

Bachelor's Thesis

**ML-Modeler: A Web Application for Automated
Analysis on Cyber Attacks**

Andreas Demosthenous

University of Cyprus



Department of Computer Science

May 2023

UNIVERSITY OF CYPRUS
DEPARTMENT OF COMPUTER SCIENCE

**ML-Modeler: A Web Application for Automated
Analysis on Cyber Attacks**

Andreas Demosthenous

Supervisor
George Pallis

The Diploma Thesis was submitted for partial fulfilment of the requirements for
obtaining the degree of Computer Science of the Department of Computer Science of
the University of Cyprus

May 2023

ACKNOWLEDGMENTS

I would like to thank and express my appreciation to my supervisor, Professor George Pallis, who gave me the opportunity to work on a topic that is very important and falls within my career goals. He supported me in everything I needed, and he was always available for questions.

My special thanks also go to Ms. Ioanna Georgiou, who is part of LINC in the department of computer science at University of Cyprus. She supported and helped me with a lot of problems I encountered. Her instructions and suggestions were important in the completion of this research. Finally, I would like to thank my family and friends for being supportive during my studies.

Abstract

The increasing frequency and complexity of cyber-attacks have made it challenging for organizations to analyze large volumes of data to identify threats and respond to them quickly. This thesis proposes the implementation of a platform for automating the data analysis of cyber-attacks. The platform allows the data analyst perform several operations on their datasets including feature selection as well as model training and tuning. The proposed platform is designed to be scalable, flexible, and easy to use, providing a comprehensive solution for organizations of all sizes. During this research, the platform was an effective tool in reducing the time and resources required for data analysis, while aiding in keeping the results in an organized fashion.

Additionally, this research includes an analysis of several cyber-attack datasets to identify the most important network related features and correlations between them as well as the most efficient machine learning algorithms to detect such attacks. The results of this analysis demonstrate that certain features, such as the protocol, payload size, flow duration are critical in identifying and classifying attacks. Furthermore, tree-based machine learning algorithms such as Random Forest and Extreme Gradient Boost are effective in accurately detecting and classifying various types of cyber-attacks.

The proposed platform, combined with the results of the analysis, represents a significant contribution to the field of cyber security and data science, and it has the potential to revolutionize the way organizations approach data analysis and threat detection.

Table of Context

Chapter 1 Introduction	1
1.1 Motivation	1
1.2 System Description	2
1.3 Contribution	2
1.4 Chapter Outline	3
 Chapter 2 Background	 4
2.1 Cyber Attacks Landscape.....	4
2.1.1 DDoS Attacks.....	4
2.1.2 Brute Force Attacks.....	13
2.2 Existing Mitigation Approaches.....	14
2.3 Manual Data Analysis.....	15
2.3.1 Python.....	16
2.3.2 Feature Selection.....	19
2.3.3 Machine Learning.....	22
2.4 Benefits of Automation.....	24
 Chapter 3 Related Work	 26
3.1 Similar Platforms.....	26
3.1.1 DataRobot.....	26
3.1.2 RapidMiner.....	27
 Chapter 4 System Architecture	 30
4.1 System Requirements.....	30
4.2 System Design.....	31
4.2.1 Architecture.....	32
4.2.2 Data Storage.....	33
4.2.3 Components.....	37
4.3 Deployment.....	42
4.3.1 Web Server.....	42
4.3.2 Database.....	43
4.3.3 PHP.....	43
4.3.4 Python.....	44

4.4 System Workflow.....	44
4.4.1 Dataset Upload.....	45
4.4.2 Feature Selection.....	48
4.4.3 Model Training and Tuning.....	55
Chapter 5 System Evaluation	60
5.1.1 Datasets.....	60
5.1.1.1 CIC-DDoS2019.....	60
5.1.1.2 CIC-IDS2017	61
5.1.2 Methodology.....	62
5.1.3 Experiments & Results.....	65
Chapter 6 Conclusion & Future work	86
6.1 Insights & Conclusions.....	86
6.2 Future work.....	88
Bibliography.....	89
Appendix A	92
Appendix B	96

Chapter 1

Introduction

1.1 Motivation.....	1
1.2 System Description.....	2
1.3 Contribution.....	2
1.4 Chapter Outline.....	3

1.1 Motivation

In recent years, the frequency and severity of cyber-attacks have increased dramatically [40], leading to significant financial and reputational damage for affected organizations. These attacks pose a substantial risk to organizations, resulting in considerable financial losses and reputational damage. Among the most prevalent types of attacks, Distributed Denial of Service (DDoS) attacks have gained prominence. These attacks involve overwhelming a target system with an enormous volume of traffic from multiple sources. Detecting and analyzing such attacks present substantial challenges that demand advanced analytics tools and techniques.

To address these challenges, I developed a comprehensive platform that automates critical tasks involved in the analysis of cyber-attack datasets. By doing so, the platform aims to empower data analysts, enhancing their efficiency and accuracy in identifying these attacks and their distinctive characteristics. Moreover, the platform provides capabilities for training and fine-tuning models, enabling data analysts to effectively respond and mitigate the impact of these attacks. By automating various aspects of the analysis process, the platform will significantly streamline the work of data analysts, allowing them to save time and resources. It will facilitate the swift and accurate identification of DDoS attacks, enabling timely countermeasures and reducing the potential damage inflicted on targeted systems.

The main problem my work addresses is the lack of tools available, that allow an easy and fast analysis of cyber-attack datasets. Most enterprise tools with similar functionality focus on deploying accurate machine learning models without providing much insight and information about the data processing. My platform is more research oriented, since it gives the analyst an insight into the datasets by allowing them to identify feature relationships, test several algorithms against several dataset configurations and experiment with different algorithms hyperparameters to better understand how they work.

1.2 System Description

The platform allows analysts to upload their network traffic datasets and perform several tasks to identify the characteristics of the attack. It includes a user-friendly interface that enables interaction with the system and access to the results of the analysis in real-time.

The core of the platform is a machine learning engine that includes a variety of algorithms for cyber-attack classification. With the platform, data analysts can perform several tasks, including data preprocessing, feature analysis, feature selection, model training, model evaluation and model tuning. Furthermore, the platform allows analysts to download the results of the analysis, such as the identified attack characteristics, the performance and efficiency of the trained models as well as the values for the best hyper-parameter sets of their models, to further process them.

1.3 Contribution

The main contribution of this work is the development of a platform that automates the process of analyzing cyber-attack datasets. It provides a comprehensive solution for data analysts to identify attack characteristics quickly and accurately, enabling them to take effective action to mitigate the attacks.

In my chosen use case on cyber-attacks which mainly include DDoS attacks but also brute force and port scans, I have demonstrated the effectiveness of the platform in analyzing real-world datasets efficiently and quickly reporting the findings of the analysis as well as being able to organize and visualize them in an easy to interpret way on the platform.

In addition, I have conducted a series of experiments on 2 publicly available datasets (CIC-DDoS2019 [3], CIC-IDS2017 [4]) using the platform to evaluate it and enrich the research community with my conclusions. Specifically, I compared the performance of 2 feature correlation algorithms (Spearman vs Pearson), 2 best-feature selection algorithms (Mutual Info. vs ANOVA) and 5 machine learning algorithms (DTC vs RFC vs XGBC vs KNN vs LDA) on the aforementioned datasets. My experiments and conclusions form an important contribution to the research community.

1.4 Chapter Outline

Firstly, Chapter 2 is focused on explaining several types of cyber-attacks analyzed in this use case, their characteristics from the network traffic perspective and the existing approaches used for distinguishing malicious traffic from the normal. In addition, it discusses the different approaches used by data analysts to process and analyze traffic and the potential benefits of automation.

Chapter 3 includes some other systems with similar functionality and their contribution in automation and discusses how my platform compares to their functionality.

Chapter 4 includes the complete architecture of the system, including the platform design, backend processing as well as the connectivity with the database. It will describe the overall structure, components, and functionalities of the platform.

In chapter 5, I am discussing the datasets used to evaluate the platform, as well as the experiments performed and my results. I provide an analysis of the performance and effectiveness of the platform in analyzing cyber-attacks while making the job of the data analyst easier and more efficient.

Closing, in chapter 6 I summarize the key findings and conclusions from my study. I supply an overall assessment of the platform, its strengths and limitations, and its potential applications. Additionally, I discuss future work that can be done to further improve and develop the platform as well as further analysis that can be performed on the analyzed data.

Chapter 2

Background

2.1 Cyber Attacks Landscape.....	4
2.1.1 DDoS Attacks.....	4
2.1.2 Brute Force Attacks.....	13
2.2 Existing Mitigation Approaches.....	14
2.3 Manual Data Analysis.....	15
2.3.1 Python.....	16
2.3.2 Feature Selection.....	19
2.3.3 Machine Learning.....	22
2.4 Benefits of Automation.....	24

2.1 Cyber Attacks Landscape

[2] Cyber-attacks are malicious activities carried out by individuals or groups with the intention of stealing or damaging data, disrupting services, or causing other types of harm. The landscape of cyber-attacks is constantly evolving, with new threats and vulnerabilities appearing all the time. To better understand the types of attacks that can occur, it is important to examine some of the most common ones. The following section supplies a thorough explanation of DDoS attacks to better understand their inner workings.

2.1.2 DDoS Attacks

[1] A DDoS (Distributed Denial of Service) attack is a type of cyber-attack that targets a website or online service by overwhelming it with traffic from multiple sources. In a DDoS attack, a large number of devices or computers are controlled by the attacker to send traffic to the target website, making it unavailable to legitimate users. The traffic sent can be in the form of data packets, requests for information, or even malicious payloads. DDoS attacks are often used by cybercriminals to disrupt or disable online services, extort money, or damage

the reputation of a business or organization. These attacks can cause significant financial losses, downtime, and reputational damage for the victim.

One of the largest DDoS attacks happened in the last decade affected GitHub in 2015 [41]. The largest DDoS attack ever at the time, this one also happened to target GitHub. This politically motivated attack lasted several days and adapted itself around implemented DDoS mitigation strategies. The DDoS traffic originated in China and specifically targeted the URLs of two GitHub projects aimed at circumventing Chinese state censorship. It is guessed the attack's intent was to pressure GitHub into eliminating those projects. The attack traffic was created by injecting JavaScript code into the browsers of everyone who visited Baidu, China's most popular search engine. Other sites who were using Baidu's analytics services were also injecting the malicious code. This code was causing the infected browsers to send HTTP requests to the targeted GitHub pages. In the attack's aftermath, it was determined that the malicious code was not originating from Baidu but was added by an intermediary service.

DDoS Variants

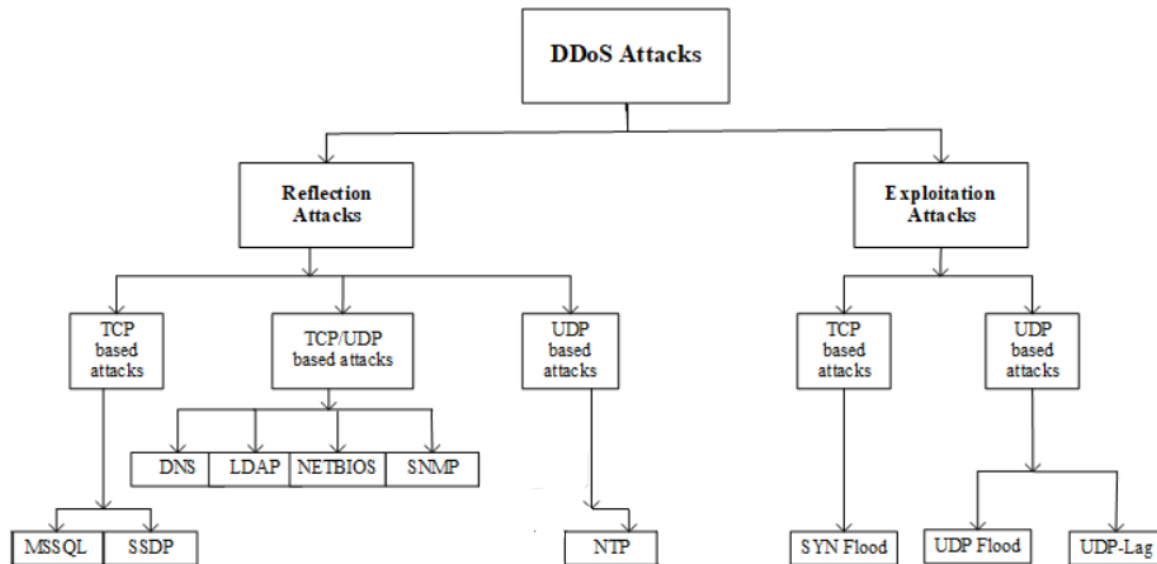
There are several variants of DDoS attacks, each with its own method of execution. The two main variants are reflective and exploitation DDoS attacks that abuse several services.

Reflective DDoS attacks [3] involve the attacker sending a request to a server with a spoofed IP address of the target network. The server responds to the spoofed IP address, flooding the target network with an overwhelming amount of traffic.

On the other hand, Exploitation DDoS attacks [3] exploit vulnerabilities in software or hardware to overload the target network. In this type of attack, the attacker sends a malicious packet to the target network, causing it to crash or become unavailable. Additionally, there are IoT-based DDoS attacks, where IoT devices such as cameras or routers are compromised and used to launch DDoS attacks. All these variants pose a significant threat to organizations and can cause significant damage if not properly mitigated.

There are several other variants, but for the purposes of this research the focus is more on the exploitation and reflection variants which are shown in figure 2.1 and explained in more detail in the following subsections.

Figure 2.1: DDoS Attack Taxonomy [3]



Reflective DDoS

Reflection DDoS attacks [3] are a type of DDoS attack that takes advantage of open servers or devices that respond to certain types of requests with large amounts of data. In a reflection attack, the attacker sends a request to an open server or device, spoofing the source IP address of the request to make it appear as if it is coming from the target. The server or device responds with a large amount of data, which is then directed at the target, overwhelming it with traffic. Some common types of services abused by reflection attacks and analyzed in this research include the following:

DNS DDoS: Exploits vulnerabilities in the Domain Name System (DNS) by sending requests to DNS servers with spoofed source IP addresses of the target network. The DNS server responds to the spoofed IP addresses, flooding the target network with traffic and overwhelming its resources.

- **LDAP DDoS:** Exploits vulnerabilities in the Lightweight Directory Access Protocol (LDAP) by sending requests to LDAP servers with spoofed source IP addresses of the target network. The LDAP server responds to the spoofed IP addresses, flooding the target network with traffic.
- **MSSQL DDoS:** Exploits vulnerabilities in the Microsoft SQL Server by sending requests to MSSQL servers with spoofed source IP addresses of the target network. The MSSQL server

responds to the spoofed IP addresses, flooding the target network with traffic and causing it to become unavailable.

- **NTP DDoS:** Exploits vulnerabilities in the Network Time Protocol (NTP) by sending requests to NTP servers with spoofed source IP addresses of the target network. The NTP server responds to the spoofed IP addresses, flooding the target network with traffic and overwhelming its resources.
- **NetBIOS DDoS:** Exploits vulnerabilities in the Network Basic Input/Output System by sending requests to NETBIOS servers with spoofed source IP addresses of the target network. The NetBIOS server responds to the spoofed IP addresses, flooding the target network with traffic and causing it to become unavailable.
- **SNMP DDoS:** Exploits vulnerabilities in the Simple Network Management Protocol (SNMP) by sending requests to SNMP servers with spoofed source IP addresses of the target network. The SNMP server responds to the spoofed IP addresses, flooding the target network with traffic and overwhelming its resources.
- **SSDP DDoS:** Exploits vulnerabilities in the Simple Service Discovery Protocol (SSDP) by sending requests to SSDP servers with spoofed source IP addresses of the target network. The SSDP server responds to the spoofed IP addresses, flooding the target network with traffic and causing it to become unavailable.

Exploitation DDoS

Exploitation DDoS attacks [3] are a type of cyber-attack that exploits vulnerabilities in software or hardware to overload the target network. There are several methods of executing an exploitation DDoS attack, including **TCP SYN flood**, **UDP flood**, and **UDP lag** which are explained below and later analyzed in this research.

- **TCP SYN flood** attacks exploit the three-way handshake process that occurs when two devices establish a connection over the internet. The attacker sends a large number of SYN packets with spoofed source IP addresses to the target network. The target network responds with SYN-ACK packets to the spoofed IP addresses, but since the source IP addresses are fake, the target network is left waiting for a response that never comes, tying up resources and making the network unavailable.

- **UDP flood** attacks involve sending a large number of UDP packets to the target network. Since UDP does not require a handshake or verification of receipt, the target network simply responds to each packet, regardless of its validity. This flood of incoming packets can overwhelm the network and cause it to become unavailable.
- **UDP lag** attacks exploit the fact that UDP packets do not require a handshake, allowing the attacker to send a large number of packets to the target network with spoofed source IP addresses. The target network responds to each packet with an ICMP Destination Unreachable message, which can cause the network to become unavailable due to the high volume of incoming traffic.

DDoS Generation Tools

DDoS generation tools include a variety of specialized software tools that allow attackers to launch large-scale attacks. These tools are designed to exploit vulnerabilities in web servers and web applications and can be used to launch a variety of DDoS attacks such as HTTP floods, SYN floods, and UDP floods. Attackers can use these tools to generate a massive amount of traffic, overloading the target server and causing it to become unresponsive. In some cases, attackers may use these tools to launch multiple simultaneous attacks, targeting distinct parts of the network infrastructure. **GoldenEye**, **Hulk**, and **SlowHTTPTest** are all types of Denial-of-Service attack tools used to target web servers and later analyzed thoroughly in this research and are going to be described in the sections below.

GoldenEye

GoldenEye [6] is a Denial-of-Service (DoS) tool that is used to launch attacks against web servers. Additionally, it can perform a Distributed Denial-of-Service (DDoS) attack by leveraging a network of compromised machines, known as a botnet. It generates a large amount of HTTP traffic by sending multiple requests to the target server, exploiting vulnerabilities in web servers that cannot handle large numbers of requests.

GoldenEye can perform several types of attacks, including HTTP GET and POST floods, slowloris attacks, and TCP connection attacks. HTTP GET and POST floods overwhelm the target server with a large number of requests, while slowloris attacks send a limited number

of requests but keep them open for as long as possible, taking up server resources and preventing legitimate requests from being processed. TCP connection attacks flood the server with TCP connection requests, creating a backlog that eventually crashes the server.

It additionally includes advanced features such as IP spoofing, user agent and referrer spoofing, and the ability to rotate attack vectors. These features make it harder for security tools to detect and mitigate the attack and can cause additional damage to the target server.

Hulk

Hulk [7] is another DDoS tool that is used to launch attacks against web servers. It is designed to generate a large amount of traffic by repeatedly sending GET and POST requests to the target server, exploiting weaknesses in the HTTP protocol and poorly written web applications.

Hulk can perform different types of attacks, including HTTP GET and POST floods, slowloris attacks, and TCP connection attacks.

Hulk also includes advanced features such as the ability to randomize the User-Agent and Referrer headers in order to evade detection, and the ability to use multiple concurrent connections to the target server. These features make it harder for security tools to detect and mitigate the attack and can cause additional damage to the target server.

SlowHTTPTest

Like the previously mentioned tools, SlowHTTPTest [8] can be used to launch distinct types of attacks, including Slow Read, Slow Loris, and Slow Post attacks against web servers. Slow Read attacks exploit the HTTP protocol by sending incomplete requests, causing the server to wait for more data that never arrives. Slow Post attacks send HTTP POST requests to the server but do not complete them, again taking up server resources and preventing legitimate requests from being processed.

Furthermore, SlowHTTPTest includes features that can be used to evade detection and mitigation, such as the ability to randomize the User-Agent and Referrer headers and to use multiple concurrent connections to the target server.

DDoS Traffic Experimental Analysis

DDoS attacks have certain characteristics that allow data analytics and machine learning to distinguish them from normal traffic. In the paper “Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy” [3], it was studied how several machine learning algorithms like Random Forest and Naïve Bayes perform on a DDoS classification dataset with 11 different DDoS attack types: UDP-lag, DNS, MSSQL, LDAP, NetBIOS, NTP, SSDP, SNMP, Syn and UDP. The data was gathered by the authors by simulating the attacks using an attack and victim network and monitored the traffic. They later used the CICFlowMeter [9] Tool to convert the captured traffic in a processable format.

DDoS Traffic Characteristics

According to aforementioned research analysis, the best features for each DDoS kind were extracted using RandomForestRegressor [10] class of scikit-learn and are shown in figure 2.2

Name	Feature	Weight	Mean
UDP-lag	ACK Flag Count	0.125438	0.86545908
	Init_Win_bytes_forward	0.002093	5061.324632
	min_seg_size_forward	0.000795	-3967113.958
	Fwd IAT Mean	0.000612	1109423.738
	Fwd IAT Max	0.000471	3310515.822
TFTP	Fwd IAT Mean	0.000207	541101.4798
	min_seg_size_forward	0.000198	-34648586.17
	Fwd IAT Max	0.000151	1562350.615
	Flow IAT Max	0.000129	1562493.187
	Flow IAT Mean	0.000124	540958.2437
WebDDoS	ACK Flag Count	0.043991	0.330296128
	Init_Win_bytes_forward	0.009357	21747.29613
	Fwd Packet Length Std	0.002881	62.69322463
	Packet Length Std	0.002068	108.2461488
	min_seg_size_forward	0.000872	32.00
DNS	Max Packet Length	1.139858	1378.802657
	Fwd Packet Length Max	0.127708	1378.773093
	Fwd Packet Length Min	0.007794	1378.522451
	Average Packet Size	0.005849	2067.363444
	Min Packet Length	0.003487	1378.521706
Benign	ACK Flag Count	0.020021	0.172801294
	Flow IAT Min	0.016769	11817.55752
	Init_Win_bytes_forward	0.003182	7560.598157
	Fwd Packet Length Std	0.001786	39.79290701
	Packet Length Std	0.001678	88.27355725
MSSQL	Fwd Packets/s	0.000204	1676967.356
	Protocol	4.60E-05	N/A
LDAP	Max Packet Length	1.278323	1463.73827
	Fwd Packet Length Max	0.143219	1463.728043
	Fwd Packet Length Min	0.008736	1463.717027
	Average Packet Size	0.006532	2194.906258
	Min Packet Length	0.003909	1463.716847
NetBIOS	Fwd Packets/s	0.000172	1580350.263
	min_seg_size_forward	7.20E-05	-41140208.12
	Protocol	4.60E-05	N/A
	Fwd Header Length	3.50E-05	-82335716.29
	Fwd Header Length.1	3.20E-05	-82335716.29
NTP	Subflow Fwd Bytes	0.106481	27903.20181
	Length of Fwd Packets	0.058022	27903.20181
	Fwd Packet Length Std	0.001081	25.03898396
	min_seg_size_forward	0.000707	-8471708.317
	Flow IAT Min	0.000573	438.699766
SSDP	Destination Port	0.000671	33266.62516
	Fwd Packet Length Std	0.000597	14.90294147
	Packet Length Std	0.000232	14.2504337
	Protocol	4.60E-05	N/A
	min_seg_size_forward	1.20E-05	-44212168.85
SNMP	Max Packet Length	1.152048	1386.280102
	Fwd Packet Length Max	0.129074	1386.258583
	Fwd Packet Length Min	0.007879	1386.226774
	Average Packet Size	0.005912	2079.140846
	Min Packet Length	0.003526	1386.226619
Syn	ACK Flag Count	0.145834	0.999478603
	Init_Win_bytes_forward	0.002432	5837.969261
	min_seg_size_forward	0.000872	20.00076092
	Fwd IAT Total	0.000571	8086250.716
	Flow Duration	0.000409	8086316.455
UDP	Destination Port	0.000699	33284.8418
	Fwd Packet Length Std	0.000615	15.28026139
	Packet Length Std	0.000239	14.61817097
	min_seg_size_forward	9.80E-05	-39820342.76
	Protocol	4.50E-05	N/A

Table 2.1: Best Features per attack type [3]

The authors concluded that ‘packet Length Std’ is one of the most influential features for benign traffic. One of the reasons is that there is more variation in the size of packets in benign traffic in comparison to different DDoS attacks, because DDoS attacks are conducted by automated tools and botnets and usually, they produce fixed-size or similar packets. Therefore, it’s an important feature to consider when training a machine learning model.

For the SYN DDoS, the analysts concluded that the two most influential features are 'ACK Flag Count' and 'Flow Duration', because this attack works by not responding to the server with the expected ACK code (it exploits a TCP protocol weakness).

Moreover, for the MSSQL DDoS, according to the analysis, two main influential features are the 'Protocol' and 'Fwd Packets/s' which makes sense, because the attacker abuses the Microsoft SQL Server Resolution Protocol (MC-SQLR) and sends millions of packets to the victim.

Furthermore, the authors have discovered that IAT (time interval between 2 packets) related features are considered as influential features in many attacks such as TFTP, UDP-lag and NTP. According to them, one of the reasons is that many DDoS attacks show bursty behavior(unstable) in sending packets to the victims, unlike benign traffic that usually does not show any bursty behavior. The bursty behavior affects the arrival rate, and so it affects IAT related features, and this can be a reason they are influential features for detecting DDoS attacks.

To make a successful DDoS attack the authors argue that an attacker needs to send more packet than the victim can handle. Also, attackers use diverse types of packets, such as SYN or ICMP packets to send many malicious packets all of which are similar in size and small because of the low cost of computing resources but are not like the packets in a benign flow. So, the minimum segment size of the packets in a malicious flow would be less than the packets in a benign flow.

DDoS Detection with Machine Learning

For training a machine learning model, the authors tested ID3, Random Forest, Naïve Bayes, and Multinomial Logistic Regression, algorithms that I will describe briefly in the following paragraphs.

[3] **Random Forest** is a machine learning algorithm that combines two ideas of decision tree and ensemble learning. The forest contains many decision trees that use randomly picked data attributes as their input. The forest has a collection of trees with controlled variance.

Finally, the result of a classification can be decided by majority voting or weighted voting. One of the advantages of random forest is that the variance of the model decreases as the number of trees in the forest increases, while the bias remains the same. Furthermore, random forest has many other advantages such as sparse number of parameters and resistance to over-fitting.

Naïve Bayes is a probabilistic classifier based on Bayes Theorem with strong independence assumptions between features. Assuming that features are not correlated with each other is not a true assumption in many problems and it can conversely affect the accuracy of the classifier. The main advantage of Naïve Bayes is that it is an online algorithm (can update its model in real time) and its training can be completed in linear time.

Multinomial Logistic Regression is a classification method that uses the main idea of logistic regression to classify multiclass problems. Logistic regression is a predictive analysis like other regression analyses. Logistic regression can describe data and explain the relationship between features and classes.

The performance examination results are shown in Table 2.2. The authors used five-fold cross validation for their experiments. Based on their experiments ID3 took few minutes to be trained and classify the testing set. Random forest with 100 trees took more than 15 hours for the same process. In addition, multinomial logistic regression took more than 2 days to be trained and classify the testing test. In addition, according to the weighted average of the three-evaluation metrics (Pr, Rc, F1), the highest accuracy belongs to random forest and ID3 algorithms. Also, in terms of recall ID3 won first place by far. Logistic regression achieves the worst result overall. Considering the execution time and the evaluation metrics ID3 is the best algorithm with the shortest execution time and highest accuracy.

Algorithm	Pr	Rc	F1
ID3	0.78	0.65	0.69
RF	0.77	0.56	0.62
Naïve Bayes	0.41	0.11	0.05
Logistic regression	0.25	0.02	0.04

Table 2.2: Performance examination results

Brute Force attacks are another type of cyber-attack included in my research and is described below.

2.1.2 Brute Force Attacks

Brute force attacks [5] are a type of cyber-attack that involves trying every possible password or combination of characters until the correct one is found. The goal of a brute force attack is to gain unauthorized access to a system or account by bypassing its authentication mechanisms. Brute force attacks can be carried out manually, but they are more commonly automated using specialized software or tools.

FTP Patator and SSH Patator are two types of brute-force attacks later examined in this research that aim to gain unauthorized access to FTP and SSH servers, respectively. The attacks involve using a program or tool, such as Patator [11], to automate the login attempts with a list of username and password combinations until a valid one is found. The goal is to exploit weak or commonly used credentials that may have been set by the server administrator or the users themselves.

Brute Force Variants

In the case of FTP Patator, the attack generates a high volume of traffic to the FTP server, as the tool sends multiple login requests per second. The traffic generated by FTP Patator can be characterized by a high number of TCP connections and a significant increase in the number of successful and failed login attempts compared to normal FTP traffic.

Similarly, in SSH Patator, the attacker uses a program or tool to try multiple username and password combinations to gain unauthorized access to SSH servers. SSH Patator generates a high volume of traffic that can be characterized by a high number of TCP connections and a significant increase in the number of failed login attempts compared to normal SSH traffic.

Brute Force Traffic Characteristics

One of the most obvious traffic characteristics of brute force attacks is the high volume of traffic generated by the attack. This is because the attacker tries multiple login combinations in a brief period of time, resulting in a significant increase in traffic.

Another traffic characteristic of brute force attacks is the repeated attempts to login using different usernames and passwords. This generates traffic that is highly correlated with the login attempts. This traffic can be analyzed to identify patterns in the login attempts, such as the specific usernames and passwords that the attacker is trying to use.

In addition to these traffic characteristics, brute force attacks often have a distinct traffic pattern when compared to normal traffic. This pattern is often characterized by a higher number of login attempts from a single IP address. This is because attackers often use a single IP address to carry out the attack, resulting in a high number of login attempts from that IP address.

2.2 Existing Mitigation Approaches

Detecting and preventing cyber-attacks is a critical aspect of maintaining the security of computer systems and networks. One way to identify and prevent these attacks is through the detection of malicious traffic. There are several approaches to detecting malicious traffic, including signature-based detection, anomaly-based detection, and behavior-based detection.

Signature-based detection [12] involves the use of predefined patterns, or signatures, to identify known malicious traffic. These signatures are created based on the characteristics of known attacks, such as specific network protocols, packet contents, or IP addresses. Signature-based detection is effective for identifying known threats, but it is less useful for detecting new or unknown attacks.

Anomaly-based detection [12] on the other hand, focuses on identifying traffic that deviates from normal patterns. This approach involves the use of statistical models to identify unusual patterns in network traffic, such as a sudden increase in data transfer or unusual patterns of data flow. Anomaly-based detection is effective in detecting unknown attacks, but it can also generate false positives, as normal network activity may appear anomalous.

Behavior-based detection [13] is a newer approach that uses machine learning techniques to identify suspicious behavior on a network. This approach involves the use of algorithms that learn from network behavior over time and can identify patterns that may indicate an attack. Behavior-based detection is particularly effective in detecting advanced persistent threats (APTs) that can go undetected by other methods.

This research focuses on Behavior-based detection where a data analyst has to go through the process of analyzing traffic datasets and build machine learning models that can precisely

detect such attacks. This paper proposes automation on the tasks of the analyst to make the model creation easier and faster.

2.3 Manual Data Analysis

Manual data analysis [14] is a major step in identifying patterns and trends in cyber-attack traffic datasets. To build effective models, analysts must go through several phases, including data cleaning, data exploration, feature engineering, and model training and evaluation.

The first phase, data cleaning, involves removing any incomplete or irrelevant data from the dataset. This step is important to ensure that the data is accurate and unbiased. Once the data is cleaned, the analyst moves on to data exploration, where they analyze the dataset to identify patterns, correlations, and outliers. This phase helps the analyst gain a better understanding of the data and identify potential variables that may be important for modeling.

After data exploration, the analyst moves on to feature engineering, which involves selecting and transforming variables to improve the model's accuracy. This phase may include creating new variables, transforming variables, or selecting variables that are most relevant to the model.

Once the data has been pre-processed and the variables selected, the analyst can move on to model training and evaluation. In this phase, the analyst selects an appropriate machine learning algorithm and trains it on the dataset. The model is then evaluated using a validation dataset to determine its accuracy, precision, recall, and F1 score.

Throughout these phases, manual data analysis requires the analyst to have a deep understanding of the dataset, the variables, and the machine learning algorithms used. This process is iterative and may require several rounds of testing and refinement before a final model is selected.

2.3.1 Python

Python is one of the most popular programming languages for data analysis due to its versatility and user-friendly libraries. With the help of libraries like NumPy, Pandas, Sklearn etc., Python provides powerful tools for data manipulation, exploration, and visualization. These libraries allow for easy loading of various data formats such as CSV, Excel, SQL databases, etc. Python also enables users to perform complex calculations and statistical analyses.

Libraries

For the implementation of the system, I used several Python libraries mainly for data analysis and interaction with the database.

Scikit-learn

Scikit-learn [26], known as sklearn, is a popular open-source Python library that provides efficient tools for data mining, machine learning, and data analysis. It is built on top of NumPy, SciPy, and Matplotlib, and it integrates well with other libraries such as Pandas. Scikit-learn offers a wide range of machine learning algorithms, including classification, regression, clustering, and dimensionality reduction that were used in this project. It also provides various data preprocessing techniques, model selection and evaluation, and model optimization methods.

I used several methods for preprocessing, feature selection, model training and tuning.

Pandas

Pandas [27] is a popular open-source Python library that provides efficient tools for data manipulation, analysis, and cleaning. It offers powerful data structures, such as Data Frames and Series, that allow users to work with data in a flexible and intuitive manner. With Pandas, users can easily load and store data from various file formats such as CSV, Excel, SQL databases, and more. Additionally Pandas enables users to perform various data operations, including merging, filtering, aggregating, and pivoting. It provides easy-to-use functions for handling missing data and dealing with time-series data.

I used Pandas mainly for the Data Frame structure, preprocessing and feature selection.

NumPy

NumPy [28] is an open-source Python library for scientific computing that provides efficient tools for numerical operations and array processing. It provides a high-performance array object, which enables users to perform fast and efficient computations on copious amounts of data. Furthermore, NumPy offers a wide range of mathematical functions for numerical operations, such as linear algebra, Fourier transform, and random number generation. The library is highly optimized for performance, making it an essential tool for scientific computing, data analysis, and machine learning.

I used it mainly for dataset operations and preprocessing.

Vaex

Vaex [29] is a Python library that provides a fast and memory-efficient way to work with large datasets. It is designed to handle billions of rows of data with ease, making it an ideal tool for big data processing and analysis. Vaex provides a high-performance DataFrame API that allows users to perform data manipulations and calculations on their data, similar to Pandas. However, unlike Pandas, Vaex operates on disk-based or memory-mapped files, which means that it can handle large datasets without consuming enormous amounts of RAM.

I used Vaex to perform the dataset initialization on the user's uploaded dataset to gain basic information about the dataset. Unfortunately, I couldn't find a wrapper method for the `cross_validation` [30] method that I used for the model training, and it would be too much work to remake the system to use a vaex compatible method for the training.

XGBoost

XGBoost [31] is a popular open-source software library that provides a fast and efficient implementation of gradient boosting algorithms. XGBoost uses a combination of gradient boosting and regularized learning to improve accuracy and prevent overfitting, making it a

powerful tool for predictive modeling. Additionally, it supports parallel processing and can be easily integrated with other machine learning frameworks like scikit-learn.

Since Scikit-learn doesn't have support for the XGBC algorithm I used this library to be able to use it.

PyMongo

PyMongo [32] is a library that provides a Python interface for working with MongoDB, a popular NoSQL document-oriented database. PyMongo allows users to easily interact with MongoDB databases from within their Python applications, providing a high level of flexibility and ease-of-use. With PyMongo, users can perform a wide range of database operations, including inserting, updating, and deleting data, as well as querying the database for specific documents or collections.

I used PyMongo to interact with the database retrieve the requested experiments save the results, etc.

Others

I used several other libraries to regulate and monitor the created experiments as well as provide feedback information about the running processes. These libraries include sys, psutil, os, subprocess, atexit, json and time.

2.3.2 Feature Selection

Feature selection is a crucial aspect of data analysis that involves identifying the most relevant and useful variables in a dataset. The goal of feature selection is to reduce the dimensionality of the data, making it easier to interpret and analyze. In other words, it helps to identify which features have the most significant impact on the outcome variable of interest.

Constant Features

Constant features are the features in a dataset that have a constant value for all observations. Such features are also referred to as zero-variance or quasi-constant features. Constant features provide no useful information for the analysis, as they do not vary across the dataset and thus do not contribute to the prediction of the outcome variable. Moreover, including constant features in the analysis can lead to problems such as overfitting and bias. Therefore, it is essential to identify and remove constant features from the dataset before applying any analysis. There are various methods for detecting constant features, such as calculating the variance or coefficient of variation for each feature. Such method is available in scikit-learn library [26] in python and is used in this study to calculate the constant features of the dataset. Once the constant features are identified, they can be removed from the dataset, reducing the dimensionality of the data and improving the accuracy and efficiency of the analysis.

Highly Correlated Features

Highly correlated feature selection is another important technique in data analysis, which involves identifying and selecting a subset of features that have a strong linear relationship with other features. Highly correlated features can lead to multicollinearity, which can cause instability in the model and affect the accuracy of the results. Therefore, it is essential to identify and remove highly correlated features before applying any analysis. Several techniques are available to detect highly correlated features, such as correlation coefficient, scatter plot, and heatmaps.

For the purposes of this research, correlation coefficient was measured to identify highly correlated features. Pandas' library offers the corr method [22] that calculates the correlation coefficient for each feature of the given dataset. There are several algorithms that can be used for this purpose, but for the purposes of this research I use Pearson and Spearman. Once the highly correlated features are identified, they can be either removed from the dataset or combined into a single feature to reduce dimensionality and improve the accuracy of the analysis. By selecting only the most relevant and non-redundant features, highly correlated feature selection can help improve the interpretability and generalizability of the analysis results.

Pearson

Pearson correlation [23] measures the linear relationship between two continuous variables. It computes the correlation coefficient, which ranges from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation. Pearson correlation assumes that the variables are normally distributed and have a linear relationship.

Spearman

Spearman correlation [24] measures the monotonic relationship between two variables, which means it can capture non-linear relationships as well. It computes the Spearman rank correlation coefficient, which ranges from -1 to 1, where -1 indicates a perfect negative monotonic relationship, 0 indicates no monotonic relationship, and 1 indicates a perfect positive monotonic relationship. Spearman correlation does not assume any specific distribution of the variables and is more robust to outliers than Pearson correlation.

In simple words, Pearson can only detect linear correlations between variables and spearman can also detect variables relationships in a more general sense.

Best Features

K-Best feature selection is a technique used in data analysis to select the best features in a dataset based on a statistical test. The approach involves selecting the top k features with the highest scores based on a statistical test, such as chi-square, ANOVA, or mutual information. The technique is particularly useful in high-dimensional datasets where there are many features, and selecting the most relevant ones can improve model accuracy and reduce overfitting.

K-Best feature selection can be implemented in Python using the scikit-learn library, which provides a feature selection method named SelectKBest [25] with various functions to perform univariate statistical tests for feature selection.

The 3 most common algorithms used are described below:

Chi-2

Chi-2 [34] is a statistical test used to determine whether there is a significant association between two categorical variables. The test calculates the difference between the observed and expected frequencies of the variables and measures the significance of this difference using a p-value. While Chi-square is a useful technique for feature selection in datasets with categorical variables, it may not be effective for DDoS detection datasets that typically contain continuous variables, such as network traffic data.

In DDoS detection, the variables of interest are usually the traffic characteristics, such as packet size, packet rate, and connection rate. These variables are typically continuous and do not fit the assumptions of the Chi-2 test, which assumes that the variables are categorical and have discrete values. Therefore, using it for feature selection in DDoS datasets may not yield accurate results and could lead to the selection of irrelevant features or the exclusion of important ones.

ANOVA

ANOVA (Analysis of Variance) [35] is a statistical test used to determine whether there are significant differences between the means of two or more groups in a dataset. ANOVA is a useful technique for feature selection in DDoS detection datasets that typically contain continuous variables, such as network traffic data. ANOVA can be used to test whether there are significant differences in the means of a particular traffic characteristic, such as packet size or packet rate, between the normal and DDoS traffic groups.

ANOVA works by calculating the sum of squares between and within the groups and comparing them using an F-test to determine whether there is significant variation in the means. The technique assumes that the variables are normally distributed, and the groups are independent and have equal variances.

Mutual Information

Mutual information [36] measures the mutual dependence between two variables by calculating the reduction in uncertainty of one variable given the knowledge of the other. Mutual information is a powerful technique for feature selection in DDoS detection datasets

because it can capture both linear and nonlinear relationships between the traffic characteristics, which are often critical indicators of DDoS attacks.

ANOVA and Mutual Information tests are used and compared extensively in the experimental analysis of DDoS datasets that contain network traffic features.

2.3.3 Machine Learning

Machine learning algorithms are becoming increasingly popular for detecting cyber-attacks such as DDoS attacks. These algorithms are effective at identifying patterns and anomalies in network traffic that may indicate a cyber-attack, which can help organizations respond quickly and mitigate potential damage. Various machine learning algorithms, such as random forest, decision trees, XGB classifier, KNN, and LDA, can be utilized to detect cyber-attacks. These algorithms work by analyzing features of network traffic, building models based on training data, and making predictions based on new data. In practice, the effectiveness of a machine learning algorithm for detecting cyber-attacks will depend on factors such as the quality of training data, the algorithm's hyperparameters, and the nature of the attack being detected.

Decision Trees

Decision trees [16] are a machine learning algorithm that can be used for both classification and regression tasks. Decision trees work by recursively partitioning the data into subsets based on the values of the features, with the goal of minimizing impurity in each subset. In the context of cyber-attacks, decision trees can be used to identify features in network traffic that are indicative of a DDoS attack. Decision trees have been shown to be effective at detecting DDoS attacks, as reported by Chen, Y., Pei, J., and Li, D. [16] in their paper where they proposed an efficient and low-latency system on DDoS detection based on Decision Trees.

Random Forest

Random forest [17] is a machine learning algorithm that uses a collection of decision trees to make predictions. Random forest works by constructing many decision trees, each using a randomly selected subset of the features and samples from the dataset. The final prediction

is made by taking the majority vote of all the decision trees. In the context of cyber-attacks, random forest can be used to identify patterns in network traffic that are indicative of a DDoS attack. Random forest has been shown by Chen, Y., Hou, J., Li, Q., and Long, H. [17] to be effective at detecting several types of DDoS attacks such as UDP, TCP and ICMP floods.

Extreme Gradient Boost

XGB classifier, also known as extreme gradient boosting, is a machine learning algorithm that similarly with random forest uses an ensemble of decision trees to make predictions. XGB classifier [18] works by constructing a series of decision trees, each building on the mistakes of the previous trees, with the goal of minimizing the loss function. In the context of cyber-attacks, XGB classifier can be used to identify patterns in network traffic that are indicative of a DDoS attack. XGB classifier has been shown to be effective at detecting DDoS attacks with high accuracy, as reported in the paper: "XGBoost Classifier for DDoS Attack Detection and Analysis in SDN-Based Cloud" by Chen, Z., Jiang, F., Cheng, Y., Gu, X., Liu, W., and Peng, J [18]

K-Nearest Neighbors

K-nearest neighbors [19] is a machine learning algorithm that can be used for both classification and regression tasks. KNN works by finding the k closest training examples to a given test example and using their labels to make a prediction. In the context of cyber-attacks, KNN can be used to identify patterns in network traffic that are indicative of a DDoS attack. KNN has been shown by Vu, N. H., Choi, Y., & Choi, M. [19] to be effective at detecting DDoS attacks with high accuracy.

Linear Discriminant Analysis

LDA, or linear discriminant analysis [20], is a machine learning algorithm that can be used for classification tasks. LDA works by projecting the data onto a lower-dimensional space that maximizes the separation between the classes. LDA can be used to identify features in network traffic that are indicative of a DDoS attack and has been shown to be effective at detecting DDoS attacks with high accuracy, as reported by Thapngam, T., Yu, S., & Zhou, W. in their paper: "DDoS discrimination by Linear Discriminant Analysis (LDA)" [20].

2.4 Benefits of Automation

Automating the process of data analysis [15] can bring several benefits to the analysts. One of the primary benefits of automation is the speed and efficiency with which it can process large volumes of data. By automating data analysis, organizations can process massive amounts of data much faster than manual methods, which can be time-consuming and error prone.

In addition to speed and efficiency, automation can improve the accuracy and consistency of data analysis. Automated systems can be programmed to apply the same criteria and methodology consistently to all data, eliminating potential human error and bias. This consistency can improve the quality of data analysis and the reliability of the resulting insights.

Another benefit of automation is its ability to identify and respond to potential threats in real-time. Automated systems can monitor network traffic and identify potential threats as they occur, enabling organizations to respond quickly and proactively to prevent damage or mitigate the impact of an attack.

Automated systems can also improve scalability and cost-effectiveness. As the volume of data increases, organizations can easily scale up their automated systems to process and analyze the data efficiently. This can reduce the need for additional personnel and resources, which can be costly.

Finally, automation can free up human analysts to focus on higher-level tasks that require human intuition and problem-solving skills. Automated systems can handle routine tasks, such as data processing and cleaning, allowing analysts to focus on more complex tasks, such as developing and refining machine learning models.

Chapter 3

Related Work

3.1 Similar Platforms.....	26
3.1.1 DataRobot.....	26
3.1.2 RapidMiner.....	27

3.1 Similar Platforms

Several online platforms and tools are available for running data analysis experiments and machine learning tasks.

3.1.1 DataRobot

DataRobot [38] is an enterprise-grade automated machine learning platform that enables organizations to build, deploy, and manage machine learning models at scale.

It automates the entire machine learning workflow, from data preparation to model deployment, which reduces the time and resources required for developing machine learning models. Additionally, it offers a wide range of machine learning algorithms, including deep learning, gradient boosting, and random forests, which enables users to build complex models with ease.

DataRobot is a cloud-based platform that can scale to handle large datasets and complex models. It also supports parallel processing and distributed computing. It provides transparency into the machine learning models it creates, allowing users to understand how the models work and how they make predictions. In addition, it allows users to customize the machine learning models by tuning the hyperparameters and selecting specific features for the model.

Even though DataRobot is a great tool for automating data analysis and machine learning, it has a few drawbacks. First of all, it can be expensive for small organizations and individual users. It requires some technical expertise to use, especially for configuring the platform and interpreting the results. Furthermore, it may require additional software or systems to integrate with existing workflows and data sources. The tool automates many aspects of the machine learning workflow, which may limit the control users have over the process. Lastly,

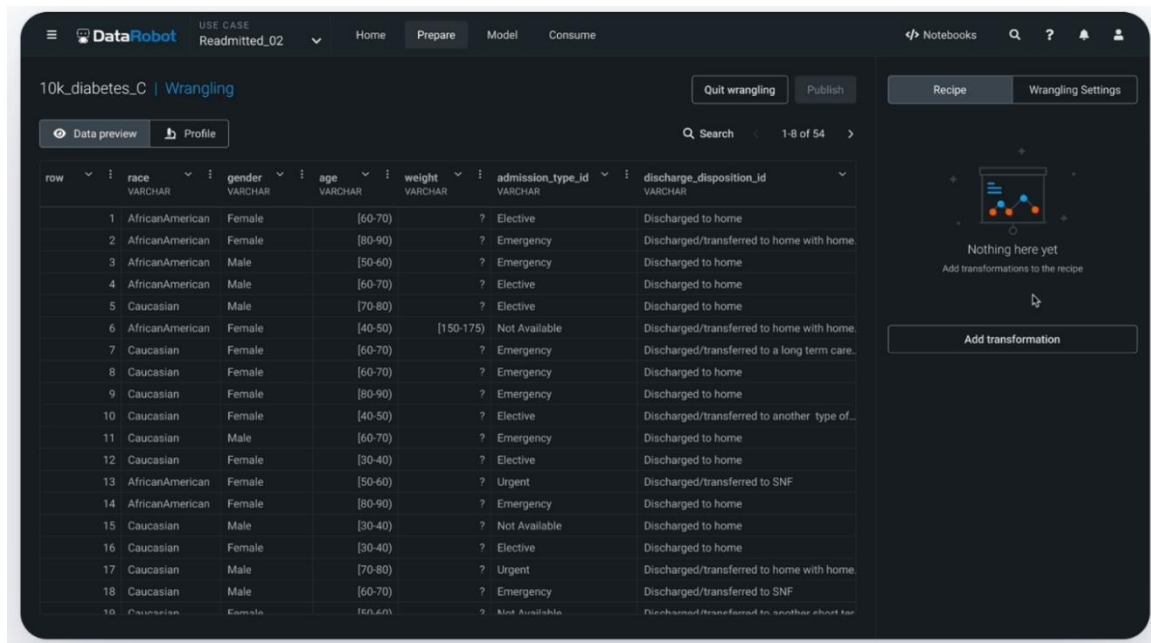


Figure 3.1: DataRobot's Interface

it provides a structured environment for machine learning, which may not be suitable for users who require more flexibility in their approach.

Figure 3.1 shows a screenshot from the DataRobot's interface that allows data transformation operations specified by the user. There are many other enterprise tools providing similar functionality with DataRobot, but in my opinion they lack simplicity which can make deployment and use of these tools difficult and expensive.

3.1.2 RapidMiner

RapidMiner [42] is a powerful and comprehensive data science platform that empowers organizations to extract valuable insights and make data-driven decisions. It provides a range of tools and functionalities for data preparation, machine learning, predictive analytics, and deployment. With its intuitive visual interface and extensive library of algorithms, RapidMiner

offers both beginner-friendly features and advanced capabilities for data scientists and analysts.

One of the key strengths of RapidMiner is its data preparation capabilities. It allows users to import, transform, and cleanse data from various sources, ensuring data quality and consistency. The platform provides a wide range of operators for data manipulation, feature engineering, and data integration, enabling users to effectively prepare their data for analysis and modeling. It also offers a variety of machine learning algorithms and techniques, including classification, regression, clustering, association rules, and text mining. These algorithms are easily accessible through a drag-and-drop interface, allowing users to build complex predictive models without requiring extensive coding knowledge. Additionally, RapidMiner supports the integration of external libraries and custom code, providing flexibility for advanced users who want to extend the platform's capabilities.

Another notable feature of RapidMiner is its Auto Model functionality. With Auto Model, users can automate the process of model selection and hyperparameter optimization. This feature saves time and effort by automatically exploring different algorithms and configurations to find the best-performing models for a given task. It allows users to quickly experiment with diverse options and identify the most effective approaches without the need for manual trial and error.

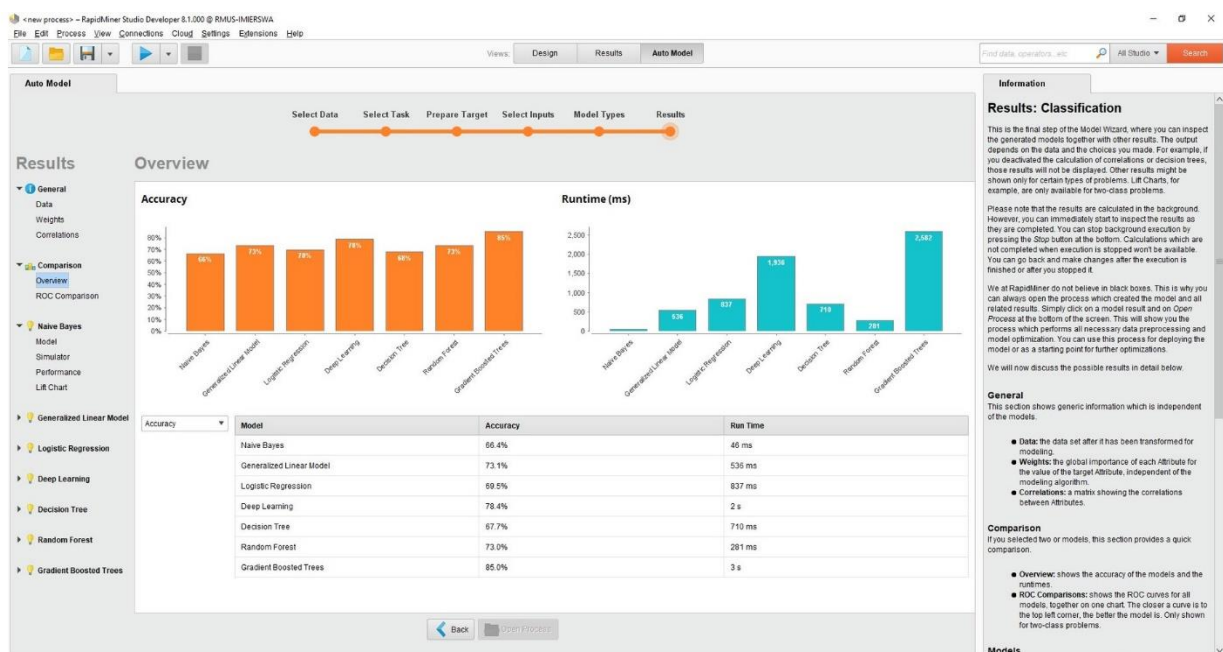


Figure 3.2: RapidMiner's Interface

RapidMiner supports seamless integration with popular data storage systems, databases, and other data science tools, enabling users to leverage their existing infrastructure and workflows. Furthermore, it provides options for deploying models into production environments, making it easier to operationalize and scale data-driven solutions.

Figure 3.2 shows how RapidMiner's compares the performance of several machine learning algorithms.

Both DataRobot and RapidMiner automate the process of machine learning in much better ways since they are enterprise tools meant to provide value to their customers. Even though they are reliable they have the following limitations:

1. They are priced tools and their usability depends on the clients' budget.
2. Their main objective is to build and deploy machine learning models(industrial), rather than analyzing the datasets to better understand the data being processed(research).

I believe my platform is more research oriented and is supposed to allow the user to learn as many things as possible about the data themselves rather than deploying an accurate model. It gives the analyst insights about variable relationships, algorithms performances under certain configurations. Other than that, it is supposed to provide a service to the users rather than requiring the user provide the resources for the experiments. For example, RapidMiner is an executable running on the user's computer, which might be a limitation factor.

My platform is a much more simplified version of these enterprise tools and is mainly meant for research purposes and it offers slightly different functionalities (feature selection) therefore it is difficult to be directly compared to them. For this reason, I am keeping this chapter short, and I am not going into further comparison between these tools and my platform.

Chapter 4

System Architecture

4.1 System Requirements.....	30
4.2 System Design.....	31
4.2.1 Architecture.....	32
4.2.2 Data Storage.....	33
4.2.3 Components.....	38
4.3 Deployment.....	42
4.3.1 Web Server.....	42
4.3.2 Database.....	43
4.3.3 PHP.....	43
4.3.4 Python.....	44
4.4 System Workflow.....	44
4.4.1 Dataset Upload.....	47
4.4.2 Feature Selection.....	48
4.4.3 Model Training and Tuning.....	55

4.1 System Requirements

The system has several requirements in order to be deployed and function properly. The hardware components required include a computer or server with a sufficient processor, RAM, and storage capacity to handle large datasets and process the requested experiments. As far as the software requirements are concerned, an operating system, a web server, and a database management system are needed. That's on hardware and software requirements. For the required technologies I have used I am explaining further below.

Back-End

As far as the backend is concerned, the system can be deployed in any operating system and any web server supported by that operating system. I used **Windows 10 Pro** with the **Apache**

web server that came together with the XAMP installation. For the DBMS, **MongoDB** (v 1.35.0) is required to be installed in the system as well as the necessary databases and collections must be created (I discuss in detail in the deployment section).

PHP (v 8.0) is the backend programming language I used for client-server communication, therefore it needs to be present on the system as well as its **mongodb module** that will support connectivity with the database. Additionally, PHP needs to be configured, so it can receive a large amount of data from the clients otherwise the dataset upload might fail.

Python 3 is the second backend programming language I used for the data analysis tasks, so it also needs to be installed and its installation path should be added (if not added automatically) in the PATH environment variable, so python programs can be started only with the 'python' keyword.

There are several python libraries I used that don't come together with the python installation and are listed below:

- Scikit-learn
- Pandas
- NumPy
- XGBoost
- PyMongo
- Vaex
- Psutil

These libraries need to be exclusively installed, but more specific installation and deployment steps are listed in the Deployment section.

Front-End

The Front-End of the system is structured with HTML, styled with CSS and the interactivity is added with JavaScript. Some styling components are taken from the Bootstrap [33] framework that is loaded from a CDN on the client's browser. Messages are exchanged between the client and the server using the JSON format. The requirements for the front-end are handled by the web server, therefore only the website's files need to be present in the directory used by the webserver to serve the files to the clients.

The user can interact with the system using any device with any browser with JavaScript enabled. Those are all the required technologies to allow the platform to function. Specific installation details are provided in the deployment section.

4.2 System Design

The high-level idea of the system is to provide an interface for the user to upload their dataset on the server and then be able to request experiments on the datasets and get back the results.

4.2.1 Architecture

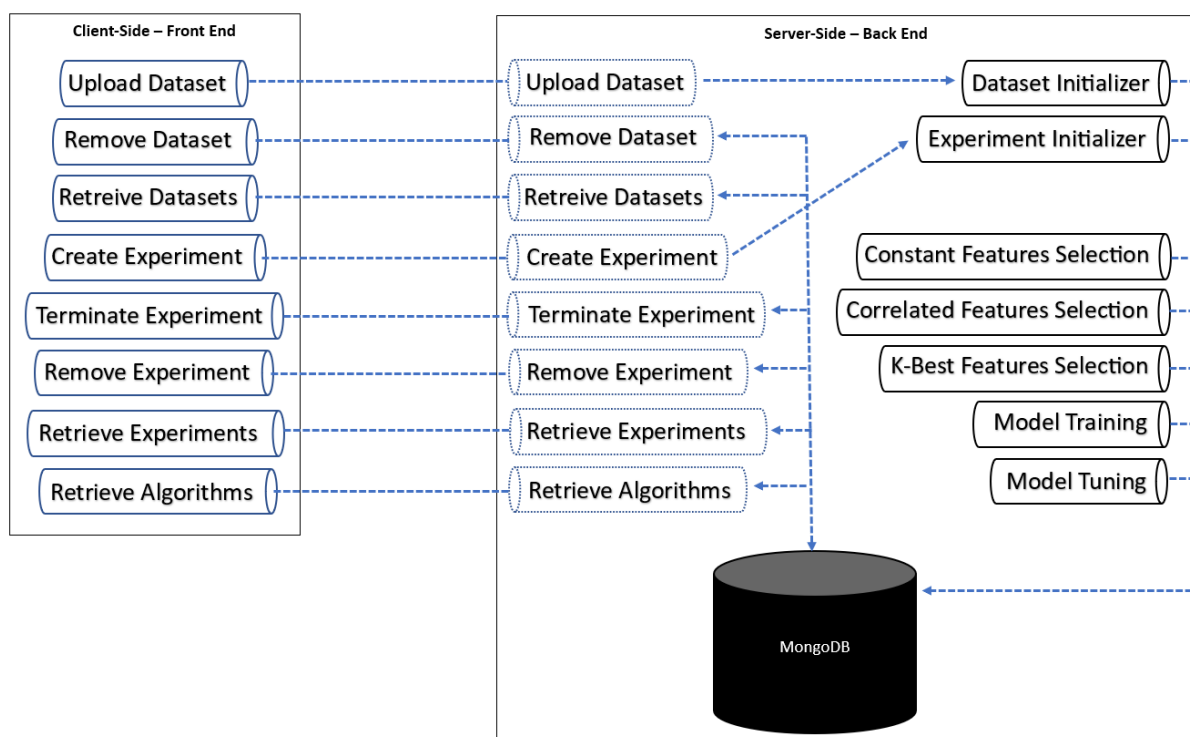


Figure 4.1 System Architecture

Figure 4.1 displays the high-level idea of the system with the different components it supports.

The client side has several components that are responsible for uploading and removing datasets on the server as well as retrieving information about them. In addition, it allows creating experiments with different options/parameters, terminate a running experiment, remove an experiment entirely from the server as well as retrieve all the experiments from the server. Also, it can retrieve all the available machine learning algorithms with their relative information supported by the system.

The requested operation, once selected by the user with its corresponding parameters, is encoded and sent to the server via an AJAX request. The requested server-side resource (PHP file) is executed and handles each request. Each server-side resource creates a connection to the database to either retrieve or save information.

Each one of the system's components are broken down and analyzed in the 4.2.3 subchapter.

4.2.2 Data Storage

The database I decided to use is a mongo database, which is a NoSQL database. The reason I made this choice is because I have a large amount of data to save for each experiment and there are many nested fields, which would make the implementation of a traditional SQL database more complex. In addition, each experiment has the form of a JSON object in the front-end side, therefore it makes more sense to save it in the same form in a database, which is what I did.

Experiments

The first collection I have is the Experiments collection. As the name implies, each experiment is saved in this collection.

```
_id: ObjectId('6415a7e25c780000f10075e1')
id: "6415a7e2426fc"
name: "exp_8209"
type: "Constant Features Selection"
dataset_Obj: Object
  id: "64159a077dd09"
  name: "CIC-DDoS2019"
  target_classes: Array
  features: Array
  target: "Label"
configurations_list: Array
subsets_list: Array
preprocessing_list: Array
models_list: Array
hyperparam_models_list: Array
corr_models_list: Array
kbest_models_list: Array
date: "2023-03-18T12:00:34.269Z"
memUsage: "0.00%"
cpuUsage: "0.00%"
status: "Completed"
duration: 523423
pid: 5528
dataset_info: Object
constant_features_results: Array
```

Figure 4.2: Experiment Sample in the Database

Figure 4.2 Shows how an experiment is saved in the database. `_id` is the unique id that is automatically assigned to the experiment by the Mongo DBMS. `Id` is the id I assign to the experiment to be able to distinguish it from the rest. `Name` is the name of the experiment assigned by the user. `Type` is the type of the experiment and the `dataset_Obj` entry holds information of the dataset of the experiment such as the target classes and the features of the dataset.

Moving on, `configuration_list` holds the information about how many samples to be loaded for each target class e.g. 100000 normal traffic samples. `subsets_list` holds the different features subsets specified by the user for the current experiment. `preprocessing_list` holds the information about the different preprocessing options selected by the user. `models_list` holds the training models specified for a model training experiment. Such information includes the algorithm, `k` for cross validation and the chosen hyperparameters for the algorithm.

Regarding the list holding that will hold experiment related parameters, `hyperparam_models_list` holds the ranges to test for each parameter specified for a hyperparameter tuning experiment. `corr_models_list` holds the algorithms and correlation thresholds specified for a correlated feature selection experiment. `kbest_models_list` holds the algorithms and number of best features to calculate for a best features selection experiment. `Date` is the date of the experiment creation.

Furthermore, `memUsage` is the current RAM usage(percentage) of the experiment. This field is updated on runtime by the monitor process of each running experiment. Similarly, with `cpuUsage` which is about the current CPU usage of the running experiment. `Status` is the status of the experiment which can be 'Draft' for saved but not started experiments, 'Running', for currently running experiments, 'Terminated', for experiments that terminated either normally or unexpectedly. 'Completed' for completed experiments and 'Queued' for submitted experiments but not started yet by the operating system. An experiment might be delayed from starting because of lack of resources. Such experiments remain in the 'Queued'.

`Duration` is the duration the entire experiment took to complete or terminate and is updated at the end of the experiment. The `pid` is the pid of the process running the experiment. This information is used to terminate the experiment if requested by the user, but also for

debugging purposes. `dataset_info` holds information about the portion of the dataset selected, as it is possible to select a portion of the dataset and not all of it. This information is the result of the 'describe' method of the panda's library when executed on the dataset which is basically statistical information about each feature. Lastly, the `constant_features_results` hold the results of the constant feature selection experiment.

Figure 4.3 shows an example of what information the 'describe' method obtains and how it is displayed on the platform.

/	Src Port	Dst Port	Protocol	Flow Duration	Tot Fwd Pkts	Tot Bwd Pkts	TotLen Fwd Pkts	TotLen Bwd Pkts
count	200000.00	200000.00	200000.00	200000.00	200000.00	200000.00	200000.00	200000.00
mean	49612.35	3972.05	7.87	7966888.48	105.09	4.87	3475.35	3312.54
std	17833.43	13189.72	4.24	25882825.26	3547.37	123.71	113506.03	176036.24
min	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
25%	50503.00	80.00	6.00	1216.00	1.00	1.00	0.00	0.00
50%	54378.00	80.00	6.00	7132.50	1.00	1.00	0.00	0.00
75%	59552.25	443.00	6.00	150444.75	3.00	4.00	250.00	488.00
max	65534.00	65534.00	17.00	119999976.00	248799.00	23182.00	7961568.00	31947804.00

Figure 4.3: 'Describe' method output

Datasets

The Datasets collection holds information about the uploaded datasets.

```

_id: ObjectId('64159a075c780000f10075db')
name: "CIC-DDoS2019"
target_index: "-1"
type: "Classification"
date: "2023-03-18T11:01:27.409Z"
status: "Completed"
memUsage: "0.00%"
cpuUsage: "0.00%"
id: "64159a077dd09"
pid: 2388
▸ features: Array
  target: "Label"
▾ targets: Array
  0: "DrDoS_NetBIOS"
  1: "UDP_lag"
  2: "WebDDoS"
  3: "Syn"
  4: "DrDoS_NTP"
  5: "DrDoS_SNMP"
  6: "DrDoS_SSDP"
  7: "BENIGN"
  8: "DrDoS_LDAP"
  9: "DrDoS_MSSQL"
  10: "DrDoS_UDP"
  11: "TFTP"
  12: "DrDoS_DNS"
duration: 241040.59529304504

```

Figure 4.4: Dataset Sample in the Database

Figure 4.4 shows an entry of a dataset in the database. 'name' is the dataset's name specified by the user. 'target_index' is the column's index that represents the variable to be predicted. Since the platform only supports classification datasets, the target index must correspond to a column that has no more than 50 different values. Otherwise, the target is deemed to be non-categorical and the dataset initialization fails. Since python supports negative indexing, -1 corresponds to the last column, -2 to the column before the last one etc.

Additionally, 'date' is the upload data of the dataset. status is the status of the dataset's initialization. 'memUsage' and 'cpuUsage' fields hold the RAM and CPU usage information of the dataset initializer process. 'features' is a list holding the string name of each feature. That list is generated by the initializer process and appended to the database. 'target' is the name of the target column as specified by the user with the target_index. 'targets' is a list of the different classes discovered in the target column and 'duration' is the duration of the dataset's initialization.

Algorithms

The last collection I used is the Algorithms collection that holds the information for each supported classification algorithm. This collection is fixed and cannot be changed by the user in any way through the interface.

```
_id: ObjectId('641600d44a5fac677a37bc29')
name: "Extreme Gradient Boost Classifier"
name_abr: "XGBC"
▼ hyper_parameters: Array
  ▼ 0: Object
    name: "n_estimators"
    type: "num_range"
    start: 10
    stop: 1000
    incr: 10
    description: "The number of trees in the ensemble."
    slider_incr: 1
    default_value: 100
    selected_value: 100
    selected_start: 100
    selected_stop: 1000
    selected_incr: 10
  ▶ 1: Object
  ▶ 2: Object
```

Figure 4.5: Algorithms Sample in the Database

Figure 4.5 shows a sample of the XGBC's algorithm entry. 'name' is the full name of the algorithm and 'name_abr' the abbreviation of the algorithm. The 'hyper_parameters' array holds the information for each of the hyperparameters. The 'name' of the hyperparameter,

the 'type' the hyperparameter. It can be 'num_range', that indicates a range of integers or floats. 'class_range' indicates an array of the classes(strings) the hyperparameter can take. 'boolean' indicates the hyperparameter can be either true or false.

Moving on, the 'start' and 'stop' fields are the boundaries of the range the user can select for a tuning experiment. They are used to give an approximate idea of what values the hyperparameter can take. 'incr' is the default increment. For example, if the increment is 10, the attempted values for that hyperparameter would be 10, 20, 30, 40 ... The 'description' provides a brief description of the hyperparameter.

Next, we have the 'slider_incr' is used by the slider component for that hyperparameter to increase/decrease by that value when the user uses the slider. The 'default_value' is the default value of the hyperparameter. The 'selected_value', 'selected_start', 'selected_stop', 'selected_incr' fields correspond to the user's selection about the hyperparameter and can be used for a model training or tuning experiment.

4.2.3 Components

The system consists of several different components that handle the supported operation the user can request.

Upload Dataset

This component handles the upload and initialization of the dataset provided by the user. Through the interface the user is prompted to enter the name of the dataset, its type and the target index. For now, only classification datasets are supported but room for upgrade is left for other kinds, like regression. The user can then either drag and drop the files of the dataset or select and upload them through the file selector. For simplicity, only .xlsx and .csv files are supported.

When the user submits the dataset files, he can see the upload progress of each file and can decide to revert the operation at any time.

The server receives the files and saves them in a specific directory on the server before passing control to the Dataset Initializer.

Dataset Initializer

This component is responsible for parsing the recently uploaded files and obtaining basic information about them. Firstly, it assigns a unique id to the dataset and then generates a process to parse the files.

To be able to handle abnormal exits and keep track of the memory and CPU usage, for every experiment including dataset initialization, there is always a monitor process that keeps an eye on the experiment process and updates the database with the current state of the process. The monitor makes use of the psutil library in python to extract information about the experiment process and periodically update the database so the user can have feedback on the progress of the experiment.

The initializer will first load the dataset with vaex [29] which makes the operation much faster than the traditional file loading with pandas. Then it will detect the target column with the column index specified by the user and check if it represents a categorical variable. If not, it will terminate indicating the problem. It will also calculate and store in the database the feature names and the several classes of the target column.

When the initialization process of a dataset is started, the user is redirected to 'My resources' page where he can see the status of the initialization as well as terminate and remove any dataset.

Remove Dataset

This component handles the deletion of a dataset when requested by the user. When such an operation is requested the database entry corresponding to the dataset to be removed is deleted and the files are removed from the disk.

Retrieve Datasets

This component is responsible for retrieving the datasets' information that are stored in the database and serving them to the client's browser. Since the datasets are treated as JSON

objects both on the client's browser and the database, the extraction from the database and transmission is quite simple.

Create Experiment

This component allows the user to create an experiment. The experiment's type as well as the parameters of the experiment which include the dataset, the number of samples for each target class, the subsets of features to be used, the processing options as well as the algorithms and their parameters.

Experiment Initializer

Similarly with the dataset initializer, the experiment initializer will act as a monitor for the experiment's process. More specifically, when an experiment is submitted the monitor process will fork and create the corresponding process based on the requested experiment type. For example, if the user submits a constant feature selection experiment the monitor process forks into the process that can process this type of experiment and starts monitoring it.

The monitor will update the database periodically about the RAM and CPU usage of the experiment. Furthermore, it will check if the user requested the experiment's termination where it terminates the experiment process.

Terminate Experiment

This component is responsible for terminating an ongoing experiment. When the user requests to terminate an experiment, the server will update the status of the experiment in the database as terminated, so the monitor of the experiment can see that and proceed to terminate the experiment process.

Remove Experiment

This component is responsible for removing the requested experiment. When such an operation is requested, the server will remove the entry that corresponds to the experiment

to be removed. There is not any reverting mechanism for recovering deleted experiments, so this component should be used carefully.

Retrieve Experiments

This component retrieves all the experiments from the database in a sorted manner. Through the website the client can request to view the experiments in a sorted manner based on any of the characteristics of the experiment.

My Experiments

▲ id	▲ Name	▲ Dataset	▲ Type	▲ Date	▲ Duration	▲ Memory Usage	▲ CPU Usage	▲ State
6415a7e2426fc	exp_8209	CIC-DDoS2019	Constant Features Selection	3/18/2023, 2:00:34 PM	8 minutes	0.00%	0.00%	Completed

Figure 4.6: Experiments Table Header

Figure 4.6 shows the header of the experiments' table as well as one entry. By default, when the page is loaded, the experiments are retrieved based on the id column which is directly correlated with the Date of submission as the unique id corresponds to the data of submission in milliseconds. That means, by default the experiments are listed from the most recent to the oldest one.

When a user clicks on the header of any column the experiments are retrieved based on that column. The order of the sorting alternates between ascending and descending where the down triangle indicates descending order and the upward triangle ascending order.

Retrieve Algorithms

This component is responsible for retrieving the information for each supported algorithm from the database and serving it on the client's browser. The component is used on page load, but the user cannot directly request it.

Constant Features Selection

This component is responsible for handling constant feature selection experiments. Firstly, it loads the dataset from the file system by reading chunks of 1 GB to avoid memory errors and

based on the user's specified configurations (number of samples per target class) it creates a pandas Dataframe.

Firstly, it applies the preprocessing options on the dataframe. Then it calculates the dataset's basic information ('describe' method) and appends it on the database along with the number of samples for each target class. Then, for each features subsets specified, it encodes all the string variables to avoid any problems with the processing.

Finally, using the VarianceThreshold [21] class from the sklearn library, it calculates the constant features of the dataset, appends them to the database entry for the experiment and repeats the process with the next subset.

Correlated Features Selection

This component is responsible for handling correlated feature selection experiments. Firstly, it loads the dataset from the file system by reading chunks of 1 GB to avoid memory errors and based on the user's specified configurations (number of samples per target class) it creates a panda Dataframe.

Firstly, it applies the preprocessing options on the dataframe. Then it calculates the dataset's basic information ('describe' method) and appends it on the database along with the number of samples for each target class.

Then, for each subset and algorithm combination specified, it encodes all the string variables to avoid any problems with the processing. Then, it runs the corr method [22] from the Pandas library [27] with the specified algorithm and gets the correlation matrix as a result. It processes the matrix and stores for each feature, all the other features that their correlation threshold is greater than the specified one, in a list. Finally, it appends the resulting list to the database and repeats the process with the next algorithm or subset specified.

K-Best Features Selection

This component is responsible for handling k-best feature selection experiments. Firstly, just like the previous components, it loads the dataset from the file system by reading chunks of

1 GB to avoid memory errors and based on the user's specified configurations (number of samples per target class) it creates a pandas Dataframe.

Firstly, it applies the preprocessing options on the dataframe. Then it calculates the dataset's basic information ('describe' method) and appends it on the database along with the number of samples for each target class.

Then, for each subset and algorithm combination specified, it encodes all the string variables to avoid any problems with the processing. Then, it runs the SelectKBest method [25] from the Pandas library with the specified algorithm and gets a list with the best features' indices as the result. Then it processes the resulting list and converts the indices to the corresponding features names. Finally, it appends the final list to the database and repeats the process with the next algorithm or subset specified.

Model Training

This component is responsible for handling model training experiments. Firstly, just like the previous components, it loads the dataset from the file system by reading chunks of 1 GB to avoid memory errors and based on the user's specified configurations (number of samples per target class) it creates a pandas Dataframe.

Firstly, it applies the preprocessing options on the dataframe. Then it calculates the dataset's basic information ('describe' method) and appends it on the database along with the number of samples for each target class.

Then, for each subset and algorithm combination specified, it encodes all the string variables to avoid any problems with the processing. Then it runs the algorithm (if it is supported) with the specified hyperparameters using the cross_validation [30] method of the sklearn library [26] and appends the results on the database.

To avoid data leakage and for simplicity, if specified in the preprocessing options, the standard scaler ([0,1] scale) is pipelined into the model and the scaling will be handled by the cross_validation method before training.

The supported algorithms of the platform are Random Forest, Decision Trees, Extreme Gradient Boost, Linear Discriminant Analysis and K-Nearest Neighbors.

Model Tuning

This component is responsible for handling model tuning experiments. Firstly, just like the previous components, it loads the dataset from the file system by reading chunks of 1 GB to avoid memory errors and based on the user's specified configurations (number of samples per target class) it creates a panda Dataframe.

Firstly, it applies the preprocessing options on the dataframe. Then it calculates the dataset's basic information ('describe' method) and appends it on the database along with the number of samples for each target class.

Then, for each subset and algorithm combination specified, it encodes all the string variables to avoid any problems with the processing. Then, it uses the GridSearchCV [37] method of the sklearn library with the hyperparameter ranges specified by the user to tune the model. Finally, it appends the tuning results to the database.

The user can see the results (training and validation accuracy) for each hyperparameter combination in a sorted manner with the most accurate set to be first and highlighted.

4.3 Deployment

In this section I provide a detailed guide on how this system can be deployed on a server. The guide will be based on the operating system I used which is windows 10.

4.3.1 Web Server

I used XAMP 8.0(<https://www.apachefriends.org/download.html>) which comes with PHP 8 and Apache web server pre-installed. The public directory for the website's files is located at the root directory of the xamp installation in the htdocs folder. So, after placing the website's files in this directory they should be accessible through the webserver.

4.3.2 Database

The DBMS I used, is MongoDB (v 1.13.0) (<https://www.mongodb.com/docs/manual/>) as well as the MongoDB Compass(<https://www.mongodb.com/try/download/compass>) which is a GUI that allows easier interaction with the database. MongoDB Compass is the equivalent of phpMyAdmin for an SQL DB.

After installation, the database should be running on localhost, on port 27017. There should be a database named MLWebsite with 3 collections named Experiments, Datasets and Algorithms. Experiments and Datasets can be left empty as they will be filled on user's request, but the Algorithms collection should be imported from MongoDB compass using the Algorithms.json file contained in the system's root folder. This collection contains information on the 5 algorithms used for the experimental analysis.

4.3.3 PHP

PHP 8 comes pre-installed with XAMP and the external dependencies needed for the interaction with the mongo database are included in the system's files, but a mongo db extension(<https://pecl.php.net/package/mongodb/1.13.0/windows>)needs to be added in the php configuration. I download the PHP 8.0, thread safe x64 configuration zip file and I copy the php_mongodb.dll file in the xamp/php/ext directory which contains the extension files for php. Then for the extension to activate the line extension=php_mongodb.dll needs to be added in the php.ini file in the xamp/php directory.

Also, the php should be configured to allow the reception of large and many files from the network. The following lines should be added (modified if already existing):

- upload_max_filesize=50000M
- max_file_uploads=30
- post_max_size=50000M

These lines will allow uploading up to 50000Mbytes and up to 30 files per session.

4.3.4 Python

Python 3 (<https://www.python.org/downloads/>) needs to be installed and its installation path should be added (if not added automatically) in the PATH environment variable, so python programs can be started only with the 'python' keyword.

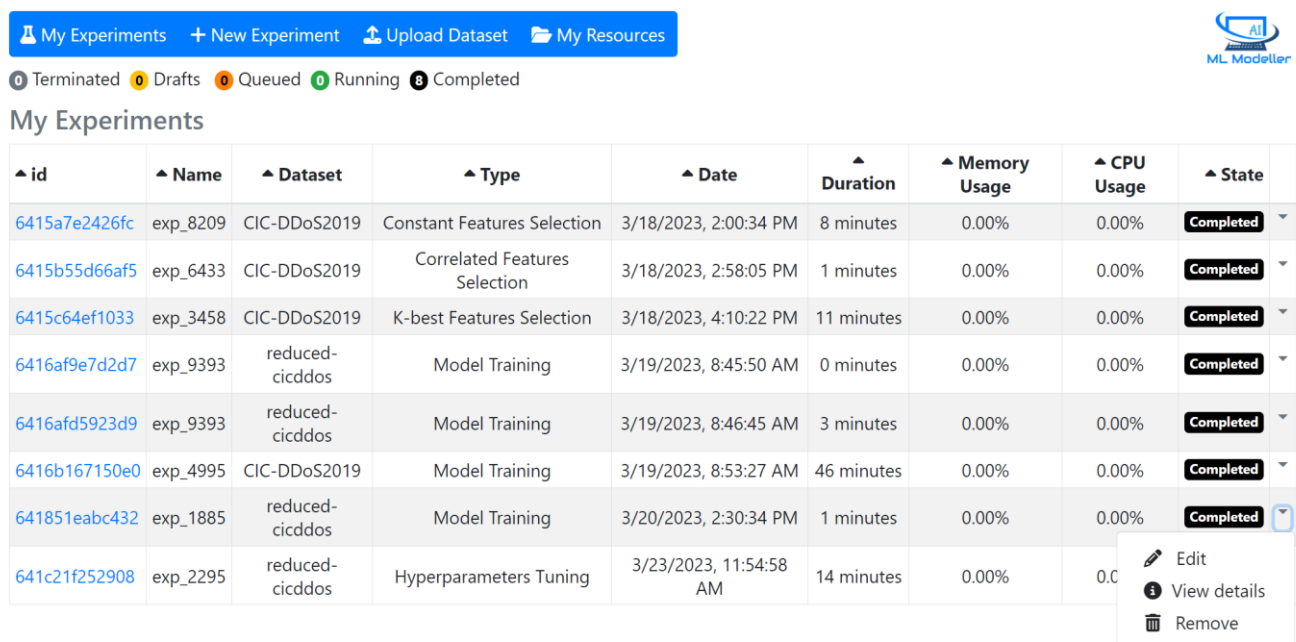
The python libraries used are the following and can be install with the pip tool:

- pip install scikit-learn
- pip install xgboost
- pip install numpy
- pip install pandas
- pip install vaex
- pip install psutil
- pip install pymongo

After all these steps, when the Apache server is started from the XAMP control panel, the platform should be accessible through <http://localhost/ML-Website/>

4.4 System Workflow

In this section, I provide a workflow of the system and demonstrate how a data analyst can use it to perform several operations and benefit from it.



id	Name	Dataset	Type	Date	Duration	Memory Usage	CPU Usage	State
6415a7e2426fc	exp_8209	CIC-DDoS2019	Constant Features Selection	3/18/2023, 2:00:34 PM	8 minutes	0.00%	0.00%	Completed
6415b55d66af5	exp_6433	CIC-DDoS2019	Correlated Features Selection	3/18/2023, 2:58:05 PM	1 minutes	0.00%	0.00%	Completed
6415c64ef1033	exp_3458	CIC-DDoS2019	K-best Features Selection	3/18/2023, 4:10:22 PM	11 minutes	0.00%	0.00%	Completed
6416af9e7d2d7	exp_9393	reduced-cicddos	Model Training	3/19/2023, 8:45:50 AM	0 minutes	0.00%	0.00%	Completed
6416afd5923d9	exp_9393	reduced-cicddos	Model Training	3/19/2023, 8:46:45 AM	3 minutes	0.00%	0.00%	Completed
6416b167150e0	exp_4995	CIC-DDoS2019	Model Training	3/19/2023, 8:53:27 AM	46 minutes	0.00%	0.00%	Completed
641851eabc432	exp_1885	reduced-cicddos	Model Training	3/20/2023, 2:30:34 PM	1 minutes	0.00%	0.00%	Completed
641c21f252908	exp_2295	reduced-cicddos	Hyperparameters Tuning	3/23/2023, 11:54:58 AM	14 minutes	0.00%	0.00%	Completed

Figure 4.7: Main Experiments Page

Figure 4.7 shows the main page that retrieves the experiments from the database and displays their characteristics on a table. On top, there are the main functionalities supported which include viewing and managing the experiments, creating a new experiment, uploading a dataset and viewing/managing the uploaded datasets.

Below the main buttons, are displayed the number of terminated, drafted, queued, running and completed experiments.

For each experiment, there are certain operations supported as shown by the open popup including, removing, editing and viewing the results of an experiment. For running experiments there is the additional option to terminate it which sends a signal to the server to terminate the experiment's process.

4.4.1 Dataset Upload

In this section, I demonstrate a sample workflow for uploading a dataset on the server.

My Experiments + New Experiment Upload Dataset My Resources

ML Modeller

Specify the new dataset parameters

Enter the dataset name, type, target column index and select the dataset files .

Dataset name

Type

Target column index

Drag and drop .xlsx or .csv files here, or click to select files

6 files selected: [clear all](#)

- DrDoS_DNS.csv | text/csv | 1.99 GB
- DrDoS_LDAP.csv | text/csv | 874.81 MB
- DrDoS_MSSQL.csv | text/csv | 1.76 GB
- DrDoS_SNMP.csv | text/csv | 2.02 GB
- DrDoS_SSDP.csv | text/csv | 1.17 GB
- DrDoS_UDP.csv | text/csv | 1.40 GB

Upload

Figure 4.8: Upload Dataset Page

Figure 4.8 shows how the upload dataset page looks after filling the required input fields and setting 6 files to be uploaded for the dataset. The input files will be merged upon processing and the target column set to -1 indicates the target column will be the last column of each file. For each selected file, the name, type and size are shown and can be removed by using the trashcan button.

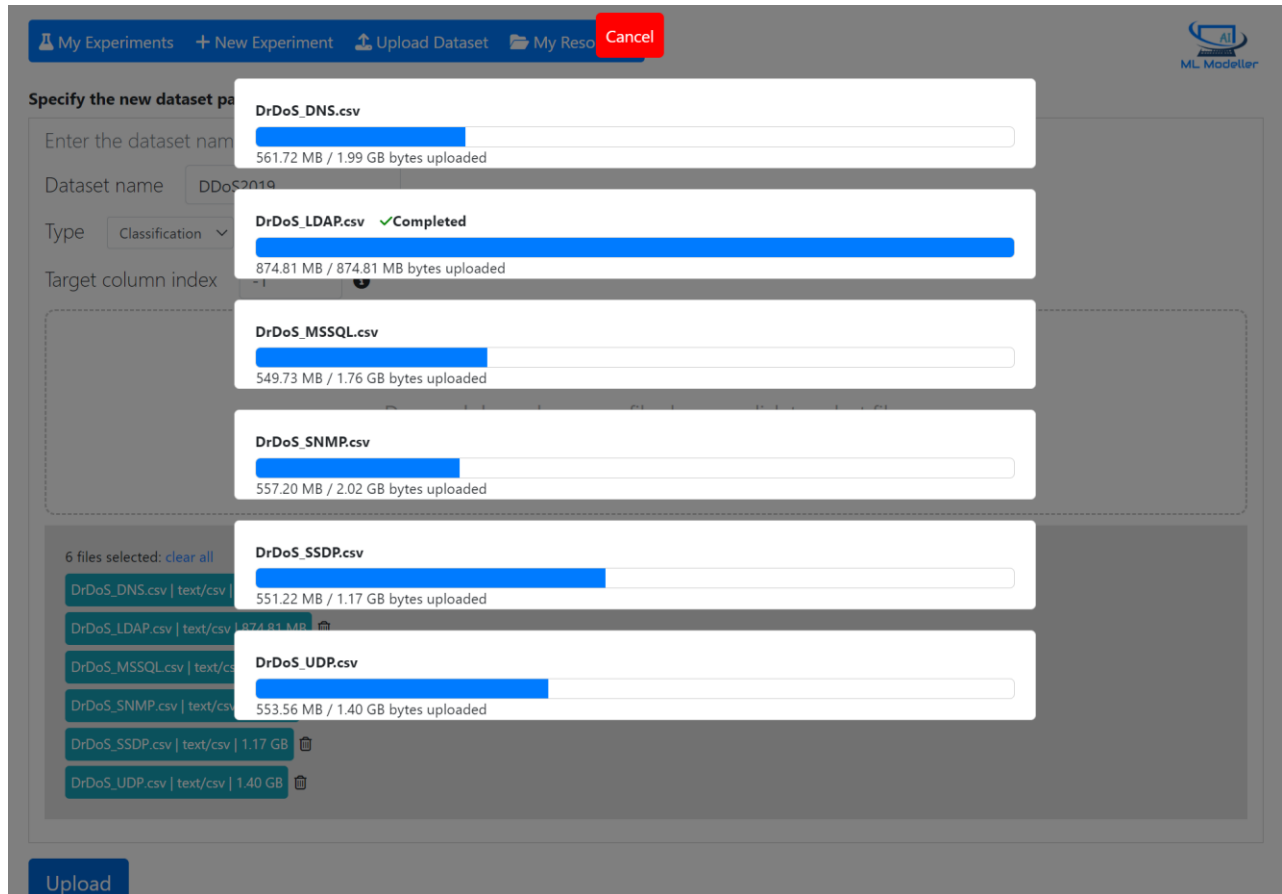


Figure 4.9: Upload Dataset Progress Page

Figure 4.9 displays the page after the uploading has been started by the user. It shows the upload progress for each file. The operation can be reverted any time using the cancel button.

id	Name	Type	Date	Duration	Memory Usage	CPU Usage	State
64159a077dd09	CIC-DDoS2019	Classification	3/18/2023, 1:01:27 PM	4 minutes	0.00%	0.00%	Initialized
6415a67902659	reduced-cicddos	Classification	3/18/2023, 1:54:32 PM	0 minutes	0.00%	0.00%	Initialized
6446749811d18	DDoS2019	Classification	4/24/2023, 3:22:47 PM	0 minutes	2.37%	3.00%	Initializing

Figure 4.10: Initializing Dataset Page

After the files are successfully uploaded on the server, the dataset initialization starts and the user is redirected to 'My Resources' page where he can manage the datasets uploaded on the server as well as view the progress of the initialization.

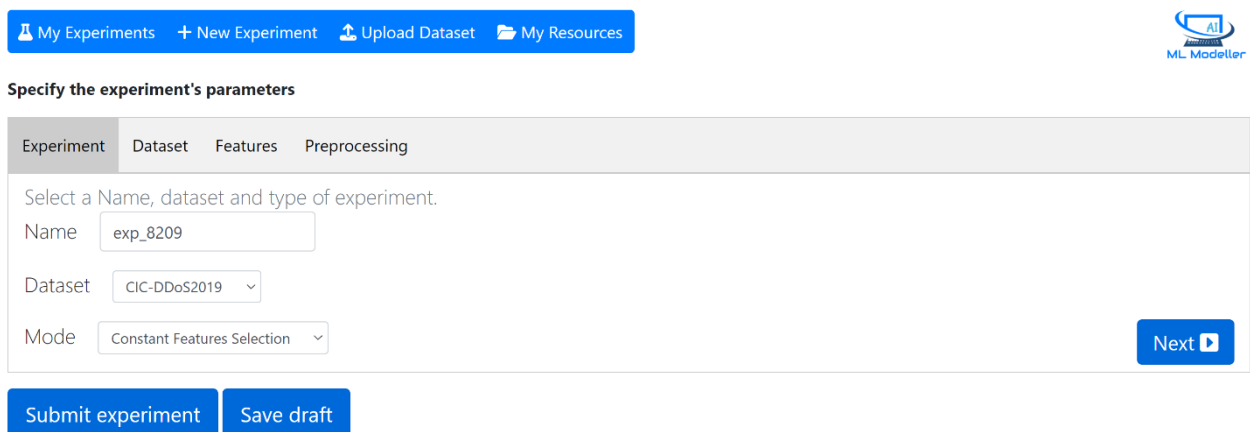
When the status of the dataset is set to Initialized, it can be used for processing. The creation of experiments will be demonstrated with examples in the next subchapters.

4.4.2 Feature Selection

In this section I will demonstrate how the platform can be used for the creation of experiments involving feature selection. Such operations include constant feature, correlated feature and k-best feature selection.

Constant Features

This section demonstrates how a constant feature selection experiment can be created.



The screenshot displays the 'ML Modeller' web interface. At the top, a navigation bar includes links for 'My Experiments', '+ New Experiment', 'Upload Dataset', and 'My Resources'. Below this, a section titled 'Specify the experiment's parameters' contains a tabbed interface with 'Experiment', 'Dataset', 'Features', and 'Preprocessing' tabs. The 'Experiment' tab is active, showing a form with the following fields: 'Name' (text input with 'exp_8209'), 'Dataset' (dropdown menu with 'CIC-DDoS2019' selected), and 'Mode' (dropdown menu with 'Constant Features Selection' selected). A 'Next' button is located to the right of the 'Mode' field. At the bottom of the form, there are two buttons: 'Submit experiment' and 'Save draft'.

Figure 4.11: Experiment Details Tab

Figure 4.11 shows the initial tab of the experiment creation page. The user can fill in the name of the experiment, select the dataset and the kind of experiment which in this case is a constant feature selection.

Figure 4.12 displays the next tab that allows the user to enter the number of samples per target class. For this example, all samples are selected from the BENIGN traffic, 50,000 random samples from the NetBios DDoS and 50,000 from the MSSQL DDoS.

My Experiments
New Experiment
Upload Dataset
My Resources

ML Modeller

Specify the experiment's parameters

Experiment
Dataset
Features
Preprocessing

Select target class and quantity.

CIC-DDoS2019
DrDoS_SNMP
10000
Add to experiment

12 configurations selected: clear all

CIC-DDoS2019 | DrDoS_NetBIOS | 100000
CIC-DDoS2019 | UDP-Iag | 100000
CIC-DDoS2019 | WebDDoS | 100000
CIC-DDoS2019 | Syn | 100000
CIC-DDoS2019 | DrDoS_NTP | 100000
CIC-DDoS2019 | DrDoS_SNMP | 100000
CIC-DDoS2019 | DrDoS_SSDP | 100000
CIC-DDoS2019 | BENIGN | 100000
CIC-DDoS2019 | DrDoS_LDAP | 100000
CIC-DDoS2019 | DrDoS_MSSQL | 100000
CIC-DDoS2019 | DrDoS_UDP | 100000
CIC-DDoS2019 | DrDoS_DNS | 100000

Next

Submit experiment
Save draft

Figure 4.12: Dataset Details Tab

My Experiments
New Experiment
Upload Dataset
My Resources

ML Modeller

Specify the experiment's parameters

Experiment
Dataset
Features
Preprocessing

Select the feature subsets.

86 features selected: clear all select all

Add to experiment

Flow ID

Total Backward Packets

Bwd Packet Length Mean

Fwd IAT Mean

Fwd PSH Flags

Max Packet Length

URG Flag Count

Fwd Avg Packets/Bulk

Init_Win_bytes_forward

Idle Std

Source IP

Total Length of Fwd Packets

Bwd Packet Length Std

Fwd IAT Std

Bwd PSH Flags

CWE Flag Count

Fwd Avg Bulk Rate

Init_Win_bytes_backward

Idle Max

Source Port

Total Length of Bwd Packets

Flow Bytes/s

Fwd IAT Max

Fwd URG Flags

Packet Length Std

ECE Flag Count

Bwd Avg Bytes/Bulk

act_data_pkt_fwd

Idle Min

Destination IP

Fwd Packet Length Max

Flow Packets/s

Fwd IAT Min

Bwd URG Flags

Packet Length Variance

Down/Up Ratio

Bwd Avg Packets/Bulk

min_seg_size_forward

SimilarHTTP

Destination Port

Fwd Packet Length Min

Flow IAT Mean

Bwd IAT Total

Fwd Header Length

FIN Flag Count

Average Packet Size

Bwd Avg Bulk Rate

Active Mean

Inbound

Protocol

Fwd Packet Length Mean

Flow IAT Std

Bwd IAT Mean

Bwd Header Length

SYN Flag Count

Avg Fwd Segment Size

Subflow Fwd Packets

Active Std

Timestamp

Fwd Packet Length Std

Flow IAT Max

Bwd IAT Std

Fwd Packets/s

RST Flag Count

Avg Bwd Segment Size

Subflow Fwd Bytes

Active Max

Flow Duration

Bwd Packet Length Max

Flow IAT Min

Bwd IAT Max

Bwd Packets/s

PSH Flag Count

Fwd Header Length.1

Subflow Bwd Packets

Active Min

Total Fwd Packets

Bwd Packet Length Min

Fwd IAT Total

Bwd IAT Min

Min Packet Length

ACK Flag Count

Fwd Avg Bytes/Bulk

Subflow Bwd Bytes

Idle Mean

1 subsets selected: clear all

86 features = [Flow ID, Source IP, Source Port, Destination IP, Destination Port, Protocol, Timestamp, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min, Fwd IAT Total, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Fwd Header Length, Bwd Header Length, Fwd Packets/s, Bwd Packets/s, Min Packet Length, Max Packet Length, Packet Length Mean, Packet Length Std, Packet Length Variance, FIN Flag Count, SYN Flag Count, RST Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Average Packet Size, Avg Bwd Segment Size, Fwd Header Length.1, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Subflow Fwd Packets, Subflow Fwd Bytes, Subflow Bwd Packets, Subflow Bwd Bytes, Subflow Bwd Bytes, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Idle Min, Idle Max, Idle Min, SimilarHTTP, Inbound]

Next

Submit experiment
Save draft

Figure 4.13: Features Tab

Figure 4.13 displays the features tab which allows the user to enter select the subsets of features to be used. For each subset selected the experiment will be repeated.

Specify the experiment's parameters

Experiment
Dataset
Features
Preprocessing

Select the preprocessing parameters.

☐ Convert negative values to 0
☐ Remove samples containing null values
☐ Scale numerical data to the [0, 1] standard scale.

Submit experiment
Save draft

Figure 4.14: Preprocessing options tab

Figure 4.14 displays the preprocessing options available which include converting the negative values to 0, not using samples with at least 1 null feature and scaling numerical data to the standard [0,1] scale.

My Experiments
+ New Experiment
Upload Dataset
My Resources

0 Terminated
0 Drafts
0 Queued
1 Running
8 Completed

My Experiments

id	Name	Dataset	Type	Date	Duration	Memory Usage	CPU Usage	State
64468305b2e01	constant_features_exp	CIC-DDoS2019	Constant Features Selection	4/24/2023, 4:24:21 PM	0 minutes	4.08%	5.78%	Running
641c21f252908	exp_2295	reduced-cicddos	Hyperparameters Tuning	3/23/2023, 11:54:58 AM	14 minutes	0.00%	0.00%	Completed
641851eabc432	exp_1885	reduced-cicddos	Model Training	3/20/2023, 2:30:34 PM	1 minutes	0.00%	0.00%	Completed
6416b167150e0	exp_4995	CIC-DDoS2019	Model Training	3/19/2023, 8:53:27 AM	46 minutes	0.00%	0.00%	Completed
6416afd5923d9	exp_9393	reduced-cicddos	Model Training	3/19/2023, 8:46:45 AM	3 minutes	0.00%	0.00%	Completed
6416af9e7d2d7	exp_9393	reduced-cicddos	Model Training	3/19/2023, 8:45:50 AM	0 minutes	0.00%	0.00%	Completed
6415c64ef1033	exp_3458	CIC-DDoS2019	K-best Features Selection	3/18/2023, 4:10:22 PM	11 minutes	0.00%	0.00%	Completed
6415b55d66af5	exp_6433	CIC-DDoS2019	Correlated Features Selection	3/18/2023, 2:58:05 PM	1 minutes	0.00%	0.00%	Completed
6415a7e2426fc	exp_8209	CIC-DDoS2019	Constant Features Selection	3/18/2023, 2:00:34 PM	8 minutes	0.00%	0.00%	Completed

Figure 4.15: Experiments Tab

Figure 4.15 shows the newly created experiment when the user submits it. In case he saves it as draft the experiment will still be saved in the database as draft but won't execute.

Experiment #644685a7dea59

Name **exp_8209**
 Type **Constant Features Selection**
 Status **Completed**
 Pid **5528**
 Date submitted **2023-04-24T16:24:21.269Z**

Dataset CIC-DDoS2019

/	Source Port	Destination Port	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bw
count	1031769.00	1031769.00	1031769.00	1031769.00	1031769.00	1031769.00	1031769.00	1031769.00
mean	100485.93	21884.06	32175.95	14.80	2223017.38	10.12	0.29	4096.24
std	142454.22	23653.31	19348.24	4.41	13221323.62	318.71	9.45	21955.05
min	0.00	0.00	0.00	0.00	1.00	1.00	0.00	0.00
25%	12477.00	748.00	15377.00	17.00	1.00	2.00	0.00	458.00
50%	42883.00	6138.00	32232.00	17.00	1.00	2.00	0.00	1160.00
75%	120244.00	45269.00	49049.00	17.00	150.00	2.00	0.00	2944.00
max	805213.00	65532.00	65535.00	17.00	119999992.00	100148.00	4602.00	15266416.00

Figure 4.16: Results Page

Figure 4.16 displays the results page loaded when the view details button is clicked for the constant features selection experiment. There are some basic information about the experiment shown and then information about the dataset discussed previously.

Target class	Samples
BENIGN	31330
DrDoS_DNS	100000
DrDoS_LDAP	100000
DrDoS_MSSQL	100000
DrDoS_NTP	100000
DrDoS_NetBIOS	100000
DrDoS_SNMP	100000
DrDoS_SSDP	100000
DrDoS_UDP	100000
Syn	100000
UDP-lag	100000
WebDDoS	439

Figure 4.17 shows a table also included in the results page that displays the number of samples used per target class. In this case even though the specified sample size for BENIGN traffic was 100,000 there was a maximum of 31,330 BENIGN samples.

Results			
#	Algorithm	Features	
0	Variance Threshold	86 Features	

12 Constant Features
 Bwd PSH Flags
 Fwd URG Flags
 Bwd URG Flags
 FIN Flag Count
 PSH Flag Count
 ECE Flag Count
 Fwd Avg Bytes/Bulk
 Fwd Avg Packets/Bulk
 Fwd Avg Bulk Rate
 Bwd Avg Bytes/Bulk
 Bwd Avg Packets/Bulk
 Bwd Avg Bulk Rate

Export to Excel

Copy Features

Figure 4.18: Results table

Figure 4.17: Class distribution

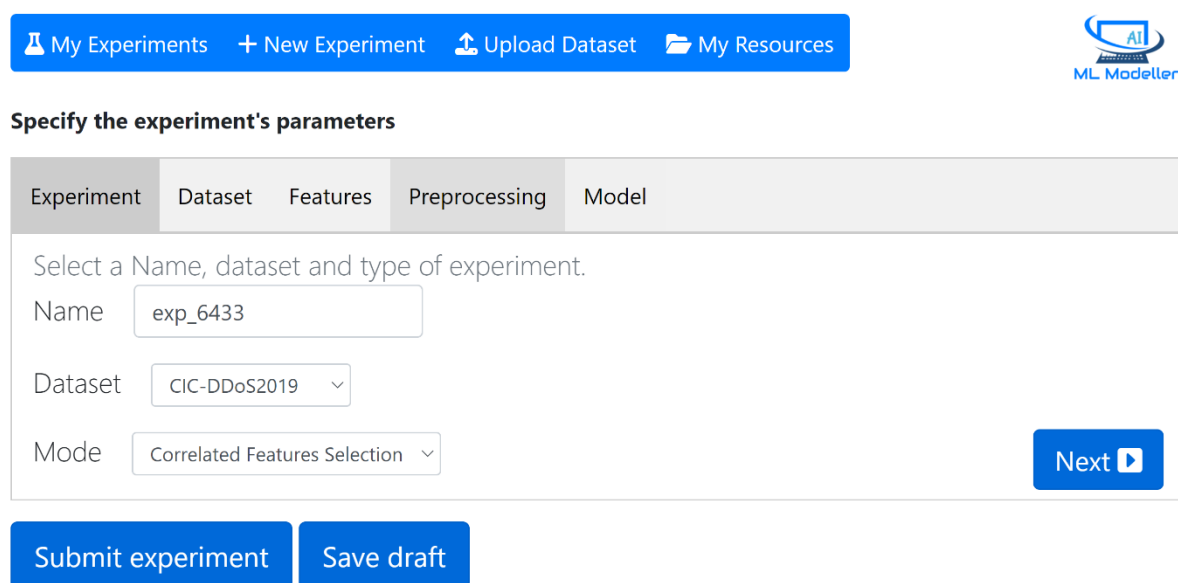
Figure 4.18 shows the results obtained from the experiment. Out of the 86 features subset chosen for the experiment, 12 features remained constant.

The constant features were extracted with variance threshold and they can be copied with the copy button on the far right. When copied, they can then be unselected to not be used in a correlated features selection experiment.

Finally, the results table, class distribution table and dataset information tables can be downloaded as an excel document using the 'Export to Excel' button.

Correlated Features

This section demonstrates how a correlated features experiment can be submitted.



The screenshot shows the ML Modeller interface. At the top, there is a navigation bar with links: 'My Experiments', '+ New Experiment', 'Upload Dataset', and 'My Resources'. To the right of the navigation bar is the 'ML Modeller' logo. Below the navigation bar, the section is titled 'Specify the experiment's parameters'. This section contains a tabbed interface with five tabs: 'Experiment', 'Dataset', 'Features', 'Preprocessing', and 'Model'. The 'Experiment' tab is currently selected. Inside the 'Experiment' tab, there is a form with the following fields: 'Name' (text input with value 'exp_6433'), 'Dataset' (dropdown menu with value 'CIC-DDoS2019'), and 'Mode' (dropdown menu with value 'Correlated Features Selection'). To the right of these fields is a blue 'Next' button with a right arrow icon. Below the 'Specify the experiment's parameters' section, there are two buttons: 'Submit experiment' and 'Save draft'.

Figure 4.19: Correlated Features Experiment

Figure 4.19 shows the initial tab for a correlated features experiment. Notice that for this experiment there is another tab, not included in the previous type of experiment. Dataset, Features and Preprocessing tabs are the same as previously demonstrated, so I will not go over them again.

Figure 4.20 displays the model tab. Here the user can specify which algorithms to test (for each features subset specified) to find correlations between features. Features that are highly correlated do not provide any value to the model and therefore, it is better to exclude them. In the figure, Pearson [23] and Spearman [24] algorithms are used to find features that are at least 95% correlated.

Specify the experiment's parameters

Experiment
Dataset
Features
Preprocessing
Model

Select an algorithm and correlation threshold (0: no correlation, 1: absolute correlation).

Pearson
0.80
+ Add to experiment

2 models selected: [clear all](#)

Pearson | threshold = 0.95
Spearman | threshold = 0.95

Submit experiment
Save draft

Figure 4.20: Correlated Model Tab

Results

#	Algorithm	Correlation Threshold	Features	Correlated Features	Correlations:
0	Pearson	95%	74 Features	26 Correlated Features	1. Fwd Packet Length Min 2. Fwd Pk
1	Spearman	95%	74 Features	39 Correlated Features	1. Fwd Packet Length Min 2. Fwd Pk

Export to Excel

Figure 4.21: Correlated Features Results

Figure 4.21 displays the results of the correlated feature selection experiment. Pearson algorithm detected 26 features with at least 95% correlation with at least one other feature of the dataset and Spearman detected 39 features with at least 95% correlation with at least one other feature of the dataset.

Also, for each one of the features with at least one correlation, it is possible to see which features are the ones that are correlated with. As shown in figure 4.22, Fwd IAT Mean is 95% correlated with Flow IAT Mean and Flow IAT Std, so it might not be useful keeping all 3 of them in the model.

Figure 4.22: Correlated Features' Correlations


Correlations:				
1. Fwd Packet Length Min	2. Fwd Packet Length Mean	3. Flow IAT Std	4. Fwd IAT Total	5. Fwd IAT Mean
1. Fwd Packet Length Min	2. Fwd Packet Length Mean	3. Bwd Packet Length Max	4. Bwd Packet Length Mean	5. Fwd IAT Mean Flow IAT Mean Flow IAT Std

This is an example test case and further details about the features as well as conclusions about the features correlations of cyber-attacks are analyzed in the experimental analysis in the next chapter.

Best Features

This section demonstrates how a best features experiment can be submitted.

[My Experiments](#) [+ New Experiment](#) [Upload Dataset](#) [My Resources](#)



Specify the experiment's parameters

Experiment	Dataset	Features	Preprocessing	Model
Select a Name, dataset and type of experiment.				
Name	<input type="text" value="exp_3458"/>			
Dataset	<input type="text" value="CIC-DDoS2019"/>			
Mode	<input type="text" value="K-best Features Selection"/>			
				Next

[Submit experiment](#) [Save draft](#)

Figure 4.23: Best Features Experiment

Figure 4.23 shows the initial tab for a best features experiment. Similarly, to a correlated features experiment dataset configuration, features subsets, preprocessing options and algorithms can be specified. Dataset, Features and Preprocessing tabs are the same as previously demonstrated, so I will not go over them again.

Figure 4.24 displays the model tab. Here the user can specify which algorithms to test (for each features subset specified) to find the best features for each features' subset specified. The most influential features to the target variable are the most important ones and should increase the accuracy and efficiency of the model. In the figure, ANOVA(F-Test) [35] is used to find the 5-best, 10-best, ..., 50-best features. As mentioned before, chi-2 [34] and mutual information tests [36] are supported and can be used for best features extraction.

My Experiments + New Experiment Upload Dataset My Resources

Specify the experiment's parameters

Experiment Dataset Features Preprocessing Model

Select an algorithm and k for the k best features to be computed.

F-Test 5 + Add to experiment

10 models selected: clear all

- F-Test | k = 5
- F-Test | k = 10
- F-Test | k = 15
- F-Test | k = 20
- F-Test | k = 25
- F-Test | k = 30
- F-Test | k = 35
- F-Test | k = 40
- F-Test | k = 45
- F-Test | k = 50

Submit experiment Save draft

Figure 4.24: Best Features Algorithms Selection

Results

#	Algorithm	K	Features	K-Best Features
0	F-Test	5	47 Features	5 K-Best Features
1	F-Test	5	60 Features	5 K-Best Features
2	F-Test	10	47 Features	10 K-Best Features
3	F-Test	10	60 Features	10 K-Best Features
4	F-Test	15	47 Features	15 K-Best Features
5	F-Test	15	60 Features	15 K-Best Features
6	F-Test	20	47 Features	20 K-Best Features
7	F-Test	20	60 Features	20 K-Best Features
8	F-Test	25	47 Features	25 K-Best Features
9	F-Test	25	60 Features	25 K-Best Features
10	F-Test	30	47 Features	30 K-Best Features
11	F-Test	30	60 Features	30 K-Best Features
12	F-Test	35	47 Features	35 K-Best Features
13	F-Test	35	60 Features	35 K-Best Features
14	F-Test	40	47 Features	40 K-Best Features
15	F-Test	40	60 Features	40 K-Best Features
16	F-Test	45	47 Features	45 K-Best Features
17	F-Test	45	60 Features	45 K-Best Features
18	F-Test	50	47 Features	50 K-Best Features
19	F-Test	50	60 Features	50 K-Best Features

Export to Excel

Figure 4.25: Best Features Results

Figure 4.25 displays the results of the best feature selection experiment. There were 2 features subsets (with 60 and 47 features) specified and for each of the 2 subsets, the 5-best, ..., 50-best features were calculated. The best features can be copied using the right-most button and be pasted in a model-training or tuning experiment very easily.

4.4.3 Model Training and Tuning

This section demonstrates how a correlated features experiment can be submitted.

My Experiments + New Experiment Upload Dataset My Resources

Specify the experiment's parameters

Experiment Dataset Features Preprocessing Model

Select a Name, dataset and type of experiment.

Name exp_7100

Dataset reduced-cicddos

Mode Model Training

Next

Submit

Figure 4.26: Model Training Experiment

Figure 4.26 shows the initial tab for a model training experiment. The user can select that he wants a model training or tuning experiment and just like the previously described experiments, continue with the rest parameters. Dataset, Features and Preprocessing tabs are the same as previously demonstrated, so I will not go over them again.

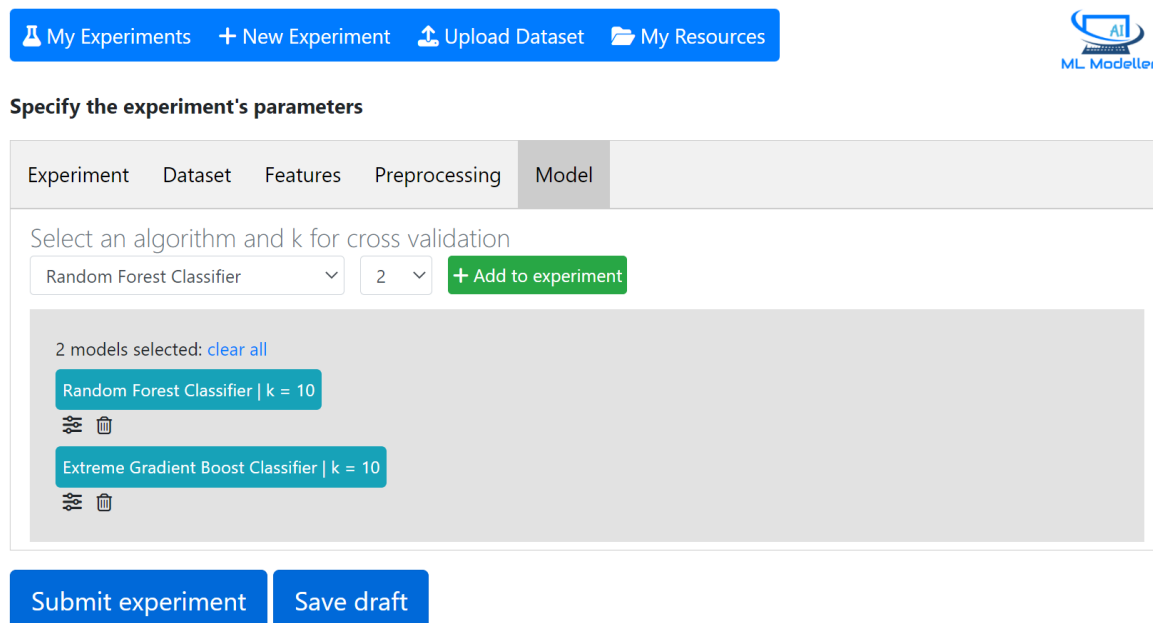


Figure 4.27: Model Training Algorithms

Figure 4.27 displays the model tab. Here the user can specify which training algorithms to test (for each features subset specified) to compare accuracies. In the figure, Random Forest and Extreme Gradient Boost Classifiers are used with 10-kfold cross validation for more accurate results. The user can remove an algorithm using the trash can icon and select the hyperparameters to be used for each algorithm. By default, the default hyperparameters for each algorithm are preselected.

Figure 4.28 shows the popped-up area the user can use to specify the hyperparameters to be used for an algorithm. The information for each algorithm is retrieved from the database to be used for this popup. The user can type in the value of each hyperparameter or use the increase and decrease buttons to do the same thing. The info icon shows information about each hyperparameter hinting to the user on how to use it. The user can then save the hyperparameters and submit the experiment.

Figure 4.29 Shows how the results of the model training look like for each of the 2 subsets specified and the 2 algorithms.

Enter the hyper-parameters for Random Forest Classifier | k = 10

1. n_estimators (default value = 100) ⓘ

☐ Use Default Value

2. criterion (default value = gini)
☒ gini ☐ entropy
☐ Use Default Value

3. max_depth (default value = None) ⓘ

☐ Use Default Value

4. min_samples_split (default value = 2) ⓘ

The maximum depth of each decision tree in the forest. A smaller value can prevent overfitting, but too small a value can lead to underfitting.

Save

Figure 4.28: Algorithm Hyperparameters Selection

Results

#	Algorithm	K-Fold	Features	Training Duration	Training Accuracy	Training Precision	Training Recall	Training F1	Validation Accuracy	Validation Precision	Validation Recall
0	RFC	10	5 Features	8 minutes	Mean = 73.03%	Mean = 78.62%	Mean = 73.03%	Mean = 72.37%	Mean = 63.34%	Mean = 67.60%	Mean = 63.34%
1	RFC	10	10 Features	10 minutes	Mean = 86.01%	Mean = 89.76%	Mean = 86.01%	Mean = 85.58%	Mean = 75.38%	Mean = 78.10%	Mean = 75.38%
2	XGBC	10	5 Features	2 hours, 3 minutes	Mean = 71.70%	Mean = 74.95%	Mean = 71.70%	Mean = 70.83%	Mean = 67.17%	Mean = 69.75%	Mean = 67.17%

Export to Excel

Figure 4.29: Model Training Results

The experiment, that its results is shown in the last figure, is still in progress and the last training with XGBC and 10 feature subset is not shown as it is still running. As shown from the figure the training duration, accuracy, precision, recall and F1 average scores for both training and validation splits from the 10 repetitions(10-fold).

Hyperparameter Tuning is the last operation supported by the platform. The experiment creation is almost identical to the model training, but instead of setting the hyperparameters for each algorithm, the user will set the range of values for each hyperparameters to try, as well as the increment. As shown in figure 4.30, for a Random Forest Classifier a range of 100 to 200 with an increment of 10 will be used for n_estimators ([100, 110, ...,200]) and both 'gini' and 'entropy' values for the criterion hyperparameter.

Enter the hyper-parameter ranges for Random Forest Classifier | k = 10

1. n_estimators (default value = 100) ⓘ

Range

Increment **Executions**

2. criterion (default value = gini)

☒ gini ☐ entropy

Executions

3. max_depth (default value = None) ⓘ

Save

Figure 4.30: Hyperparameter Tuning Experiment

Note that the tuning will include every possible combination of the specified ranges and will be repeated 10 times (10-fold) to increase the accuracy of the results.

Results

Algorithm	K-Fold	Features	Mean Training Duration	Mean Training Accuracy	Mean Validation Accuracy	Ranking	Hyperparameters:			
Random Forest Classifier	10	5 Features	38.03 seconds	61.67161975%	61.62156516%	1	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	29.71 seconds	61.58195875%	61.57534771999999%	2	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	36.54 seconds	61.57116527999999%	61.54792065%	3	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	35.20 seconds	61.57238332%	61.54557711%	4	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	19.26 seconds	61.56602874%	61.53852560000001%	5	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	23.93 seconds	61.49238606%	61.48916164%	6	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	24.80 seconds	61.50030754%	61.470358520000005%	7	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	27.22 seconds	61.42788162%	61.395944%	8	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	31.70 seconds	61.41691469%	61.394363350000006%	9	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	32.11 seconds	61.42857797%	61.38262566%	10	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	20.25 seconds	61.42831751%	61.34188030999999%	11	bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	14.89 seconds	61.213655%	61.22201014999999%	12	bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	25.88 seconds	61.22105418%	61.21574381%	13	bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	24.21 seconds	61.21940023%	61.18127168%	14	bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	12.93 seconds	61.21409025%	61.17187029999999%	15	bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	16.22 seconds	61.215917610000005%	61.15620755%	16	bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3
Random Forest Classifier	10	5 Features	17.93 seconds	61.13853118%	61.096653900000005%	17	bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3

Figure 4.31: Hyperparameter Tuning Results

Figure 4.31 shows how the results of a tuning experiment look like. For each Algorithm-Subset combination there is a table with all hyperparameter combinations tried, the mean duration of the 10 iterations the mean training accuracy, mean validation accuracy and the ranking of

the combination according to the validation accuracy. The results are sorted with the most accurate combination coming first.

Hyperparameters:

bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 200	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 150	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 180	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 190	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 100	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 120	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 130	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 140	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 170	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 160	oob_score = false
bootstrap = true	criterion = entropy	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 110	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 110	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 200	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 180	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 100	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 120	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 130	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 160	oob_score = false
bootstrap = true	criterion = gini	max_depth = 5	max_features = 0.3	min_samples_leaf = 1	min_samples_split = 2	n_estimators = 190	oob_score = false

Figure 4.32: Hyperparameter Tuning Results Hyperparameters

The tuning in the example only included 'n_estimators' and criterion ranges and as indicated by figure 4.32, when 'entropy' is used instead of 'gini', it leads to higher accuracy.

This is the last operation I implemented for the platform. The next chapter includes an experimental analysis of cyber-attack traffic mainly DDoS, using the platform to automate the process.

Chapter 5

System Evaluation

5.1.1 Datasets.....	60
5.1.1.1 CIC-DDoS2019.....	60
5.1.1.2 CIC-IDS2017.....	61
5.1.2 Methodology.....	62
5.1.3 Experiments & Results.....	64

5.1.1 Datasets

I used 2 cyber-attack related datasets to analyze and use to evaluate my system. Both datasets contain network related features and were generated with CICFlowMeter [9], a particularly useful dataset generation tool that takes the .pcap files, containing the captured network traffic of a system and generates excel files that can be further analyzed by data analysts.

5.1.1.1 CIC-DDoS2019

The CIC-DDoS2019 [3] is a publicly available dataset of network traffic captures generated by the Canadian Institute for Cybersecurity, designed for evaluating and analyzing the effectiveness of DDoS attack detection and mitigation methods.

The dataset contains both benign and malicious traffic captures. The benign traffic is from a variety of sources, such as web browsing, email, and file transfers. The malicious traffic consists of various DDoS attacks, such as UDP flood, TCP flood, and HTTP flood attacks. The dataset was collected in a controlled environment using a custom-built network emulator that simulates a real-world network environment.

The dataset contains over 80 million records of network traffic with over 20 GB of generated traffic in .csv format. It includes features extracted from the captured traffic, such as flow duration, packet length, and protocol type, which can be used to train machine learning models for DDoS detection and mitigation.

The DDoS traffic types analyzed that are included in the dataset besides normal(benign) traffic and are mentioned in the background section are the following:

- NetBIOS DDoS
- UDP-Lag DDoS
- Syn DDoS
- NTP DDoS
- SNMP DDoS
- SSDP DDoS
- LDAP DDoS
- MSSQL DDoS
- UDP DDoS
- DNS DDoS

The dataset contains a total of 86 network related features and a target column, specifying the DDoS kind (or benign for normal traffic) that are briefly described in Appendix A

5.1.1.2 CIC-IDS2017

CIC-IDS2017 [4] is similar in terms of structure to CIC-DDoS2019. It is a publicly available dataset created by the Canadian Institute for Cybersecurity containing a set of network traffic captures designed for evaluating and analyzing the effectiveness of intrusion detection systems (IDSs).

The dataset contains both benign and malicious traffic captures. The benign traffic is from a variety of sources, such as web browsing, email, and file transfers. Malicious traffic consists of several types of attacks, such as DDoS and brute forcing.

The dataset was collected in a controlled environment using a custom-built network emulator that simulates a real-world network environment and it contains more than 1 GB of generated traffic in .csv format.

The dataset was generated with the same tool (CICFlowMeter) as the first dataset and it initially has the same kinds of network features briefly explained in Appendix A.

The traffic types analyzed that are included in the dataset besides normal(benign) are the following:

- DDoS

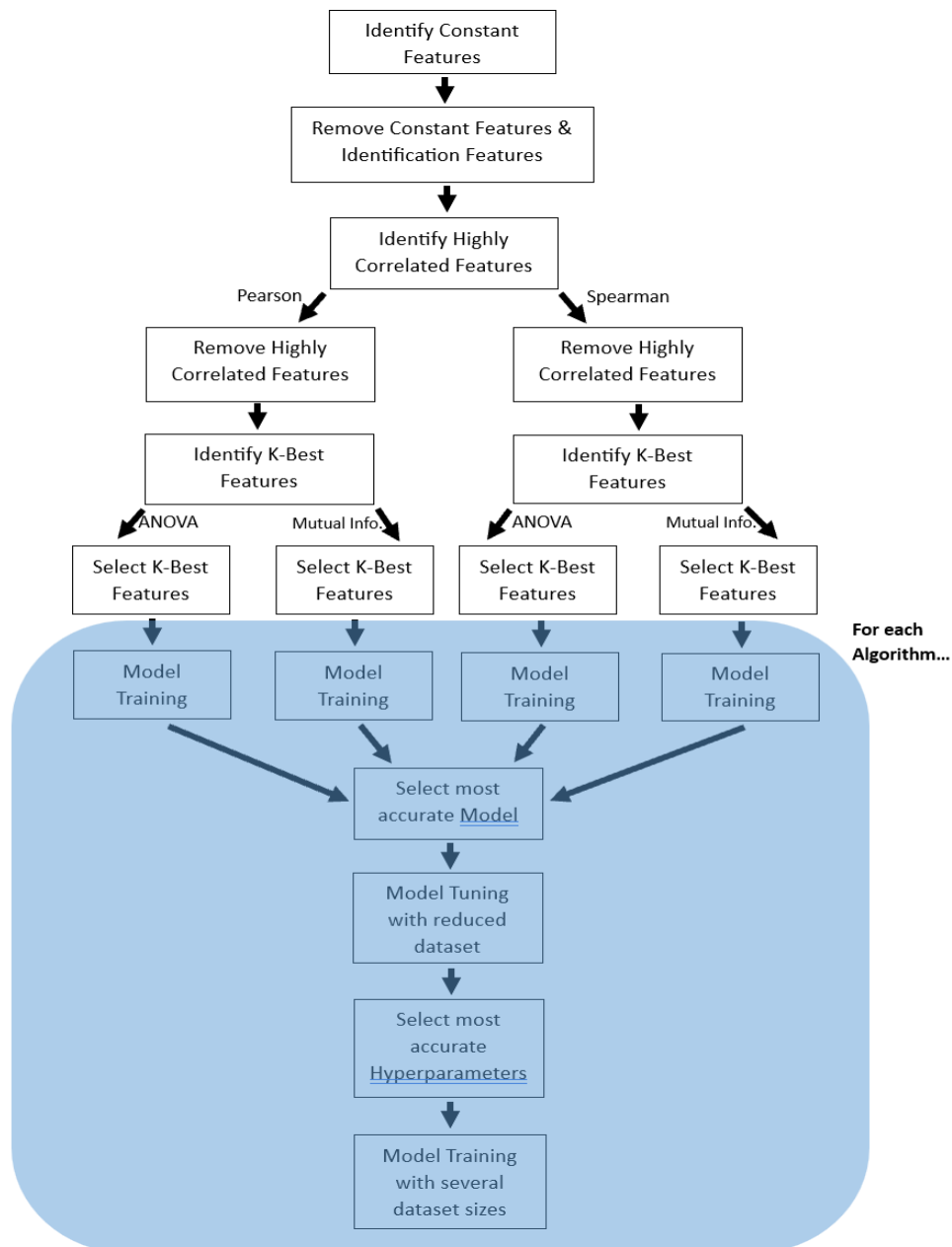
- DoS GoldenEye
- DoS Hulk
- DoS SlowHttpTest
- Dos Slowloris
- FTP-Patator (brute force attack)
- SSH-Patator (brute force attack)
- Port Scan

5.1.2 Methodology

The methodology I have followed for the most part, to analyze each dataset, is demonstrated in figure 5.1.

Firstly, I am identifying the constant features of the datasets and removing them since they don't provide any value to the analysis. Then I remove features that act as identifiers and therefore they can either leak information about the target or be unable to generalize and make predictions in another dataset. Such features are Source/Destination IPs, and timestamps of the attack that cannot be used to make generic conclusions about DDoS traffic. Then I am using Pearson [23] and Spearman [24] algorithms to detect and remove highly correlated features. Such features don't provide any value to the model and removing them can make the training more efficient. Then I am using ANOVA and Mutual Information tests to detect the best of the remaining features of the dataset and for each classification algorithm (XGBC, Random Forest, Decision Trees, LDA, KNN) I am training (with default hyperparameters) to find the most accurate subset. Then I am taking the most accurate (as small as possible) subset of best features and tune each algorithm with a reduced version of the dataset (to finish tuning faster) to find the best hyperparameters. Finally, I use the best hyperparameters to train the model with different dataset sizes and examine the accuracy.

Figure 5.1: Methodology Diagram

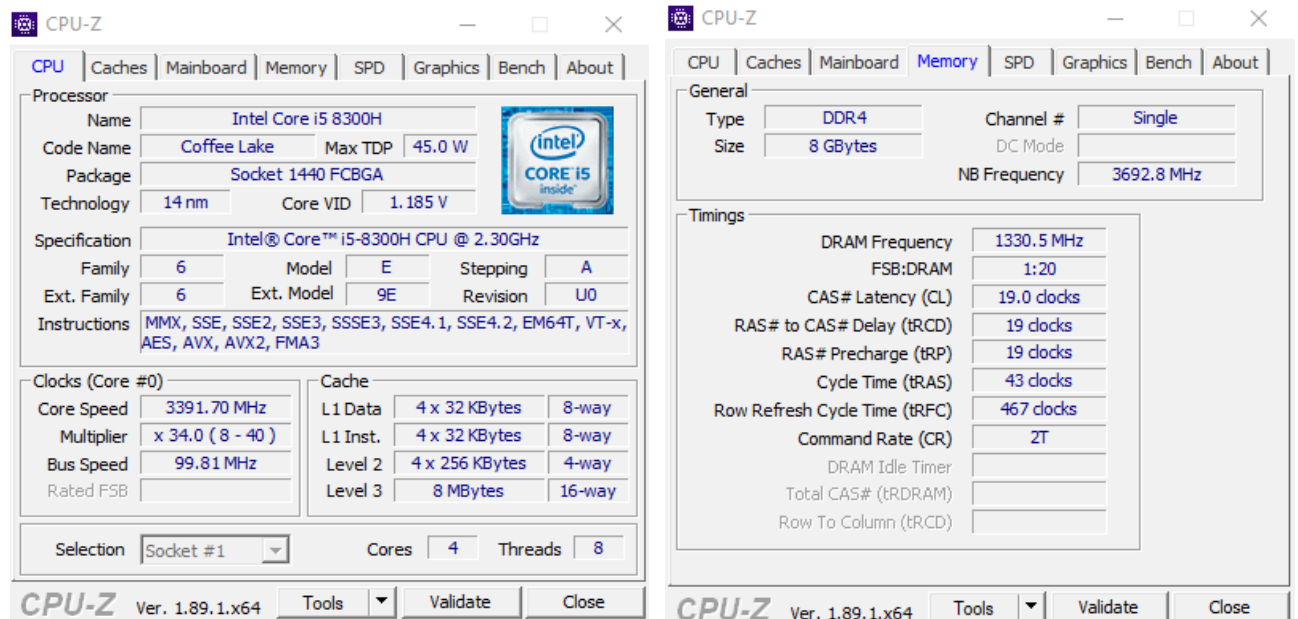


As an initial approach in understanding the first dataset I followed a slightly different approach than the one shown in figure 5.1. First, I removed the constant and identification features as well as the union of the features selected as highly correlated with Pearson and Spearman. Then I used chi-2, f-tests and mutual information tests to extract the best 5 to 25 features. Then I tried several machine learning algorithms to get an idea of the ones that perform better with different dataset sizes.

After having a deeper look at my initial approach, I realized that I made some mistakes. The first mistake is that I didn't use cross-validation to ensure the dataset is thoroughly trained, so my result might have been off. Moreover, I realized that chi-2 is not the best algorithm to use for my dataset as it contained mostly continuous variables (numerical) instead of categorical ones that chi-2 is made for. Even though my initial approach was not fully

accurate, it gave me an idea on which algorithms can perform relatively well to use in my next approach. In Appendix B, in the 'Initial Approach' section I have put the graphs and results extracted from that approach, but I will not go into much detail on these results as I will do so in my updated and better approach.

The following figures display the system I performed all my experiments. Apparently, it is not very efficient, but due to technical difficulties it was my last resort.



The next subchapter includes a detailed description of all my experiments as well as the results I obtained using the proposed system.

5.1.3 Experiments & Results

In this section, I will describe every experiment/operation I conducted to analyze the aforementioned datasets and present my results. For simplicity I will refer to CIC-DDoS2019 dataset as the 1st dataset and the CIC-IDS2017 as the 2nd dataset.

To identify the constant features, I submitted constant feature selection experiments for the 2 datasets. For each attack kind of the 1st dataset, I used 100,000 random samples and about 30,000 samples for the normal traffic, as that was the maximum number of samples in the dataset. For each attack kind of the 2nd dataset, I used 5,000 - 10,000 random samples and 10,000 samples for normal traffic. The yellow highlighted features are features that remained

constant in both datasets, the green ones remained constant only in the 1st one and the blue one only in the second one.

1. Bwd PSH Flags(*)
2. Fwd URG Flags(*)
3. Bwd URG Flags
4. FIN Flag Count
5. PSH Flag Count
6. ECE Flag Count
7. Fwd Avg Bytes/Bulk(*)
8. Fwd Avg Packets/Bulk(*)
9. Fwd Avg Bulk Rate(*)
10. Bwd Avg Bytes/Bulk(*)
11. Bwd Avg Packets/Bulk(*)
12. Bwd Avg Bulk Rate(*)
13. CWE Flag Count

(*) Constant features detected by another research.

According to research conducted related to the 2nd dataset by Arnaud R., Eloïse C., Florent C. and Pascal L. [39], due to miscalculation of features by the CIC-Flowmeter tool, some features are always null. These features are marked with a * in the aforementioned list. I decided to exclude certain attacks included in the 2nd dataset, with small number of samples (<1000) compared to other attacks and their nature couldn't be compared to DDoS attacks. I kept the attacks that essentially overwhelm the server with useless requests. Other than DDoS, there are brute force attacks, port scans as well as DoS attacks generated by certain tools. So, in my case I got 2 extra constant features (CWE Flag Count, Bwd URG Flags) to remove.

In general, I think features remaining constant under specific scenarios and datasets can give insights on how attacks work, but features (PSH, ECE flags etc.) unused in some attacks and setups can be used in others.

So, after removing the constant features of each dataset I decided to remove features that serve identification purposes. Such features include the FlowID which is a string concatenation of other features (flowId = sourceIP + "-" + destinationIP + "-" + srcPort + "-" + dstPort + "-" + protocol), the source and destination IP's as well as the source and destination ports. These features can be unique in any attack scenario/setup and training a model based on them will not be able to generalize and will be biased on IP's and ports that can take any value. Also, the timestamp feature that identifies the time of the flow doesn't provide any

value and can leak information about the target. To sum up, the removed features are the following:

- FlowID
- Source IP
- Destination IP
- Source Port
- Destination Port
- Timestamp

To make my model lighter I ran correlated features experiments to detect features that are highly correlated with other features, therefore keeping all of them is unnecessary for the model. To verify this theory, I later checked the accuracy with all the features compared to the dataset after removing constant and correlated ones.

For each attack kind of the 1st dataset, I used 100,000 random samples and about 30,000 samples for the normal traffic, as that was the maximum number of samples in the dataset. For each attack kind of the 2nd dataset, I used 5,000 - 10,000 random samples and 10,000 samples for normal traffic. To better understand cyber-attacks and detect any correlations with the identification features (source/destination IP's etc.) I included them in the correlated features experiments.

According to the results of the 1st dataset and Pearson algorithm, out of 74 features, 26 features are 95% correlated with at least another feature in the dataset. The following list shows only a few correlations, but you can find the complete one in the 'Highly Correlated Features Lists' section in Appendix B.

1. Fwd Packet Length Min: Fwd Packet Length Max
2. Fwd Packet Length Mean: Fwd Packet Length Max, Fwd Packet Length Min
3. Flow IAT Std: Flow IAT Mean
4. Fwd IAT Total: Flow Duration
5. Fwd IAT Mean: Flow IAT Mean, Flow IAT Std
6. Fwd IAT Std: Flow IAT Mean, Flow IAT Std, Flow IAT Max, Fwd IAT Mean
7. Fwd IAT Max: Flow IAT Max, Fwd IAT Std
8. Fwd IAT Min: Flow IAT Min
9. Bwd IAT Std: Bwd IAT Mean
10. Bwd Header Length: Total Backward Packets

According to the results of the 1st dataset and Spearman algorithm, out of 74 features, 39 features are 95% correlated with at least another feature in the dataset. The following list shows only a few correlations, but you can find the complete one in the 'Highly Correlated Features Lists' section in Appendix B.

1. Fwd Packet Length Min: Fwd Packet Length Max
2. Fwd Packet Length Mean: Fwd Packet Length Max, Fwd Packet Length Min
3. Bwd Packet Length Max: Total Length of Bwd Packets
4. Bwd Packet Length Mean: Total Length of Bwd Packets, Bwd Packet Length Max
5. Flow Packets/s: Flow Duration
6. Flow IAT Mean: Flow Duration, Flow Packets/s
7. Flow IAT Max: Flow Duration, Flow Packets/s, Flow IAT Mean
8. Fwd IAT Mean: Fwd IAT Total
9. Fwd IAT Std: Total Fwd Packets
10. Fwd IAT Max: Fwd IAT Total, Fwd IAT Mean

According to the results of the 2nd dataset and Pearson algorithm, out of 73 features, 22 features are 95% correlated with at least another feature in the dataset. The following list shows only a few correlations, but you can find the complete one in the 'Highly Correlated Features Lists' section in Appendix B.

1. Fwd Packet Length Std: Fwd Packet Length Max
2. Bwd Packet Length Mean: Bwd Packet Length Max
3. Bwd Packet Length Std: Bwd Packet Length Max, Bwd Packet Length Mean
4. Fwd IAT Total: Flow Duration
5. Fwd IAT Max: Flow IAT Max
6. Bwd IAT Max: Bwd IAT Total
7. Fwd Header Length: Total Fwd Packets
8. Bwd Header Length: Total Backward Packets
9. Fwd Packets/s: Flow Packets/s
10. Packet Length Std: Max Packet Length, Packet Length Mean

According to the results of the 2nd dataset and Spearman algorithm, out of 73 features, 30 features are 95% correlated with at least another feature in the dataset. The following list shows only a few correlations, but you can find the complete one in the 'Highly Correlated Features Lists' section in Appendix B.

1. Fwd Packet Length Max: Total Length of Fwd Packets
2. Fwd Packet Length Mean: Total Length of Fwd Packets, Fwd Packet Length Max
3. Bwd Packet Length Max: Total Length of Bwd Packets
4. Bwd Packet Length Mean: Total Length of Bwd Packets, Bwd Packet Length Max
5. Flow Packets/s: Flow Duration
6. Flow IAT Mean: Flow Packets/s
7. Flow IAT Max: Flow Duration, Flow Packets/s, Flow IAT Mean
8. Fwd IAT Mean: Fwd IAT Total
9. Fwd IAT Max: Fwd IAT Total, Fwd IAT Mean
10. Bwd IAT Mean: Bwd IAT Total

Since Spearman can detect more general relationships between variables, the correlated features are typically more than those identified by Pearson. Moreover, the lists mentioned above can create graphs that can better visualize the relationships of the features since correlation has both directions. The double direction is not visible in the lists as I won't to keep at least one instance of the 2 correlated variables. For example, if A: B meaning A is correlated with B, there will not be B: A even if it is valid, as I don't want to exclude both A and B from the following experiments.

For the purposes of this research, I won't go into further research on why these correlations exist in these datasets, but I will analyze the effect of eliminating these features in the creation of a more efficient machine learning model that can detect these attacks.

Moving on, I am left with 2 subsets for each dataset:

Subset 1 = features – constantFeatures - identificationFeatures - pearsonCorrelatedFeatures

Subset 2 = features – constantFeatures - identificationFeatures - spearmanCorrelatedFeatures

For each subset I want to identify those features that are the most important ones. Ideally, I am looking for the least number of features that can train the most accurate model. So in the next step I will use ANOVA and Mutual Information Tests to identify the 5 best up to 50 best features of the subsets.

Target class	Samples
BENIGN	31330
DrDoS_DNS	100000
DrDoS_LDAP	100000
DrDoS_MSSQL	100000
DrDoS_NTP	100000
DrDoS_NetBIOS	100000
DrDoS_SNMP	100000
DrDoS_SSDP	100000
DrDoS_UDP	100000
Syn	100000
UDP-lag	100000

Figure 5.2: Methodology Diagram

As mentioned in the background, network related features are typically continuous and do not fit the assumptions of the Chi-2 test, which assumes that the variables are categorical and have discrete values. Therefore, using it for feature selection in these datasets may not yield accurate results and could lead to the selection of irrelevant features or the exclusion of important ones. Figure 5.2 displays the class distribution for each DDoS class. There was far more DDoS traffic than normal one so I tried keeping it balanced.

The complete lists of the results of the K-Best feature selection on the 1st dataset can be found in the 'Best Features Lists' section in Appendix B since they were large and not individually analyzed.

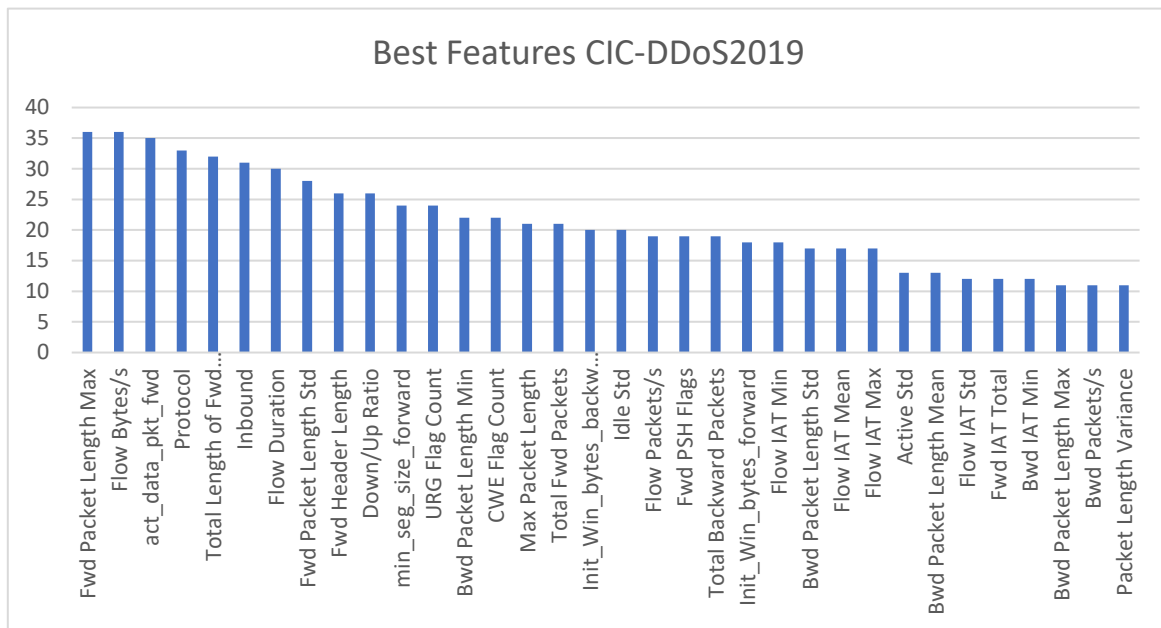


Figure 5.3: Best features histogram

In order to rank the features of the 1st dataset, for every occurrence of each feature in a k-best feature subset calculated by either ANOVA or Mutual information I was giving 1 point to the feature. The final rank is shown in figure 5.3.

The maximum packet length of the incoming(fwd) traffic was ranked as the most important feature. This is because DDoS attacks typically keep the same small packet length to send as many requests as possible. Also, it could be that each attack type might have used a unique upper bound for its packets, therefore this feature can be a good predictor. At the same ranking is the speed of the flow (Bytes/s). DDoS attacks usually have higher and constant flow rates in order to be more effective, so as expected this feature is on the top of the list.

The "act_data_pkt_fwd" feature is a metric that measures the number of actual data packets forwarded by the network in a flow. In other words, it counts the number of incoming packets that contain actual data, rather than control or management information. Several types of DDoS attack traffic will contain a fixed number of data packets or no data packets at all to be lightweight and allow higher transmission rates. In contrast, normal traffic has a higher and more unpredictable number of data packets.

Protocol as expected is another feature that plays significant role, since many DDoS types, abuse services that operate under specific protocols therefore it can be a good predictor on

the DDoS target. If we are talking about DDoS detection instead of DDoS classification though, it could be a misleading feature.

The Total Length of Fwd packets is another important feature as it represents the total number of bytes sent by the source. DDoS attacks normally have a large number of bytes sent, compared to normal traffic. Inbound comes after and is very similar as it represents the number of bytes received by the destination. It looks like those 2 features should have been detected as highly correlated since the number of bytes sent should be the same as the number of bytes received. It is quite common for packet losses to occur, especially when the network gets congested by DDoS traffic or unreliable protocols are used (UDP).

Flow duration is generally higher in DDoS attacks as threat actors want to keep the network as busy as possible and the standard deviation of the number of sent bytes is usually lower in DDoS attacks as this number is closer to the average. That is because DDoS attacks typically generate many small requests with fixed length. In contrast, normal traffic's size patterns are unpredictable and the standard deviation is normally higher.

To better understand each DDoS traffic individually and also be able to compare them with existing research, I additionally extracted the most influential features for each DDoS class mixed with normal traffic.

In their research, Iman S., Arash H. L., Saqib H., and Ali A. G. [3] on the same dataset, have listed the most influential features for each DDoS class. Their results cannot be directly compared to mine since, my results include the most influential feature extracted from the entire dataset, but to be able to rank the features as a total, for every appearance of each feature in the top 5 list of a DDoS class of the research I added a point to that feature.

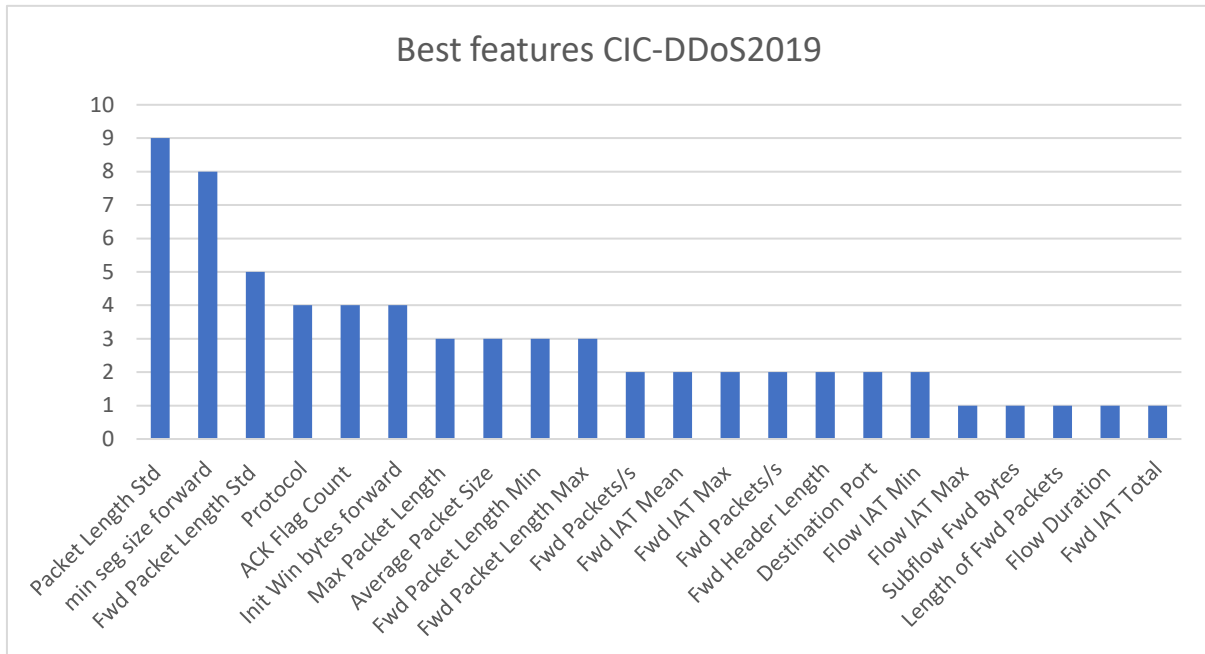


Figure 5.4: Histogram of the best features from another research

Figure 5.4 displays the results of my ranking of the best features of the research. The features appeared mostly is packet length std which represents the standard deviation of the packets size sent/received in the flow. Such a feature as explained previously can be a good indicator of the traffic sent by threat actors since their packets typically have fixed length. According to my analysis on highly correlated features, this feature is at least 95% linearly correlated with the standard deviation of the packets' size forwarded (Fwd Packet Length Std), therefore I decided to exclude the first one.

“Min_seg_size_fwd” refers to the minimum TCP segment's (not the entire packet) size. As shown by the histogram it was an important feature according to the researchers. That's because some types of DDoS attacks use small TCP segment sizes to consume network resources and disrupt legitimate traffic. Since it is ranked with a value of 8, that means there are 8 out of the 12 DDoS attacks that use most probably small TCP segment sizes.

Protocol and ACK Flag count were also prominent features where in my analysis they were deemed highly correlated, so I only kept protocol. I noticed that the maximum value of any flag counter in the dataset was 1. So, it could be either 0 or 1, which I am not sure exactly why as a flow can be described as a sequence of packets with similar characteristics that serve the same purpose. So, I would expect to see a small but not always 1 or 0 count for the network

flags. It could be a miscalculation of CICFlowMeter just like other constant features mentioned previously.

Just for the record, I performed another experiment to identify the best features using ANOVA, but this time including the identification features, that were excluded for the model creation. These features include the FlowID, Source IP, Destination IP, Source port, Destination port and the timestamp.

According to ANOVA, Source IP and Timestamp were in the top 5 features, Source port and FlowID in the top 10 features, Destination IP in the top 15 features and surprisingly destination port was in the top 35 features.

Target class	Samples
BENIGN	10000
DDoS	10000
DoS GoldenEye	10000
DoS Hulk	10000
DoS Slowhttptest	5499
DoS slowloris	5796
FTP-Patator	7935
PortScan	10000
SSH-Patator	5897

Figure 5.5: Class Distribution

Regarding the 2nd dataset, I repeated similar process to identify and rank the best features.

The CIC-IDS2017 dataset contains traffic related to DDoS, DoS generated by certain tools, Brute force against SSH and FTP authentication as well as Port scans. The attacks analyzed are not all identical to DDoS attacks analyzed at the previous dataset, but they share the same nature of overwhelming the server with useless traffic. Figure 5.5 displays the number of samples/flows used per attack type. There was more traffic for DDoS and Benign(normal) traffic, but I tried to keep it balanced.

The results include the calculated best features with ANOVA(f-tests) and mutual information tests for up to the 30-best features and can be found in the 'Best Features Lists' section in Appendix B since they were large and not individually analyzed.

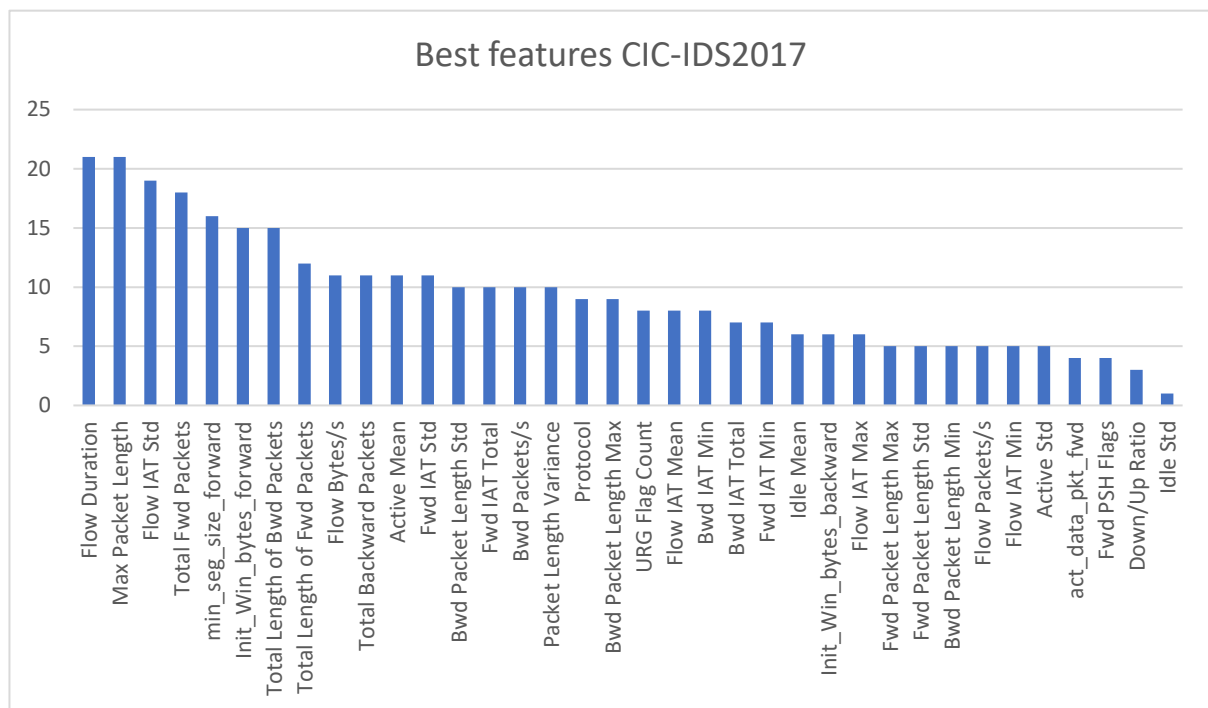


Figure 5.6: Histogram of the best features of CIC-IDS2017 Dataset

Similarly with the 1st dataset I made a ranking for each feature following the same method. Figure 5.6 shows the histogram of the best features of the 2nd dataset. This dataset was similar in nature to the 1st one from the point of view of having enormous amounts of traffic transmitted.

Flow duration sits at the top with similar reasons as with the previous dataset. Threat actors typically keep the channel busy either because they are performing a DDoS attack, a port scan or a Brute force attack.

Max packet length was an equally important feature according to my ranking. This feature refers to the maximum packet size in bytes observed in the flow and was at the top of my ranking of the 1st dataset. This feature as shown by the correlation analysis, is highly correlated with Packet length std, the standard deviation of the packet size. Both Packet length std and Max packet length can tell us information about the traffic transmitted and as we know the traffic of the attacks being analyzed tends to have a more consistent structure, therefore the max packet length tends to remain fixed at lower values for each attack and the standard deviation lower. Compared to normal traffic the maximum packet length tends to be higher and unpredictable.

Flow IAT std is another important feature that represents the time standard deviation of the time between 2 packets sent in the flow. When its value is lower, it means there is traffic sent at constant pace (time between 2 packets closer to the average) which can be a characteristic of some attacks. Sometime threat actors try to make their traffic seem as normal as possible, therefore constant pace doesn't necessarily mean suspicious traffic.

Total Fwd Packets and min_seg_size_fwd are some other key features seen in the previous analysis. since they indicate the amount of traffic sent. It is worth noting that in this dataset protocol is not considered as important as the 1st dataset. That is probably because the traffic classes in this case are not so protocol related, therefore it might not be as useful. Also, 'act_data_pkt_fwd' which indicates the number of actual data packets sent, is not as important as the DDoS specialized dataset. That could be as in this case there are other types of attacks like brute forcing or port scanning that need to use data packets to carry out the attack. Therefore this feature might not be a good predictor.

This concludes my analysis of the best features for cyber-attack datasets. Now it's time to test them and see how they perform on detecting these attacks after being trained with several machine learning algorithms. The classification algorithms I used, are Decision trees, Random Forest, Extreme Gradient Boost, Linear Discriminant Analysis and K-Nearest Neighbors. For each algorithm, I trained a model with several feature's subsets selected with Mutual Information Tests and F-tests (ANOVA) and subsets that have been formed after the removal of highly correlated features selected by Pearson and Spearman algorithms (before k-best selection with Mut. Info and ANOVA).

To reduce the computational costs, just like the previous experiments I used a subset of the 1st dataset shown in figure 5.2 and the 2nd dataset in figure 5.5. Since the 2 datasets describe slightly different attacks, the targets number and total datasets are different, they are not directly comparable. So, each algorithm was trained (with its default hyperparameters) with several features subset of several lengths (5 up to 50) that were obtained using Mutual Info., ANOVA, Spearman and Pearson. The resulted accuracies are then obtained to compare the performance of the algorithms.

To sum up, this evaluation compares 5 different machine learning algorithms, 2 K-Best feature selection algorithms and 2 Highly correlated Features selection algorithms based on the

validation accuracy metric obtained after the training. This process is repeated for the 2nd dataset to verify the results.

Appendix B contains a large number of line graphs that show the Training, Validation accuracies and Training duration for each comparable set of algorithms (Mutual Info vs ANOVA, Pearson vs Spearman, KNN vs LDA vs RFC ...) in relation to the feature subset's length. I decided to keep the main text lighter and exclude these graphs but rather use bar charts that demonstrate the aggregated comparison between each set of algorithms without considering subsets' length.

Figure 5.7 shows a bar chart that summarizes the performance of each algorithm on the 1st dataset (CIC-DDoS2019). The blue bar represents the average of the validation accuracies obtained by subsets selected by Mutual Information tests and the orange bar subsets selected by ANOVA. Therefore, we have a comparison of these 2 algorithms.

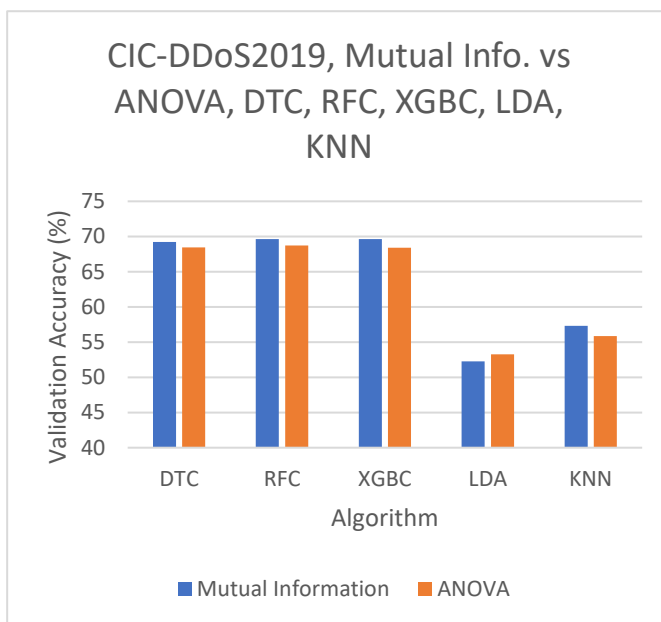


Figure 5.7: Bar chart of the average validation accuracy per algorithm

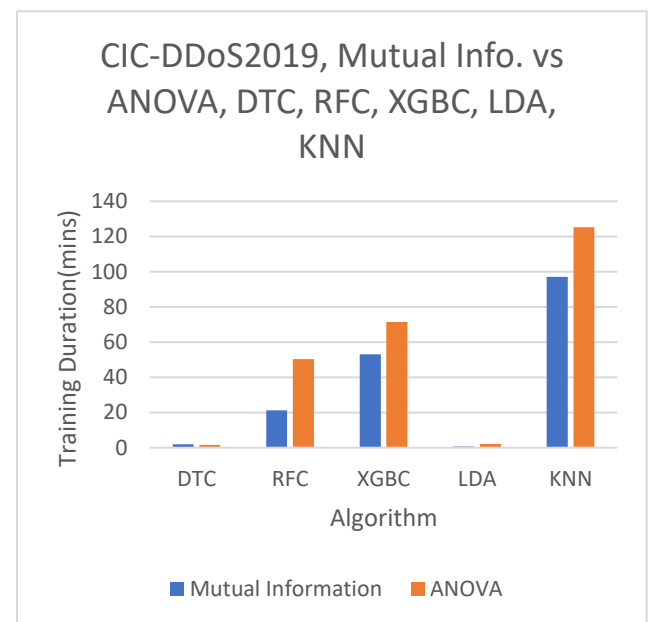


Figure 5.8: Bar chart of the average training duration per algorithm

As the chart on figure 5.7 indicates, mutual information was able to select features that produced slightly more accurate results for almost every algorithm in the first dataset. I believe this is the case since Mutual Info tests as previously mentioned, can capture both linear and nonlinear relationships between the traffic characteristics compared to ANOVA which can only capture linear relationships between variables. This observation can be seen in the more detailed graphs provided in appendix B where it is shown that for lower subset lengths, ANOVA selected subsets are significantly less accurate than Mutual Info. Selected

subsets, but the difference decreases as the subset length increases and after certain lengths the difference disappears. An example of this observation can be seen in figure 5.9.

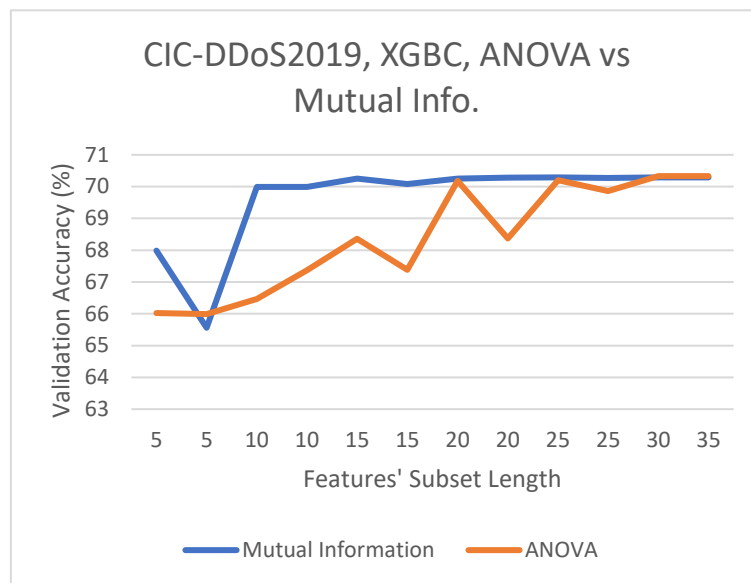


Figure 5.9: Line graph showing the validation accuracy in relation to the subset length on XGBC.

This observation cannot be seen in LDA where, accuracy is about the same for both algorithms (slightly higher for ANOVA selected subsets).

Figure 5.8 displays the average training duration (in minutes) taken for each algorithm to complete the training. Surprisingly, training occurred by subsets selected by mutual information tests completed much faster or equal compared to ANOVA in every algorithm.

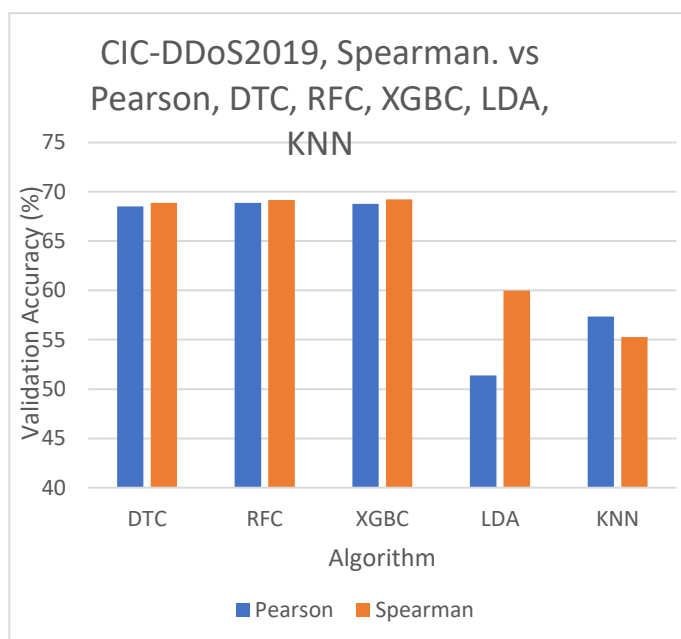


Figure 5.10: Bar chart of the average validation accuracy per algorithm

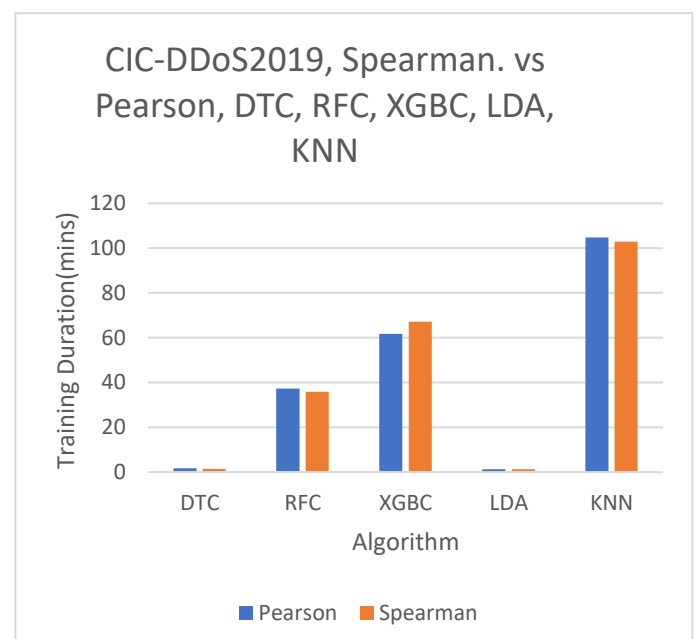


Figure 5.11: Bar chart of the average training duration per algorithm

Figure 5.10 shows a bar chart that summarizes the performance of each algorithm on the 1st dataset (CIC-DDoS2019). The blue bar represents the average of the validation accuracies obtained by subsets that occurred after the exclusion of highly correlated features selected by Pearson and the orange one by Spearman. I want to compare the accuracies obtained after using these 2 feature correlation algorithms to evaluate them.

As shown in figure 5.10 DTC, RFC, XGBC are slightly more accurate when used with Spearman where LDA is almost 2 times more accurate. I am assuming this is the case since Spearman measures the monotonic relationship between two variables, which means it can capture both linear and non-linear relationships as well. This concept is like the previous comparison with Mutual Info. vs ANOVA. That can justify why Spearman produced almost 2 times more correlations in both dataset than Pearson. So, I believe it is safe to conclude that algorithms that can capture several kinds of relationships between variables produce more accurate results and it can be a better option to choose.

Figure 5.11 shows the training duration when Spearman selected features are excluded vs Pearson's. As shown with Pearson the training of RFC was slightly faster, but the opposite occurred with XGBC and KNN, so I can't really make a generic conclusion about the speed.

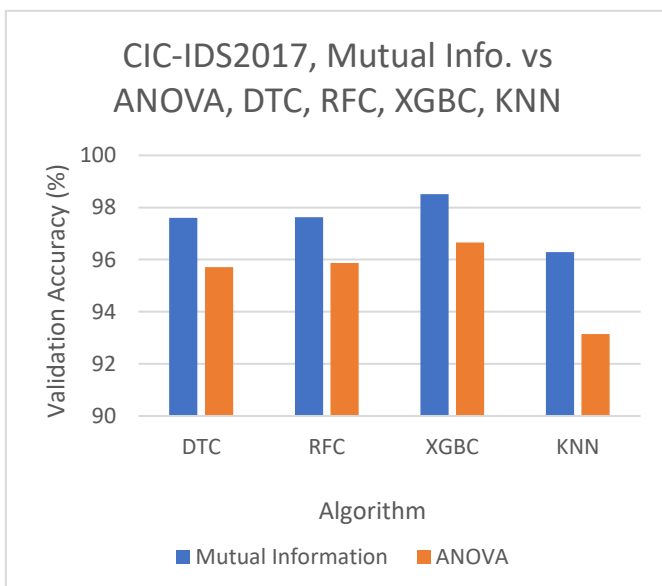


Figure 5.12: Bar chart of the average validation accuracy per algorithm (2nd dataset)

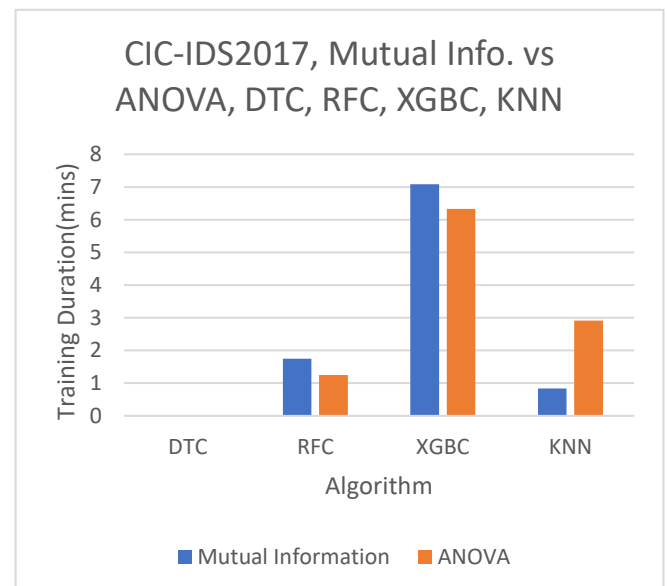


Figure 5.13: Bar chart of the average training duration per algorithm (2nd dataset)

By creating the same graphs for the second dataset, it is verified that mutual information is slightly more accurate than ANOVA. I had to abandon LDA since the training would take an infinite amount of time. I believe some features' distributions on this dataset could be the reason LDA was taking so long but I couldn't figure it out so I didn't use it on the second dataset.

Figure 5.13 can't verify the results of the 1st dataset where Mutual Info. was significantly faster, but I believe this is because the size of the dataset is smaller and therefore the training duration much shorter to see significant differences.

Finally, to complete this set of comparisons, figure 5.13 shows that Pearson was slightly more accurate which is the opposite of my observation in the first dataset. Therefore I cant say confidently which of these 2 algorithms can more precisely identify variable correlations. Unfortunately, I didn't have LDA in the comparison, that made the most difference in the previous graph (figure 5.10).

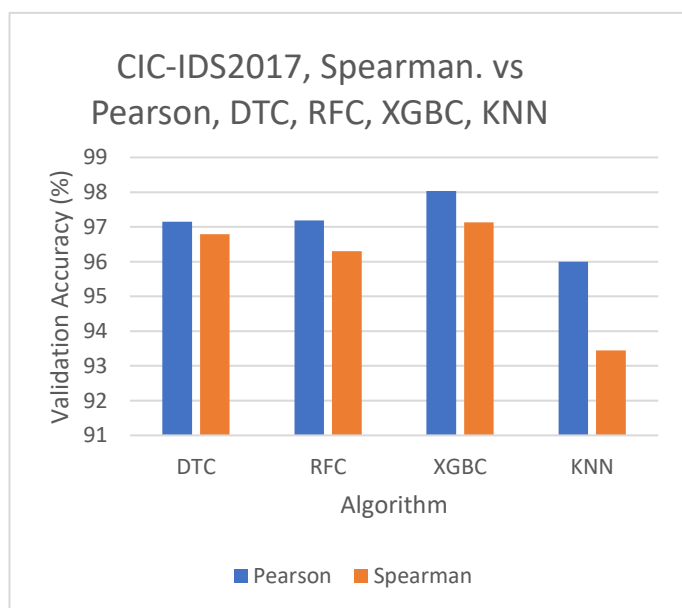


Figure 5.13: Bar chart of the average validation accuracy per algorithm (2nd dataset)

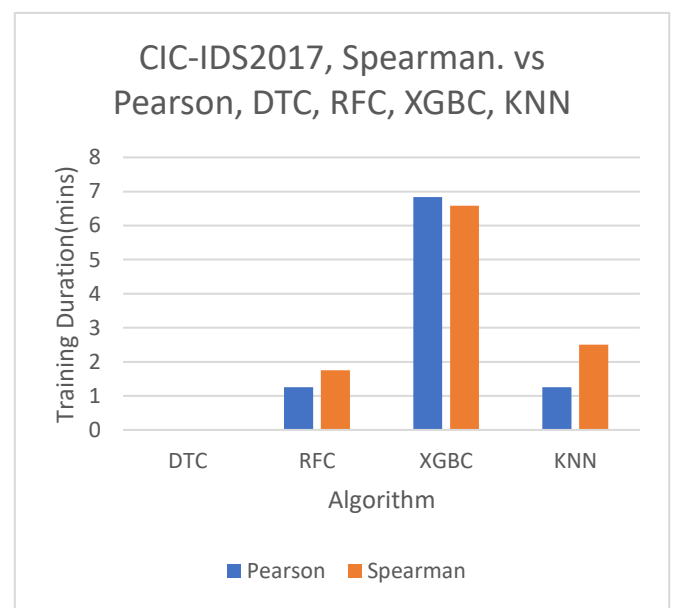


Figure 5.14 Bar chart of the average training duration per algorithm (2nd dataset)

As shown by the previous bar charts some algorithms are more accurate than others. In order to measure each algorithms performance I made the following box plots:

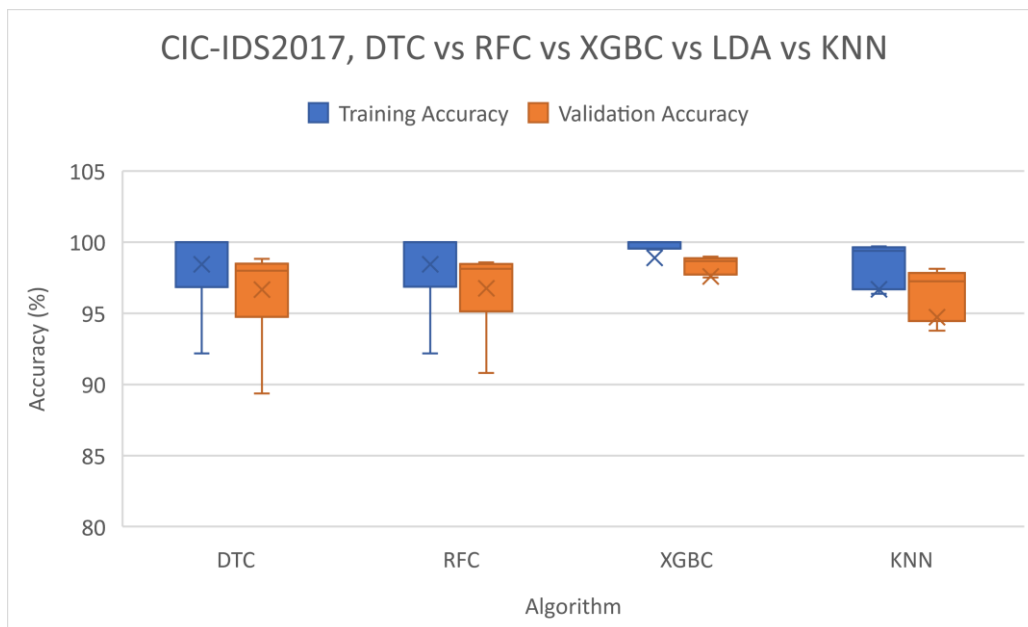


Figure 5.15 Box plot comparing algorithm's accuracies (validation & training) on 2nd dataset

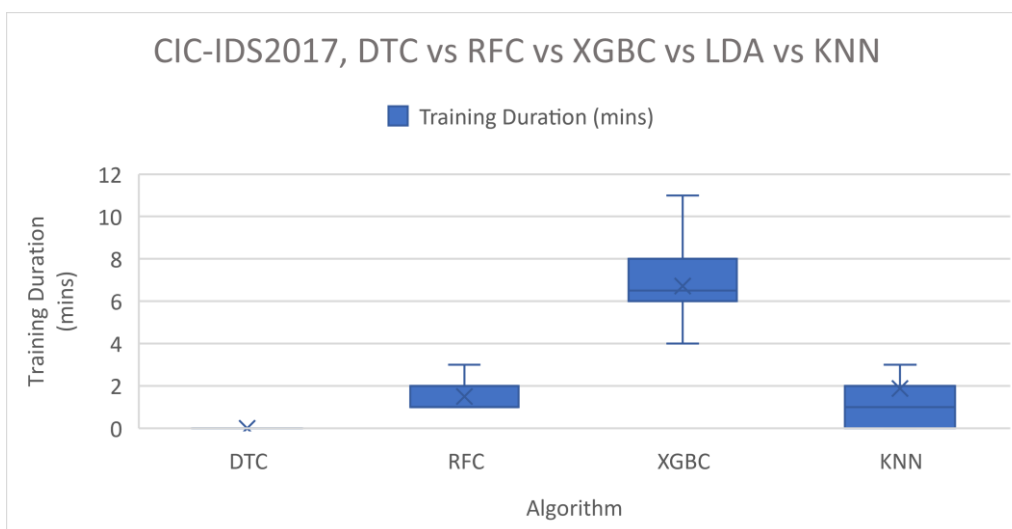


Figure 5.16 Box plot comparing algorithm's training durations on 2nd dataset

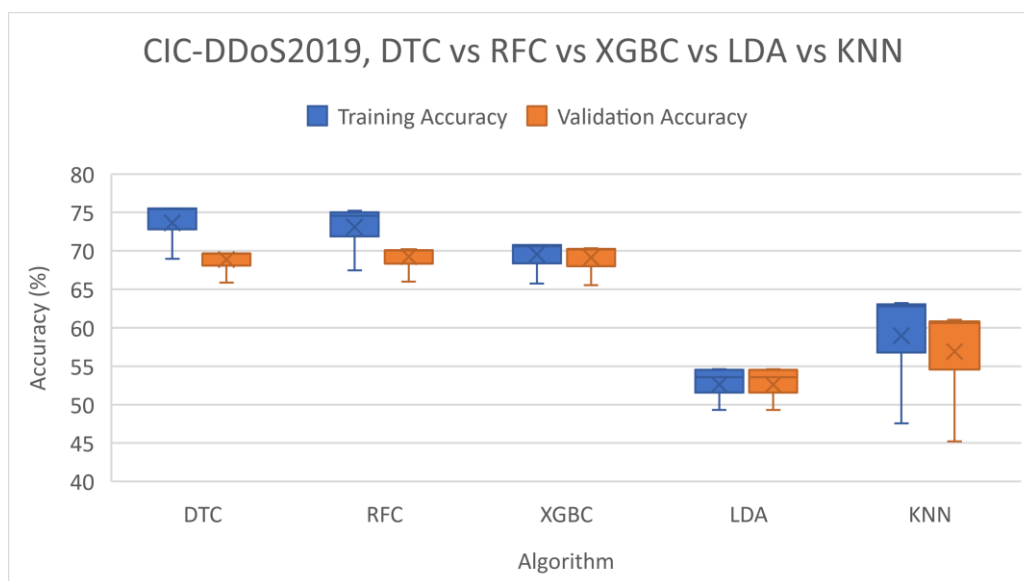


Figure 5.17 Box plot comparing algorithm's accuracies (validation & training) on 1st dataset

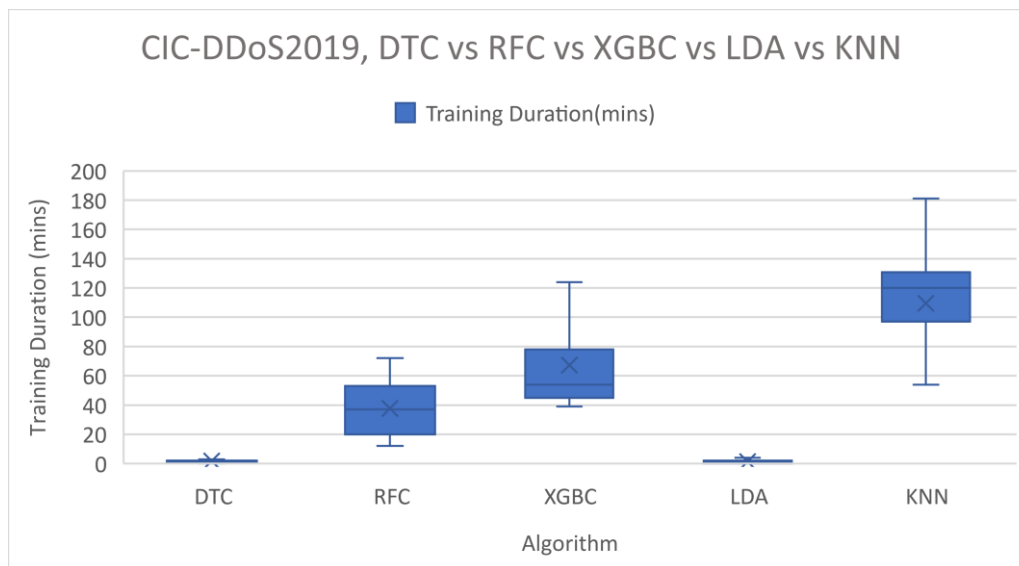


Figure 5.18 Box plot comparing algorithm's accuracies (validation & training) on 1st dataset

Figure 5.15 shows how each algorithm (DTC, RFC, XGBC, KNN) performed on the 2nd dataset. This dataset was significantly smaller than the 1st one with fewer target classes so the accuracy was higher. As shown by the figure the validation accuracy was close to 100% and as expected the training accuracy is always higher than the validation accuracy. XGBC was the most accurate algorithm with the least overfitting compared to the rest.

Figure 5.16 shows the training duration summary for each algorithm for the 2nd dataset. XGBC on average took about 7 minutes to train the dataset and the rest algorithms about 1-2 minutes.

Figure 5.17 shows how each algorithm (DTC, RFC, XGBC, LDA, KNN) performed on the 1st dataset. As shown, tree-based algorithms (RFC, DTC, XGBC) were the most accurate with around 70% accuracy. LDA with about 50-55% accuracy and KNN with 55-60% accuracy. It is worth noting that DTC and RFC were significantly overfitted compared to XGBC which makes sense as XGBC is an updated, better version of the other 2 tree-based algorithms, that is specifically designed to avoid overfitting. Also, LDA seems to have 0 overfitting and KNN being slightly overfitted.

Finally, figure 5.18 shows the training duration summary for each algorithm for the 1st dataset, where KNN was the slowest with an average of 110 minutes (about 2 hours) followed by XGBC, RFC, DTC and LDA. It seems KNN didn't perform that well, considering its training

duration. Overall, Tree-based algorithms seem to have a superior performance on detecting these cyber-attacks.

Target class	Samples
BENIGN	31330
DrDoS_DNS	9678
DrDoS_LDAP	9808
DrDoS_MSSQL	9731
DrDoS_NTP	9937
DrDoS_NetBIOS	9696
DrDoS_SNMP	9977
DrDoS_SSDP	9812
DrDoS_UDP	9883
Syn	8753
UDP-lag	9037

Now that I have an idea of how each algorithm performs and what the most accurate features' subsets are for each algorithm, I will perform hyperparameter tuning to identify whether different combinations of hyperparameters can improve each algorithm's performance. For each algorithm I have selected the subset with the highest validation accuracy produced with as small length as possible. I decided to perform the tuning only for the 1st dataset (CIC-DDoS2019), as the second one's accuracy was almost perfect. In order to test a larger number of different combinations within a reasonable amount of time, I used a reduced version of the 1st dataset. The number of samples for each DDoS class is shown in figure 5.19.

Figure 5.19 Target class distribution for tuning experiment.

Algorithm	Combinations	K-Fold	Feature's Subset
DTC	2160	5	10 features = { Protocol, FlowDuration, TotalFwdPackets, TotalLengthofFwdPackets, FwdPacketLengthMax, FlowBytes/s, FlowIATStd, FwdIATTotal, FwdHeaderLength, act_data_pkt_fwd }
RFC	2880	5	15 features = { Protocol, FlowDuration, TotalFwdPackets, TotalBackwardPackets, TotalLengthofFwdPackets, FwdPacketLengthMax, FwdPacketLengthStd, FlowBytes/s, FlowIATStd, FlowIATMin, FwdIATTotal, FwdHeaderLength, act_data_pkt_fwd, min_seg_size_forward, Inbound }
XGBC	864	5	20 features = { Protocol, FlowDuration, TotalLengthofFwdPackets, FwdPacketLengthMax, FwdPacketLengthStd, BwdPacketLengthMin, BwdPacketLengthStd, FlowBytes/s, FlowIATStd, FwdIATTotal, FwdPSHFlags, FwdHeaderLength, URGFlagCount, CWEFlagCount, Down/UpRatio, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, IdleStd, Inbound }
LDA	1650	5	15 features = { Protocol, FlowDuration, TotalLengthofFwdPackets, FwdPacketLengthMax, FwdPacketLengthStd, BwdPacketLengthMin, FlowBytes/s, FlowIATStd, FwdIATTotal, FwdPSHFlags, URGFlagCount, CWEFlagCount, Down/UpRatio, act_data_pkt_fwd, Inbound }
KNN	960	5	15 features = { Protocol, FlowDuration, TotalLengthofFwdPackets, FwdPacketLengthMax, FwdPacketLengthStd, BwdPacketLengthMin, FlowBytes/s, FlowIATStd, FwdIATTotal, FwdPSHFlags, URGFlagCount, CWEFlagCount, Down/UpRatio, act_data_pkt_fwd, Inbound }

Table 5.3: Table showing tuning subset & other characteristics per algorithm

Table 5.3 shows the features subsets with higher validation accuracies and as small length as possible selected based on the previous training experiments. I used $k = 5$ for cross validation to complete the tuning faster and be able to test a larger range of hyperparameters.

Table 5.4 on the next page shows the ranges used for each hyperparameter of each algorithm as well as their values extracted from the most accurate combination. For every algorithm, all possible combinations that can be formed from all the ranges are tested. The total amount of combinations per algorithm are shown on Table 5.3. I tried using larger ranges for the most influential parameters per algorithm while keeping the ones unrelated to accuracy within a small range or fixed at the default value.

Algorithm	Hyperparameter	Range	Best Value
DTC	ccp_alpha	0	0
DTC	criterion	[gini, entropy]	entropy
DTC	max_depth	[5, 10, 15]	10
DTC	max_features	[0.1, 0.2, 0.3, 0.4]	0.4
DTC	max_leaf_nodes	[23, 24, 25, 26, 27]	27
DTC	min_impurity_decrease	[0, 0.1, 0.2]	0
DTC	min_samples_leaf	[1, 2, 3]	1
DTC	min_samples_split	[2, 7]	2
DTC	random_state	74	74
DTC	splitter	best	best
RFC	bootstrap	TRUE	TRUE
RFC	criterion	[gini, entropy]	entropy
RFC	max_depth	[10, 12, 14, 16]	14
RFC	max_features	[0.1, 0.2, 0.3, 0.4]	0.4
RFC	min_samples_leaf	[1, 2, 3]	2
RFC	min_samples_split	[2, 3, 4]	3
RFC	n_estimators	[90, 100, 110, ... , 180]	90
RFC	oob_score	FALSE	FALSE
XGBC	n_estimators	[80, 100, 120]	gbtree
XGBC	learning_rate	[0.05, 0.06, 0.07, 0.08, 0.09, 0.10]	0.9
XGBC	max_depth	[3, 6]	0
XGBC	subsample	[0.7, 0.8, 0.9, 1.0]	0.1
XGBC	colsample_bytree	[0.8, 0.9]	6
XGBC	booster	gbtree	2
XGBC	gamma	0	120
XGBC	min_child_weight	[1, 2, 3]	0
XGBC	reg_alpha	0	1
XGBC	reg_lambda	1	0.8
LDA	n_components	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	1

LDA	shrinkage	[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]	0.1
LDA	solver	[svd, lsqr, eigen]	lsqr
LDA	tol	[0.000001, 0.010001, 0.020001, 0.030001, 0.040001]	0.000001
KNN	algorithm	[auto, ball_tree, kd_tree, brute]	auto
KNN	leaf_size	[20, 30, 40, 50, 60]	30
KNN	metric	[euclidean, manhattan, minkowski, chebyshev]	manhattan
KNN	n_jobs	-1	-1
KNN	n_neighbors	[5, 10, 15, 20, 25, 30]	30
KNN	p	2	2
KNN	weights	[uniform, distance]	distance

Table 5.4: Table showing tuning ranges per algorithm per parameter

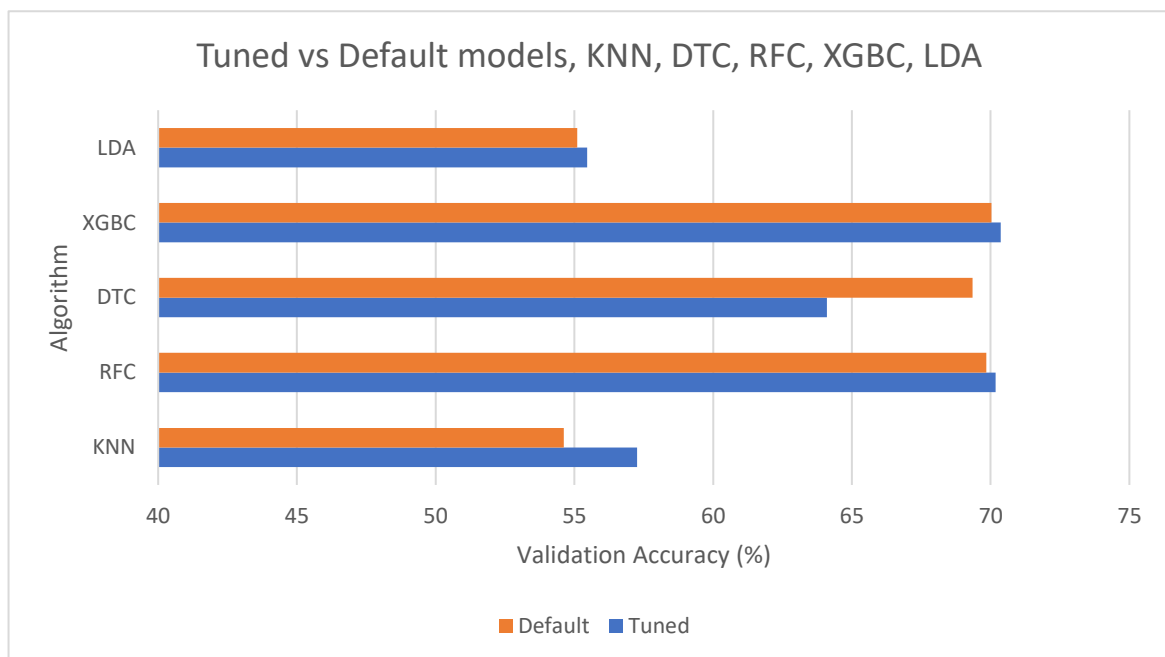


Figure 5.20: Bar chart comparing default and tuned models for each algorithm.

My hyperparameter tuning was performed in a reduced subset of the dataset (~100000 samples) to try as many combinations as possible in a reasonable time. In order to evaluate the extracted hyperparameters I trained each model with the default parameters and the optimal ones with several dataset sizes and took the average validation accuracy. Figure 5.20 is a bar chart showing the average accuracy of every algorithm with the tuned and default hyperparameters. KNN observed the most increase (+3%) in accuracy and LDA, RFC and XGBC a smaller one (+1%). Surprisingly, DTC performed significantly worse (-5%) on average with the tuned parameters. It seems that the extracted parameters were only optimal in the size used for the tuning, but with different sample sizes the accuracy dropped drastically. I believe this is an interesting finding that analysts should be aware of with their analysis.

As I have previously mentioned, in my first approach I have examined how different dataset sizes affect each algorithm's performance, but since that approach was deemed inaccurate, I will attempt to retry it. In my first approach I observed a decrease in accuracy as the dataset size increases. To examine the performance of each algorithm in relation to the size of the dataset I used the tuned models with several sample sizes to re-examine this theory.

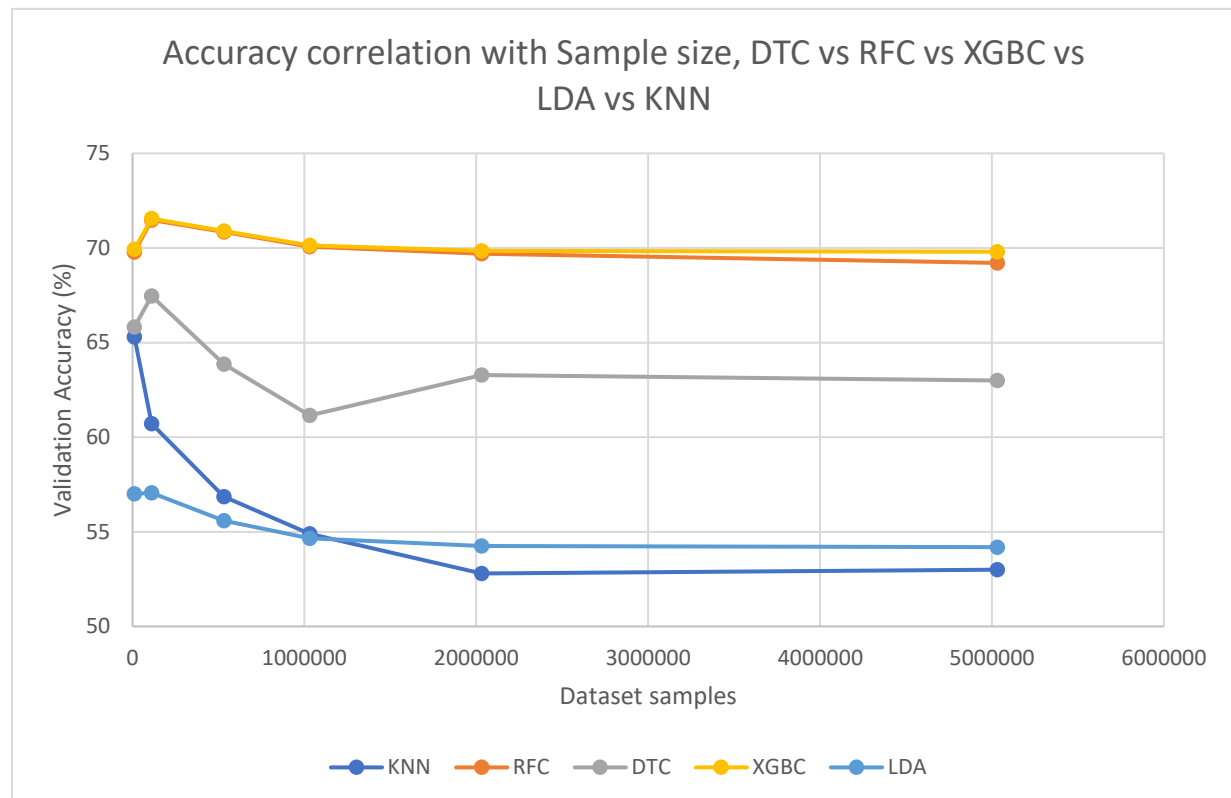


Figure 5.21: Scatter plot showing the validation accuracy in relation to the dataset size.

Figure 5.21 is a scatter plot showing how the validation accuracy of the tuned algorithms is affected by the dataset's size. For the tree-based algorithms as the graph shows there is a slight increase and then a decrease that slowly stabilizes, but not completely. This is because the smallest dataset size is not enough to train a model therefore the accuracy increases. Then, there is a noticeable decrease, that is very noticeable on KNN that is probably due to overfitting. This means that the algorithms are becoming too specialized for the training data and are not generalizing well to new, unseen data. Furthermore, since KNN is a more complex algorithm and takes a large amount of time to train, it is most likely that it struggles learning with a large dataset. It is also worth noting that RFC and XGBC performed almost identically, which makes sense as they are similar.

Figure 5.22 shows how the training duration is affected by the dataset's size. As expected, a linear increase can be observed except from LDA and DTC that they were very fast and the increase is not really noticeable on the plot.

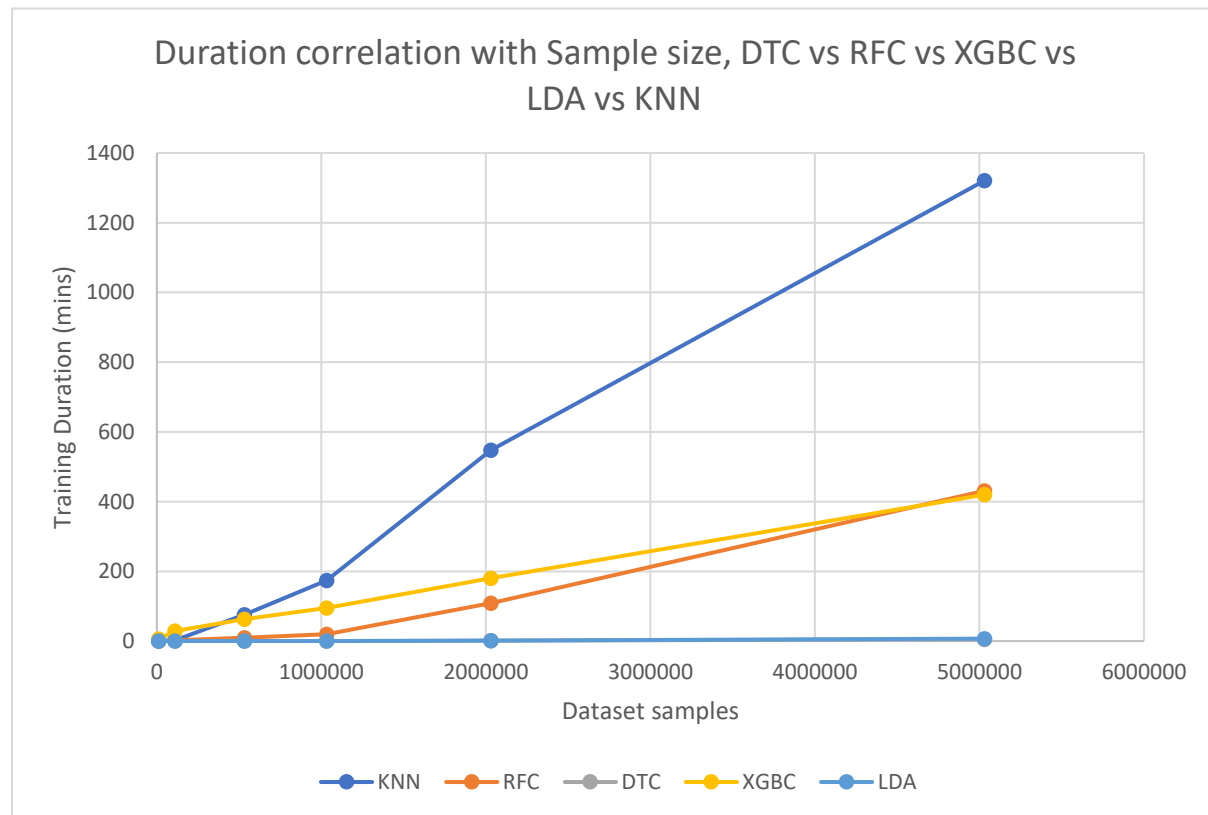


Figure 5.22: Scatter plot showing the training duration in relation to the dataset size.

Overall KNN was by far the slowest algorithm and DTC with LDA the fastest. RFC and XGBC seemed to have a much higher accuracy and even though KNN took so long to train, it was not very successful.

This concludes my experimental evaluation of cyber-attacks, mainly DDoS attacks using my platform and in the next section I will summarize the most important conclusions and insights I extracted from my analysis.

Chapter 6

Conclusion and Future work

6.1 Insights & Conclusions.....	86
6.2 Future work.....	88

6.1 Insights & Conclusions

Throughout this research I managed to draw several conclusions. Before summarizing them, I want to mention that this work has been about 70% system development and 30% experimenting and analyzing the datasets. Therefore, my analysis has not been the most perfect and thorough as it could be.

Firstly, the most important conclusion is that automating the process of data analysis can be very extremely for the analyst. The platform allowed me to perform a lot of experiments to analyze the datasets very easily and quickly. Other than that, I could have all my results organized on the website and saved in the database ready to be extracted and further processed in excel.

Regarding my results on cyber-attacks, as I have analyzed in the previous section, some features can be very influential in detecting the attacks. Such features include the protocol being used, the standard deviation of the payload, the transmission rate and the total duration of the flow.

Another important insight I extracted is that all algorithms performed the best when the number of features used was about 10 to 20. When larger subsets were used the accuracy wouldn't improve, which makes sense since many features are useless for the model. When the subset's length was below 10 the accuracy was significantly worse for each model.

In my research I compared a set of correlation algorithms (Pearson, Spearman) and best feature selection algorithms (Mutual info, ANOVA). Regarding the latter I can confidently say

that mutual information tests are able to select the best features more accurately in cyber-attack datasets since in most cases the selected features produced higher accuracies. My comparison of Pearson and Spearman could not reach a confident conclusion regarding the accuracy of the produced models since my results between the 2 datasets had conflicts. Since these algorithms were used to detect highly correlated features, their accuracy would most probably not have a direct impact on the resulting models' accuracies. This is because even if I keep 2 highly correlated features in my dataset, the accuracy will not necessarily drop.

Later in my analysis I compared the ability of 5 machine learning algorithms (DTC, RFC, XGBC, LDA and KNN) to make accurate predictions on the datasets' targets. I concluded that Tree based algorithms (DTC, RFC and XGBC) were the most effective with XGBC being the most accurate and with the least overfitting compared to DTC and XGBC. To my surprise, the average accuracy (~70%) of these algorithms was much higher than I would expect. To be able to distinguish 11 different DDoS attacks and normal traffic with 70% accuracy seems a bit too high and I suspect that certain features could be leaking information about it. For example, what if the simulated attacker transmitted each request at the same constant rate. The machine learning model can learn to identify an attack based on an almost constant (as I eliminated 100% constant features) data rate or an almost constant payload size. I believe, in order to build realistic models that can generalize well in completely unknown traffic several datasets should be combined generated on completely different network setups.

After training, I tuned each algorithm to find the best hyperparameters and see how they perform on average with the default ones. I concluded that every tuned algorithm showed a slight increase in accuracy except DTC. DTC dropped about 5% with the tuned parameters as with different sample sizes than the one used for tuning the algorithm couldn't perform that well with the chosen hyperparameters.

I then proceeded to examine how accuracy changes with the dataset's size. I concluded that as the amount of training data increases the accuracy drops. I believe, this is due to overfitting since when an algorithm sees a large number of samples, it 'memorizes' certain patterns and then is not able to see other ones and generalize.

Finally, the CICFlowMeter tool that is converting raw traffic files to excel datasets might not be accurate and some features might be miscalculated leading to unrealistic models.

6.1 Future work

Through this project I have created a good basis for maybe a much more powerful automation tool. My platform can be extended to support regression datasets, a larger number of algorithms and be more customizable. There are certain API's that allow the creation of charts and plots that can be automatically displayed for each experiment comparing it to others etc. Also, another interesting feature would be the capability to perform predictions on pretrained models saved on the server. The user can select an already trained model and insert new data to predict. Therefore, I believe my work can be used for the future development of even more useful features.

Regarding the analysis of cyberattacks I believe there are several experiments that can be performed to further understand how such datasets can or cannot be used for actual attack prevention. I believe it should be examined whether certain features of the dataset other than the identification features that are obvious (IPs, ports etc.) can leak information about the target and therefore be unable to generalize in unknown data. As I have mentioned before some features might remain in a specific range within an attack, that is not necessarily a characteristic of the attack in general. In addition, other data analysis techniques for feature engineering such as PCA can be evaluated against these types of attacks compared to the methodology of custom feature selection that I used. Finally, I think it is important to consider a greater number of datasets to cross validate the insights of my current work.

Bibliography

1. Nazario, J. (2008). DDoS attack evolution. *Network Security*, 2008(7), 7-10.
2. Li, Y., & Liu, Q. (2021). A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments. *Energy Reports*, 7, 8176-8186.
3. Sharafaldin, I., Lashkari, A. H., Hakak, S., & Ghorbani, A. A. (2019, October). Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)* (pp. 1-8). IEEE.
4. Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018.
5. Dave, K. T. (2013). Brute-force attack 'seeking but distressing'. *Int. J. Innov. Eng. Technol. Brute-force*, 2(3), 75-78.
6. Shorey, T., Subbaiah, D., Goyal, A., Sakxena, A., & Mishra, A. K. (2018, September). Performance comparison and analysis of slowloris, goldeneye and xerxes ddos attack tools. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 318-322). IEEE.
7. Sharma, A. DDOS AND HULK ATTACKS IN WEB APPLICATIONS WITH DETECTION MECHANISM
8. Khomutovskyy, S. (2021). *Detection and mitigation of slow HTTP POST attack* (Doctoral dissertation, Instytut Informatyki).
9. Lashkari, A. H., Zang, Y., Owhuo, G., Mamun, M. S. I., & Gil, G. D. (2017). CICFlowMeter. *GitHub*. [vid. 2021-08-10]. Dostupné z: <https://github.com/ahlashkari/CICFlowMeter/blob/master/ReadMe.txt>.
10. Segal, M. R. (2004). Machine learning benchmarks and random forest regression.
11. Chalyi, O., & Kolomytsev, M. (2022). Comparison of Tools for Web-Application Brute Forcing. *Theoretical and Applied Cybersecurity*, 4(1).
12. Otoum, Y., & Nayak, A. (2021). As-ids: Anomaly and signature based ids for the internet of things. *Journal of Network and Systems Management*, 29, 1-26.
13. Visoottiviseth, V., Sakarin, P., Thongwilai, J., & Choobanjong, T. (2020, November). Signature-based and behavior-based attack detection with machine learning for home IoT devices. In *2020 IEEE REGION 10 CONFERENCE (TENCON)* (pp. 829-834). IEEE.
14. Ott, R. L., & Longnecker, M. T. (2015). *An introduction to statistical methods and data analysis*. Cengage Learning.
15. Shyr, J., & Spisic, D. (2014). Automated data analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(5), 359-366.
16. Chen, Y., Pei, J., & Li, D. (2019, May). DETPro: a high-efficiency and low-latency system against DDoS attacks in SDN based on decision tree. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)* (pp. 1-6). IEEE.
17. Chen, Y., Hou, J., Li, Q., & Long, H. (2020, December). DDoS attack detection based on random forest. In *2020 IEEE International Conference on Progress in Informatics and Computing (PIC)* (pp. 328-334). IEEE.
18. Chen, Z., Jiang, F., Cheng, Y., Gu, X., Liu, W., & Peng, J. (2018, January). XGBoost classifier for DDoS attack detection and analysis in SDN-based cloud. In *2018 IEEE*

- international conference on big data and smart computing (bigcomp)* (pp. 251-256). IEEE.
19. Vu, N. H., Choi, Y., & Choi, M. (2008, April). DDoS attack detection using K-Nearest Neighbor classifier method. In *Proceedings of the 4th IASTED International Conference on Telehealth/Assistive Technologies. Baltimore, Maryland, USA* (pp. 248-253).
 20. Thapngam, T., Yu, S., & Zhou, W. (2012, January). DDoS discrimination by linear discriminant analysis (LDA). In *2012 International Conference on Computing, Networking and Communications (ICNC)* (pp. 532-536). IEEE.
 21. Scikit-learn.org – Variance Threshold https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html
 22. Pandas – pandas.DataFrame.corr <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html>
 23. Cohen, I., Huang, Y., Chen, J., Benesty, J., Benesty, J., Chen, J., ... & Cohen, I. (2009). Pearson correlation coefficient. *Noise reduction in speech processing*, 1-4.
 24. Myers, L., & Sirois, M. J. (2004). Spearman correlation coefficients, differences between. *Encyclopedia of statistical sciences*, 12.
 25. Scikit-learn.org – SelectKBest https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
 26. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
 27. McKinney, W., & others. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51–56).
 28. Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357-362.
 29. Breddels, M. A., & Veljanoski, J. (2018). Vaex: big data exploration in the era of gaia. *Astronomy & Astrophysics*, 618, A13.
 30. Scikit-learn – cross_validation https://scikit-learn.org/stable/modules/cross_validation.html
 31. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). New York, NY, USA: ACM. <https://doi.org/10.1145/2939672.2939785>
 32. PyMongo - <https://pymongo.readthedocs.io/en/stable/>
 33. Hesterberg, T. (2011). Bootstrap. *Wiley Interdisciplinary Reviews: Computational Statistics*, 3(6), 497-526.
 34. Lancaster, H. O., & Seneta, E. (2005). Chi-square distribution. *Encyclopedia of biostatistics*, 2.
 35. St, L., & Wold, S. (1989). Analysis of variance (ANOVA). *Chemometrics and intelligent laboratory systems*, 6(4), 259-272.
 36. Kraskov, A., Stögbauer, H., & Grassberger, P. (2004). Estimating mutual information. *Physical review E*, 69(6), 066138.
 37. GridSearchCV - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
 38. DataRobot <https://www.datarobot.com/platform/>

39. Rosay, A., Cheval, E., Carlier, F., & Leroux, P. (2022, February). Network intrusion detection: A comprehensive analysis of CIC-IDS2017. In *8th International Conference on Information Systems Security and Privacy* (pp. 25-36). SCITEPRESS-Science and Technology Publications.
40. Chadd, A. (2018). DDoS attacks: past, present and future. *Network Security*, 2018(7), 13-15.
41. CloudFare - Famous DDoS attacks | The largest DDoS attacks of all time
<https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/>
42. RapidMiner - <https://rapidminer.com/>

Appendix A

Each sample represents the features of a flow between 2 nodes. CICFlowMeter will group packets (out of .pcap or other network log file) that have similar characteristics (rate, payload size, source/dest IP's and Ports, protocol) into one flow and calculate the features of that flow(dataset row).

*Forward direction: source to destination

*Backward direction: destination to source

*Bulk rate: rate of packets/bytes transferred as a single unit/group

*Flow rate: rate of packets/bytes transferred in a flow

<u>Feature Name</u>	<u>Description</u>
FlowID	String concatenation of other features (flowId = sourceIP + "-" + destinationIP + "-" + srcPort + "-" + dstPort + "-" + protocol)
Source IP	IP address of the sender
Destination IP	IP address of the receiver
Source Port	Port number used by the sender
Destination Port	Port number used by the receiver
Protocol	Number identifying the protocol of the connection
Flow duration	Duration of the flow in Microseconds
act_data_pkt_fwd	Number of actual data packets forwarded from the source to destination. Management/Control packets are not included.
Inbound	Total number of bytes received by the destination. It's not always the same as total length of Fwd Packet feature since packet losses might occur during transmission.
total Fwd Packet	Total packets in the forward direction
total Bwd packets	Total packets in the backward direction
total Length of Fwd Packet	Total size of packets in forward direction
total Length of Bwd Packet	Total size of packets in backward direction
Fwd Packet Length Min	Minimum size of packets in forward direction
Fwd Packet Length Max	Maximum size of packets in forward direction
Fwd Packet Length Mean	Mean size of packets in forward direction

Fwd Packet Length Std	Standard deviation size of packets in forward direction
Bwd Packet Length Min	Minimum size of packets in backward direction
Bwd Packet Length Max	Maximum size of packets in backward direction
Bwd Packet Length Mean	Mean size of packets in backward direction
Bwd Packet Length Std	Standard deviation size of packets in backward direction
Flow Bytes/s	Number of flow bytes per second
Flow Packets/s	Number of flow packets per second
Flow IAT Mean	Mean time between two packets sent in the flow
Flow IAT Std	Standard deviation time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Flow IAT Min	Minimum time between two packets sent in the flow
Fwd IAT Min	Minimum time between two packets sent in the forward direction
Fwd IAT Max	Maximum time between two packets sent in the forward direction
Fwd IAT Mean	Mean time between two packets sent in the forward direction
Fwd IAT Std	Standard deviation time between two packets sent in the forward direction
Fwd IAT Total	Total time between two packets sent in the forward direction
Bwd IAT Min	Minimum time between two packets sent in the backward direction
Bwd IAT Max	Maximum time between two packets sent in the backward direction
Bwd IAT Mean	Mean time between two packets sent in the backward direction
Bwd IAT Std	Standard deviation time between two packets sent in the backward direction
Bwd IAT Total	Total time between two packets sent in the backward direction
Fwd PSH flags	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
Bwd PSH Flags	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
Fwd URG Flags	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)

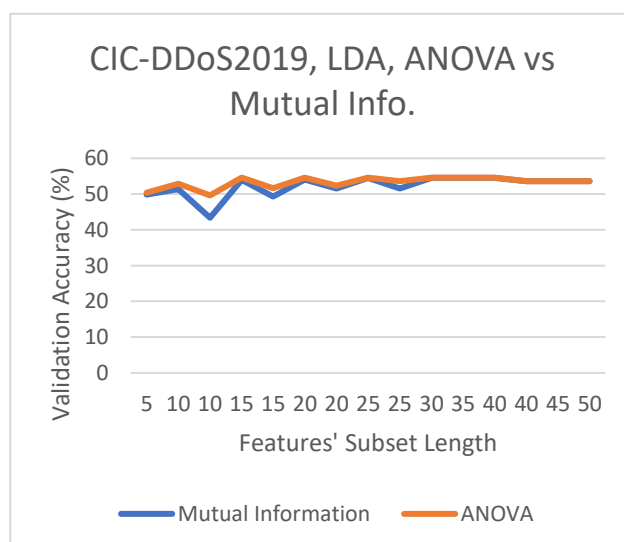
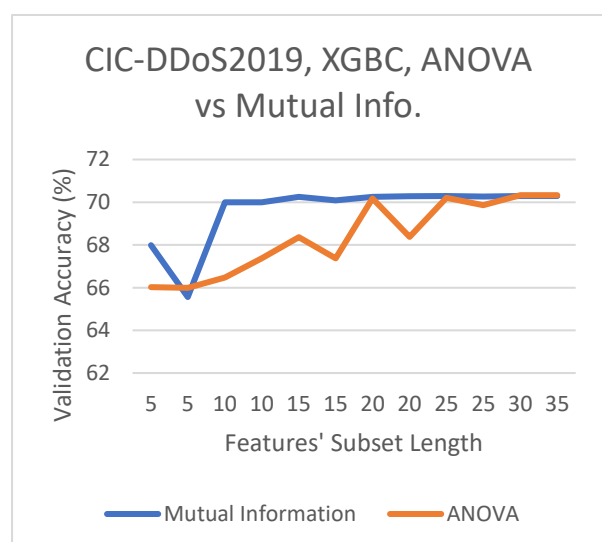
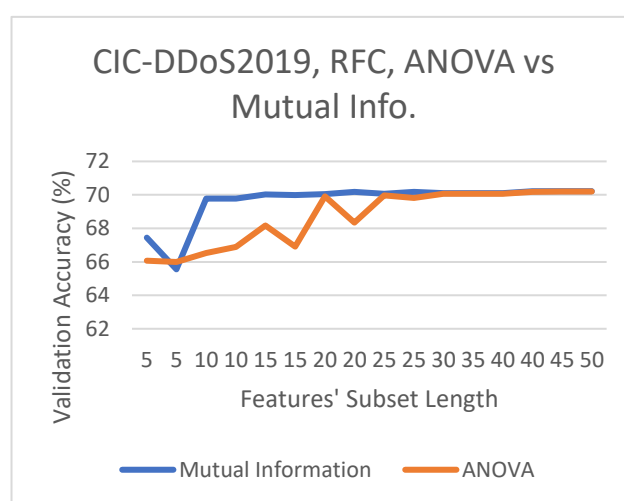
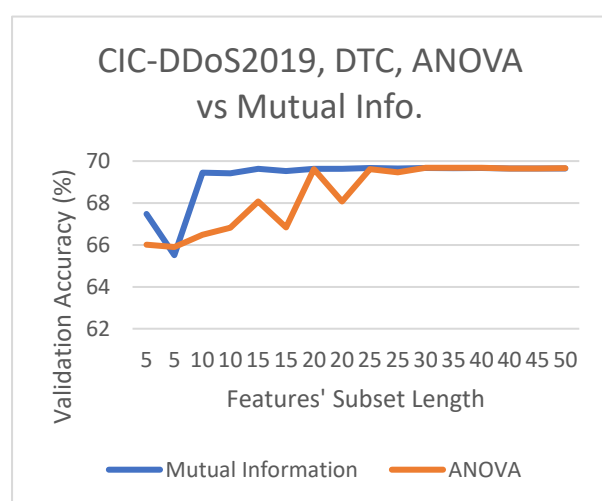
Bwd URG Flags	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
Fwd Header Length	Total bytes used for headers in the forward direction
Bwd Header Length	Total bytes used for headers in the backward direction
FWD Packets/s	Number of forward packets per second
Bwd Packets/s	Number of backward packets per second
Packet Length Min	Minimum length of a packet
Packet Length Max	Maximum length of a packet
Packet Length Mean	Mean length of a packet
Packet Length Std	Standard deviation length of a packet
Packet Length Variance	Variance length of a packet
FIN Flag Count	Number of packets with FIN
SYN Flag Count	Number of packets with SYN
RST Flag Count	Number of packets with RST
PSH Flag Count	Number of packets with PUSH
ACK Flag Count	Number of packets with ACK
URG Flag Count	Number of packets with URG
CWR Flag Count	Number of packets with CWR
ECE Flag Count	Number of packets with ECE
down/Up Ratio	Download and upload ratio
Average Packet Size	Average size of packets
Fwd Segment Size Avg	Average size observed in the forward direction
Bwd Segment Size Avg	Average size observed in the backward direction
Fwd Bytes/Bulk Avg	Average number of bytes bulk rate in the forward direction
Fwd Packet/Bulk Avg	Average number of packets bulk rate in the forward direction
Fwd Bulk Rate Avg	Average rate at which bulk transfers are occurring in the forward direction.
Bwd Bytes/Bulk Avg	Average rate at which bulk transfers are occurring in the backward direction.
Bwd Packet/Bulk Avg	Average number of packets bulk rate in the backward direction
Bwd Bulk Rate Avg	Average number of bulk rate in the backward direction

Subflow Fwd Packets	The average number of packets in a sub flow in the forward direction
Subflow Fwd Bytes	The average number of bytes in a sub flow in the forward direction
Subflow Bwd Packets	The average number of packets in a sub flow in the backward direction
Subflow Bwd Bytes	The average number of bytes in a sub flow in the backward direction
Fwd Init Win bytes	The total number of bytes sent in initial window in the forward direction
Bwd Init Win bytes	The total number of bytes sent in initial window in the backward direction
Fwd Act Data Pkts	Count of packets with at least 1 byte of TCP data payload in the forward direction
Fwd Seg Size Min	Minimum segment size observed in the forward direction
Active Min	Minimum time a flow was active before becoming idle
Active Mean	Mean time a flow was active before becoming idle
Active Max	Maximum time a flow was active before becoming idle
Active Std	Standard deviation time a flow was active before becoming idle
Idle Min	Minimum time a flow was idle before becoming active
Idle Mean	Mean time a flow was idle before becoming active
Idle Max	Maximum time a flow was idle before becoming active
Idle Std	Standard deviation time a flow was idle before becoming active
min_seg_size_fwd	Minimum segment size observed in the forward direction of the flow
Label	Target variable identifying the kind of network traffic(attack, normal)

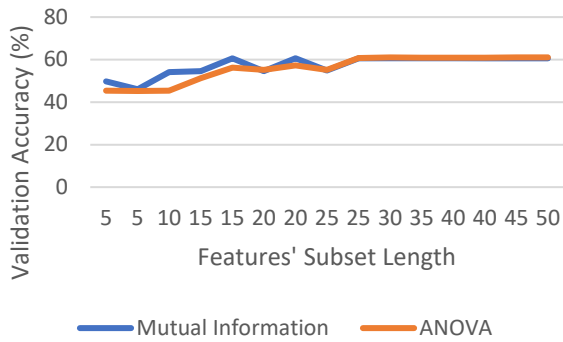
Appendix B

This section contains all the graphs I created with my results from the training experiments. They were a lot, so I decided to exclude them from the main text, but rather refer them to support my conclusions. The graphs mainly highlight differences between accuracies obtained by several machine learning algorithms (DTC, RFC, XGBC, LDA, KNN), comparing features obtained with ANOVA and Mutual Information as well as features obtained after the exclusion of highly correlated (useless) features obtained by Spearman and Pearson algorithms.

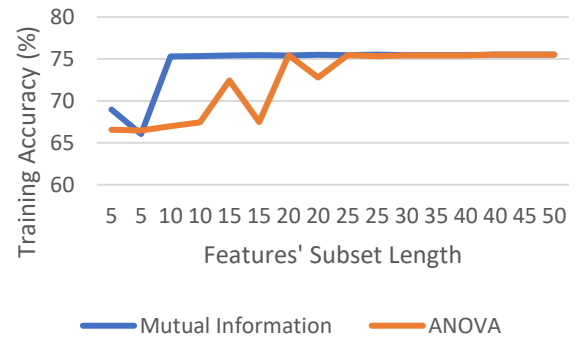
1. CIC-DDoS2019 Dataset:



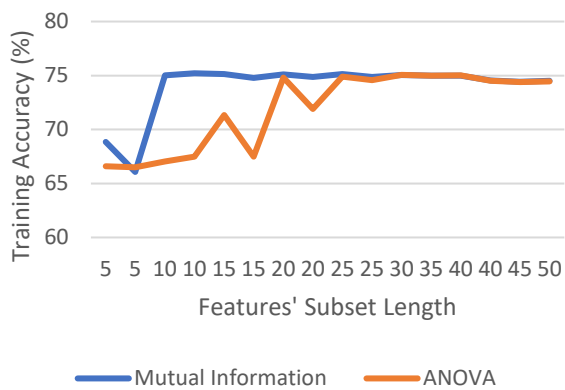
CIC-DDoS2019, KNN, ANOVA vs Mutual Info.



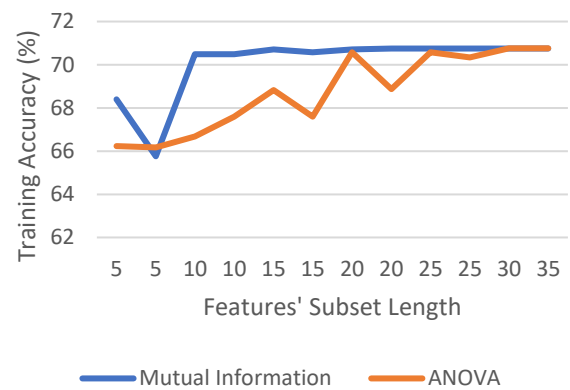
CIC-DDoS2019, DTC, ANOVA vs Mutual Info.



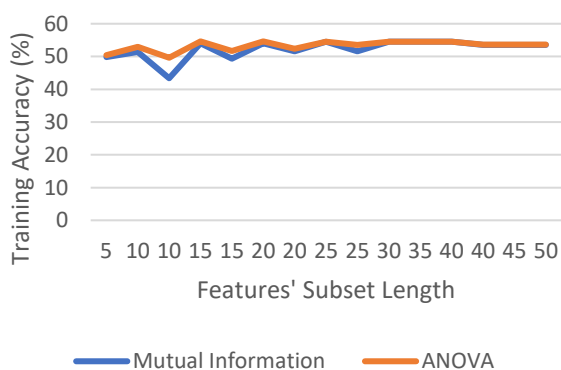
CIC-DDoS2019, RFC, ANOVA vs Mutual Info.



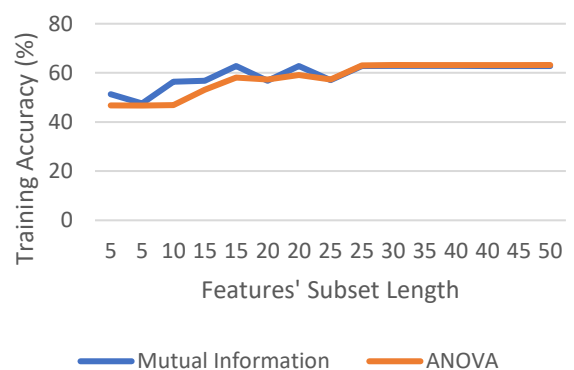
CIC-DDoS2019, XGBC, ANOVA vs Mutual Info.



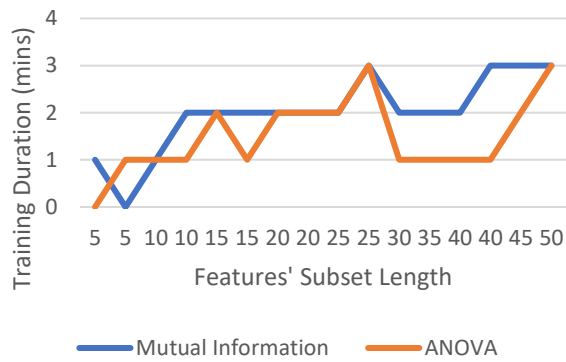
CIC-DDoS2019, LDA, ANOVA vs Mutual Info.



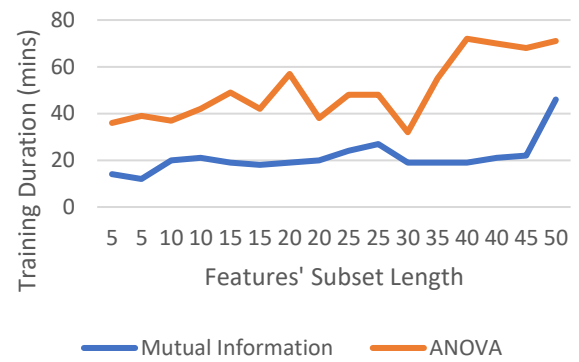
CIC-DDoS2019, KNN, ANOVA vs Mutual Info.



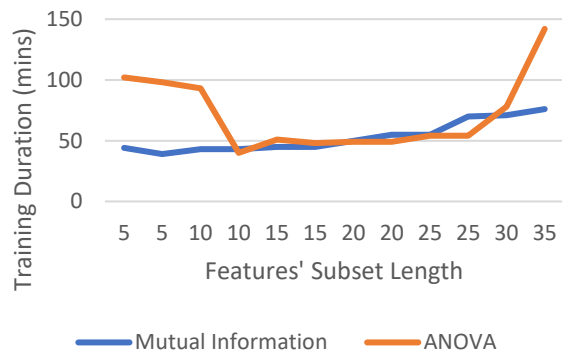
CIC-DDoS2019, DTC, ANOVA vs Mutual Info.



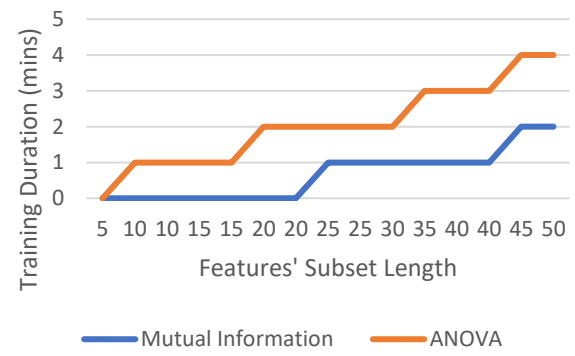
CIC-DDoS2019, RFC, ANOVA vs Mutual Info.



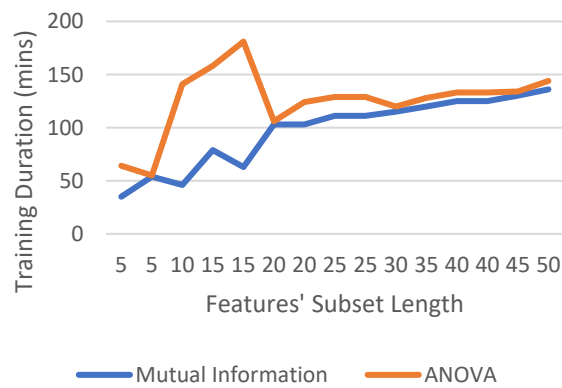
CIC-DDoS2019, XGBC, ANOVA vs Mutual Info.



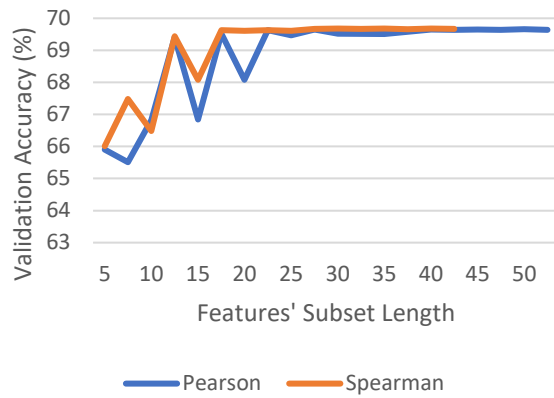
CIC-DDoS2019, LDA, ANOVA vs Mutual Info.



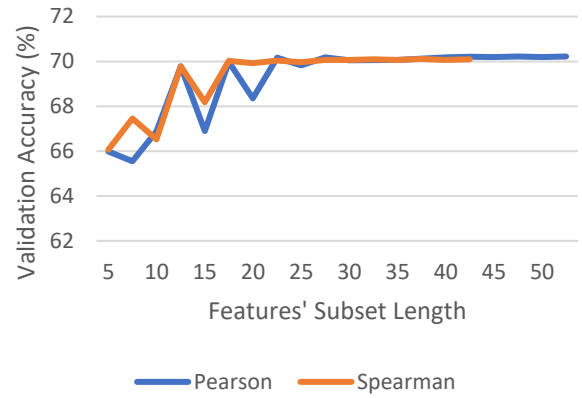
CIC-DDoS2019, KNN, ANOVA vs Mutual Info.



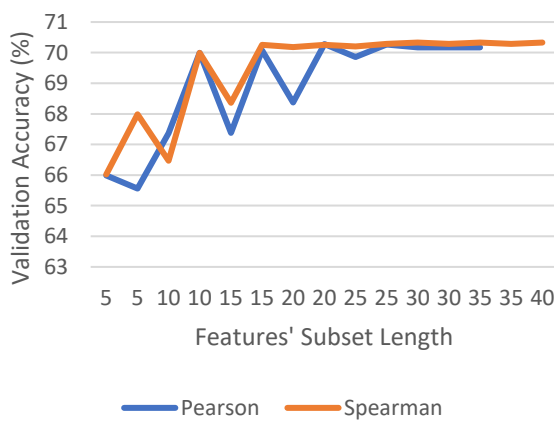
CIC-DDoS2019, DTC, Pearson vs Spearman



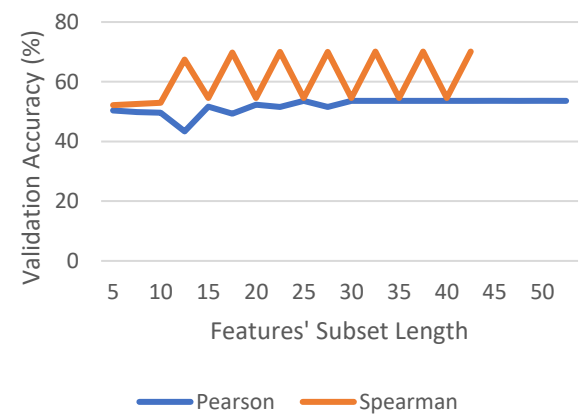
CIC-DDoS2019, RFC, Pearson vs Spearman



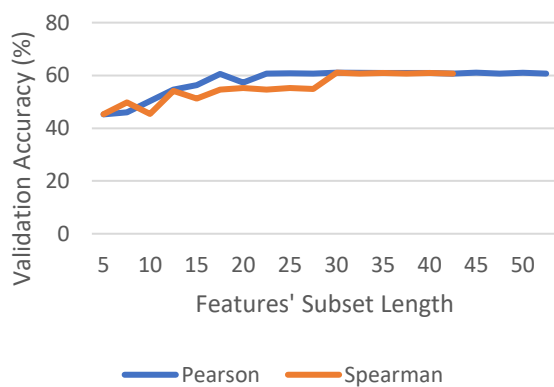
CIC-DDoS2019, XGBC, Pearson vs Spearman



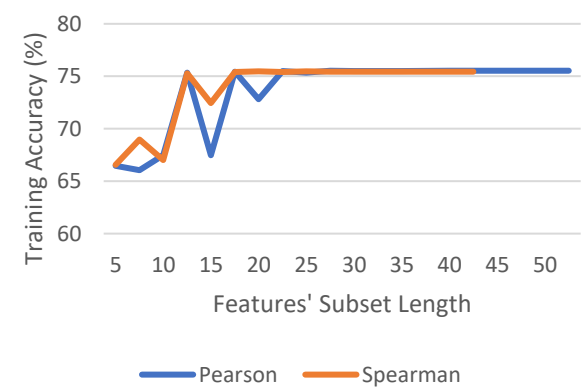
CIC-DDoS2019, LDA, Pearson vs Spearman



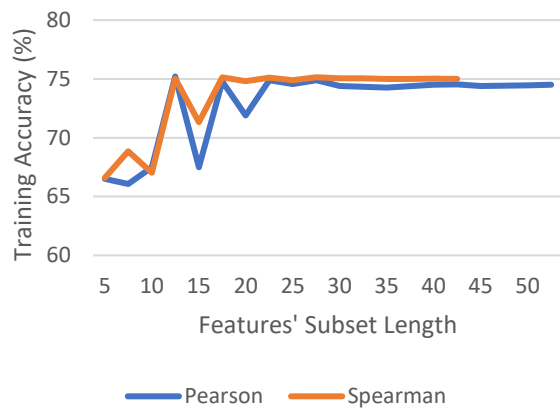
CIC-DDoS2019, KNN, Pearson vs Spearman



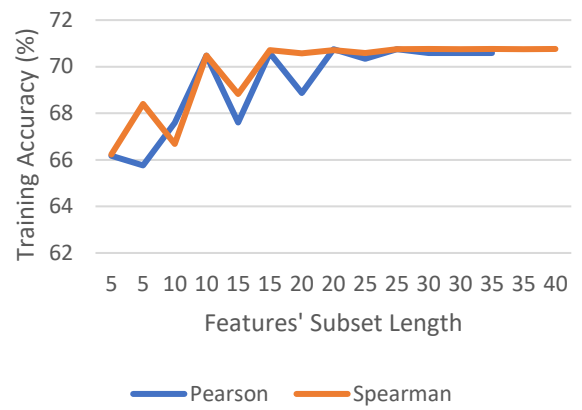
CIC-DDoS2019, DTC, Pearson vs Spearman



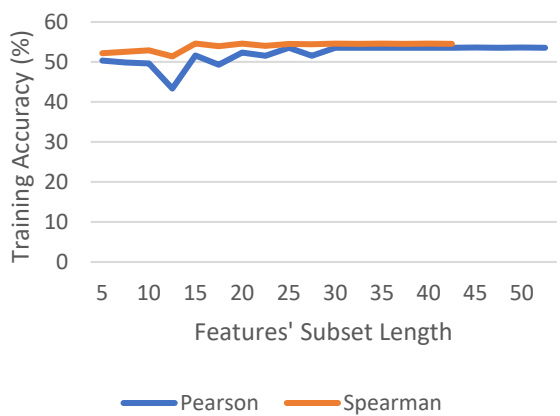
CIC-DDoS2019, RFC, Pearson vs Spearman



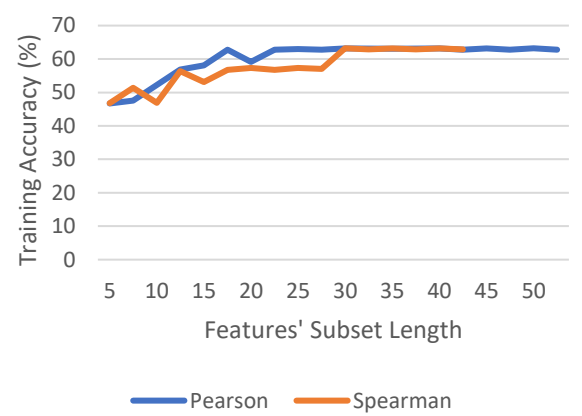
CIC-DDoS2019, XGBC, Pearson vs Spearman



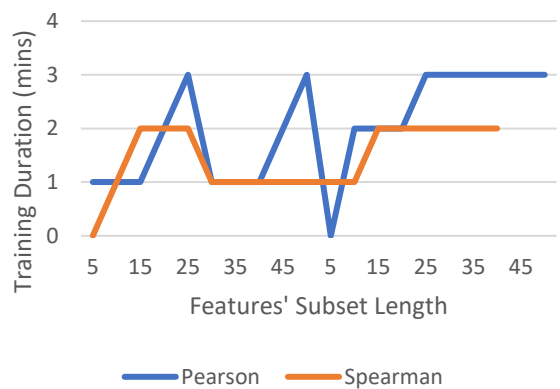
CIC-DDoS2019, LDA, Pearson vs Spearman



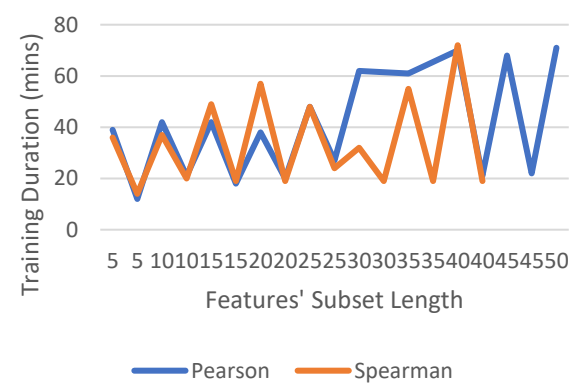
CIC-DDoS2019, KNN, Pearson vs Spearman



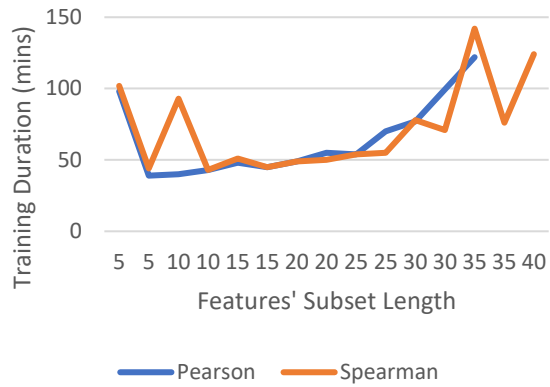
CIC-DDoS2019, DTC, Pearson vs Spearman



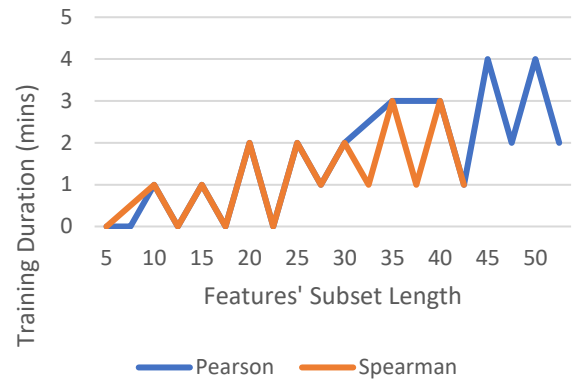
CIC-DDoS2019, RFC, Pearson vs Spearman



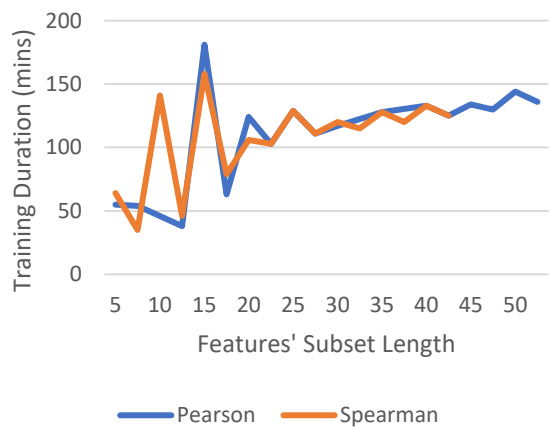
CIC-DDoS2019, XGBC, Pearson vs Spearman



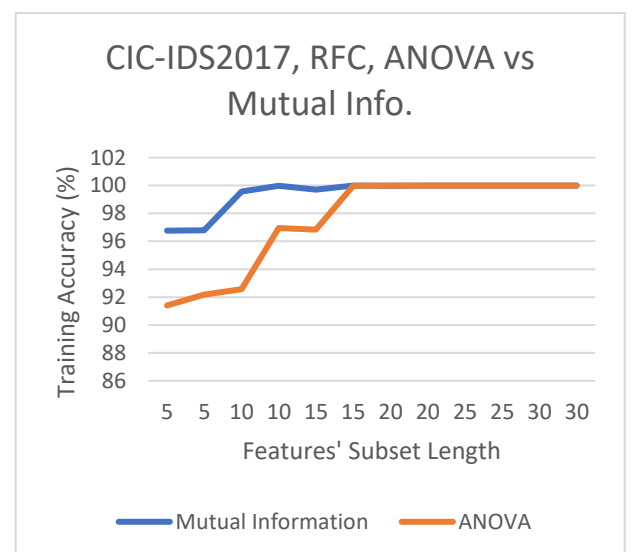
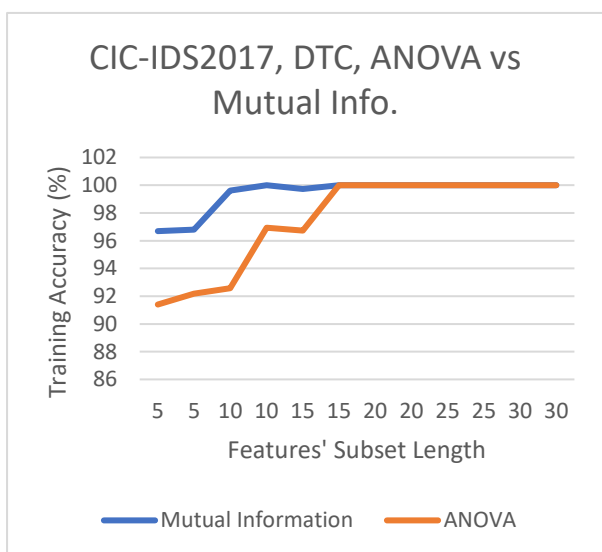
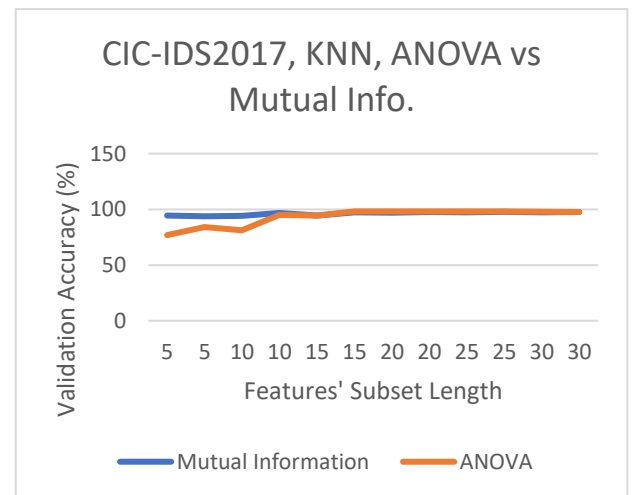
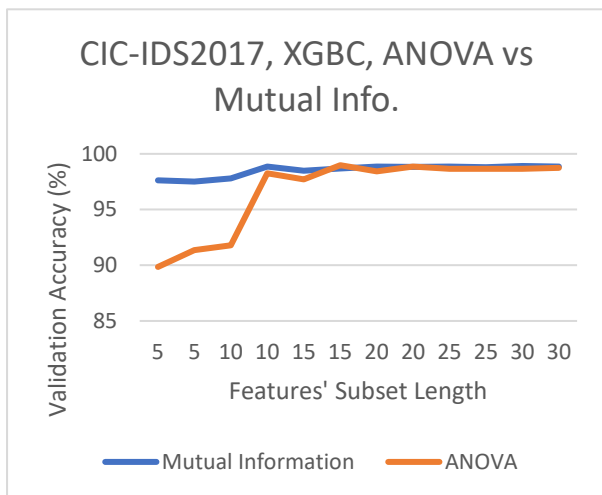
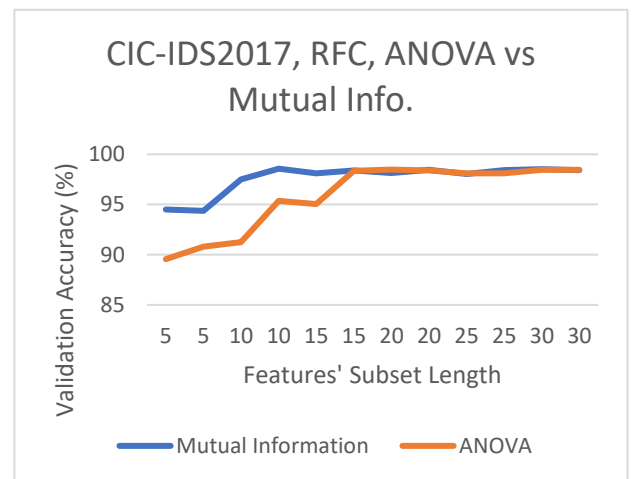
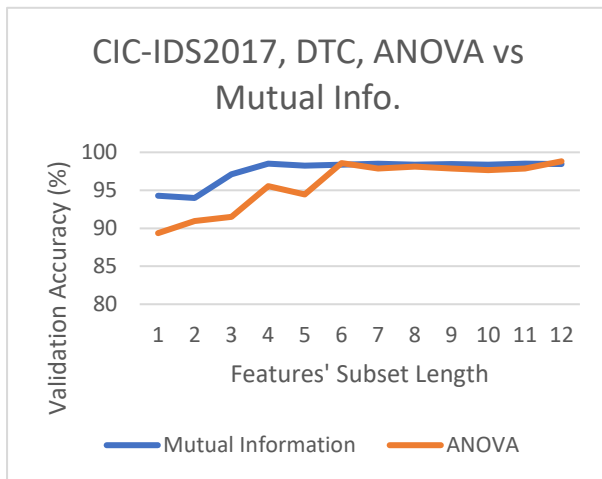
CIC-DDoS2019, LDA, Pearson vs Spearman

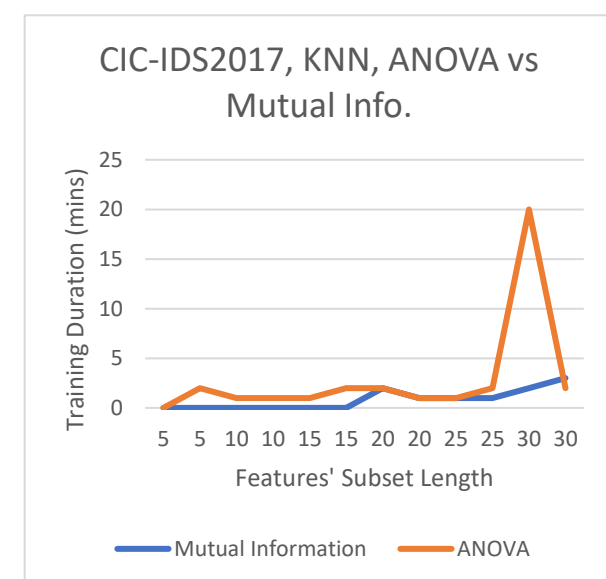
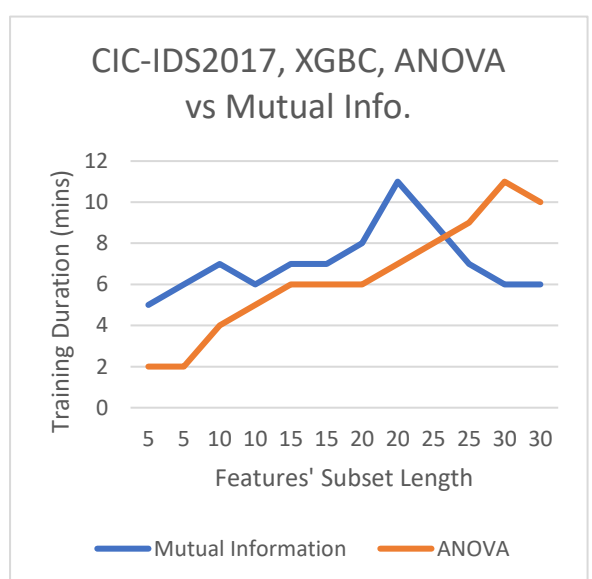
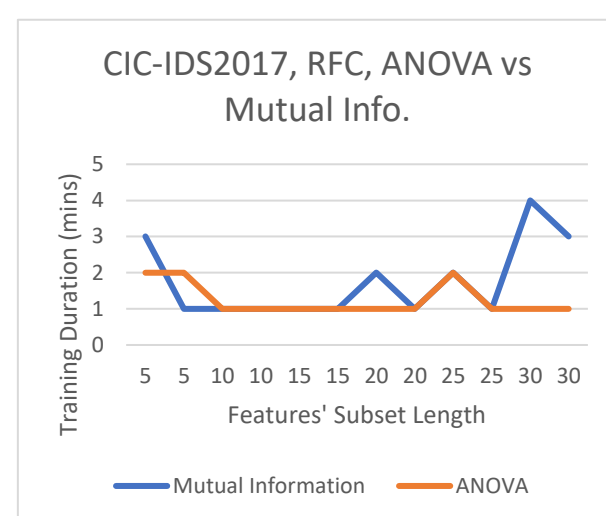
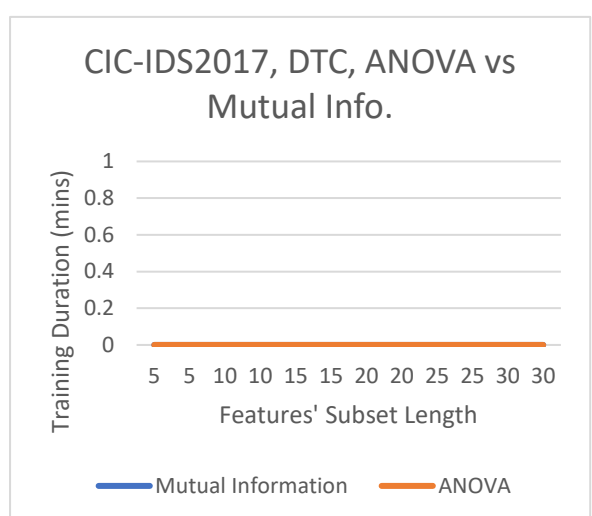
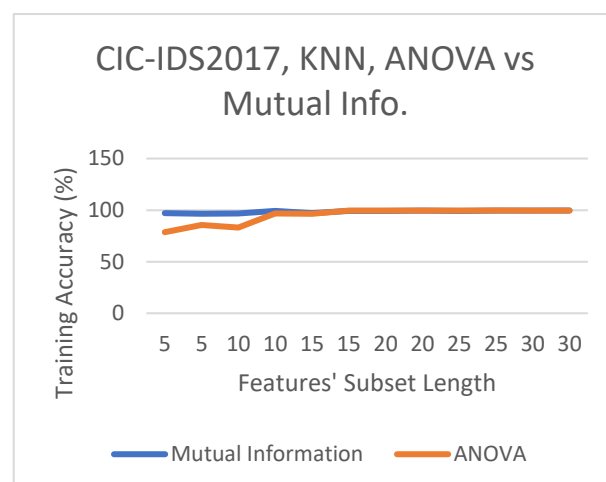
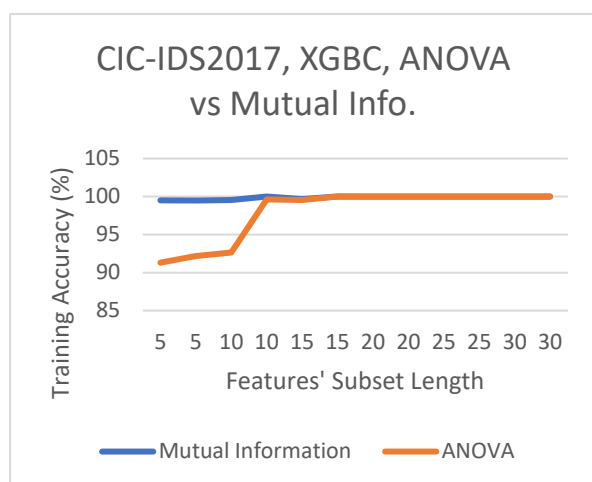


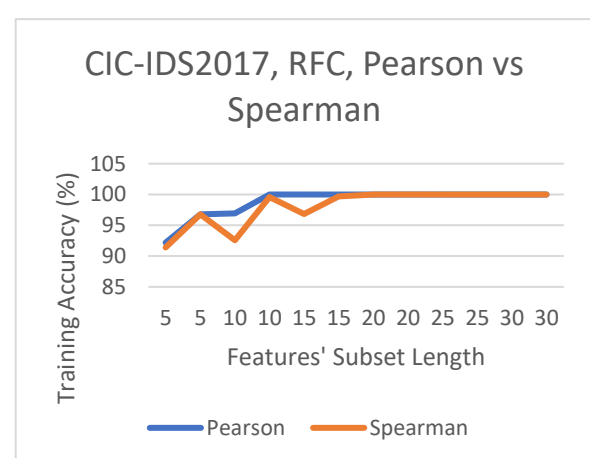
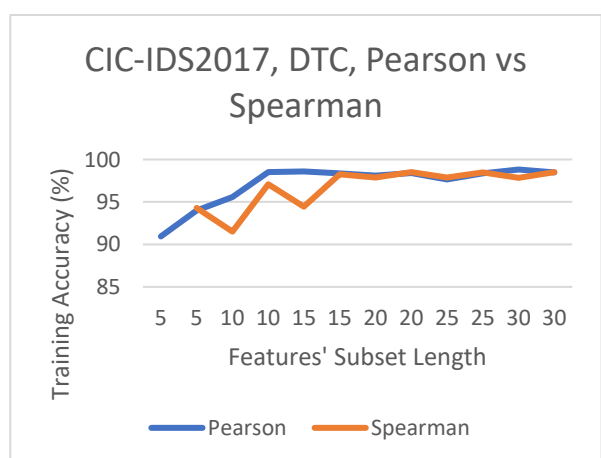
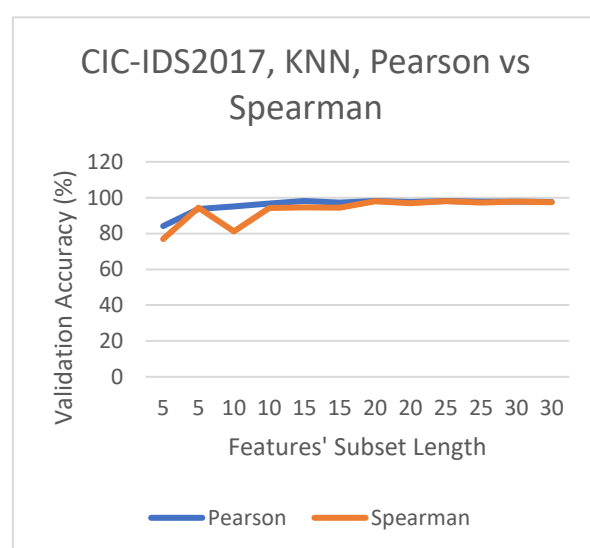
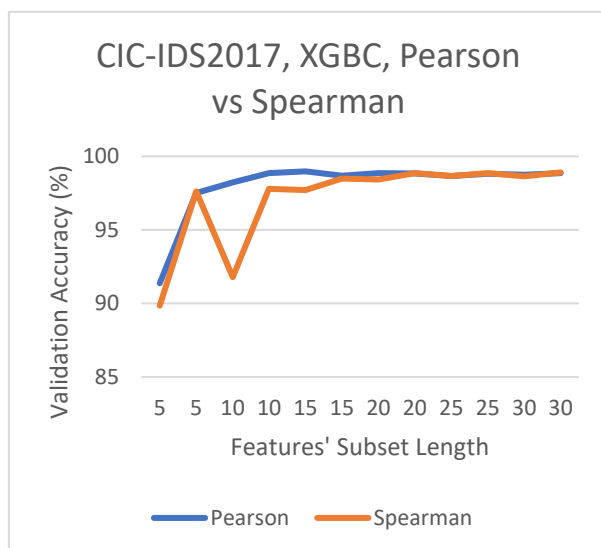
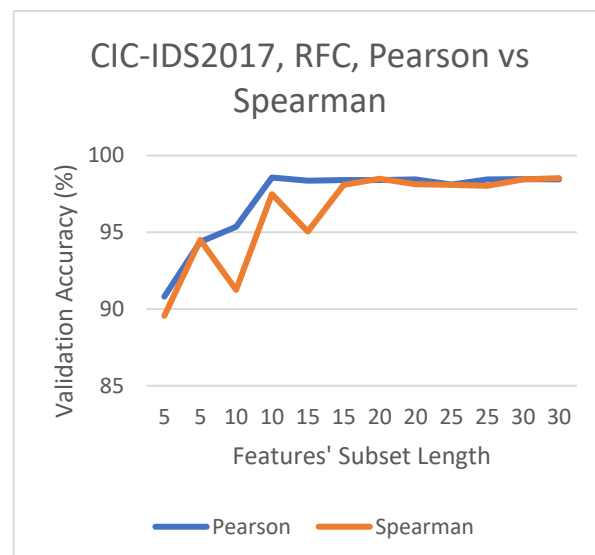
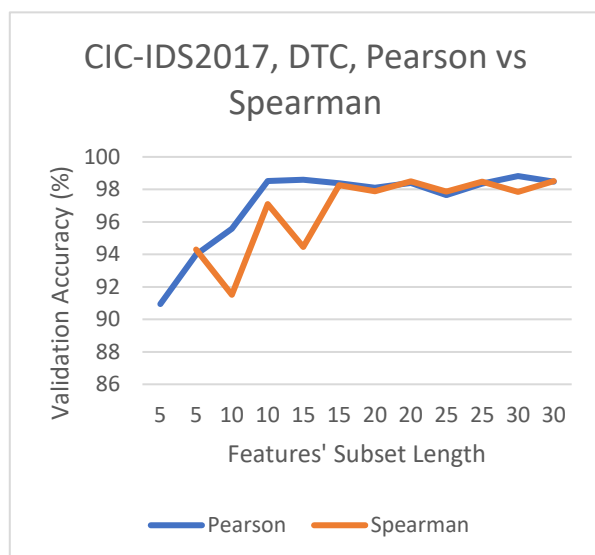
CIC-DDoS2019, KNN, Pearson vs Spearman

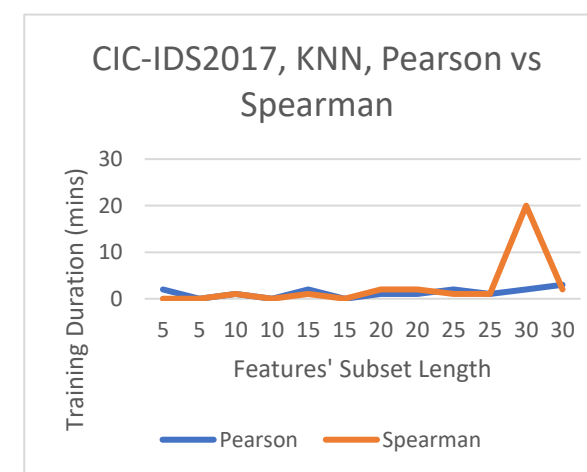
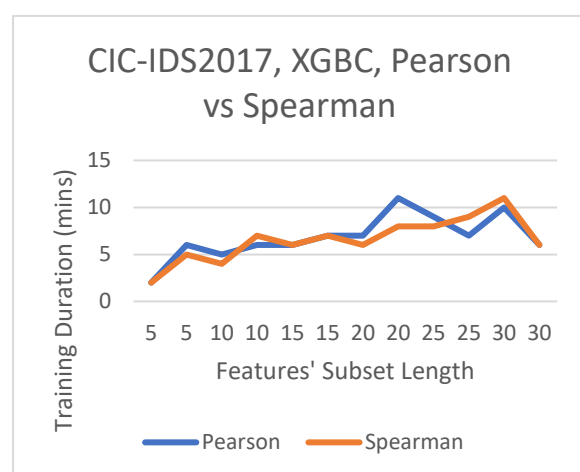
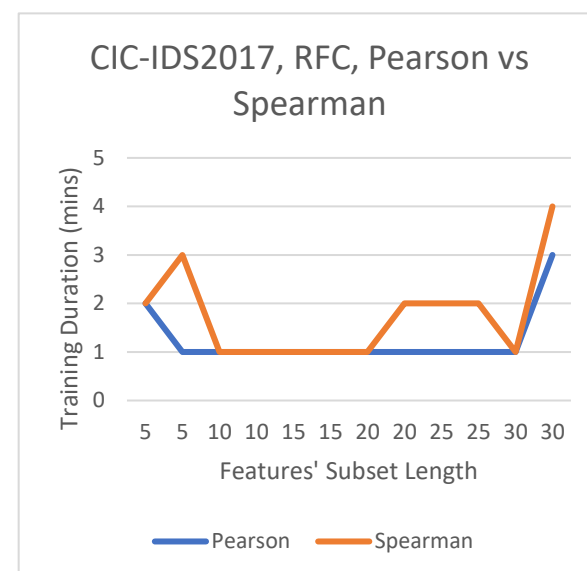
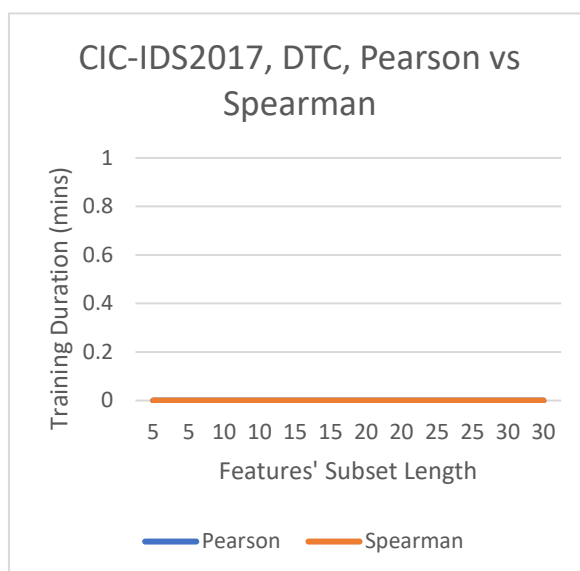
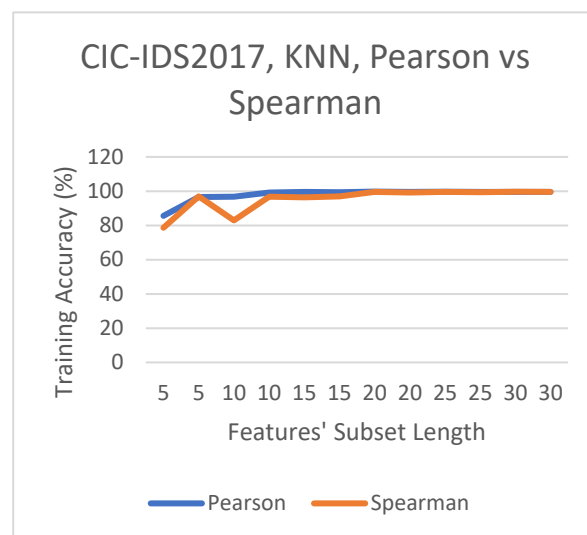
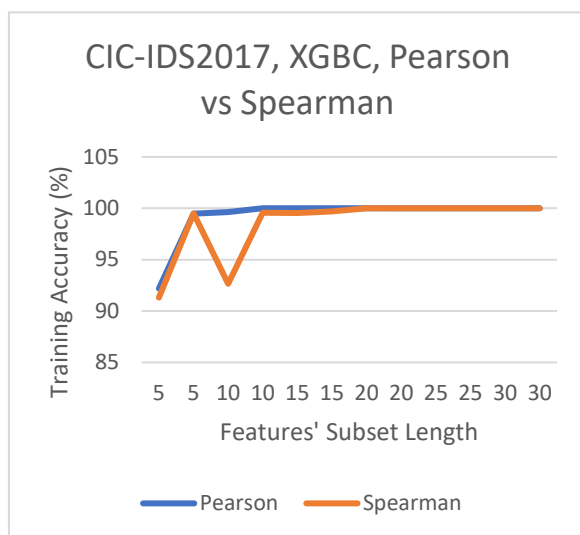


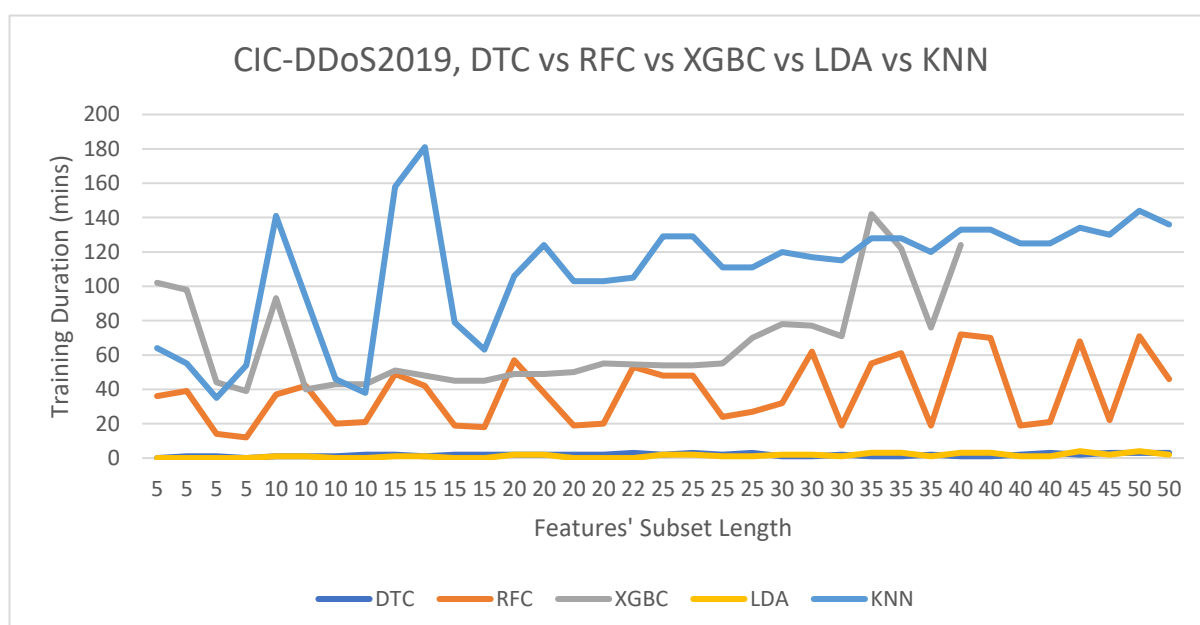
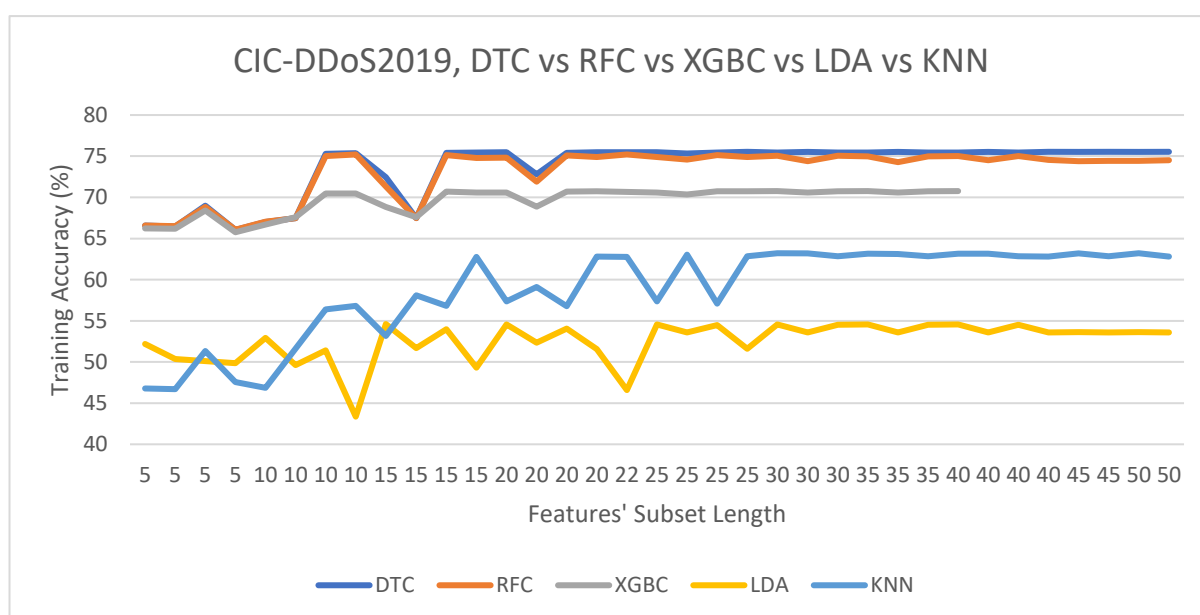
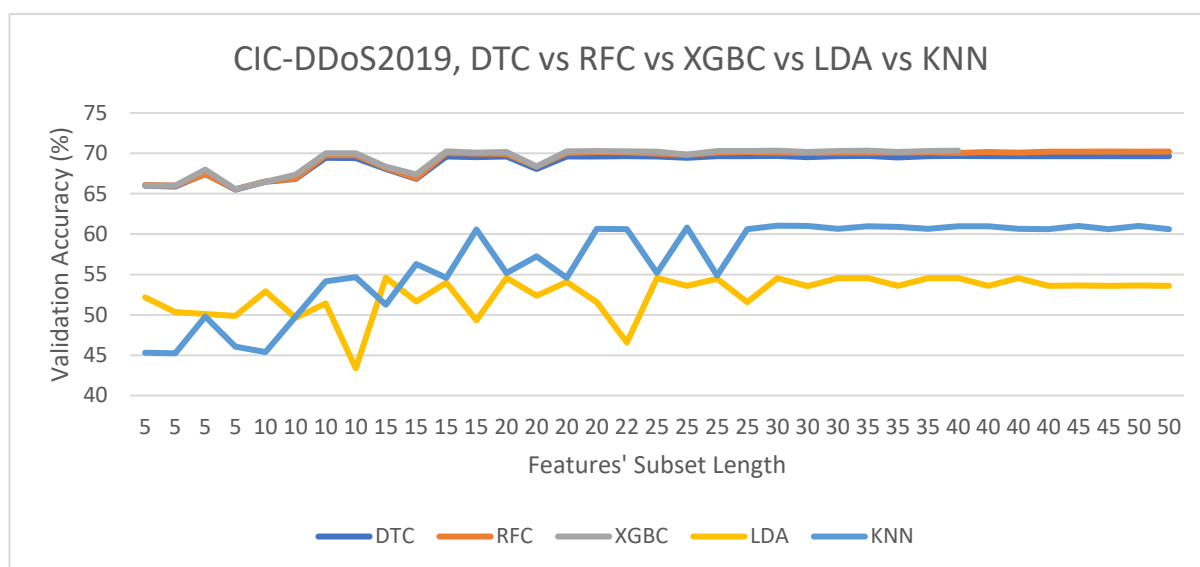
2. CIC-IDS2017 Dataset:

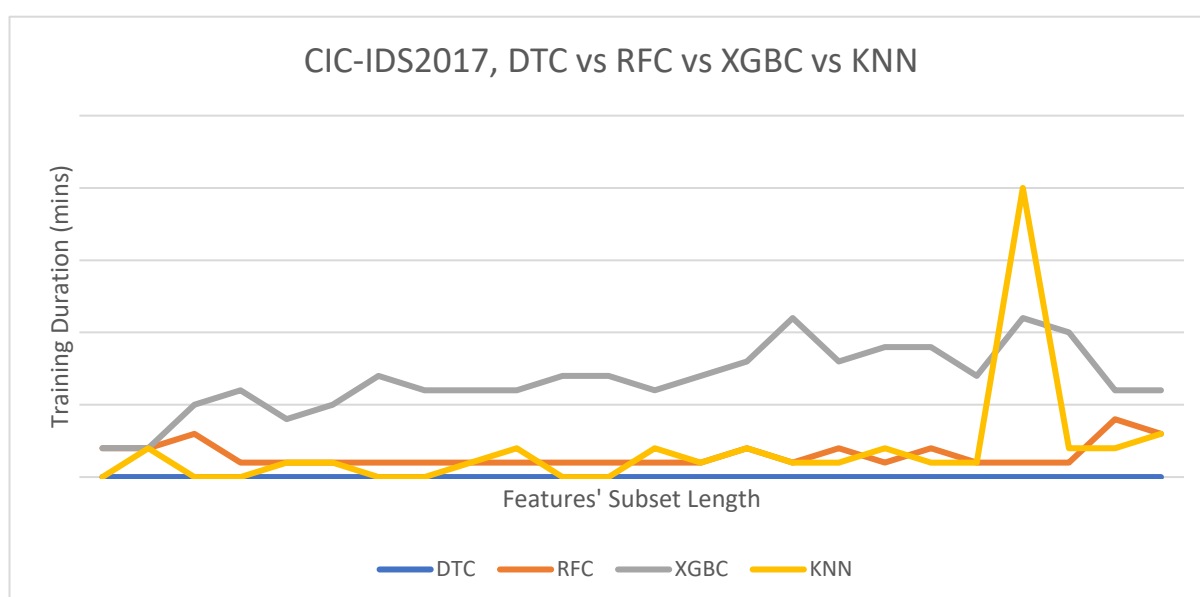
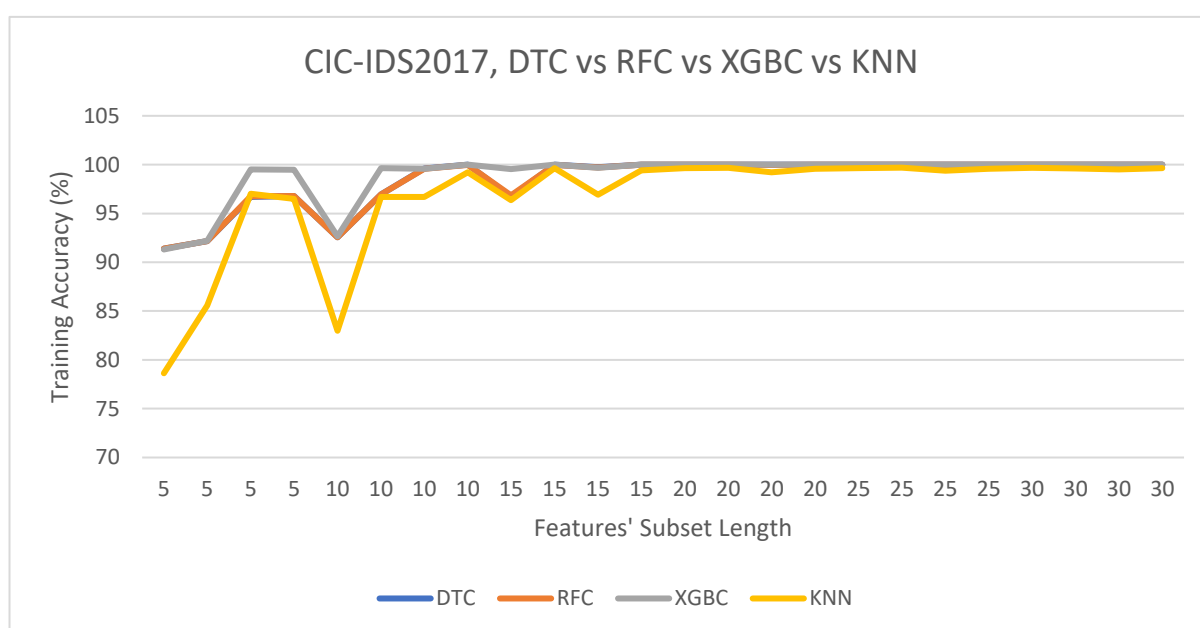
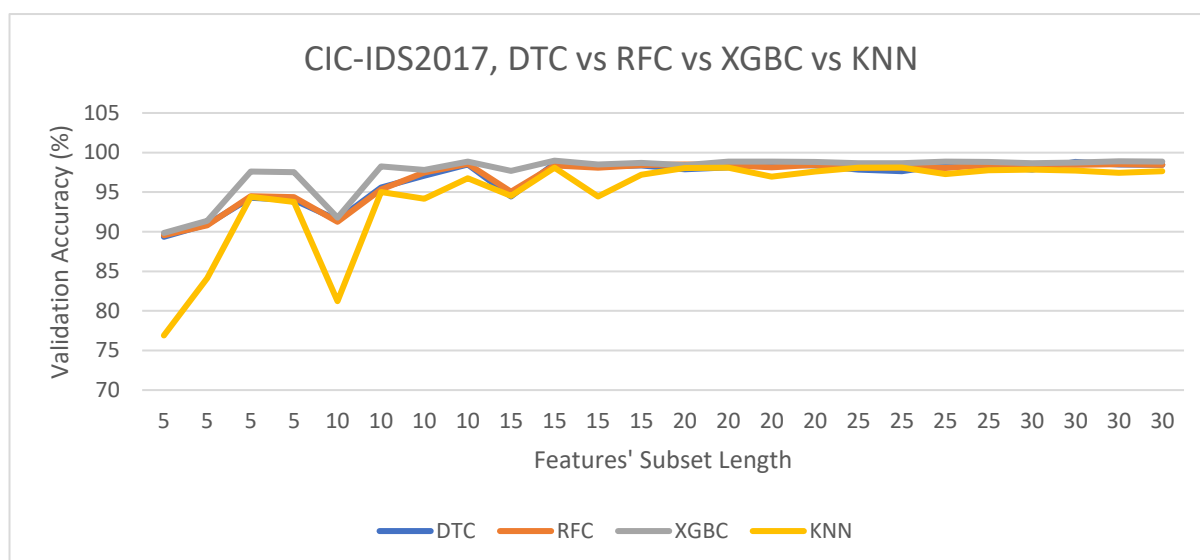






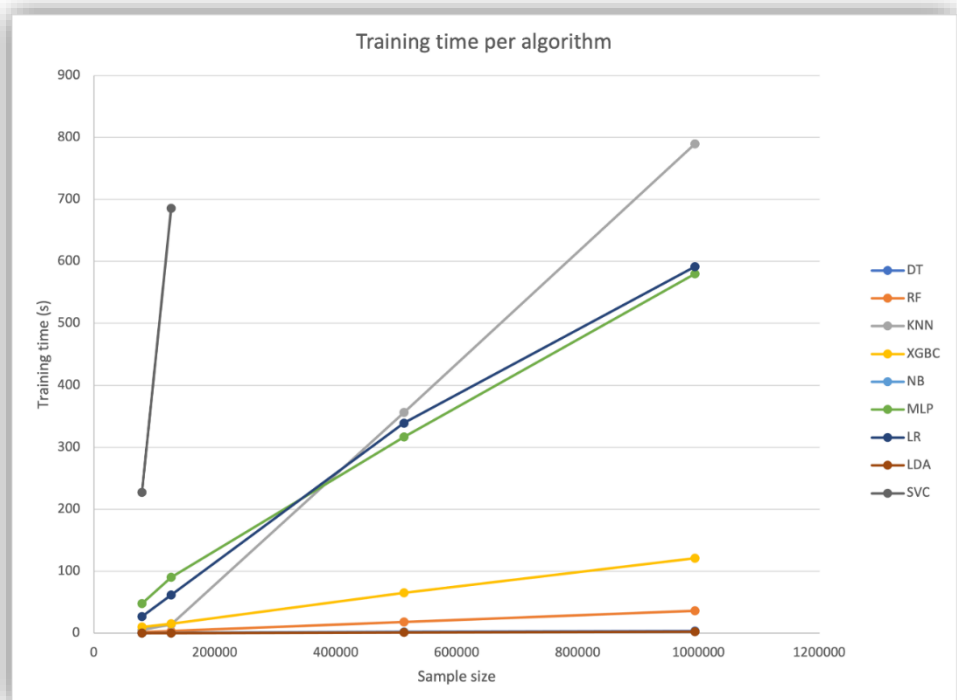


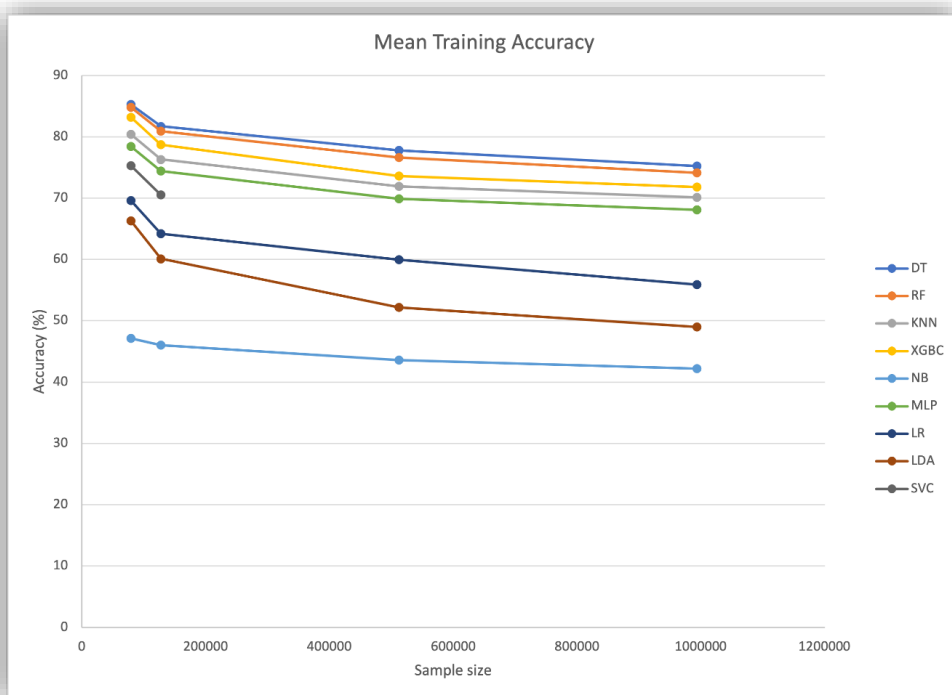
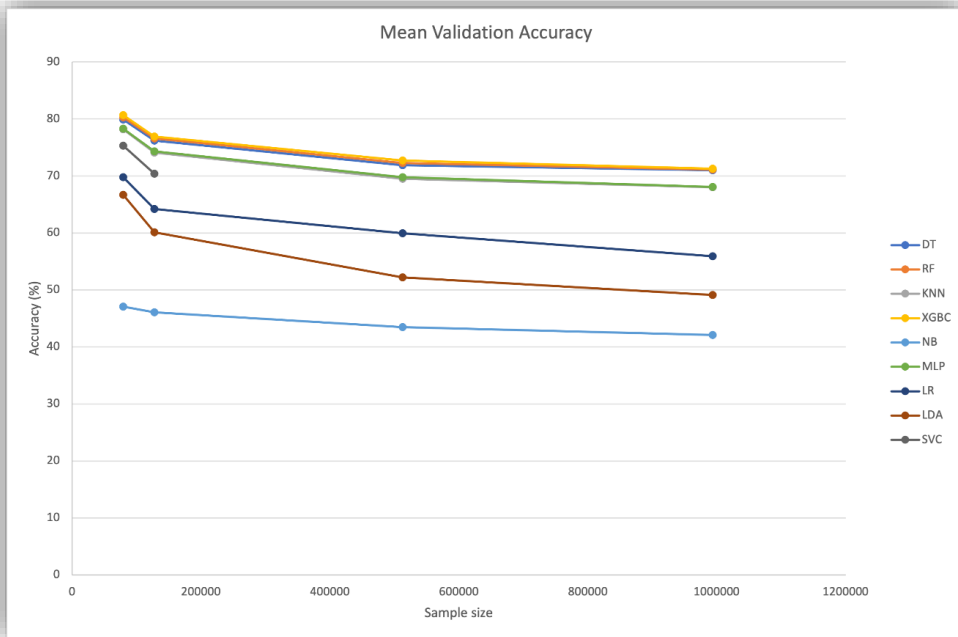




Initial Approach

1) N = 5,000 -> 79,520 samples									
Algorithm name:	Decision Tree	Random Forest	KNN	XGBC	Naive Bayes	MLP	LR	LDA	SVC
Mean training duration(S):	0.232	1.943	5.27	9.791	0.168	47.665	27.047	0.15	227.34
Mean Training accuracy:	0.853	0.848	0.804	0.832	0.471	0.784	0.696	0.663	0.753
Best Training Accuracy:	0.871	0.87	0.821	0.849	0.62	0.807	0.74	0.702	0.783
Best Training Subset:	(S = 16, k = 22)	(S = 16, k = 22)	(S = 8, k = 15)	(S = 16, k = 22)	(S = 12, k = 20)	(S = 16, k = 22)	(S = 13, k = 25)	(S = 12, k = 20)	(S = 16, k = 22)
Mean Validation Accuracy:	0.799	0.803	0.782	0.807	0.471	0.783	0.698	0.667	0.753
Best Validation Accuracy:	0.811	0.815	0.794	0.818	0.621	0.805	0.742	0.708	0.783
Best Validation Subset:	(S = 16, k = 22)	(S = 16, k = 22)	(S = 8, k = 15)	(S = 13, k = 25)	(S = 12, k = 20)	(S = 16, k = 22)	(S = 13, k = 25)	(S = 12, k = 20)	(S = 16, k = 22)
2) N = 10,000 -> 127,642 samples									
Algorithm name:	Decision Tree	Random Forest	KNN	XGBC	Naive Bayes	MLP	LR	LDA	SVC
Mean training duration(S):	0.346	3.213	13.992	14.937	0.291	90.261	61.724	0.252	685.637
Mean Training accuracy:	0.817	0.809	0.763	0.787	0.46	0.744	0.642	0.601	0.705
Best Training Accuracy:	0.836	0.835	0.781	0.803	0.63	0.76	0.683	0.638	0.738
Best Training Subset:	(S = 16, k = 22)	(S = 16, k = 22)	(S = 9, k = 15)	(S = 16, k = 22)	(S = 5, k = 10)	(S = 11, k = 20)	(S = 11, k = 20)	(S = 12, k = 20)	(S = 16, k = 22)
Mean Validation Accuracy:	0.762	0.766	0.741	0.769	0.461	0.743	0.642	0.601	0.704
Best Validation Accuracy:	0.772	0.776	0.756	0.781	0.63	0.761	0.683	0.639	0.739
Best Validation Subset:	(S = 14, k = 25)	(S = 14, k = 25)	(S = 9, k = 15)	(S = 13, k = 25)	(S = 5, k = 10)	(S = 12, k = 20)	(S = 11, k = 20)	(S = 12, k = 20)	(S = 16, k = 22)
3) N = 50,000 -> 512,527 samples									
Algorithm name:	Decision Tree	Random Forest	KNN	XGBC	Naive Bayes	MLP	LR	LDA	SVC
Mean training duration(S):	2.021	17.88	356.408	65.0485	1.6045	316.4855	339.058	1.422	
Mean Training accuracy:	0.778	0.766	0.719	0.736	0.436	0.699	0.5995	0.522	
Best Training Accuracy:	0.786	0.784	0.72	0.739	0.531	0.705	0.615	0.532	
Best Training Subset:	(S = 16, k = 22)	(S = 16, k = 22)	(S = 16, k = 22)	(S = 16, k = 22)	(S = 10, k = 15)	(S = 14, k = 25)	(S = 8, k = 15)	(S = 8, k = 15)	
Mean Validation Accuracy:	0.719	0.723	0.695	0.727	0.435	0.698	0.5995	0.522	
Best Validation Accuracy:	0.722	0.725	0.698	0.729	0.531	0.704	0.615	0.532	
Best Validation Subset:	(S = 16, k = 22)	(S = 12, k = 20)	(S = 8, k = 15)	(S = 12, k = 20)	(S = 10, k = 15)	(S = 12, k = 20)	(S = 8, k = 15)	(S = 8, k = 15)	
4) N = 100,000 -> 993,731 samples									
Algorithm name:	Decision Tree	Random Forest	KNN	XGBC	Naive Bayes	MLP	LR	LDA	SVC
Mean training duration(S):	3.513	36.237	789.689	120.68	2.463	579.727	591.638	2.186	
Mean Training accuracy:	0.752	0.741	0.701	0.718	0.422	0.681	0.559	0.49	
Best Training Accuracy:	0.777	0.774	0.714	0.728	0.552	0.7	0.597	0.532	
Best Training Subset:	(S = 16, k = 22)	(S = 16, k = 22)	(S = 16, k = 22)	(S = 16, k = 22)	(S = 2, k = 5)	(S = 13, k = 25)	(S = 11, k = 20)	(S = 12, k = 20)	
Mean Validation Accuracy:	0.71	0.711	0.68	0.713	0.421	0.681	0.559	0.491	
Best Validation Accuracy:	0.718	0.721	0.691	0.723	0.552	0.699	0.598	0.533	
Best Validation Subset:	(S = 16, k = 22)	(S = 13, k = 25)	(S = 8, k = 15)	(S = 16, k = 22)	(S = 2, k = 5)	(S = 13, k = 25)	(S = 11, k = 20)	(S = 12, k = 20)	





Highly Correlated Features Lists

According to the results of the 1st dataset and Pearson algorithm, out of 74 features, the following 26 features are 95% correlated with at least another feature in the dataset:

1. Fwd Packet Length Min: Fwd Packet Length Max

2. Fwd Packet Length Mean: Fwd Packet Length Max, Fwd Packet Length Min
3. Flow IAT Std: Flow IAT Mean
4. Fwd IAT Total: Flow Duration
5. Fwd IAT Mean: Flow IAT Mean, Flow IAT Std
6. Fwd IAT Std: Flow IAT Mean, Flow IAT Std, Flow IAT Max, Fwd IAT Mean
7. Fwd IAT Max: Flow IAT Max, Fwd IAT Std
8. Fwd IAT Min: Flow IAT Min
9. Bwd IAT Std: Bwd IAT Mean
10. Bwd Header Length: Total Backward Packets
11. Fwd Packets/s: Flow Packets/s
12. Min Packet Length: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean
13. Packet Length Mean: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length
14. RST Flag Count: Fwd PSH Flags
15. ACK Flag Count: Protocol
16. Average Packet Size: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length, Packet Length Mean
17. Avg Fwd Segment Size: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length, Packet Length Mean, Average Packet Size
18. Avg Bwd Segment Size: Bwd Packet Length Mean
19. Fwd Header Length.1: Fwd Header Length
20. Subflow Fwd Packets: Total Fwd Packets
21. Subflow Fwd Bytes: Total Length of Fwd Packets
22. Subflow Bwd Packets: Total Backward Packets, Bwd Header Length
23. Subflow Bwd Bytes: Total Length of Bwd Packets
24. Idle Mean: Flow IAT Max, Fwd IAT Std, Fwd IAT Max
25. Idle Max: Flow IAT Max, Fwd IAT Std, Fwd IAT Max, Idle Mean
26. Idle Min: Idle Mean

According to the results of the 1st dataset and Spearman algorithm, out of 74 features the following 39 features are 95% correlated with at least another feature in the dataset:

1. Fwd Packet Length Min: Fwd Packet Length Max
2. Fwd Packet Length Mean: Fwd Packet Length Max, Fwd Packet Length Min
3. Bwd Packet Length Max: Total Length of Bwd Packets
4. Bwd Packet Length Mean: Total Length of Bwd Packets, Bwd Packet Length Max
5. Flow Packets/s: Flow Duration
6. Flow IAT Mean: Flow Duration, Flow Packets/s
7. Flow IAT Max: Flow Duration, Flow Packets/s, Flow IAT Mean
8. Fwd IAT Mean: Fwd IAT Total
9. Fwd IAT Std: Total Fwd Packets
10. Fwd IAT Max: Fwd IAT Total, Fwd IAT Mean
11. Fwd IAT Min: Flow IAT Min
12. Bwd IAT Total: Total Backward Packets
13. Bwd IAT Mean: Total Backward Packets, Bwd IAT Total
14. Bwd IAT Max: Total Backward Packets, Bwd IAT Total, Bwd IAT Mean
15. Bwd IAT Min: Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max
16. Bwd Header Length: Total Backward Packets
17. Fwd Packets/s: Flow Duration, Flow Packets/s, Flow IAT Mean, Flow IAT Max
18. Bwd Packets/s: Total Backward Packets, Bwd Header Length
19. Min Packet Length: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean
20. Max Packet Length: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length

21. Packet Length Mean: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length, Max Packet Length
22. Packet Length Std: Fwd Packet Length Std
23. Packet Length Variance: Fwd Packet Length Std, Packet Length Std
24. RST Flag Count: Fwd PSH Flags
25. ACK Flag Count: Protocol
26. Average Packet Size: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length, Max Packet Length, Packet Length Mean
27. Avg Fwd Segment Size: Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Min Packet Length, Max Packet Length, Packet Length Mean, Average Packet Size
28. Avg Bwd Segment Size: Total Length of Bwd Packets, Bwd Packet Length Max, Bwd Packet Length Mean
29. Fwd Header Length.1: Fwd Header Length
30. Subflow Fwd Packets: Total Fwd Packets, Fwd IAT Std
31. Subflow Fwd Bytes: Total Length of Fwd Packets
32. Subflow Bwd Packets: Total Backward Packets, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd Header Length, Bwd Packets/s
33. Subflow Bwd Bytes: Total Length of Bwd Packets, Bwd Packet Length Max, Bwd Packet Length Mean, Avg Bwd Segment Size
34. Init_Win_bytes_forward: Protocol, ACK Flag Count
35. Active Max: Active Mean
36. Active Min: Active Mean, Active Max
37. Idle Mean: Active Mean, Active Max, Active Min
38. Idle Max: Active Mean, Active Max, Active Min, Idle Mean
39. Idle Min: Active Mean, Active Max, Active Min, Idle Mean, Idle Max

According to the results of the 2nd dataset and Pearson algorithm, out of 73 features the following 22 features are 95% correlated with at least another feature in the dataset:

1. Fwd Packet Length Std: Fwd Packet Length Max
2. Bwd Packet Length Mean: Bwd Packet Length Max
3. Bwd Packet Length Std: Bwd Packet Length Max, Bwd Packet Length Mean
4. Fwd IAT Total: Flow Duration
5. Fwd IAT Max: Flow IAT Max
6. Bwd IAT Max: Bwd IAT Total
7. Fwd Header Length: Total Fwd Packets
8. Bwd Header Length: Total Backward Packets
9. Fwd Packets/s: Flow Packets/s
10. Packet Length Std: Max Packet Length, Packet Length Mean
11. SYN Flag Count: Fwd PSH Flags
12. ECE Flag Count: RST Flag Count
13. Average Packet Size: Packet Length Mean
14. Avg Fwd Segment Size: Fwd Packet Length Mean
15. Avg Bwd Segment Size: Bwd Packet Length Max, Bwd Packet Length Mean, Bwd Packet Length Std
16. Subflow Fwd Packets: Total Fwd Packets, Fwd Header Length
17. Subflow Fwd Bytes: Total Length of Fwd Packets
18. Subflow Bwd Packets: Total Backward Packets, Bwd Header Length
19. Subflow Bwd Bytes: Total Length of Bwd Packets
20. Idle Mean: Flow IAT Max, Fwd IAT Max
21. Idle Max: Flow IAT Max, Fwd IAT Max, Idle Mean
22. Idle Min: Idle Mean

According to the results of the 2nd dataset and Spearman algorithm, out of 73 features the following 30 features are 95% correlated with at least another feature in the dataset:

1. Fwd Packet Length Max: Total Length of Fwd Packets
2. Fwd Packet Length Mean: Total Length of Fwd Packets, Fwd Packet Length Max
3. Bwd Packet Length Max: Total Length of Bwd Packets
4. Bwd Packet Length Mean: Total Length of Bwd Packets, Bwd Packet Length Max
5. Flow Packets/s: Flow Duration
6. Flow IAT Mean: Flow Packets/s
7. Flow IAT Max: Flow Duration, Flow Packets/s, Flow IAT Mean
8. Fwd IAT Mean: Fwd IAT Total
9. Fwd IAT Max: Fwd IAT Total, Fwd IAT Mean
10. Bwd IAT Mean: Bwd IAT Total
11. Bwd IAT Max: Bwd IAT Total, Bwd IAT Mean
12. Fwd Header Length: Total Fwd Packets
13. Bwd Header Length: Total Backward Packets
14. Fwd Packets/s: Flow Duration, Flow Packets/s, Flow IAT Mean, Flow IAT Max
15. Packet Length Mean: Max Packet Length
16. Packet Length Std: Max Packet Length
17. Packet Length Variance: Max Packet Length, Packet Length Std
18. SYN Flag Count: Fwd PSH Flags
19. ECE Flag Count: RST Flag Count
20. Average Packet Size: Max Packet Length, Packet Length Mean
21. Avg Fwd Segment Size: Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean
22. Avg Bwd Segment Size: Total Length of Bwd Packets, Bwd Packet Length Max, Bwd Packet Length Mean
23. Subflow Fwd Packets: Total Fwd Packets, Fwd Header Length
24. Subflow Fwd Bytes: Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean, Avg Fwd Segment Size
25. Subflow Bwd Packets: Total Backward Packets, Bwd Header Length
26. Subflow Bwd Bytes: Total Length of Bwd Packets, Bwd Packet Length Max, Bwd Packet Length Mean, Avg Bwd Segment Size
27. Active Max: Active Mean
28. Active Min: Active Mean, Active Max
29. Idle Max: Idle Mean
30. Idle Min: Idle Mean, Idle Max

Best Features Lists

Best features of the 1st dataset (CIC-DDoS2019):

*Subset 1(53 features) is the subset originating from the removal of the correlated features identified by the Pearson algorithm, whereas subset 2(40 features) from the removal of the correlated features identified by Spearman. The features are not sorted from the best to worst in each subset and their descriptions can be found in Appendix A

#	Algorithm	K	Subset	Best Features
0	F-Test	5	2	{ Protocol, Fwd Packet Length Max, Flow Bytes/s, act_data_pkt_fwd, Inbound }
1	F-Test	5	1	{ Protocol, Fwd Packet Length Max, Flow Bytes/s, Max Packet Length, Inbound }
2	F-Test	10	2	{ Protocol, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Flow Bytes/s, URG Flag Count, CWE Flag Count, Down/Up Ratio, act_data_pkt_fwd, Inbound }
3	F-Test	10	1	{ Protocol, Fwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Max Packet Length, URG Flag Count, Down/Up Ratio, Init_Win_bytes_forward, act_data_pkt_fwd, Inbound }
4	F-Test	15	2	{ Protocol, Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Fwd PSH Flags, URG Flag Count, CWE Flag Count, Down/Up Ratio, act_data_pkt_fwd, Inbound }
5	F-Test	15	1	{ Protocol, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Mean, Flow Bytes/s, Flow Packets/s, Max Packet Length, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, act_data_pkt_fwd, Inbound }
6	F-Test	20	2	{ Protocol, Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Fwd PSH Flags, Fwd Header Length, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Idle Std, Inbound }
7	F-Test	20	1	{ Protocol, Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Mean, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Fwd PSH Flags, Max Packet Length, Packet Length Std, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, act_data_pkt_fwd, Inbound }
8	F-Test	25	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Bwd IAT Std, Fwd PSH Flags, Fwd Header Length, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
9	F-Test	25	1	{ Protocol, Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max,

				Fwd PSH Flags, Max Packet Length, Packet Length Std, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Idle Std, Inbound }
10	F-Test	30	2	{ Protocol, Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Max Packet Length, Packet Length Std, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Idle Std, Inbound }
11	F-Test	30	1	{ Protocol, Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Max Packet Length, Packet Length Std, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Idle Std, Inbound }
12	F-Test	35	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Bwd IAT Std, Fwd PSH Flags, Fwd Header Length, SYN Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
13	F-Test	35	1	{ Protocol, Flow Duration, Total Backward Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Max, Active Min, Idle Std, Inbound }

14	F-Test	40	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Bwd IAT Std, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Fwd Header Length, FIN Flag Count, SYN Flag Count, PSH Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
15	F-Test	40	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, SYN Flag Count, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Inbound }
16	F-Test	45	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, SYN Flag Count, URG Flag Count, CWE Flag Count, Down/Up Ratio, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Inbound }
17	F-Test	50	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Bwd Packets/s,

				Max Packet Length, Packet Length Std, Packet Length Variance, FIN Flag Count, SYN Flag Count, PSH Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Inbound }
18	Mutual Information Test	5	2	{ Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Flow Bytes/s, act_data_pkt_fwd }
19	Mutual Information Test	5	1	{ Total Length of Fwd Packets, Fwd Packet Length Max, Flow Bytes/s, Max Packet Length, act_data_pkt_fwd }
20	Mutual Information Test	10	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Fwd Header Length, act_data_pkt_fwd }
21	Mutual Information Test	10	1	{ Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Fwd Header Length, Max Packet Length, act_data_pkt_fwd }
22	Mutual Information Test	15	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Fwd Header Length, act_data_pkt_fwd, min_seg_size_forward, Inbound }
23	Mutual Information Test	15	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Fwd Header Length, Max Packet Length, Packet Length Std, Packet Length Variance, Init_Win_bytes_forward, act_data_pkt_fwd }
24	Mutual Information Test	20	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Fwd Header Length, Down/Up Ratio, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Idle Std, Inbound }
25	Mutual Information Test	20	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, Init_Win_bytes_forward, act_data_pkt_fwd, min_seg_size_forward, Inbound }
26	Mutual Information Test	25	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Bwd

				IAT Std, Fwd Header Length, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
27	Mutual Information Test	25	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, Init_Win_bytes_forward, act_data_pkt_fwd, min_seg_size_forward, Inbound }
28	Mutual Information Test	30	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Bwd IAT Std, Fwd PSH Flags, Bwd URG Flags, Fwd Header Length, SYN Flag Count, URG Flag Count, CWE Flag Count, Down/Up Ratio, Bwd Avg Bytes/Bulk, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
29	Mutual Information Test	30	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, Down/Up Ratio, Init_Win_bytes_forward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Max, Active Min, Inbound }
30	Mutual Information Test	35	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Bwd IAT Std, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Fwd Header Length, SYN Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
31	Mutual Information Test	35	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Mean, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT

				Mean, Bwd IAT Max, Bwd IAT Min, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, URG Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Max, Active Min, Idle Std, Inbound }
32	Mutual Information Test	40	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Bwd IAT Std, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Fwd Header Length, FIN Flag Count, SYN Flag Count, PSH Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Idle Std, Inbound }
33	Mutual Information Test	40	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, URG Flag Count, CWE Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Inbound }
34	Mutual Information Test	45	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, URG Flag Count, CWE Flag Count, Down/Up Ratio, Fwd Avg Bulk Rate, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Inbound }

35	Mutual Information Test	50	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Std, Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Flow IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Max, Bwd IAT Min, Fwd PSH Flags, Bwd PSH Flags, Bwd URG Flags, Fwd Header Length, Bwd Packets/s, Max Packet Length, Packet Length Std, Packet Length Variance, SYN Flag Count, URG Flag Count, CWE Flag Count, ECE Flag Count, Down/Up Ratio, Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate, Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min, Idle Std, Inbound }
----	-------------------------	----	---	--

Best features of the 2nd dataset (CIC-IDS2017):

*Subset 1(38 features) is the subset originating from the removal of the correlated features identified by the Spearman algorithm, whereas subset 2(43 features) from the removal of the correlated features identified by Pearson. The features are not sorted from the best to worst in each subset and their descriptions can be found in Appendix A.

#	Algorithm	K	Subset	Best Features
0	F-Test	5	1	{ Total Fwd Packets, Total Length of Bwd Packets, Bwd Packet Length Std, PSH Flag Count, min_seg_size_forward }
1	F-Test	5	2	{ Total Fwd Packets, Bwd Packet Length Max, Packet Length Mean, PSH Flag Count, min_seg_size_forward }
2	F-Test	10	1	{ Total Fwd Packets, Total Length of Bwd Packets, Bwd Packet Length Std, Flow IAT Std, Fwd IAT Std, Max Packet Length, Packet Length Mean, PSH Flag Count, min_seg_size_forward, Idle Mean }
3	F-Test	10	2	{ Protocol, Flow Duration, Total Fwd Packets, Bwd Packet Length Max, Flow IAT Std, Max Packet Length, Packet Length Mean, PSH Flag Count, min_seg_size_forward, Active Mean }
4	F-Test	15	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Length of Bwd Packets, Bwd Packet Length Std, Flow IAT Std, Fwd IAT Total, Fwd IAT Std, Max Packet Length, Packet Length Mean, PSH Flag Count, URG Flag Count, min_seg_size_forward, Active Mean, Idle Mean }
5	F-Test	15	2	{ Protocol, Flow Duration, Total Fwd Packets, Bwd Packet Length Max, Flow IAT Std, Bwd IAT Mean, Max Packet Length, Packet Length Mean, Packet Length Variance, PSH Flag Count, URG Flag Count, min_seg_size_forward, Active Mean, Active Max, Active Min }

6	F-Test	20	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Bwd Packets, Bwd Packet Length Std, Flow IAT Std, Fwd IAT Total, Fwd IAT Std, Bwd IAT Min, Max Packet Length, Packet Length Mean, PSH Flag Count, ACK Flag Count, URG Flag Count, Init_Win_bytes_forward, min_seg_size_forward, Active Mean, Active Std, Idle Mean }
7	F-Test	20	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Bwd Packet Length Max, Flow IAT Mean, Flow IAT Std, Bwd IAT Mean, Bwd IAT Min, Max Packet Length, Packet Length Mean, Packet Length Variance, PSH Flag Count, ACK Flag Count, URG Flag Count, Init_Win_bytes_forward, min_seg_size_forward, Active Mean, Active Max, Active Min }
8	F-Test	25	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Bwd Packets, Bwd Packet Length Min, Bwd Packet Length Std, Flow IAT Std, Fwd IAT Total, Fwd IAT Std, Fwd IAT Min, Bwd IAT Min, Fwd PSH Flags, Min Packet Length, Max Packet Length, Packet Length Mean, FIN Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, Init_Win_bytes_forward, min_seg_size_forward, Active Mean, Active Std, Idle Mean }
9	F-Test	25	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Bwd Packet Length Max, Flow IAT Mean, Flow IAT Std, Fwd IAT Mean, Bwd IAT Mean, Bwd IAT Min, Fwd PSH Flags, Min Packet Length, Max Packet Length, Packet Length Mean, Packet Length Variance, FIN Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, Init_Win_bytes_forward, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min }
10	F-Test	30	1	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Bwd Packets, Bwd Packet Length Min, Bwd Packet Length Std, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Fwd IAT Std, Fwd IAT Min, Bwd IAT Total, Bwd IAT Std, Bwd IAT Min, Fwd PSH Flags, Bwd Packets/s, Min Packet Length, Max Packet Length, Packet Length Mean, FIN Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, Down/Up Ratio, Init_Win_bytes_forward, min_seg_size_forward, Active Mean, Active Std, Idle Mean }
11	F-Test	30	2	{ Protocol, Flow Duration, Total Fwd Packets, Total Backward Packets, Fwd Packet Length Mean, Bwd Packet Length Max, Bwd Packet Length Min, Flow IAT Mean, Flow IAT Std, Flow IAT Min, Fwd IAT Mean, Fwd IAT Min, Bwd IAT Mean, Bwd IAT Min, Fwd PSH Flags, Min Packet Length, Max Packet Length, Packet Length Mean, Packet Length Variance, FIN Flag Count, PSH Flag Count, ACK Flag Count, URG Flag Count, Down/Up Ratio, Init_Win_bytes_forward, min_seg_size_forward, Active Mean, Active Std, Active Max, Active Min }
12	Mutual Information Test	5	1	{ Flow Duration, Total Length of Fwd Packets, Flow IAT Max, Packet Length Mean, Packet Length Variance }
13	Mutual Information Test	5	2	{ Flow Duration, Total Length of Fwd Packets, Flow Bytes/s, Fwd IAT Total, Max Packet Length }
14	Mutual Information Test	10	1	{ Flow Duration, Total Length of Fwd Packets, Fwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Max, Max Packet Length, Packet Length Mean, Packet Length Variance }
15	Mutual Information Test	10	2	{ Flow Duration, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Bwd Packets/s, Max Packet Length, Init_Win_bytes_forward }
16	Mutual Information Test	15	1	{ Flow Duration, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow

				IAT Std, Flow IAT Max, Fwd IAT Mean, Bwd Packets/s, Max Packet Length, Packet Length Mean, Packet Length Variance }
17	Mutual Information Test	15	2	{ Flow Duration, Total Fwd Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Std, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Fwd IAT Std, Bwd IAT Total, Bwd Packets/s, Max Packet Length, Init_Win_bytes_forward, Init_Win_bytes_backward }
18	Mutual Information Test	20	1	{ Flow Duration, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean, Bwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Fwd IAT Mean, Fwd IAT Std, Bwd IAT Total, Bwd IAT Mean, Bwd Packets/s, Max Packet Length, Packet Length Mean, Packet Length Variance, Init_Win_bytes_forward }
19	Mutual Information Test	20	2	{ Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Std, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Fwd IAT Total, Fwd IAT Std, Fwd IAT Min, Bwd IAT Total, Bwd IAT Std, Bwd Packets/s, Max Packet Length, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward }
20	Mutual Information Test	25	1	{ Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean, Bwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Std, Bwd Packets/s, Max Packet Length, Packet Length Mean, Packet Length Variance, Init_Win_bytes_forward, Init_Win_bytes_backward }
21	Mutual Information Test	25	2	{ Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Fwd IAT Std, Fwd IAT Min, Bwd IAT Total, Bwd IAT Std, Bwd IAT Min, Bwd Packets/s, Max Packet Length, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Idle Mean }
22	Mutual Information Test	30	1	{ Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Max, Fwd Packet Length Mean, Bwd Packet Length Max, Flow Bytes/s, Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Min, Bwd IAT Total, Bwd IAT Mean, Bwd IAT Std, Bwd Packets/s, Max Packet Length, Packet Length Mean, Packet Length Variance, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Active Max }
23	Mutual Information Test	30	2	{ Flow Duration, Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Total Length of Bwd Packets, Fwd Packet Length Min, Fwd Packet Length Std, Bwd Packet Length Min, Bwd Packet Length Std, Flow Bytes/s, Flow IAT Std, Flow IAT Min, Fwd IAT Total, Fwd IAT Std, Fwd IAT Min, Bwd IAT Total, Bwd IAT Std, Bwd IAT Min, Bwd Packets/s, Min Packet Length, Max Packet Length, PSH Flag Count, Down/Up Ratio, Init_Win_bytes_forward, Init_Win_bytes_backward, act_data_pkt_fwd, min_seg_size_forward, Active Mean, Idle Mean, Idle Std }