusion Detection Model based on a Convolutional Neural Network. IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC) 6, 634–637. https://doi.org/10.1109/IAEAC50856.2021.9390930

Kabir, S., Sakib, S., Hossain, M.A., Islam, S., Hossain, M.I., 2021. A Convolutional Neural Network based Model with Improved Activation Function and Optimizer for Effective Intrusion Detection and Classification. 2021 International Conference on Advance Computing and Innovative Technologies in Engineering, ICACITE 2021 7, 373–378. https://doi.org/10.1109/ICACITE51222.2021.9404584

Kamboj, P., Trivedi, M.C., Yadav, V.K., Singh, V.K., 2017. Detection techniques of DDoS attacks: A survey. 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics, UPCON 2017 2018-Janua, 675–679. https://doi.org/10.1109/UPCON.2017.8251130

KAREHKA, R., 2012. 10 Uses of Technology in Our Daily Life - Useoftechnology [WWW Document]. URL https://useoftechnology.com/technology-today-tomorrow/ (accessed 8.20.23).

KDD Cup 1999 Data [WWW Document], n.d. URL http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed 2.18.20).

Khor, K.C., Ting, C.Y., Phon-amnuaisuk, S., 2010. Comparing Single and Multiple Bayesian Classifiers Approaches for Network Intrusion Detection. 2010 Second International Conference on Computer Engineering and Applications 2, 325–329. https://doi.org/10.1109/ICCEA.2010.214

Kishore, R., Chauhan, A., 2020. Evaluation of Deep Neural Networks for Advanced Intrusion Detection Systems, in: 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA). pp. 1–8. https://doi.org/10.1109/ICECA49313.2020.9297515

Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., Cunningham, R.K., Zissman, M.A., 2000. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. Proceedings - DARPA Information Survivability Conference and Exposition, DISCEX 2000 2, 12–26. https://doi.org/10.1109/DISCEX.2000.821506

Ludmila I. Kuncheva, James C. Bezdek, Robert P.W. Duin, 2001. Decision Templates for Multiple Classifier Fusion: An Experimental Comparison 34, 299–314.

Margaret, R., 2018. intrusion detection system (IDS) [WWW Document]. URL https://searchsecurity.techtarget.com/definition/intrusion-detection-system (accessed 9.23.19).

Microsoft fends off record-breaking 3.47Tbps DDoS attack | Ars Technica [WWW Document], n.d. URL https://arstechnica.com/information-technology/2022/01/microsoft-fends-off-record-breaking-3-47-tbps-ddos-attack/ (accessed 8.14.23).

MLC++, A Machine Learning Library in C++ [WWW Document], n.d. URL http://robotics.stanford.edu/users/ronnyk/mlc.html (accessed 8.16.23).

MLlib | Apache Spark [WWW Document], n.d. URL https://spark.apache.org/mllib/ (accessed 8.16.23).

OpenCV - Open Computer Vision Library [WWW Document], n.d. URL https://opencv.org/ (accessed 8.16.23).

Personnaz, L., Knerr, S., Gérard Dreyfus, 1990. Single-layer learning revisited: A stepwise procedure for building and training a neural network. Neurocomputing. https://doi.org/10.1007/978-3-642-76153-9

Pricing Calculator | Microsoft Azure [WWW Document], n.d. URL https://azure.microsoft.com/en-us/pricing/calculator/ (accessed 8.15.23).

PyTorch [WWW Document], n.d. URL https://pytorch.org/ (accessed 8.16.23).

R: The R Project for Statistical Computing [WWW Document], n.d. URL https://www.r-project.org/ (accessed 8.16.23).

RapidMiner | Amplify the Impact of Your People, Expertise & Data [WWW Document], n.d. URL https://rapidminer.com/ (accessed 8.16.23).

Sabhnani, M., Serpen, G., 2004. Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set. Intelligent Data Analysis 8, 403–415. https://doi.org/10.3233/ida-2004-8406

Saleem Malik Raja, K., Jeya Kumar, K., 2014. Diversified intrusion detection using Various Detection methodologies with sensor fusion 442–448. https://doi.org/10.1109/iccpeic.2014.6915405

SATzilla: Portfolio-based algorithm selection for SAT [WWW Document], 2017. URL http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/ (accessed 9.28.19).

Sazzadul Hoque, M., 2012. An Implementation of Intrusion Detection System Using Genetic Algorithm. International Journal of Network Security & Its Applications 4, 109–120. https://doi.org/10.5121/ijnsa.2012.4208

scikit-learn: machine learning in Python — scikit-learn 1.3.0 documentation [WWW Document], n.d. URL https://scikit-learn.org/stable/ (accessed 8.16.23).

Seth, S., Chahal, K.K., Singh, G., 2021. A Novel Ensemble Framework for an Intelligent Intrusion Detection System. IEEE Access 9, 138451–138467. https://doi.org/10.1109/ACCESS.2021.3116219

Shahbandayeva, L., Mammadzada, U., Manafova, I., Jafarli, S., Adamov, A.Z., 2022. Network Intrusion Detection using Supervised and Unsupervised Machine Learning. 16th IEEE International Conference on Application of Information and Communication Technologies, AICT 2022 - Proceedings 1–7. https://doi.org/10.1109/AICT55583.2022.10013594

Shorubiga, P., Shyam, R., 2023. CNN-Based Model for the HTTP Flood Attack Detection. 2023 International Conference for Advancement in Technology, ICONAT 2023 1–6. https://doi.org/10.1109/ICONAT57137.2023.10080698

Siddiqi, M.A., Pak, W., 2022. Tier-Based Optimization for Synthesized Network Intrusion Detection System. IEEE Access 10, 108530–108544. https://doi.org/10.1109/ACCESS.2022.3213937

Simone, M.P., 2009. Challenges of Managing an Intrusion Detection System (IDS) in the Enterprise. Sans Institute 27.

Stolfo, S.J., Fan, W., Lee, W., Prodromidis, A., Chan, P.K., 2000. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. Proceedings - DARPA Information Survivability Conference and Exposition, DISCEX 2000 2, 130–144. https://doi.org/10.1109/DISCEX.2000.821515

Subbulakshmi, T., Afroze, A.F., 2013. Multiple learning based classifiers using layered approach and Feature Selection for attack detection. 2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology, ICE-CCN 2013 308–314. https://doi.org/10.1109/ICE-CCN.2013.6528514

Syarif, I., Afandi, R.F., Astika Saputra, F., 2020. Feature selection algorithm for intrusion detection using cuckoo search algorithm. IES 2020 - International Electronics Symposium: The Role of Autonomous and Intelligent Systems for Human Life and Comfort 430–435. https://doi.org/10.1109/IES50839.2020.9231840

Tanmoy, S., Niva, D., 2017. Survey on Host and Network Based Intrusion Detection System. Int. J. Advanced Networking and Applications 6.

Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A., 2015. Proceedings of the 2014 7th IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2014. Proceedings of the 2014 7th IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2014 164p.

TensorFlow [WWW Document], n.d. URL https://www.tensorflow.org/ (accessed 8.16.23).

Thakkar, A., Lohiya, R., 2020. A Review of the Advancement in Intrusion Detection Datasets. Procedia Comput Sci 167, 636–645. https://doi.org/10.1016/j.procs.2020.03.330

The Global DataSphere & Its Enterprise Impact | IDC Blog [WWW Document], n.d. URL https://blogs.idc.com/2019/11/04/how-you-contribute-to-todays-growing-datasphere-and-its-enterprise-impact/ (accessed 8.14.23).

Tonni, Z.A., Mazumder, R., 2023. A Novel Feature Selection Technique for Intrusion Detection System Using RF-RFE and Bio-inspired Optimization. 2023 57th Annual Conference on Information Sciences and Systems, CISS 2023 1–6. https://doi.org/10.1109/CISS56502.2023.10089745

Urvashi, M., Jain, M.A., 2015. A survey of IDS classification using KDD CUP 99 dataset in WEKA. Int J Sci Eng Res 6, 947–954.

Weka 3 - Data Mining with Open Source Machine Learning Software in Java [WWW Document], n.d. URL https://www.cs.waikato.ac.nz/ml/weka/ (accessed 8.16.23).

What is CRISP DM? - Data Science Process Alliance [WWW Document], n.d. URL https://www.datascience-pm.com/crisp-dm-2/ (accessed 8.15.23).

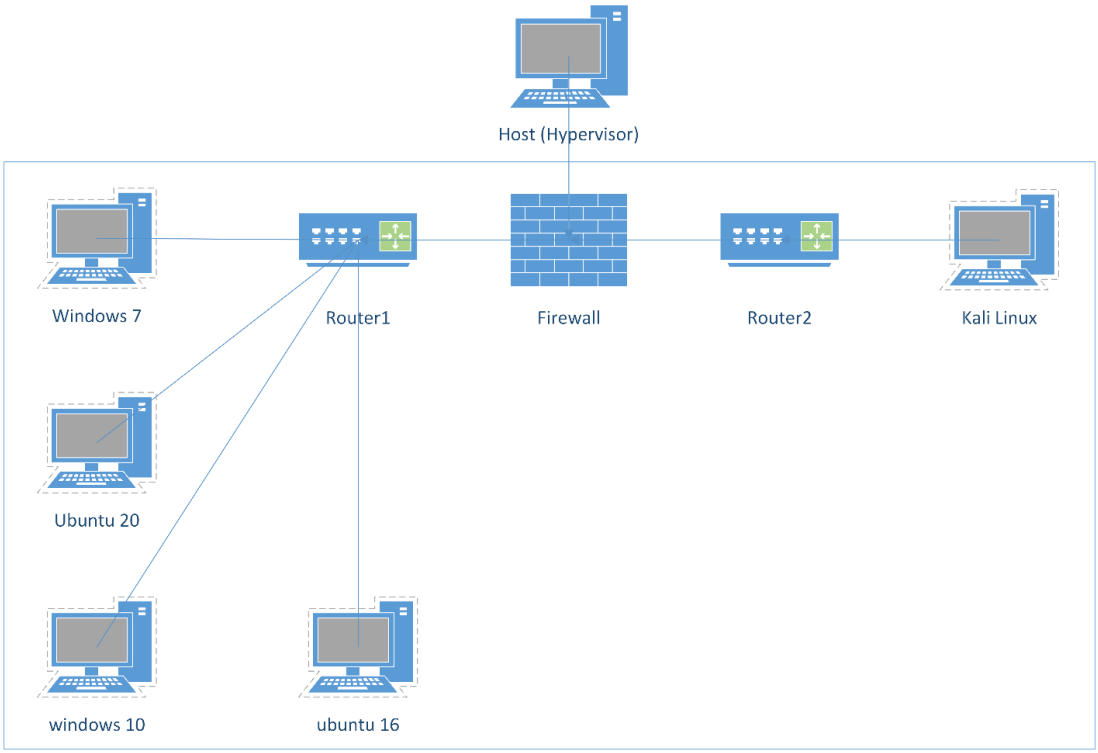
What Is Packet Capture (PCAP)? - IT Glossary | SolarWinds, n.d.

Yoo, J., Min, B., Kim, S., Shin, Dongil, Shin, Dongkyoo, 2021. Study on Network Intrusion Detection Method Using Discrete Pre-Processing Method and Convolution Neural Network. IEEE Access 9, 142348–142361. https://doi.org/10.1109/ACCESS.2021.3120839

Appendix :

building PCAP

In order to test models with real traffic data a network infrastructure was built on some servers. The infrastructure built as follows:



1- Host is the physical hypervisor that will host all the virtual machines
2- Virtual Firewall is PFSense which has three interfaces for routing
    a. Private LAN network for Kali
    b. Private LAN network for the victims
    c. WAN Network with the host to allow internet traffic
3- Virtual Ubuntu 20 for recent attacks(old attacks will not work)
    a. SSH open
    b. FTP open

c. Open SS enable

d. Apache enabled

e. Firewall disabled

4- Virtual Windows 10 for recent attacks (old attacks will not work)

a. IIS installed

5- Virtual windows 7 (exploitable for many attacks)

a. Old adobe acrobat 9.0 is installed

6- Virtual Ubuntu 16 (exploitable for many attacks)

a. SSH open

b. FTP open

c. Open SS enable

d. Apache enabled

e. Firewall disabled

The firewall was placed in the middle to force all traffic to pass through, so the data collection will be easier. The PFsense was connected to wireshark in the host via SSH to generate the PCAP on the fly.

Each virtual node in the victim side has a python script that will access a random website to simulate benign traffic

```
import time
import webbrowser
import random
import os
```

```
f = open("websites.csv", "r")
lines=f.readlines()
t_end = time.time() + 60 * 20
while time.time() < t_end:
    webbrowser.open(random.choice(lines),new=0)
    time.sleep(15)
    os.system("taskkill /im firefox.exe /f")
    os.system("taskkill /im msedge.exe /f")
    os.system("taskkill /im iexplore.exe /f")
```

Generating attacks:

Kali Linux was prepared with script to initiate attacks following attacks

- patator ssh_login
- patator ftp_login
- hulk-master
- GoldenEye
- Slowloris
- slowhttptest
- msfconsole

each attack is timed and a time stamp is recorded with timestamp, Source IP and destination IP. Once the attacks are done and the data is extracted from wireshark, the pcap fille is processed in CICFlow to extract features and generate flows ready

for machine learning. From the matching timestamsp and destination IP we can label the flows with the corresponding attack in the scipt.

Failed Attempts:

1- SVM

SVM is binary classifier by in its core. In order, to test with Multiclass, the library (e1071) was used to build SVM model. Unfortunately, this library consumed a lot of resources and time without getting an out.

2- KNN

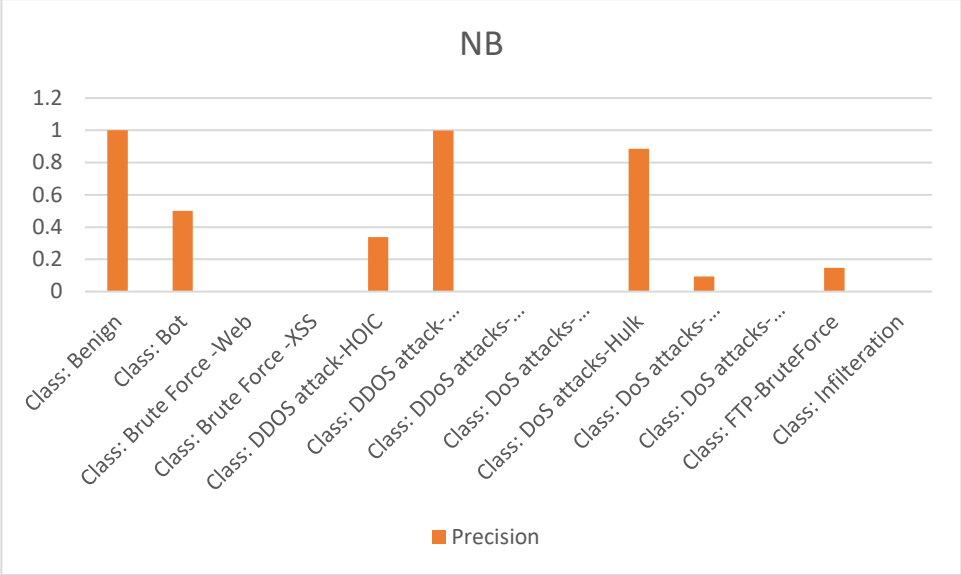
This model was built, but the results were not optimistic to be included.

Naïve Bayes

Naïve Bayes was used in RStudio using the H2o Package. This function or algorithm depends on the assumption that each predictor is independent and a Gaussian distribution with numeric predictors that has mean, and STD computed in the training set. The same goes as we did the past models, we have provided the function with Training Data to build it.

Parameter	*Value*	*Description*
model_id	NBModel	Destination id for this model; auto-generated if not specified.
nfolds	5	Number of folds for K-fold cross-validation (0 to disable or >= 2).
seed	1234	Seed for pseudo random number generator (only used for cross-validation and
fold_assignment	Random	Cross-validation fold assignment scheme, if fold_column is not specified. The 'S on the response variable, for classification problems.
response_column	Class	Response variable column.
ignored_columns		Names of columns to ignore for training.

Accuracy	Kappa
0.26684434	0.21447607
AccuracyLower	
AccuracyUpper	
0.26563995	0.26805133
AccuracyNull	
AccuracyPValue	
0.06666667	0.00000000
McnemarPValue	
NaN	



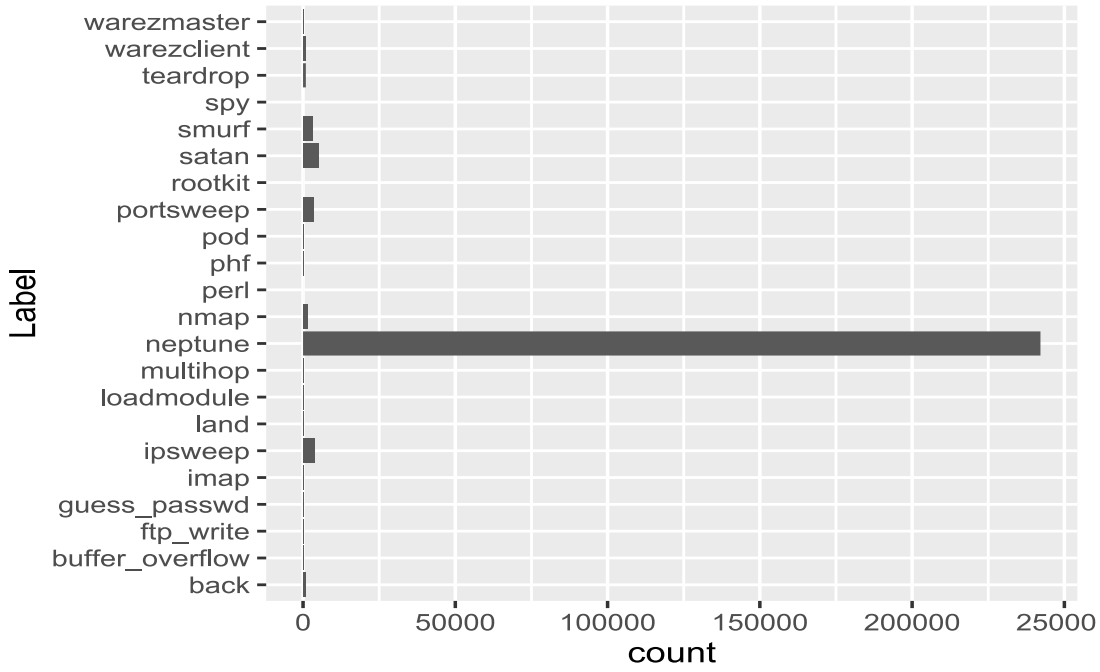
Test on KDDCUP

the KDDcup dataset was preprocessed by removing the duplicates and empty rows.

The following represent the dataset classes without the normal class.

names	count	Percentage
back	968	0.090
buffer_overflow	30	0.003
ftp_write	8	0.001
guess_passwd	53	0.005
imap	12	0.001
ipsweep	3723	0.346
land	19	0.002
loadmodule	9	0.001
multihop	7	0.001
neptune	242149	22.526
nmap	1554	0.145

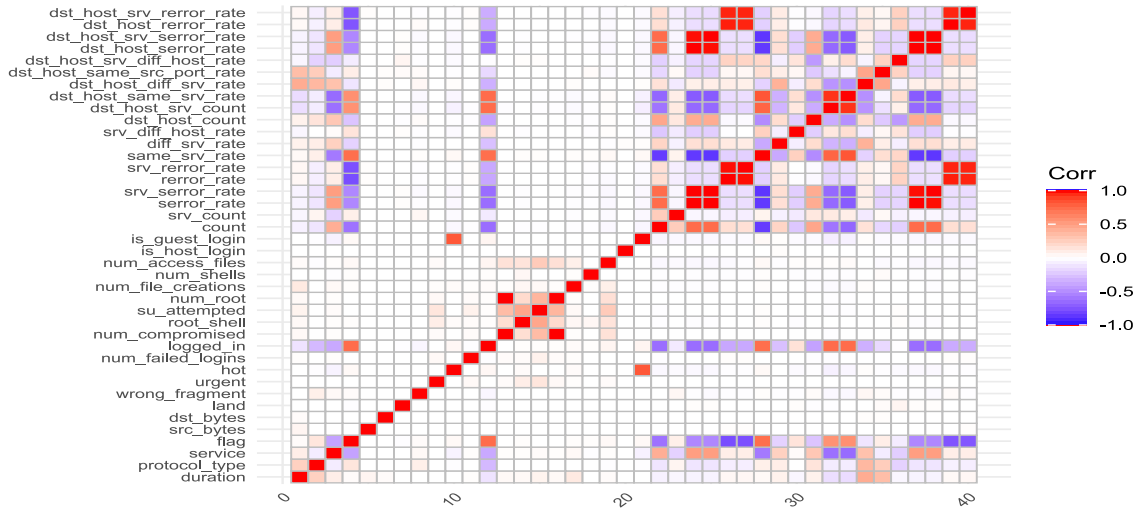
normal	812814	75.611
perl	3	0.000
phf	4	0.000
pod	206	0.019
portsweep	3564	0.332
rootkit	10	0.001
satan	5019	0.467
smurf	3007	0.280
spy	2	0.000
teardrop	918	0.085
warezclient	893	0.083
warezmaster	20	0.002
Total	1074992	100.000



In the above graph, we can see the distribution of threat types, and we clearly find that the Neptune attacks are has a huge count compared to the rest. This is maybe a result of removing duplicates and missing values.

Correlation heatmap:

We can see that most of the correlated features are the features related to the error rate and count. And these relations are expected since they are mostly derived from the error count.

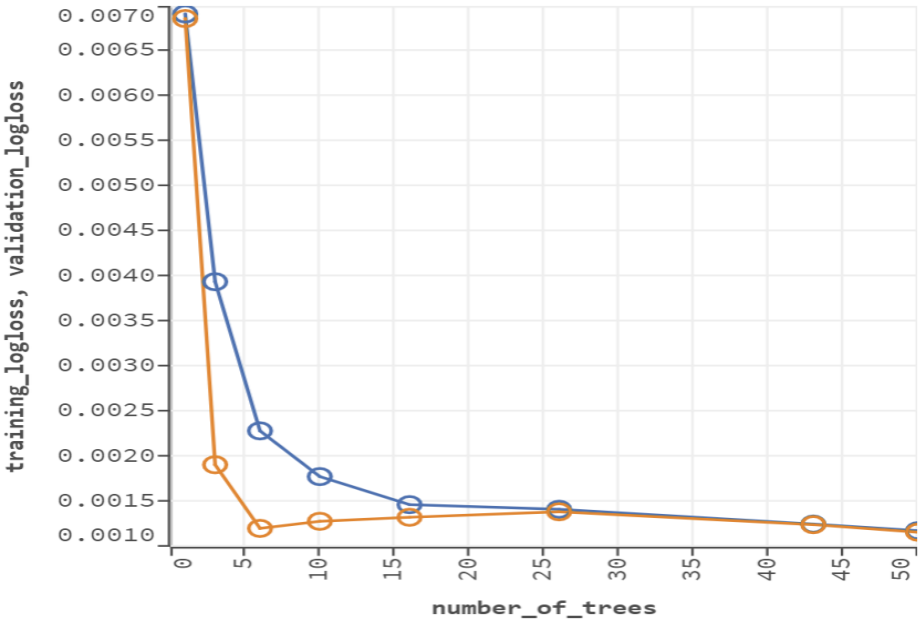


First Run using Distributed Random Forest

At this stage, a distributed random forest model is created with the complete set of features. This step will help to determine or select features, so we can reduce the Dataset with the same accuracy. Also, it will assist in having a benchmark to compare with the reduced model.

Distributed Random Forest Training:

▼ SCORING HISTORY - LOGLOSS



Training matrix :

This is the training matrix of the distributed random forest with a training size equal to 0.75 of the Dataset.

model	**drf-62b6f06b-971b-4e1b-837c-27a467942b9b**
model_checksum	$ (272,627,320,440,848,000.00)
frame	frame_0.750
frame_checksum	$ 7,848,675,700,881,100,000.00
description	Metrics reported on Out-Of-Bag training samples
model_category	Multinomial
scoring_time	$ 1,617,119,055,988.00
predictions	·
MSE	$ 0.00
RMSE	$ 0.01
nobs	$ 3,673,643.00
custom_metric_name	·
custom_metric_value	$ -
r2	$ 1.00
logloss	$ 0.00
mean_per_class_error	$ 0.38
AUC	NaN
pr_auc	NaN
multinomial_auc_table	·
multinomial_aucpr_table	·

Training Confusion matrix

	back	buffer_overflow	ftp_write	guess_passwd	imap	ipsweep	land	loadmodule	multihop	neptune	nmap	normal	perl	phf	pod	portsweep	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster	Error	Rate	Precision
back	1642	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0.003	5 / 1,647	1
buffer_overflow	0	13	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	2	0	0.48	44555	0.87
ftp_write	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	1	44353	NaN
guess_passwd	0	0	0	41	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0.0465	15738	1
imap	0	0	0	0	2	0	0	0	0	0	2	3	0	0	0	0	0	0	0	0	0	0	0	0.7143	44323	1
ipsweep	0	0	0	0	0	9358	0	0	0	0	1	35	0	0	0	0	0	0	0	0	0	0	0	0.0038	36 / 9,394	1
land	0	0	0	0	0	0	15	0	0	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0.1667	44273	0.75
loadmodule	0	1	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	1	0	1	44416	0
multihop	0	1	0	0	0	0	0	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	44321	NaN
neptune	0	0	0	0	0	0	0	0	0	80392	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1 / 803,993	1
nmap	0	0	0	0	0	17	0	0	0	0	1676	24	0	0	0	0	0	0	0	0	0	0	0	0.0239	41 / 1,717	1
normal	0	0	0	0	0	5	5	0	0	0	0	729167	0	0	0	1	0	3	1	0	0	13	0	0	28 / 729,195	1
perl	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
phf	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	44258	NaN
pod	0	0	0	0	0	0	0	0	0	0	0	3	0	0	196	0	0	0	0	0	0	0	0	0.0151	3 / 199	1
portsweep	0	0	0	0	0	0	0	0	0	1	0	12	0	0	0	7805	0	2	0	0	0	0	0	0.0019	15 / 7,820	1
rootkit	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	1	44479	NaN
satan	0	0	0	0	0	1	0	0	0	0	0	74	0	0	0	0	0	11803	0	0	0	0	0	0.0063	75 / 11,878	1
smurf	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	2106128	0	0	0	0	0	3 / 2,106,131	1
spy	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
teardrop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	738	0	0	0	0 / 738	1
warezclient	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	743	0	0.0595	47 / 790	0.98
warezmaster	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	9	0.3077	44299	1
Total	1642	15	0	41	2	9381	20	1	0	803995	1679	729422	0	0	196	7806	0	11808	2106129	0	738	759	9	0.0001	315 / 3,673,643	
Recall	1	0.52	0	0.95	0.29	1	0.83	0	0	1	0.98	1	0	0	0.98	1	0	0.99	1	0	1	0.94	0.69			

Validation Matrix:

0.25 of the Dataset was used for validation.

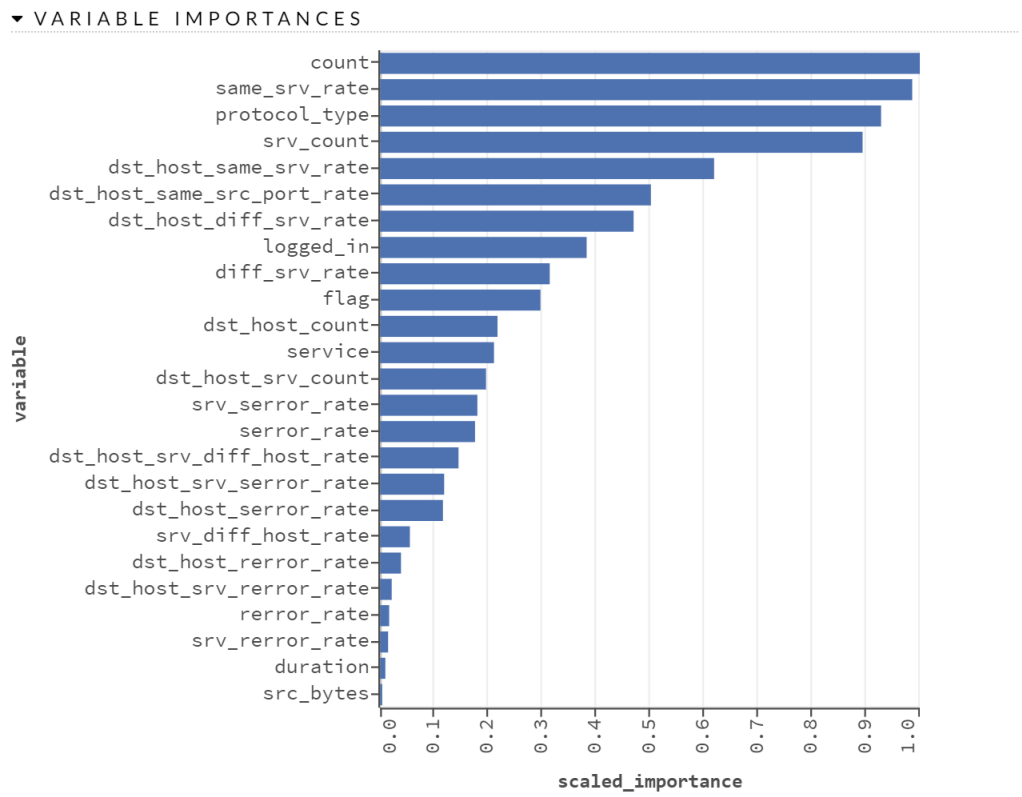
model	**drf-62b6f06b-971b-4e1b-837c-27a467942b9b**
model_checksum	-2.72627E+17
frame	frame_0.250
frame_checksum	3.28803E+18
description	·
model_category	Multinomial
scoring_time	1.61712E+12
predictions	·
MSE	0.000123
RMSE	0.011074
nobs	1224788
custom_metric_name	·
custom_metric_value	0
r2	0.999993
logloss	0.001162
mean_per_class_error	0.276288
AUC	NaN
pr_auc	NaN
multinomial_auc_table	·
multinomial_aucpr_table	·

Confusion matrix :

	back	buffer_overflow	ftp_write	guess_passwd	imap	ipsweep	land	loadmodule	multihop	neptune	nmap	normal	perl	phf	pod	portsweep	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster	Error	Rate	Precision
back	554	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0.0036	2 / 556	1
buffer_overflow	0	3	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0.4	44232	1
ftp_write	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
guess_passwd	0	0	0	9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0.1	44206	1
imap	0	0	0	0	3	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0.4	44232	1
ipsweep	0	0	0	0	0	3073	0	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0	0	0.0045	14 / 3,087	1
land	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 3	0.6
loadmodule	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
multihop	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
neptune	0	0	0	0	0	0	0	0	0	268024	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 / 268,024	1
nmap	0	0	0	0	0	12	0	0	0	0	573	14	0	0	0	0	0	0	0	0	0	0	0	0.0434	26 / 599	1
normal	0	0	0	0	0	2	2	0	0	0	0	243579	0	0	0	0	0	0	0	0	0	3	0	0	7 / 243,586	1
perl	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
phf	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
pod	0	0	0	0	0	0	0	0	0	0	0	2	0	0	63	0	0	0	0	0	0	0	0	0.0308	23774	1
portsweep	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	2584	0	1	0	0	0	0	0	0.0035	9 / 2,593	1
rootkit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
satan	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	3993	0	0	0	0	0	0.0052	21 / 4,014	1
smurf	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	701754	0	0	0	0	0	1 / 701,755	1
spy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0Rate: spy	NaN
teardrop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	239	0	0	0.0083	2 / 241	1
warezclient	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	214	0	0.0696	16 / 230	0.99
warezmaster	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	5	0.2857	44234	1
Total	554	3	0	9	3	3087	5	0	0	268025	573	243671	0	0	63	2584	0	3996	701754	0	239	217	5	0.0001	115 / 1,224,788	
Recall	1	0.6	0	0.9	0.6	1	1	0	0	1	0.96	1	0	0	0.97	1	NaN	0.99	1	NaN	0.99	0.93	0.71			

Variable Importance

At this stage, we can determine which factors we can use for the second test with random forest. Based on the scaled_importance, we have selected factors that have a value greater than 0.3.



Variable Importance Table

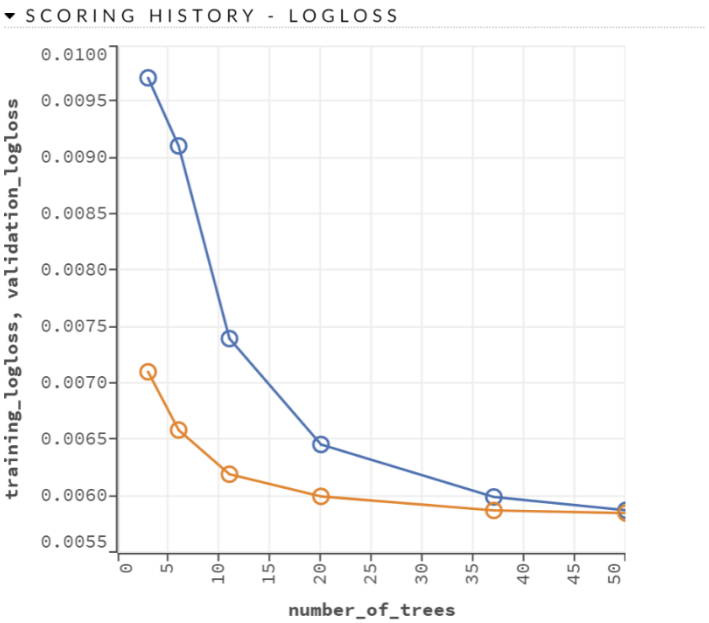
variable	relative_importance	scaled_importance	percentage
count	11859210	1	0.1263
same_srv_rate	11688206	0.9856	0.1245
protocol_type	11003613	0.9279	0.1172
srv_count	10597117	0.8936	0.1128
dst_host_same_srv_rate	7336097	0.6186	0.0781
dst_host_same_src_port_rate	5948306	0.5016	0.0633
dst_host_diff_srv_rate	5567899.5	0.4695	0.0593
logged_in	4536479.5	0.3825	0.0483
diff_srv_rate	3725651.75	0.3142	0.0397
flag	3521670	0.297	0.0375
dst_host_count	2578363.75	0.2174	0.0275
service	2501159.5	0.2109	0.0266
dst_host_srv_count	2326045.25	0.1961	0.0248
srv_serror_rate	2136325.75	0.1801	0.0227
serror_rate	2085115.375	0.1758	0.0222
dst_host_srv_diff_host_rate	1722246.375	0.1452	0.0183
dst_host_srv_serror_rate	1406472.5	0.1186	0.015
dst_host_serror_rate	1380117.125	0.1164	0.0147
srv_diff_host_rate	653914.875	0.0551	0.007
dst_host_rerror_rate	458241.8125	0.0386	0.0049
dst_host_srv_rerror_rate	254065.2188	0.0214	0.0027
rerror_rate	198456.9219	0.0167	0.0021
srv_rerror_rate	175007.0625	0.0148	0.0019
duration	115320.625	0.0097	0.0012
src_bytes	45980.3047	0.0039	0.0005

hot	34724.5781	0.0029	0.0004
wrong_fragment	32910.9648	0.0028	0.0004
num_compromised	13082.9023	0.0011	0.0001
dst_bytes	5480.2305	0.0005	0.0001
is_guest_login	1813.0879	0.0002	0
num_failed_logins	932.4665	0.0001	0
land	364.8605	0	0
num_root	332.1861	0	0
root_shell	278.7813	0	0
num_file_creations	221.6001	0	0
num_access_files	182.6476	0	0
num_shells	68.4239	0	0
su_attempted	54.1832	0	0
urgent	28.6472	0	0
is_host_login	0	0	0

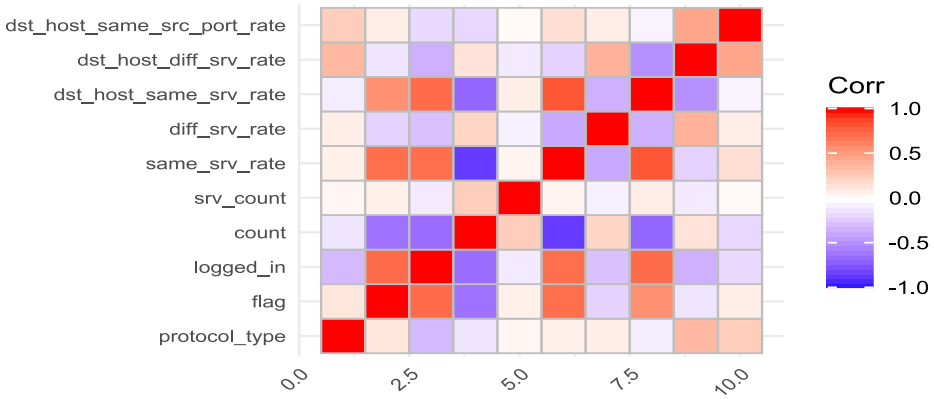
Distributed Random Forest After reduction

At this stage, we have reduced the Dataset based on the most important classes or factors. A distributed random forest is built with this reduction.

Scoring of the Distributed Random forest after feature reduction



Correlation After Reduction



Model Training Matrix

model	drf-e1fea77c-d26d-42cb-b329-aad94a5cc57a
model_checksum	-7.30695E+17
frame	frame_0.750
frame_checksum	7.84868E+18
description	Metrics reported on Out-Of-Bag training samples
model_category	Multinomial
scoring_time	1.61712E+12
predictions	·
MSE	0.001356
RMSE	0.03683
nobs	3673643
custom_metric_name	·
custom_metric_value	0
r2	0.999917
logloss	0.005877
mean_per_class_error	0.6047
AUC	NaN
pr_auc	NaN
multinomial_auc_table	·
multinomial_aucpr_table	·

Confusion Matrix for Training

	back	buffer_overflow	ftp_write	guess_passwd	imap	ipsweep	land	loadmodule	multihop	neptune	nmap	normal	perl	phf	pod	portsweep	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster	Error	Rate	Precision
back	73	0	0	0	0	0	0	0	0	0	0	1573	0	0	0	0	0	0	0	0	0	1	0	0.9557	1,574 / 1,647	0.89
buffer_overflow	0	0	0	0	0	0	0	0	0	1	0	23	0	0	0	0	0	0	0	0	0	1	0	1	25 / 25	NaN
ftp_write	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	1	44353	NaN
guess_passwd	0	0	0	36	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0.1628	15888	0.95
imap	0	0	0	0	1	0	0	0	0	0	1	5	0	0	0	0	0	0	0	0	0	0	0	0.8571	44354	1
ipsweep	0	0	0	0	0	6739	0	0	0	0	1	2648	0	0	1	0	0	0	5	0	0	0	0	0.2826	2,655 / 9,394	0.9
land	0	0	0	0	0	0	0	0	0	2	0	16	0	0	0	0	0	0	0	0	0	0	0	1	18 / 18	NaN
loadmodule	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	1	44416	0
multihop	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	1	44321	NaN
neptune	0	0	0	0	0	0	0	0	0	803979	0	13	0	0	0	0	0	1	0	0	0	0	0	0	14 / 803,993	1
nmap	0	0	0	0	0	733	0	0	0	0	906	74	0	0	3	0	0	0	1	0	0	0	0	0.4723	811 / 1,717	0.97
normal	7	0	0	2	0	3	0	1	0	10	24	729043	0	0	10	6	0	17	6	0	1	65	0	0.0002	152 / 729,195	0.99
perl	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
phf	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	44258	NaN
pod	1	0	0	0	0	48	0	0	0	0	0	26	0	0	108	0	0	1	15	0	0	0	0	0.4573	91 / 199	0.77
portsweep	0	0	0	0	0	0	0	0	0	2	0	15	0	0	0	7802	0	1	0	0	0	0	0	0.0023	18 / 7,820	1
rootkit	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	1	44479	NaN
satan	0	0	0	0	0	0	0	0	0	2	0	93	0	0	3	3	0	11775	2	0	0	0	0	0.0087	103 / 11,878	1
smurf	0	0	0	0	0	0	0	0	0	0	0	21	0	0	15	0	0	1	2106094	0	0	0	0	0	37 / 2,106,131	1
spy	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	2 / 2Rate: spy	NaN
teardrop	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	722	0	0	0.0217	16 / 738	1
warezclient	1	0	0	0	0	0	0	0	0	0	0	542	0	0	0	0	0	0	0	0	0	247	0	0.6873	543 / 790	0.79
warezmaster	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0	0	1	13 / 13	NaN
Total	82	0	0	38	1	7523	0	1	0	803996	932	734163	0	0	140	7811	0	11796	2106123	0	723	314	0	0.0017	6,118 / 3,673,643	
Recall	0.04	0	0	0.84	0.14	0.72	0	0	0	1	0.53	1	0	0	0.54	1	0	0.99	1	0	0.98	0.31	0			

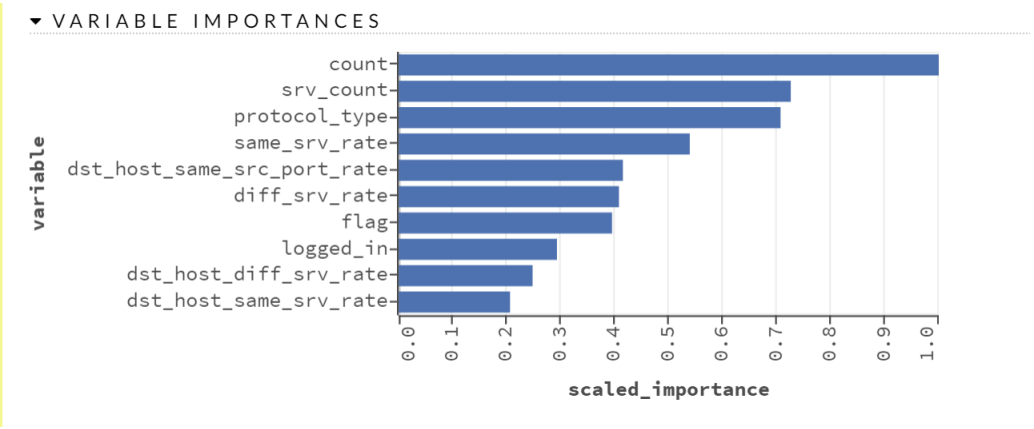
Validation Metric

model	drf-e1fea77c-d26d-42cb-b329-aad94a5cc57a
model_checksum	-7.30695E+17
frame	frame_0.250
frame_checksum	3.28803E+18
description	·
model_category	Multinomial
scoring_time	1.61712E+12
predictions	·
MSE	0.001375
RMSE	0.037088
nobs	1224788
custom_metric_name	·
custom_metric_value	0
r2	0.999916
logloss	0.005852
mean_per_class_error	0.511234
AUC	NaN
pr_auc	NaN
multinomial_auc_table	·
multinomial_aucpr_table	·

Confusion Matrix for Validation

	back	buffer_overflow	ftp_write	guess_passwd	imap	ipsweep	land	loadmodule	multihop	neptune	nmap	normal	perl	phf	pod	portsweep	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster	Error	Rate	Precision
back	18	0	0	0	0	0	0	0	0	0	0	537	0	0	0	0	0	0	0	0	0	1	0	0.9676	538 / 556	0.9
buffer_overflow	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	1	0	1	44321	NaN
ftp_write	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
guess_passwd	0	0	0	8	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0.2	44237	0.8
imap	0	0	0	0	2	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0.6	44260	1
ipsweep	0	0	0	0	0	2225	0	0	0	0	0	862	0	0	0	0	0	0	0	0	0	0	0	0.2792	862 / 3,087	0.89
land	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	44258	NaN
loadmodule	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
multihop	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
neptune	0	0	0	0	0	0	0	0	0	268018	0	6	0	0	0	0	0	0	0	0	0	0	0	0	6 / 268,024	1
nmap	0	0	0	0	0	262	0	0	0	0	293	40	0	0	4	0	0	0	0	0	0	0	0	0.5109	306 / 599	0.95
normal	2	0	0	2	0	1	0	0	0	3	16	243537	0	0	2	1	0	7	1	0	0	14	0	0.0002	49 / 243,586	0.99
perl	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	44229	NaN
phf	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
pod	0	0	0	0	0	18	0	0	0	0	0	7	0	0	38	0	0	1	1	0	0	0	0	0.4154	27 / 65	0.81
portsweep	0	0	0	0	0	0	0	0	0	1	0	11	0	0	0	2580	0	1	0	0	0	0	0	0.005	13 / 2,593	1
rootkit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
satan	0	0	0	0	0	0	0	0	0	0	0	31	0	0	0	0	0	3983	0	0	0	0	0	0.0077	31 / 4,014	1
smurf	0	0	0	0	0	0	0	0	0	0	0	4	0	0	3	0	0	0	701748	0	0	0	0	0	7 / 701,755	1
spy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
teardrop	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	233	0	0	0.0332	8 / 241	1
warezclient	0	0	0	0	0	0	0	0	0	0	0	170	0	0	0	0	0	0	0	0	0	60	0	0.7391	170 / 230	0.79
warezmaster	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	1	44384	NaN
Total	20	0	0	10	2	2506	0	0	0	268022	309	245240	0	0	47	2581	0	3992	701750	0	233	76	0	0.0017	2,045 / 1,224,788Total: Rate	

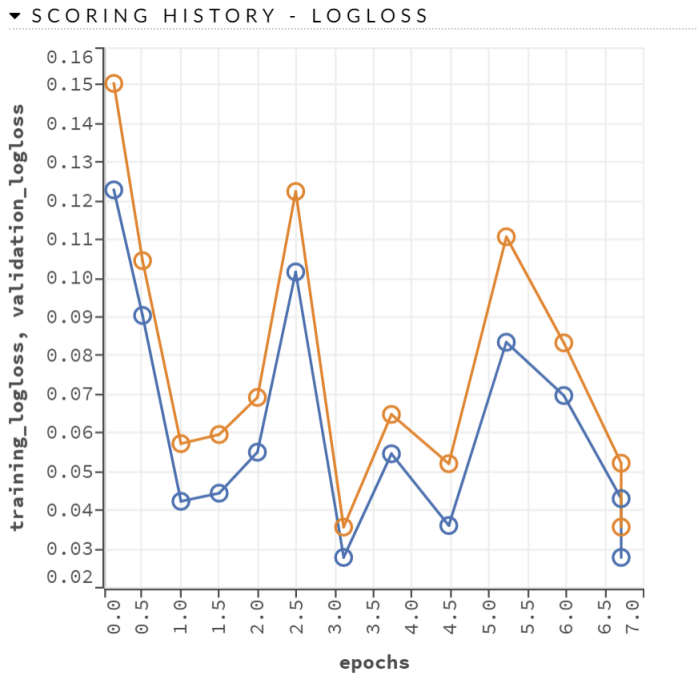
Variable Importance



Variable Importance Table

variable	relative_importance	scaled_importance	percentage
count	20127888	1	0.2026
srv_count	14606322	0.7257	0.147
protocol_type	14226244	0.7068	0.1432
same_srv_rate	10842952	0.5387	0.1092
dst_host_same_src_port_rate	8352236	0.415	0.0841
diff_srv_rate	8204572	0.4076	0.0826
flag	7946105	0.3948	0.08
logged_in	5893682	0.2928	0.0593
dst_host_diff_srv_rate	4986479	0.2477	0.0502
dst_host_same_srv_rate	4146581	0.206	0.0417

Test With Deep learning (DNN) on reduced features



Training matrix

model	deeplearning-0594f7fe-b0b5-457a-abc8-78fb91d7c0b6
model_checksum	-1.8E+18
frame	.
frame_checksum	0
description	Metrics reported on temporary training frame with 9976 samples
model_category	Multinomial
scoring_time	1.62E+12
predictions	.
MSE	0.004302
RMSE	0.065587
nobs	9976
custom_metric_name	.
custom_metric_value	0
r2	0.997001
logloss	0.02805

mean_per_class_error	0.235354
AUC	NaN
pr_auc	NaN
multinomial_auc_table	·
multinomial_aucpr_table	·

Confusion Matrix

	back	buffer_overflow	ftp_write	guess_passwd	imap	ipsweep	land	loadmodule	multihop	neptune	nmap	normal	perl	phf	pod	portsweep	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster	Error	Rate	Precision
back	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	1	44384	NaN
buffer_overflow	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
ftp_write	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
guess_passwd	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
imap	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
ipsweep	0	0	0	0	0	25	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0.2424	12267	0.78
land	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
loadmodule	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
multihop	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
neptune	0	0	0	0	0	0	0	0	0	2222	0	0	0	0	0	0	0	2	0	0	0	0	0	0.0009	2 / 2,224	1
nmap	0	0	0	0	0	5	0	0	0	0	3	2	0	0	0	0	0	0	0	0	0	0	0	0.7	44387	1
normal	0	0	0	0	0	0	0	0	0	1	0	7569	0	0	0	0	0	8	0	0	0	0	0	0.0012	9 / 7,578	1
perl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
phf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
pod	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
portsweep	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31	0	1	0	0	0	0	0	0.0313	11689	1
rootkit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
satan	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	38	0	0	0	0	0	0.05	14642	0.75
smurf	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0.0541	13547	1
spy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
teardrop	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	6	0	0	0.3333	44264	1
warezclient	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	4 / 4	NaN
warezmaster	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
Total	0	0	0	0	0	32	0	0	0	2223	3	7595	0	0	0	31	0	51	35	0	6	0	0	0.0047	47 / 9,976	
Recall	0	NaN	NaN	0	NaN	0.76	NaN	NaN	0	1	0.3	1	NaN	NaN	NaN	0.97	NaN	0.95	0.95	NaN	0.67	0	NaN			

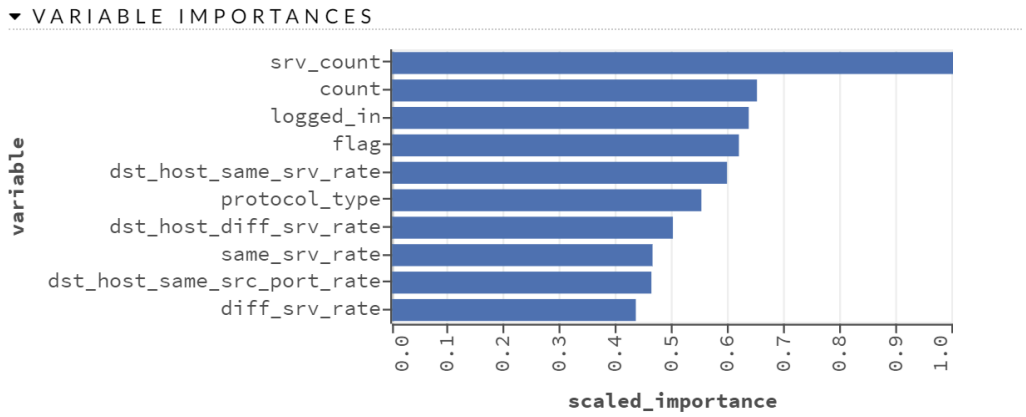
Validation Metric

model	deeplearning-0594f7fe-b0b5-457a-abc8-78fb91d7c0b6
model_checksum	-1.8E+18
frame	frame_0.250
frame_checksum	1.02E+18
description	Metrics reported on full validation frame
model_category	Multinomial
scoring_time	1.62E+12
predictions	·
MSE	0.005857
RMSE	0.076533
nobs	268677
custom_metric_name	·
custom_metric_value	0
r2	0.996096
logloss	0.035909
mean_per_class_error	0.544477
AUC	NaN
pr_auc	NaN
multinomial_auc_table	·
multinomial_aucpr_table	·

Validation Confusion Matrix

	back	buffer_overflow	ftp_write	guess_passwd	imap	ipsweep	land	loadmodule	multihop	neptune	nmap	normal	perl	phf	pod	portsweep	rootkit	satan	smurf	spy	teardrop	warezclient	warezmaster	Error	Rate	Precision
back	0	0	0	0	0	0	0	0	0	0	0	250	0	0	0	0	0	0	0	0	0	0	0	1	250 / 250	NaN
buffer_overflow	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	44258	NaN
ftp_write	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	1	44321	NaN
guess_passwd	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	1	44384	NaN
imap	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
ipsweep	0	0	0	0	0	731	0	0	0	0	0	221	0	0	0	0	0	1	0	0	0	0	0	0.2329	222 / 953	0.73
land	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	44290	NaN
loadmodule	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	1	44290	NaN
multihop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
neptune	0	0	0	0	0	8	0	0	0	60371	0	15	0	0	0	1	0	193	0	0	0	0	0	0.0036	217 / 60,588	1
nmap	0	0	0	0	0	233	0	0	0	0	64	83	0	0	0	0	0	0	0	0	0	0	0	0.8316	316 / 380	0.9
normal	0	0	0	0	0	4	0	0	0	19	7	202804	0	0	0	17	0	184	2	0	20	0	0	0.0012	253 / 203,057	1
perl	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
phf	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
pod	0	0	0	0	0	9	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	1	44 / 44	NaN
portsweep	0	0	0	0	0	0	0	0	0	1	0	13	0	0	0	871	0	15	0	0	0	0	0	0.0322	29 / 900	0.98
rootkit	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	44197	NaN
satan	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	1220	0	0	7	0	0	0.0409	52 / 1,272	0.74
smurf	0	0	0	0	0	19	0	0	0	0	0	11	0	0	0	0	0	0	720	0	0	0	0	0.04	30 / 750	1
spy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0 / 0	NaN
teardrop	0	0	0	0	0	0	0	0	0	0	0	34	0	0	0	0	0	45	0	0	153	0	0	0.3405	79 / 232	0.85
warezclient	0	0	0	0	0	0	0	0	0	0	0	219	0	0	0	0	0	0	0	0	0	0	0	1	219 / 219	NaN
warezmaster	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	1	44384	NaN
Total	0	0	0	0	0	1004	0	0	0	60391	71	203762	0	0	0	889	0	1658	722	0	180	0	0	0.0065	1,743 / 268,677	
Recall	0	0	0	0	0	0.77	0	0	NaN	1	0.17	1	NaN	NaN	0	0.97	0	0.96	0.96	NaN	0.66	0	0			

Variable Importance



Variable Importance Table

variable	relative_importance	scaled_importance	percentage
srv_count	1	1	0.1691
count	0.6503	0.6503	0.11
logged_in	0.6356	0.6356	0.1075
flag	0.6181	0.6181	0.1045
dst_host_same_srv_rate	0.5969	0.5969	0.1009
protocol_type	0.5512	0.5512	0.0932
dst_host_diff_srv_rate	0.5004	0.5004	0.0846
same_srv_rate	0.4642	0.4642	0.0785
dst_host_same_src_port_rate	0.4621	0.4621	0.0782
diff_srv_rate	0.4344	0.4344	0.0735

Scripts

Create Models

```
#Libaraies

kddcup_data_corrected<-NULL

prediction<-NULL

Sample<-NULL

TrainSample<-NULL

ValidateSampel<-NULL

predictSample<-NULL

library(readr)

library(dbplyr)

library(tidyverse)

library(ranger)

library(caret)

library(mlbench)

library(dplyr)
```

```
library(magrittr)
library(nnet)
library(caret)


number2binary = function(number, noBits) {
  binary_vector = rev(as.numeric(intToBits(number)))
  if(missing(noBits)) {
    return(binary_vector)
  } else {
    binary_vector[-(1:(length(binary_vector) - noBits))]
  }
}
upsampleData = function(SSample,percentage){


  set.seed(234)
  sampledData <- upSample(x = SSample[,-ncol(SSample)],
                          y = SSample$Class)
  sampledData$Class<-as.factor(sampledData$Class)
```

```
  Sample <- sampledData %>% group_by(Class) %>%
sample_n(if_else(n()*percentage<1,1,n()*percentage))


  Sample$Class<-as.factor(Sample$Class)
  levels(Sample$Class)
  summary(Sample$Class)
  return (Sample)



}


loadCleanData<-function(){
  CIC<-read_csv(file ="CICLatest.CSV")
  names(CIC)[names(CIC) == "Label"] <- "Class"
  #importVAR<-read.csv("D:\\CIC
PHD\\DataSource\\ImportVarForReductiom.csv",header = TRUE)


  #CIC<-CIC[,importVAR$importantVal]
```

```
  return (CIC)
}
getSample<-function(percentage){
  kddcup_data_corrected$Class<-as.factor(kddcup_data_corrected$Class)

  Sample <- kddcup_data_corrected %>% group_by(Class) %>%
sample_n(if_else(n()*percentage<1,1,n()*percentage))

  Sample$Class<-as.factor(Sample$Class)
  levels(Sample$Class)
  summary(Sample$Class)
  return (Sample)
}

RFF<-function(){
```

```
 TrainSample$Class<-as.factor(TrainSample$Class)

 # rg.iris <- ranger(Class ~ ., data = TrainSample, importance =
"impurity",local.importance = TRUE)

 # rg.iris

 # library(data.table)

 # importantVAR<-as.data.table(rg.iris$variable.importance.local)[,Class :=
TrainSample$Class][,lapply(.SD,mean),by=Class]

 #
write.csv(importantVAR,"D://Porcessing//Canadian//newResults//importnatntVAR.csv")

 # ranger::importance(rg.iris)

 rg.iris <- ranger(Class ~ ., data = TrainSample, importance = "impurity")

 rg.iris

 saveRDS(rg.iris, "Models//RFModelToLoad.rds")

 sink("Results//RFModelResult.txt")

 print(rg.iris)

 sink()

 pr<-NULL
```

```
  pr$Validate<-predict(rg.iris,subset(ValidateSampel,,-c(Class)))
  pr$predict<-predict(rg.iris,subset(TrainSample,,-c(Class)))
  return(pr)
}

LoadModels<-function(){
  library(h2o)
  h2o.init()
  Deepmodel<-
h2o::h2o.loadModel("D:\\Porcessing\\kdd\\newResults\\Models\\Deeplearning")
  Allh2oData<-as.h2o(select(TrainSample,-Class))
  Data15H20<-as.h2o(ValidateSampel)
  FinalS<-as.h2o(select(FinalStageSample,-Class))
  pred <- h2o.predict(Deepmodel, Allh2oData)
  pred15<-h2o.predict(Deepmodel, Data15H20)
  FinalSS<-h2o.predict(Deepmodel,FinalS)
  pr<-NULL
  pr$NNpr15<-as.data.frame(pred15)
  pr$NNpr05<-as.data.frame(pred)
```

```
pr$NNFinal<-as.data.frame(FinalSS)



####################DRF############

RF <-h2o::h2o.loadModel("D:\\Porcessing\\kdd\\newResults\\Models\\DRF")

pred <- h2o.predict(RF, Allh2oData)

pred15<-h2o.predict(RF, Data15H20)

FinalSS<-h2o.predict(RF,FinalS)

pr$RFpr15<-as.data.frame(pred15)

pr$RFpr05<-as.data.frame(pred)

pr$RFFinal<-as.data.frame(FinalSS)

#########################GLM <- ###############

GLM <- h2o::h2o.loadModel("D:\\Porcessing\\kdd\\newResults\\Models\\GLM")

pred <- h2o.predict(GLM, Allh2oData)

pred15<-h2o.predict(GLM, Data15H20)

FinalSS<-h2o.predict(GLM,FinalS)

pr$GLMpr15<-as.data.frame(pred15)

pr$GLMpr05<-as.data.frame(pred)
```

```
pr$GLMFinal<-as.data.frame(FinalSS)

######################GBM <- ###############

GBM <- h2o::h2o.loadModel("D:\\Porcessing\\kdd\\newResults\\Models\\GBM")

pred <- h2o.predict(GBM, Allh2oData)

pred15<-h2o.predict(GBM, Data15H20)

FinalSS<-h2o.predict(GBM,FinalS)

pr$GBMpr15<-as.data.frame(pred15)

pr$GBMpr05<-as.data.frame(pred)

pr$GBMFinal<-as.data.frame(FinalSS)

######################KMEANS#########

NB<-h2o::h2o.loadModel("D:\\Porcessing\\kdd\\newResults\\Models\\NB")

pred <- h2o.predict(NB, Allh2oData)

pred15<-h2o.predict(NB, Data15H20)

FinalSS<-h2o.predict(NB,FinalS)

pr$NBpr15<-as.data.frame(pred15)

pr$NBpr05<-as.data.frame(pred)

pr$NBFinal<-as.data.frame(FinalSS)

return(pr)
```

```
}
MultipleClassificaitionH2o<-function(){
  library(h2o)
  h2o.init(max_mem_size = "500g")
  TrainSample$Class<-as.factor(TrainSample$Class)
  ValidateSampel$Class<-as.factor(ValidateSampel$Class)
  train<-as.h2o(TrainSample)
  valid<-as.h2o(ValidateSampel)
  #splits <- h2o.splitFrame(SampleH2o, c(0.90,0.08), seed=1234)
  #train  <- h2o.assign(splits[[1]], "train.hex") # 60%
  #valid  <- h2o.assign(splits[[2]], "valid.hex") # 20%
  response <- "Class"
  predictors <- setdiff(names(train), response)
  predictors
  print("deeplearning Strated")
  m3 <- h2o.deeplearning(
    model_id="NNModel",
    training_frame=train,
```

```
  validation_frame=valid,
  x=predictors,
  y=response,
  nfolds = 15,                            # 10x cross validation
  #keep_cross_validation_fold_assignment = TRUE,
  #fold_assignment = "Stratified",
  activation = "RectifierWithDropout",
  score_each_iteration = TRUE,
  hidden = c(200, 200),          # 5 hidden layers, each of 200 neurons
  epochs = 15,
  variable_importances = TRUE,
  export_weights_and_biases = TRUE,
  seed = 42


)
 # hyper_params <- list( balance_classes = c(TRUE, FALSE) )
 # grid <- h2o.grid(x = predictors, y = response, training_frame = train, validation_frame
= valid,
```

```
#                       algorithm = "deeplearning", grid_id = "NNGrid", hyper_params =
hyper_params,
#                search_criteria = list(strategy = "Cartesian"), seed = 1234)
#
# # Sort the grid models by logloss
# sorted_grid <- h2o.getGrid("NNGrid", sort_by = "logloss", decreasing = FALSE)
# sorted_grid



print("model built")


h2o.performance(m3, train=T)         ## sampled training data (from model building)
h2o.performance(m3, valid=T)         ## sampled validation data (from model building)
h2o.performance(m3, newdata=train)    ## full training data
h2o.performance(m3, newdata=valid)    ## full validation data
Allh2oData<-as.h2o(subset(TrainSample,,-c(Class)))
Data15H20<-as.h2o(ValidateSampel)
pred <- h2o.predict(m3, Allh2oData)
pred15<-h2o.predict(m3, Data15H20)
```

```
 # predG <- h2o.predict(h2o.getModel(sorted_grid@model_ids[[1]]), Allh2oData)
 # pred15G<-h2o.predict(h2o.getModel(sorted_grid@model_ids[[1]]), Data15H20)
 pr<-NULL
 pr$NNpr15<-as.data.frame(pred15)
 pr$NNpr05<-as.data.frame(pred)
 # pr$NNpr15G<-as.data.frame(pred15G)
 # pr$NNpr05G<-as.data.frame(predG)
 h2o::h2o.saveModel(m3,"Models//",force = TRUE)
 #   h2o::h2o.saveModel(h2o.getModel(sorted_grid@model_ids[[1]]),"Models//",force   =
TRUE)
 print("deeplearning results finished")
 ################RandomForest
 print("random forest started")
 RF <- h2o.randomForest(
   model_id="RFModel",
   x = predictors,
   y = response,
   ntrees = 100,
   max_depth = 500,
```

```
  min_rows = 10,
  calibrate_model = FALSE,
  calibration_frame = valid,
  binomial_double_trees = TRUE,
  training_frame = train,
  validation_frame = valid)
 # hyper_params <- list( balance_classes = c(TRUE, FALSE) )
 # grid <- h2o.grid(x = predictors, y = response, training_frame = train, validation_frame
= valid,
 #                  algorithm = "randomForest", grid_id = "RFGrid", hyper_params =
hyper_params,
 #                search_criteria = list(strategy = "Cartesian"), seed = 1234)
 #
 # # Sort the grid models by logloss
 # sorted_grid <- h2o.getGrid("RFGrid", sort_by = "logloss", decreasing = FALSE)
 # sorted_grid

 print("random forest finished")
 h2o.performance(RF, train=T)         ## sampled training data (from model building)
```

```
h2o.performance(RF, valid=T)          ## sampled validation data (from model building)
h2o.performance(RF, newdata=train)    ## full training data
h2o.performance(RF, newdata=valid)    ## full validation data
pred <- h2o.predict(RF, Allh2oData)
pred15<-h2o.predict(RF, Data15H20)
# predG <- h2o.predict(h2o.getModel(sorted_grid@model_ids[[1]]), Allh2oData)
# pred15G<-h2o.predict(h2o.getModel(sorted_grid@model_ids[[1]]), Data15H20)
pr$RFpr15<-as.data.frame(pred15)
pr$RFpr05<-as.data.frame(pred)
# pr$RFpr15G<-as.data.frame(pred15G)
# pr$RFpr05G<-as.data.frame(predG)
h2o::h2o.saveModel(RF,"Models//",force = TRUE)
#   h2o::h2o.saveModel(h2o.getModel(sorted_grid@model_ids[[1]]),"Models//",force   =
TRUE)


print("random forest results finished")
####################################
##########################GLM#############
print("GLM started")
```

```
GLM <- h2o.glm(
  model_id="GLMModel",
  family = "multinomial",
  x = predictors,
  y = response,
  training_frame = train,
  lambda = 0)



h2o.coef_norm(GLM)
h2o.performance(GLM, train=T)          ## sampled training data (from model building)
h2o.performance(GLM, valid=T)          ## sampled validation data (from model building)
h2o.performance(GLM, newdata=train)    ## full training data
h2o.performance(GLM, newdata=valid)    ## full validation data
pred <- h2o.predict(GLM, Allh2oData)
pred15<-h2o.predict(GLM, Data15H20)
```

```
pr$GLMpr15<-as.data.frame(pred15)

pr$GLMpr05<-as.data.frame(pred)

h2o::h2o.saveModel(GLM,"Models//",force = TRUE)



print("GLM finished")

#

#######################################GBM#######

print("GBM started")

GBM <- h2o.gbm(

  model_id="GBMModel",

  x = predictors,

  y = response,

  nfolds = 5,

  seed = 1111,

  keep_cross_validation_predictions = TRUE,

  training_frame = train)



# hyper_params <- list( balance_classes = c(TRUE, FALSE) )
```

```
 # grid <- h2o.grid(x = predictors, y = response, training_frame = train, validation_frame
= valid,
 #                 algorithm = "gbm", grid_id = "GBMGrid", hyper_params = hyper_params,
 #                 search_criteria = list(strategy = "Cartesian"), seed = 1234)
 #
 # # Sort the grid models by logloss
 # sorted_grid <- h2o.getGrid("GBMGrid", sort_by = "logloss", decreasing = FALSE)
 # sorted_grid


 h2o.performance(GBM, train=T)         ## sampled training data (from model building)
 h2o.performance(GBM, valid=T)         ## sampled validation data (from model building)
 h2o.performance(GBM, newdata=train)    ## full training data
 h2o.performance(GBM, newdata=valid)    ## full validation data
 pred <- h2o.predict(GBM, Allh2oData)
 pred15<-h2o.predict(GBM, Data15H20)
 # predG <- h2o.predict(h2o.getModel(sorted_grid@model_ids[[1]]), Allh2oData)
 # pred15G<-h2o.predict(h2o.getModel(sorted_grid@model_ids[[1]]), Data15H20)
 pr$GBMpr15<-as.data.frame(pred15)
 pr$GBMpr05<-as.data.frame(pred)
```

```
 # pr$GBMpr15G<-as.data.frame(pred15G)

 # pr$GBMpr05G<-as.data.frame(predG)

 h2o::h2o.saveModel(GBM,"Models//",force = TRUE)

 #   h2o::h2o.saveModel(h2o.getModel(sorted_grid@model_ids[[1]]),"Models//",force   =
TRUE)


 print("GBM finished")

 # ##########################KM#############################

 # print("k-means started")

 # KM <- h2o.kmeans(

 #   model_id="KMModel",

 #   x = predictors,

 #   training_frame = train,

 #   k = 10,

 #   estimate_k = TRUE,

 #   standardize = FALSE,

 #   seed = 1234)

 #

 # h2o.performance(KM, train=T)         ## sampled training data (from model building)
```

```
  # h2o.performance(KM, valid=T)         ## sampled validation data (from model building)
  # h2o.performance(KM, newdata=train)    ## full training data
  # h2o.performance(KM, newdata=valid)    ## full validation data
  # pred <- h2o.predict(KM, Allh2oData)
  # pred15<-h2o.predict(KM, Data15H20)
  # FinalSS<-h2o.predict(KM,FinalS)
  # pr$NBpr15<-as.data.frame(pred15)
  # pr$NBpr05<-as.data.frame(pred)
  # pr$NBFinal<-as.data.frame(FinalSS)
  # h2o::h2o.saveModel(NB,"D://Porcessing//Canadian//newResults//",force = TRUE)
  #
  #
  ##############################################################
  return(pr)
}
getSummary<-function(predictions,Truth,name){
  Truth$Class<-as.factor(Truth$Class)
  predictions<-factor(predictions,levels = levels(Truth$Class))
```

```
levels(Truth$Class)

levels(predictions)

summary(predictions)

cm<-as.matrix(table(Truth$Class, predictions))

write.csv(cm,paste("Results//",name,"-ConfusionMatrix.csv"))

src<-levels(predictions)

ppr<-levels(Truth$Class)

diff

print(cm)

n = sum(cm)

nc = nrow(cm)

diag = diag(cm)

print(sum(diag==0))

rowsums = apply(cm, 1, sum)

colsums = apply(cm, 2, sum)

p = rowsums / n

q = colsums / n

accuracy = sum(diag) / n
```

```
accuracy

precision = diag / colsums

recall = diag / rowsums

f1 = 2 * precision * recall / (precision + recall)

write.csv(data.frame(precision, recall, f1),paste("Results//",name,"-Accurecy.csv"))




Truth$Class<-as.factor(Truth$Class)

MCM<-confusionMatrix(predictions,Truth$Class)

sink(paste("Results//",name,"-ConfusionMatrix2OverAll.csv"))

print(MCM$overall)

sink()

write.csv(MCM$byClass,paste("Results//",name,"-ConfusionMatrix2ByClass.csv"))

write.csv(MCM$table,paste("Results//",name,"-ConfusionMatrix2table.csv"))

###Done
```

```
}

getAllSummery <- function (){

  ##Training

  getSummary(RFPr$Validate$predictions,ValidateSampel,"RangerLatest")

  getSummary(h2opredictions$NNpr15$predict,ValidateSampel,"NN")

  #getSummary(h2opredictions$NNpr15G$predict,ValidateSampel,"NNGrid")

  getSummary(h2opredictions$RFpr15$predict,ValidateSampel,"DRF")

  #getSummary(h2opredictions$RFpr15G$predict,ValidateSampel,"DRFGrid")

  getSummary(h2opredictions$GLMpr15$predict,ValidateSampel,"GLM")

  getSummary(h2opredictions$GBMpr15$predict,ValidateSampel,"GBM")

  #getSummary(h2opredictions$GBMpr15G$predict,ValidateSampel,"GBMGrid")




}
```

```
############Test raw###############


smp_size <- floor(0.75 * nrow(kddcup_data_corrected))


## set the seed to make your partition reproducible

set.seed(123)

train_ind <- sample(seq_len(nrow(kddcup_data_corrected)), size = smp_size)


TrainSample <- kddcup_data_corrected[train_ind, ]

ValidateSampel <- kddcup_data_corrected[-train_ind, ]

########################################


kddcup_data_corrected<-loadCleanData()

kddcup_data_corrected<-kddcup_data_corrected[Reduce(`&`,

lapply(kddcup_data_corrected, function(x) !is.na(x)  & is.finite(x))),]


#TrainSample<-readRDS("D:\\CIC PHD\\DataSource\\TrainSample.rds")

#ValidateSampel<-readRDS("D:\\CIC PHD\\DataSource\\ValidateSample.rds")
```

```
#FinalStageSample<-readRDS("D:\\CIC PHD\\DataSource\\FinalStageSample.rds")



library(caret)

library(data.table)

#k#ddcup_data_corrected2<-
downSample(kddcup_data_corrected,kddcup_data_corrected$Class)

kddcup_data_corrected$Class<-as.factor(kddcup_data_corrected$Class)

summary(kddcup_data_corrected$Class)

TrainSample<-getSample(0.7)

ValidateSampel<-setdiff(kddcup_data_corrected, TrainSample)

summary(TrainSample$Class)

summary(ValidateSampel$Class)

saveRDS(TrainSample,"\\Data\\TrainSample.rds")

saveRDS(ValidateSampel,"\\Data\\ValidateSample.rds")

kddcup_data_corrected<-NULL

kddcup_data_corrected2<-NULL

DT<-NULL

predictSample$Class<-as.factor(predictSample$Class)
```

```
library(psych)

#corPlot(subset(TrainSample,,-c(Class)))

##########################

#cor(subset(TrainSample,,-c(Class)), use = "complete.obs")

prediction<-as.data.frame(RFPr$predict$predictions)

RFPr<-RFF()

h2opredictions<-MultipleClassificaitionH2o()

saveRDS(h2opredictions,"Data\\h2opredictions.rds")

saveRDS(RFPr,"Data\\RFPr.rds")

getAllSummery()
```

```
newTester<-TrainSample

newTester$Class<-as.character(newTester$Class)

newTester$RFPred<-RFPr$predict$predictions

#levels(h2opredictions$NNpr15$predict)<-levels(kddcup_data_corrected$Class)

newTester$NNet<-(h2opredictions$NNpr05$predict)

newTester$RF<-(h2opredictions$RFpr05$predict)

newTester$GML<-(h2opredictions$GLMpr05$predict)

newTester$GBM<-(h2opredictions$GBMpr05$predict)

#newTester$NB<-(h2opredictions$NBpr05$predict)


newTester$BooleanRFPred<-if_else(newTester$RFPred==newTester$Class,1,0)

newTester$BooleanNNet<-if_else(newTester$NNet==newTester$Class,1,0)

newTester$BooleanRF<-if_else(newTester$RF==newTester$Class,1,0)

newTester$BooleanGML<-if_else(newTester$GML==newTester$Class,1,0)

newTester$BooleanGBM<-if_else(newTester$GBM==newTester$Class,1,0)

#newTester$BooleanNB<-if_else(newTester$NB==newTester$Class,1,0)
```

```
newTester<-newTester %>%
  mutate(Classifers = paste(as.character(BooleanRFPred),
                            as.character(BooleanNNet),
                            as.character(BooleanRF),
                            as.character(BooleanGML),
                            as.character(BooleanGBM),
                            #as.character(BooleanNB),

                            sep = ""))
newTester<-newTester %>%
  mutate(numericalClass=strtoi(Classifers, base = 2))


newTester$BooleanRFPred<-NULL
newTester$Classifers<-NULL
newTester$BooleanRF<-NULL
newTester$BooleanNNet<-NULL
newTester$BooleanGML<-NULL
```

```
newTester$BooleanGBM<-NULL

# newTester$BooleanNB<-NULL



newTester$RFPred<-NULL

newTester$RF<-NULL

newTester$NNet<-NULL

newTester$GML<-NULL

newTester$GBM<-NULL

newTester$NB<-NULL

newTester$Class<-NULL

newTester$numericalClass<-as.factor(newTester$numericalClass)

NUMCLASS <- ranger(numericalClass ~ ., data = newTester, importance = "impurity")

saveRDS(NUMCLASS, "D://CIC PHD//Model//FinalModel.rds")

NumericalPred<-predict(NUMCLASS,subset(FinalStageSample,,-c(Class)))

NUMCLASS$prediction.error

NumericalPred$predictions<-as.factor(NumericalPred$predictions)

#kddcup_data_corrected$numPred<-NumericalPred$predictions
```

```
summary(NumericalPred$predictions)

summary(newTester$numericalClass)

which(NumericalPred$predictions == 0)[[1]]

match(0 , NumericalPred$predictions)

length(which(NumericalPred$predictions == 0))

final[78700][1]

PPPP<-NULL

PPPP$myPred<-NumericalPred$predictions


library(binaryLogic)

PPPP$KKK<-PPPP$myPred

PPPP$KKK<-as.integer(as.character(PPPP$KKK))

PPPP$KKK<-rbind(as.binary(PPPP$KKK,n=5))

myBinaryDataframe<-do.call(rbind.data.frame, PPPP$KKK)

FinalResultWithAll<-NULL

colnames(myBinaryDataframe) <- c("BooleanRFPre",

                                 "BooleanNNet",

                                 "BooleanRF",
```

```
                    "BooleanGML",
                    "BooleanGBM"
)
# "BooleanGML",
# "BooleanGBM",
# "BooleanNB")

FinalResultWithAll$BooleanRFPre<-(myBinaryDataframe$BooleanRFPre)
FinalResultWithAll$BooleanNNet<-(myBinaryDataframe$BooleanNNet)
FinalResultWithAll$BooleanRF<-myBinaryDataframe$BooleanRF
FinalResultWithAll$BooleanGML<-myBinaryDataframe$BooleanGML
FinalResultWithAll$BooleanGBM<-myBinaryDataframe$BooleanGBM
# FinalResultWithAll$BooleanNB<-myBinaryDataframe$BooleanNB
FinalResultWithAll$RFPred<-(RFPr$FinalPR$predictions)
FinalResultWithAll$NNet<-(h2opredictions$NNFinal$predict)
FinalResultWithAll$RF<-(h2opredictions$RFFinal$predict)
FinalResultWithAll$GML<-(h2opredictions$GLMFinal$predict)
FinalResultWithAll$GBM<-(h2opredictions$GBMFinal$predict)
```

```
#FinalResultWithAll$NB<-(h2opredictions$NBFinal$predict)

#FinalResultWithAll$NNet<-unlist(FinalResultWithAll$NNet)

#library(dplyr)

asDataFrameResult<-bind_cols(FinalResultWithAll)

asDataFrameResult$RFPred[asDataFrameResult$BooleanRFPre==FALSE]<-NA

asDataFrameResult$NNet[asDataFrameResult$BooleanNNet==FALSE]<-NA

asDataFrameResult$RF[asDataFrameResult$BooleanRF==FALSE]<-NA

asDataFrameResult$GML[asDataFrameResult$BooleanGML==FALSE]<-NA

asDataFrameResult$GBM[asDataFrameResult$BooleanGBM==FALSE]<-NA

#asDataFrameResult$NB[asDataFrameResult$BooleanNB==FALSE]<-NA

filter(asDataFrameResult,BooleanNNet==FALSE &

       BooleanRF==FALSE &

       BooleanGML==FALSE &

       BooleanGBM==FALSE &

       BooleanRFPre==FALSE )

cleanResults<-asDataFrameResult

cleanResults$BooleanRFPre<-NULL

cleanResults$BooleanNNet<-NULL
```

```
cleanResults$BooleanRF<-NULL

cleanResults$BooleanGML<-NULL

cleanResults$BooleanGBM<-NULL

####Add later "RFPred",

#final<-apply(asDataFrameResult[,c("RFPred","NNet","RF")], 1, function(x)
names(table(x))[which.max(table(x))])

final<-vector("list", nrow(asDataFrameResult))

ResTable<-setDT(cleanResults)

final<-apply(ResTable,1,function(x) names(which.max(table(x))))

final<-as.data.frame(final)


cleanResults$NNet<-NULL




for (i in 1:nrow(asDataFrameResult)) {

  if (

    (asDataFrameResult$BooleanRFPre[i]==TRUE )&

    (asDataFrameResult$BooleanNNet[i] ==TRUE)&
```

```
  (asDataFrameResult$BooleanRF[i]==TRUE)
){
  tt <- (asDataFrameResult[i,c("RFPred","NNet","RF")])

  if(tt$RFPred==tt$RF & tt$RFPred==tt$NNet){
    final[[i]]<-toString(tt$RFPred)
    print("all equal")
  }
  else{
    final[[i]]<-toString(apply(tt, 1, function(x) names(table(x))[which.max(table(x))]))
  }

}

else if(asDataFrameResult$BooleanRFPre[i]==FALSE &
        asDataFrameResult$BooleanNNet[i]==FALSE &
        asDataFrameResult$BooleanRF[i]==FALSE){
  final[[i]]<-toString(asDataFrameResult[i,c("RFPred")][[1]])
```

```
  print("all false")
 }
 else if(asDataFrameResult$BooleanRFPre[i]==TRUE &
        asDataFrameResult$BooleanNNet[i]==TRUE &
        asDataFrameResult$BooleanRF[i]==FALSE){
  final[[i]]<-toString(asDataFrameResult[i,c("RFPred")][[1]])
  print("TTF")
 }
 else if(asDataFrameResult$BooleanRFPre[i]==TRUE &
        asDataFrameResult$BooleanNNet[i]==FALSE &
        asDataFrameResult$BooleanRF[i]==TRUE){
  final[[i]]<-toString(asDataFrameResult[i,c("RFPred")][[1]])
 }
 else{
  final[[i]] <- toString(apply(asDataFrameResult[i,c("RFPred","NNet","RF")], 1,
function(x) names(table(x))[which.max(table(x))]))
  print("entered")
 }
```

```
}

#finalResAsDataframe<-bind_cols(final)

#finalResAsDataframe<-transpose(finalResAsDataframe)

getSummary(final,FinalStageSample,"Tommorow_without_NNA")

getSummary(RFPr$FinalPR$predictions,FinalStageSample,"FinalRandomForesrt")

getSummary(h2opredictions$NNFinal$predict,FinalStageSample,"FinalNN")

getSummary(h2opredictions$RFFinal$predict,FinalStageSample,"FinalRFH2o")

getSummary(h2opredictions$GLMFinal$predict,FinalStageSample,"FinalGLM")

getSummary(h2opredictions$GBMFinal$predict,FinalStageSample,"FinalGBM")

# getSummary(h2opredictions$NBFinal$predict,FinalStageSample,"FinalNB")

#

levels(nfinal$results)<-levels(kddcup_data_corrected$Class)

#
```

Master Model

```
#Libaraies

kddcup_data_corrected<-NULL

prediction<-NULL
```

```
Sample<-NULL

#library(readr)

library(dbplyr)

library(tidyverse)

library(ranger)

library(caret)

#library(keras)

library(dplyr)

library(magrittr)

library(nnet)

library(caret)

library(Rcpp)



getSummary<-function(predictions,Truth,name){

  Truth$Class<-as.factor(Truth$Class)

  predictions<-factor(predictions,levels = levels(Truth$Class))

  levels(Truth$Class)

  levels(predictions)
```

```
summary(predictions)

cm<-as.matrix(table(Truth$Class, predictions))

write.csv(cm,paste("Results//",name,"-ConfusionMatrix.csv"))

src<-levels(predictions)

ppr<-levels(Truth$Class)

diff

print(cm)

n = sum(cm)

nc = nrow(cm)

diag = diag(cm)

print(sum(diag==0))

rowsums = apply(cm, 1, sum)

colsums = apply(cm, 2, sum)

p = rowsums / n

q = colsums / n

accuracy = sum(diag) / n



accuracy
```

```
  precision = diag / colsums

  recall = diag / rowsums

  f1 = 2 * precision * recall / (precision + recall)

  write.csv(data.frame(precision, recall, f1),paste("Results//",name,"-Accurecy.csv"))




  Truth$Class<-as.factor(Truth$Class)

  MCM<-confusionMatrix(predictions,Truth$Class)

  sink(paste("Results//",name,"-ConfusionMatrix2OverAll.csv"))

  print(MCM$overall)

  sink()

  write.csv(MCM$byClass,paste("Results//",name,"-ConfusionMatrix2ByClass.csv"))

  write.csv(MCM$table,paste("Results//",name,"-ConfusionMatrix2table.csv"))

  ###Done



}
```

```
LoadModels<-function(){ #fix later


  library(h2o)
  h2o.init(max_mem_size = "32g")
  print("deepModel Started")
  Deepmodel<-h2o::h2o.loadModel("Models/")
  h2o.save_mojo(Deepmodel, path = "Models/",force = TRUE)
  h2o.performance(Deepmodel,Deepmodel@p)
  print("load data")
  TrainSample$Class<-as.factor(TrainSample$Class)
  ValidateSampel$Class<-as.factor(ValidateSampel$Class)
  Allh2oData<-as.h2o(subset(TrainSample,,-c(Class)))
  Data15H20<-as.h2o(ValidateSampel)
  FinalS<-as.h2o(subset(FinalStageSample,,c(-Class)))
  print("finished load data")
  response <- "Class"
  predictors <- setdiff(names(train), response)
  pred <- h2o.predict(Deepmodel, Allh2oData)
```

```
pred15<-h2o.predict(Deepmodel, Data15H20)

FinalSS<-h2o.predict(Deepmodel,FinalS)

pr<-NULL

pr$NNpr15<-as.data.frame(pred15)

pr$NNpr05<-as.data.frame(pred)

pr$NNFinal<-as.data.frame(FinalSS)

print("deep Model finsished")


####################DRF#############

print("rdf started")

RF <-h2o::h2o.loadModel("D:\\CIC PHD\\Model\\RFGrid_model_1")

RF2 <-h2o::h2o.loadModel("D:\\CIC PHD\\Model\\RFGrid_model_2")

pred <- h2o.predict(RF, Allh2oData)

pred15<-h2o.predict(RF, Data15H20)

FinalSS<-h2o.predict(RF,FinalS)

pr$RFpr15<-as.data.frame(pred15)

pr$RFpr05<-as.data.frame(pred)

pr$RFFinal<-as.data.frame(FinalSS)
```

```
print("rdf finished")

#########################GLM <- ###############

GLM <- h2o::h2o.loadModel("D:\\Porcessing\\Canadian\\Models\\GLMModel")

print("GLM started")

pred <- h2o.predict(GLM, Allh2oData)

pred15<-h2o.predict(GLM, Data15H20)

FinalSS<-h2o.predict(GLM,FinalS)

pr$GLMpr15<-as.data.frame(pred15)

pr$GLMpr05<-as.data.frame(pred)

pr$GLMFinal<-as.data.frame(FinalSS)

print("GLM finished")

#######################GBM <- ###############

GBM <- h2o::h2o.loadModel("D:\\CIC PHD\\Model\\GBMModel")



print("GBM started")



pred <- h2o.predict(GBM, Allh2oData)

pred15<-h2o.predict(GBM, Data15H20)
```

```
  FinalSS<-h2o.predict(GBM,FinalS)
  pr$GBMpr15<-as.data.frame(pred15)
  pr$GBMpr05<-as.data.frame(pred)
  pr$GBMFinal<-as.data.frame(FinalSS)
  print("GBM finished")
  print("ranger started")
  rg.iris<-readRDS("D:\\CIC PHD\\Model\\RFModelToLoad.rds")
  pr$Validate<-predict(rg.iris,subset(ValidateSampel,,-c(Class)))
  pr$predict<-predict(rg.iris,subset(TrainSample,,-c(Class)))
  pr$FinalPR<-predict(rg.iris,subset(FinalStageSample,,-c(Class)))
  print("ranger Finished")
  return(pr)
}
TrainSample<-read_rds("D://Porcessing//Canadian//splitRDS//TrainSample.rds")
ValidateSampel<-read_rds("D://Porcessing//Canadian//splitRDS//ValidateSampel.rds")
FinalStageSample<-
read_rds("D://Porcessing//Canadian//splitRDS//FinalStageSample.rds")
h2opredictions<-LoadModels()
RFpred<-readRDS("Data\RFPr.rds")
```

```
h2opredictions<-readRDS("Data\\h2opredictions.rds")


#############add prediction lables to the training dataset

newTester<-TrainSample

newTester$Class<-as.character(newTester$Class)

newTester$RFPred<-RFPr$predict$predictions

newTester$NNet<-(h2opredictions$NNpr05$predict)

newTester$RF<-(h2opredictions$RFpr05$predict)

newTester$GML<-(h2opredictions$GLMpr05$predict)

newTester$GBM<-(h2opredictions$GBMpr05$predict)


#############compare and make the lables binary when campred to the true  Value


newTester$BooleanRFPred<-if_else(newTester$RFPred==newTester$Class,1,0)

newTester$BooleanNNet<-if_else(newTester$NNet==newTester$Class,1,0)

newTester$BooleanRF<-if_else(newTester$RF==newTester$Class,1,0)

newTester$BooleanGML<-if_else(newTester$GML==newTester$Class,1,0)

newTester$BooleanGBM<-if_else(newTester$GBM==newTester$Class,1,0)
```

```
############ merge the binary for each model ############


newTester<-newTester %>%
  mutate(Classifers = paste(as.character(BooleanRFPred),
                            as.character(BooleanNNet),
                            as.character(BooleanRF),
                            as.character(BooleanGML),
                            as.character(BooleanGBM),
                            sep = ""))
######################Remove unneccesry Values from the
DF############################
newTester$BooleanRF<-NULL
newTester$BooleanNNet<-NULL
newTester$BooleanGML<-NULL
newTester$BooleanGBM<-NULL
newTester$RFPred<-NULL
newTester$RF<-NULL
newTester$NNet<-NULL
```

```
newTester$GML<-NULL

newTester$GBM<-NULL

newTester$NB<-NULL

newTester$Class<-NULL

newTester$BooleanRFPred<-NULL

##############################

newTester$Classifers<-as.factor(newTester$Classifers)

NUMCLASS <- ranger(Classifers ~ ., data = newTester, importance = "impurity")

saveRDS(NUMCLASS, "Models\\MasterModel.rds")

NumericalPred<-predict(NUMCLASS,subset(ValidateSampel,,-c(Class)))

saveRDS(NumericalPred,"Data\\ValidatePredFinal.rds")

NumericalPred<-read_rds("D://Porcessing//Canadian//processed - Model
Data//FinalPred.rds")

NUMCLASS$prediction.error

NumericalPred$predictions<-as.factor(NumericalPred$predictions)

summary(NumericalPred$predictions)

NumericalPred$predictions[NumericalPred$predictions=="0000"]<-"1000"

###################3

DataPredF<-NULL
```

```
DataPredF$predictions<-NumericalPred$predictions

DataPredF<-as.data.frame(DataPredF)

DataPredF$predictions<-lapply(DataPredF$predictions, as.character)

myBinaryDataframe<- NULL

emptyColoumsn <- c("BooleanRFPre",

                   "BooleanNNet",

                   "BooleanRF",

                   "BooleanGML",

                   "BooleanGBM"

)

myBinaryDataframe$BooleanRFPre<-as.integer(substr(DataPredF$predictions,1,1))

myBinaryDataframe$BooleanNNet<-as.integer(substr(DataPredF$predictions,2,2))

myBinaryDataframe$BooleanRF<-as.integer(substr(DataPredF$predictions,3,3))

myBinaryDataframe$BooleanGML<-as.integer(substr(DataPredF$predictions,4,4))

myBinaryDataframe$BooleanGBM<-as.integer(substr(DataPredF$predictions,5,5))



myBinaryDataframe$RFPred<-(RFPr$Validate$predictions)

myBinaryDataframe$NNet<-(h2opredictions$NNpr15$predict)
```

```
myBinaryDataframe$RF<-(h2opredictions$RFpr15$predict)

myBinaryDataframe$GML<-(h2opredictions$GLMpr15$predict)

myBinaryDataframe$GBM<-(h2opredictions$GBMpr15$predict)



myBinaryDataframe$RFPred[myBinaryDataframe$BooleanRFPre==0]<-NA

myBinaryDataframe$NNet[myBinaryDataframe$BooleanNNet==0]<-NA

myBinaryDataframe$RF[myBinaryDataframe$BooleanRF==0]<-NA

myBinaryDataframe$GML[myBinaryDataframe$BooleanGML==0]<-NA

myBinaryDataframe$GBM[myBinaryDataframe$BooleanGBM==0]<-NA



myBinaryDataframe$BooleanRFPre<-NULL

myBinaryDataframe$BooleanNNet<-NULL

myBinaryDataframe$BooleanRF<-NULL

myBinaryDataframe$BooleanGML<-NULL

myBinaryDataframe$BooleanGBM<-NULL

library(data.table)

ResTable<-setDT(myBinaryDataframe)

final<-apply(myBinaryDataframe,1,function(x) names(which.max(table(x))))
```

```
getSummary(final,ValidateSampel,"MasterResults")
```