# Predicting DDos Attacks: An Analysis on Features and Learning

Michael Telahun

Octorber 29, 2020

**Abstract**

DDoS or Distributed Denial of Service is a common attack occurring throughout the world across the internet. Year over year these attacks become more harmful to businesses, harder to identify, and reach larger populations. Capturing the gradual and sudden changes of network traffic that signify a DDoS attack are one of the reasons which make them difficult to ascertain from benign internet traffic. Additionally, internet traffic is largely filled with nominal or benign traffic therefore the imbalanced nature of the data must also be considered when attempting to classify this kind of malicious web traffic. In this work we analyze and review the features in a public dataset for DDoS and benign internet traffic. The principle goal of this work is delving into the information present in this dataset, understand why or why not features may present themselves as characteristic to DDoS or benign traffic, and evaluate the most pertinent features in forming classification models. The analysis is based on the hypothesis that there must be subsets of features within the dataset that give strong conclusions of whether a record is benign or DDoS. The DDoS dataset originally contains 84 features in total, this amount of information also possess the issue of highly complicated dimensionality therefore methods of analysis will include the reduction of these features in order to explain which features were utilized in the classification models mentioned here. The metric for success in this work for correctly detecting DDoS traffic is F1-Score but other metrics are also discussed and mentioned here. In results we will show that the best F1-Score achieved by our models was 85% using a simple decision tree classifier with only three features: *'Flow IAT Max', and 'Bwd IAT Tot'*. These results show the ability of feature reduction, feature analysis, and parameter tuning using grid searches. We show these results with specify the rational of having a highly explainable model rather than a high dimensional and abstract solution to this problem which may rely to heavily on training and testing data. Additionally, from the initial dataset of 100,000,000 records we train on 1,400 samples showing the robustness of the final model presented here. In total we implement five methods for classifying the dataset as DDoS or Benign: *KMEANS, Decision Trees, Support Vector Machines, and Neural Networks.* We explain why we choose Decision Trees aside from having the best F1-Score of all the models as the overall choice after developing in full the discussion of these models and how there were implemented.

# 1 Predicting DDos Attacks: An Analysis on Features and Learning

## Introduction

DDoS attacks are one of the many plaguing issues on the internet today. As data continues to grow at unprecedented rates, some estimating internet traffic growth doubling annually, it is imperative that government entities, conglomerates, small businesses, and even individuals be prepared for insurmountable attacks at increasing frequencies. Cyber-attacks are also growing increasingly cunning with a vast number of tools publicly available and websites with the intention of generating harmful activity as a service [ref]. Almost as incentive, cyber-attackers find themselves with a great opportunity to incur harm on those who are rapid to adopt new technologies as the age of cloud computing has risen and steadily grown over the last couple decades. The exploitabiliy of new technologies that have not extensively been studied presents a scenario for attackers to find more methods to endanger private and public data via DDoS attacks. In many cases attacks are targeted at businesses causes servers or websites to crash due to loads that are impossible to load balance with implemented infrastructure. Some of the more notable attacks have been on government institutions, websites such as GitHub and Yahoo, and banks like PNC, Bank of America, and Wells Fargo [ref].

Perhaps the most infamous attacks of these type are Denial of Services (DoS) or in the disseminating form Distributed Denial of Service (DDoS). A DDoS attack occurs by over loading or flooding a system such that networks and servers error and shut down. These attacks intentionally lead to a loss of information, finances, and often confidence in either the target entity (individual or business) [ref]. Perhaps those at the most risk are entities that have largely migrated to cloud technologies for the majority of their data storage and application serving/hosting. That being said, business such as law firms who are often slow to adopt new technologies due to security, are still at risk.

Despite the growing number of attacks and sophistication of methods for these attacks organizations internally are growing their security to deal and prepare for the inevitable [ref]. The work here not only includes increased security measures and increased number of server nodes as preventive measure or load removal post attack as complacent but also by continuously monitoring web traffic that for indications of potential attacks [ref]. The work here has been pursued by many in the research areas of computer security. The study has become integrated into mostly the realm of cybersecurity as DDoS attacks are specific to network and domain security systems. The goal of cybersecurity, a multidisciplinary topic or field, in terms of DDoS attacks is to provide a defense in all three stages of an attack: before, during, and after, so that a business can maintain their systems on a day to day basis, immediately identify when to fight back, and ultimately find new and increasingly sophisticated methods to proactively defend a business in future attacks.

The business of researchers specifically in the realm of computer science and cybersecurity that we mainly discuss here is the identification and ultimately the classification of the DDoS attacks from a stream of network data. The classification of DDoS attacks has been presented for many systems in offline formats, online real time, for Hadoop specific systems, Apache specific systems, asdf among others [ref, ref, ref]. The goals of these method and the problems faced are often unique but have some shared underlying issues. Perhaps the most unifying issue across datasets is the highly imbalanced information in favor of benign or non-DDoS network traffic. This is a rather intuitive feature as the majority of internet traffic is not targeted attacks at entities, rather businesses and individuals utilizing resources [ref]. Another is the highly dimensional nature of information that is captured in a network that relates to both benign and DDoS behavior. In the dataset we will discuss and utilize in this work there is over eighty features in its raw form. In contrast there are datasets which lower dimensional space which could just as easily capture both DDoS and benign behavior, the nature of dimensionality in this domain is highly dependent what information is captured which is specific to the capabilities of the system at question [ref].

Learning methods we will develop in this work are aimed at predicting whether web traffic is a part of an attack or normal benign traffic. We employ mainly four methods for the prediction of network traffic: SVMs, KMEANS, Decision Trees, and a deep learning or Artificial Neural Network (ANN). We discuss these methods in terms of their ability to classify the dataset, which we prepare, and develop a simple explanation for each. The work here also attempts to select and prepare features from the dataset that are best for training a predictive model in high classification of DDoS network traffic. As the dataset is large there is a high probability of learning the dataset and not the problem, or preforming

well in terms of the training and testing data but not in live or real world scenarios. The dataset used is a synthetic or reproducible dataset which is considered the best dataset which captures DDoS attacks in terms of their behavior. That said anomalies occur and datasets which appear highly descriptive may or may not be as relevant when utilized for live procedures.

This dataset comes from a flow generator CICFlowMeter previously ISCXFlowMeter. The University of New Brunswick hosts this dataset utilizing AWS. CICFlowMeter is described as a generator for bidirectional flows (Biflow) where more than eighty statistical network traffic features can be calculated separately [ref]. The data can be generated with a large number of configurable parameters to create specific behaviors in the data. Biflows are based on the first packet which determines the source to destination and destination to source directions for the data [ref]. The dataset is generated in the form of a CSV. The features of the data are discussed in detail and their applicability to the problem are also included. A dataset is already made publicly avaialbe for immediate use in terms of classification. The generator was created in collaboration with the computer science and engineering (CSE) department at New Brunswick and the Canadian Institute for Cybersecurity (CIC). The premise of the project from which this dataset generator was created for is anomaly detection. Briefly, concept, anomaly, or novelty detection has held the interest of many, with different detectors arising over time leveraging both machine learning techniques and mathematical modeling [ref me]. As mentioned above there are many issues that plague the successful and accurate detection of DDoS attacks, many of these issues stem from the issues developed and defined in anomaly detection. The goal of this work is not to perform anomaly detection as is used in [ref, ref, ref. ref] but it is must be mentioned that the work performed here is done on data that is prepared with the intention of strictly prediction. The work related to anomaly detection typically precedes the classification or prediction phase of the work load and is its own class of research. We do not delve deep into these methods except in relationship to a full system design intended for a potential future application which is discussed in the conclusion section of this work.

## 1.1 Problem Description

The goal of this project was to carry to classify generated network traffic into two classes, benign or DDoS (Distributed Denial of Service). The topic of DDoS attacks has been generally developed in the introduction. Briefly, DDoS attacks are samples of malicious network data which are aimed at overloading or flooding a network server such that they cannot operate in their usual circumstance. This has been done maliciously to organizations such as PNC Bank, Bank of America, Yahoo, and GitHub. These types of attacks are growing in their complexity which takes this a very relevant contemporary topic. As these attacks become increasingly sophisticated, new and well developed methods are needed to handle them. Many of these methods use some learning technique such as Machine Learning to understand the features in both benign and non-benign data. In order to fully utilize any learning method data must be prepared and understood otherwise a monkey throwing a dart would have just as much success calling a function from a package and getting 80% accuracy. On top of this, we must understand which metrics may be best to evaluate the model as the "no free lunch" theorem states accuracy is not necessarily the best metric in every classification scenario. The dataset here was designed with records of "benign" or non DDoS labeled data and "DDoS" records which are labeled to be DDoS related in nature. We There are several problems or characteristics of the dataset which must be analyzed in order for a reasonable classification model can be built. The analysis and preprocessing of the data are a large part of the problem design. Much of the work presented here is based on developing an understanding of the dataset to facilitate reasonably well models that are able to classify dataset. The rest of the work her was done to determine which model could learn to perform best on the dataset. The "best" model will be judged on F1-Score primarily as that is the requirement for success in this work, but we additionally mention other metrics such as Accuracy, Precision, and Recall. We do post analysis on a test sample created from the dataset provided.

## 1.2 Review of publications

Many have found DDoS attacks at the focal point of their carriers, not just in academia, because it is a major problem that is threatening essentially every business that has cloud infrastructure or an online presence. That is to say there is much work being done in cybersecurity pertaining to DDoS attacks and it is only growing more relevant as the internet continues to grow. We review several relevant and recent works in the area for classification of DDoS attacks using learning methods. We discuss in part what we have learned from these methods and what we have done in comparison to some of their work. Additionally

we discus how these classification techniques have performed.

Osanaiye et al propose a method for DDoS detection in cloud computing based on ensemble learning named EMFFS for ensemble-based multi-filter feature selection. The combine information from information gain, gain ratios, chi-squared and ReliefF to select relevant features in their data [6]. These methods are combined to determine a one-third split of the ranked features where the result dataset ends with a total of 14 features and a threshold is set to determine which of these 14 features are selected for the final classification model. However, their results state 13 features as the best result. Their chosen model for classification is a decision tree which is based on a greedy version of divide and conquer. The decision tree is developed further in the methods of this work as it is the same method for our best model. Osanaiye et al use the NSL-KDD dataset which is a labelled benchmark for intrusion detection. It is an imbalanced dataset which makes it a strong candidate for reasonable benchmarking. The dataset contains 41 features [8] and was originally generated by the same group from the University of New Brunswick the work here is based upon. The development of these models is very well developed and explain much about how feature selection should be done when stepping through datasets of this nature. DDoS experimental results given are as high as 99.67% in accuracy, the authors do not mention any F1-Score in their work. They mainly focus time to compute as this work is heavily based on the cloud performance, false alarm rate or false positive ratio, and detection rate. They are able to achieve 0.42% false alarm rates in under a second based on their design of EMFFS. It should be mentioned that information gain alone was able to achieve almost the same performance with one additional feature [6]. They show that relief is least accurate and has the highest false alarm rate of all their preproccesing methods. Their methodology is very strong as it incorporates several methods in an ensamble fashion and they give very good results overall.

Deep learning methods are also one route many have taken in the classification of DDoS network traffic as it has certainly made waves with its ability to achieve high classification accuracy. Ujjan et al utilize a Stacked Autoencoder (SAE) which is an unsupervised method in a real time environment to detect DDoS attacks with high efficiency [9]. The methods were created to be applied in a system utilizing SNORT which is an open source Intrusion Detection System (IDS) that "sniffs" network packets for any harmful anomalies in the data. The goal of this work is to achieve higher accuracy, lower memory utilization, and lower overhead consumption which they show as being an issue in Software Defined Networking (SDN). In this work they collect their own dataset so it is hard to assess their work in comparison with others as their data is not testable. They describe in detail the method by which they collected it with enough clarity to have reproducible environment to collect a similar dataset. In general they use adaptive polling and a set of network packages to gather data into a csv which can be used for training and testing. The process here almost leaves not much room for preprocessing as the data is essentially hand created with their intention of training a classifier from the network with little overhead. They design a lot of work for the setup and design of information collection and processing of data in terms of the open source tools which are specific to network traffic and general mythologies. They simply mention that they used common features from literature and in total used 22 features. They consider their results more when varrying the sampling rate of sFlow agents when collecting data [9]. They briefly mention a simple stacked auto encoder they implemented using Tensorflow 1.4. The design is as following: 3 hidden layers (these are note the input or output layers), learning rates of 0.1, 0.01, 0.001 which they implement over several test fit the best model for their data, in the output layer they use a sigmoid function as their activation function, they use dropout at 0.75 and 0.50 which a method for "forgetting" training samples which may become biased in learning causing overfitting, and finally batch sizes of 100 and 50 which often may not appear as influential to model but is essentially the scope of information a model is able to consume. They evaluate their model largely based on true positive and false and false positive ratios. They include R-value and F-Measure or F1-Score

Bhaya et al [2] show a method for efficient clustering analysis in large datasets. Th name of there method is EM-CURE. EM being an Entropy Method and cure from clustering using REpresentative. They develop this method as a proactive method for intrusion detection. The goal of EM-Cure is in general predicting DDoS attacks by capturing entropy windows from the network packet header information. They mention that they use of cure is specific to problems that are non-spherical in shape which is good for data with outliers. This method is developed by using a set of points to generate a clustered sample and then a shrink factor is applied to shift the points in cluster to the centriod. They mentioned that the DARPA features are specified in generating the entropy windows and leave much of the discussion for preprocessing out instead refer to the method for entropy windows which they give little description about except for a graph describing the entire system. The results are impressive with an F-Measure of 97.98% for their best model. They show very low records in terms of false positives and false negatives meaning there is little to no revision required

for this mythology if implemented in an actual system. They compare their mythology with several others that have been proposed in the same problem space. They show an SVM with 97.25% accuracy but do not share the F-Measures of this or any other model in their related comparisons [2]. This is one reason we include the SVM model in our work here. We also consider that their clustering method is heavily relient on their system design or architecture for the proposed EM-CURE method. That is to say the cluster methodology alone will most likely not easily classify ddos attack unless they are in the right feature space such as the one they describe using their model.

## 1.3 Software Tools

This project was entirely done using Python 3.8.5 and the packages that can be install using the python package management system or PIP. To perform preprocessing of data libraries such as numpy, pandas and scikit-learn were used. The numpy library was used to perform simplex matrix operations. Pandas was used for dataframes as dataframes are are the typical format for preprocessing information in modern programming langugues. I used Jupyter notebook and Visual Studio Code for writing the code, work was done using Winodows OS. The majority of work is done using Scikit-Learn also known as SKLearn or Scientific Kit for Learning and submodules within it. Scikit-Learn was is intended to simplify most data mining for out of the box use and is very robust in its implementation and included submodules [7]. One of these submodules used specifically was Imblanced Learn which is a scikit-learn package which is specifically aimed at imbalanced dataset preprocessing and learning methods. These two packages essentially are you need to do preprocessing for this work, majority of work is done simply by using the packages and utilizing the properties contained on the objects. For example decision trees can be created by scikit-learn by initialized the object with the parameters you would like to fit the model with, the decision tree can be fit using the '.fit()' function and after the model is fit the model's characteristics can be ascertain by calling one of the available properties, prediction can be performed by simply calling the predict function with the testing dataset, and the post analysis (precision recall, F1-Score, and Accuracy) can be generated via the function 'classification_report' [3].

## 1.4 Preprocessing

We discuss in detail the generation and collection of the data in the introduction and continue here with the development of the rational for the processed dataset used in this work. We initially begin with a dataset of 84 features and a binary class "Label" feature which is marked "Benign" and "ddos". As mentioned one over arching requisite for this work is the classification of the dataset using learning methods. Because often learning methods are not able to process or handle categorical strings of data the first step in the preprocessing phase involves mapping the *Label* feature to binary. As we only have two classes we map {"ddos", "Benign"} to {0, 1}. This is done to simplifiy the model additionally as often models will generate a sparse probability matrix for the predicted class labels for example a label fed as {1} and the corresponding output if correctly predicted may look something like [0.2, 0.8] or [0.8]. The predicted class label is given as {1} in the first because the highest indexed value of 0.8 is at index 1. In the second the prediced class label is given because it is greater then some threshold such as 0.5 in binary classification.

Following the change of class label we take a broadened view at the dataset considering the total size of the data in terms of records. In total the dataset provided contains 760,427 records. These are single samples with labels either "ddos" or "benign" intially. We consider these number of records in relation to the two classes. In much of the related work we have mentioned datasets are highly imbalanced or skewed in terms of one class or generally subset of classes over others. The problem we face with unbalanced data can be over simplified to simply predicting the more populous class. This is because a model that fits any function to data has the inherent design by some iterative process. In deep learning stochastic gradient descent is one such method where data attempts to sift through local minima for a global or close to global minimum. These models tend to converge quickly because of how the descent is designed to converge at certain rates. This can be controlled by many parameters such as learning rate and cost function. To remove this issue in best case we can totally balance both classes (undersampling) or generate samples (oversampling) to the smaller class using methods such as Synthetic Minority Over-sampling Technique (SMOTE). SMOTE is a method which determines features in the dataset, or sub class of the data which is less populous, and tries to generate samples which are not to dissimilar from the class it is trying to create. While this method among others can work for many datasets it has the ability to generate noise in the form of samples which may resemble other classes or resemble samples of the same class so closely that the new data points are essentially duplicates. Both of

these issues bias the model to behave differently than we would like in general this would lead to overfitting of the model, reducing our success. In the case of this work we are to perform undersampling because the dataset when split by class has enough samples to still utilize almost any learning technique. In the class "Benign" or {} there are 631,275 samples and in "ddos" or {} 129,152. The undersampling technique used here was Tomek Links. We could have used random undersampling however this method is a blind stab at choosing data points and is prone to losing data which may be important in learning later on. Additionally, before we learn in the preproccessing phase the data may be skewed based on random undersampling which may lead us to make falsified conclusions. Tomek Links develops its method by considering sample in the less populous class as the initial final dataset. The data from the more populous class is then considered one by one agains the smaller class. The decision on whether to include the sample from the larger class is based on a distance function, the package used here considered nearest single sample by considering which class the sample is closest to. If the data point is closer to the less populous class it is added to the final dataset. This process repeats until there are no samples remaining closest to the less populous class.

The dataset has 84 features or dimensions which range from text, dates, categorical, and continuous values. In order to train the correct classification model we must consider each of these features for their value. Their value can me measured using many methods such as PCA, Chi-Squared statistics, F-Value via Analysis of Variance (ANOVA) and general information content. This type of analysis come later after our initial preprocessing of the data features. For this dataset we initially consider the plot of the datasets individually as we can not use a feature which is only captured in a small portion for one class data and vice versa in the other class. For example it would be unfair and unwise to use a feature which is captured only 20% of the time in benign samples and 90% of the time in the case of ddos. We also consider and remove features which are irrelevant. When reading the data documentation we see that 'Src IP', 'Src port', 'Dst IP', 'Dst Port', and 'Protocol' are all heavily manipulated to give points of generation for the dataset. These features define the source or creation of the network traffic, the destination or where the traffic is attempting to reach, and the protocol by which it will utilize the network for. These are not relevant in the case of our classification method because they do not describe any of the underlying components they are used to generate the information. We also see that Timestamp is generated based on the time the dataset was created and albeit a feature which may be very important when evaluating the dataset in epochs of time we do not include it here as the data documentation states the data was generated over one day so the samples were made to fit based on the generation of the data. It was not explored but it is entirely possible a perfect correlation can be deduced from the time a ddos is generated in this dataset because of the automated nature of generation. This generation must follow some statisical or randomized method for generation which can mathmatically be modeled in some fashion where time dependence is at its root. Additionally, we remove the 'Unamed: 0' feature as it is essentially an ID for each generated sample. That being said there are duplicates of some records the reason for this is unknown to us but we did find that there are other features unique to each sample so there is no reason to drop records with duplicate 'Unamed: 0' features, these are not duplicate records.

We then look at the number of samples in each feature per class and find that the features which are or behave as constants or are largely empty. Constants are essentially not descriptive information about a dataset, we could go so far as to not even consider them as features of the dataset. Largely empty features pose the issue of reducing to many samples and including a feature that is largely sparse begs the question of whether it is worth reducing the number of samples in the dataset to match those of the sparse feature. These samples may have important information contained in them but for example in the case of the features 'Idle Mean', 'Idle Max', 'Idle Min' have roughly 86.5% of the data as zero. and 'Idle Std' has almost 95% as zero. There are methods for filling these features but for the number of samples available and the number of missing samples it would be unreasonable to generate such a large number of data as they would fit mainly to the small percentile and not scale to the larger. We additionally remove 'Active Mean','Active Std','Active Max','Active Min', 'Init Fwd Win Byts', 'Init Bwd Win Byts', 'Fwd Act Data Pkts', 'Fwd Seg Size Min' because of these characteristics. Each of these features was considered individually but there was nothing inherently present in these features that warranted synthetic sample generation. In several cases data labeled as ddos were more skewed than in the case of the benign labeled samples. For example the feature 'Down/Up Ratio' was the value 1 for over 70% of the ddos samples and almost 50% for the benign samples. When we consider the top two values in both cases they are 95.6% and 97.1% respectively. Since these two values represent a large portion of the dataset We consider what the feature value represents. We find that it indicates the download and upload ratios. This may be particularly important if a ddos attack involve trying to upload large amounts of data. This may be one for of

malicious actively on the internet but does not focus in on the fact that the dataset is mean to represent large floods of requests or hits to a server. We remove this feature as it should not be relevant in our work with DDoS attacks. WE remove the features 'Fwd Pkt Len Min', 'Flow Byts/s', 'Fwd PSH Flags', 'Fwd URG Flags', 'Bwd URG Flags', 'Bwd Header Len','Fwd Byts/b Avg', 'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg', 'Bwd Byts/b Avg', 'Bwd Pkts/b Avg', 'Bwd Blk Rate Avg' 'Bwd Pkt Len Max', ' Bwd Pkts/s', for the same reasons as well as the majority of them being totally equal to zero and serve no purpose here. We additionally could remove several more features via this method and additionally employ more comple methods in order convey solidarity and the decision making process of the remaining features. Before we do so we remove 'FIN Flag Cnt', 'SYN Flag Cnt','RST Flag Cnt', 'PSH Flag Cnt', 'URG Flag Cnt','CWE Flag Count', 'ECE Flag Cnt' at this point as they follow the same trend resembling constants or lack information needed to be described in the case of either class collectively. Again the problem with these features is not that they do not lack value or information but they do not contain enough samples for the dataset to be meaningful. In the case of this work network traffic is constantly generated and is abundant in the real world however with the dataset we are presented with we cannot choose to include every feature if it means we will have a very small sample space. We must also consider the main issue at hand which is DDoS attack prediction. The dataset we should assume is not a trivial problem which can be solved by throwing features a neural network until we achieve 90% accuracy or higher. For these reasons among other we choose not to simply cut these features from the dataset using a method such as variance thresholding which is considered a baseline approach that removes features whose variance does not meet a desired threshold. For one this method requires we understand which variance is reasonable for each feature. Albeit would remove many of the features that would have contained zero-variance the method is rather straight forward but does not leave much room for descriptive analysis. We have analyzed the features here in detail and have gained much better understanding of how the data looks internally, what to be on the look out for in the next steps of preprocessing, and beginning to consider which learning methods may prove best for the data.

We now consider the remaining dataset after our first stage of preprocessing the initial raw dataset. Again the first step was to remove features that detracted from the format of the question and the general lack of information content. We also needed to become familiar with the dataset in order to have better understanding of how to classify benign from ddos samples. We continue here with a discussion on methods of feature selection the primary one being looking at the correlation matrix to determine which features will have strong correlation between them and we consider F-Value thresholding as a method for evaluating the dataset. The F-Value threhsold is based on the ANOVA table and a null hypothesis so we are not directly required to pick parameters for determining the feature selection. As an example variance threhsolding requires a threshold which is used to determine at what level of variance a feature in the data must have in order to be used. We develop these two methods used and discuss their ability to reduce the dataset in terms of spacial complexity.

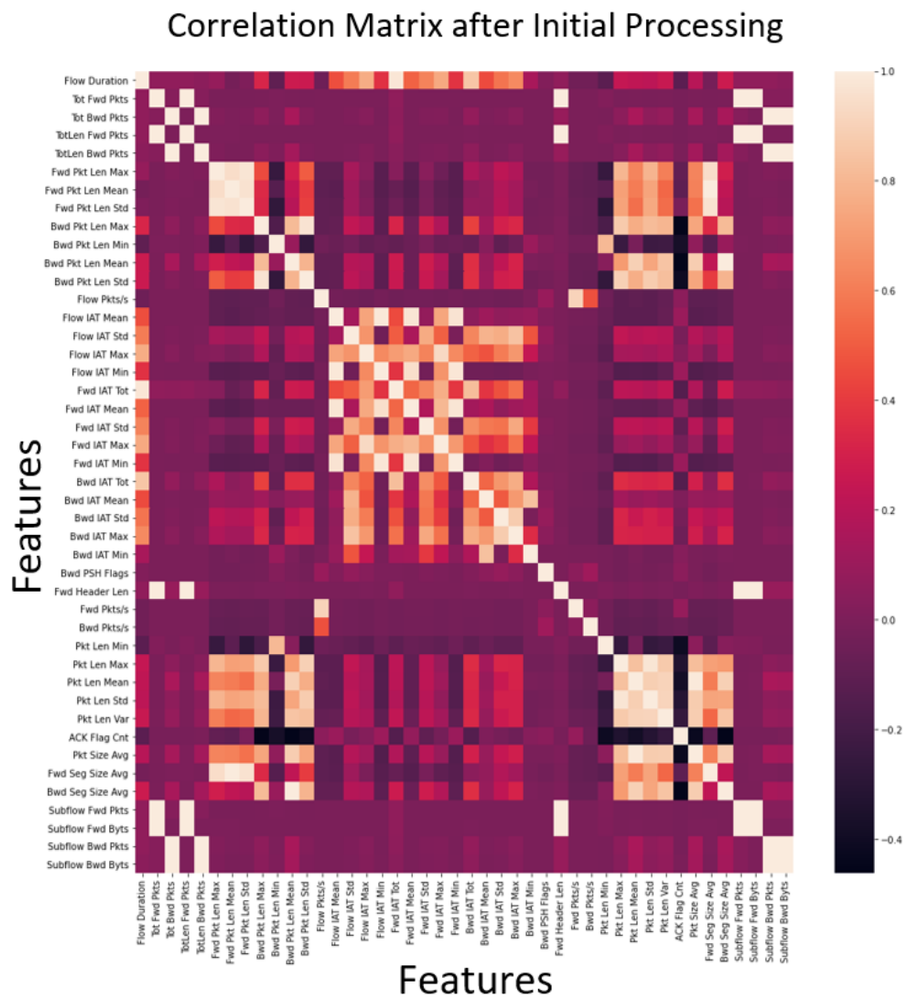


Figure 1: The correlation matrix after the initial preprocessing steps of removing largely empty or constant like features. The color bar on the right show the color relation strength. Brighter is more positively correlated. Darker is more negatively correlated. In the middle the reddish pinkish color is close to zero correlation. The ratios of users with a value greater than 2.0, noting the drop off in high number of users with increasing ratio size.

We show the correlation mattrix for the initial dataset after the first stage of preprocessing mentioned above in Figure 1.4. The correlation matrix is good for showing which features are more linearly related. From the correlation matrix we can see that either strong positively or negatively correlated features may be good features to use in the case of classification. The correlation matrix is essentially the same data divided by the diagonal of the matrix which as you can see is perfect correlation, that is each feature is perfectly correlated with itself. The whole matrix is given instead of a lower/upper triangle as a matter of preference. The method uses Pearson's correlation to generate the correlation matrix. There are some issue with Pearson's correlation which arise namely from determining dependent and independent variables. Luckily since the task is to classify DDoS or Benign samples we can forgo this issue as the features are all considered as dependent when trying to classify the class label. That is not to say that two features are dependable on another it is a matter of the application which allows this correlation matrix to be used. In the case of classification methods we prefer to avoid information that is highly correlated as it may lead to unreliable predictions, hindering the model. One other hindrance to using correlation matrices is that we are not accounting for deeper/more complex relationships which may cause two features to appear strongly correlated. That being said we observe both strongly correlated features as both potential sources for success and error in this work. The correlation matrix below in Figrue 1.4 is a slightly reduce matrix which removes much of the low correlation features. From this stage we are confortable with the matrix having some strong relationship in each feature and in total we have now 20 features to model a relationship. We could stop here and train the model with these features. We could also pray that the underlying relationships hidden within the dataset would magically be learned by whatever model we utilize.

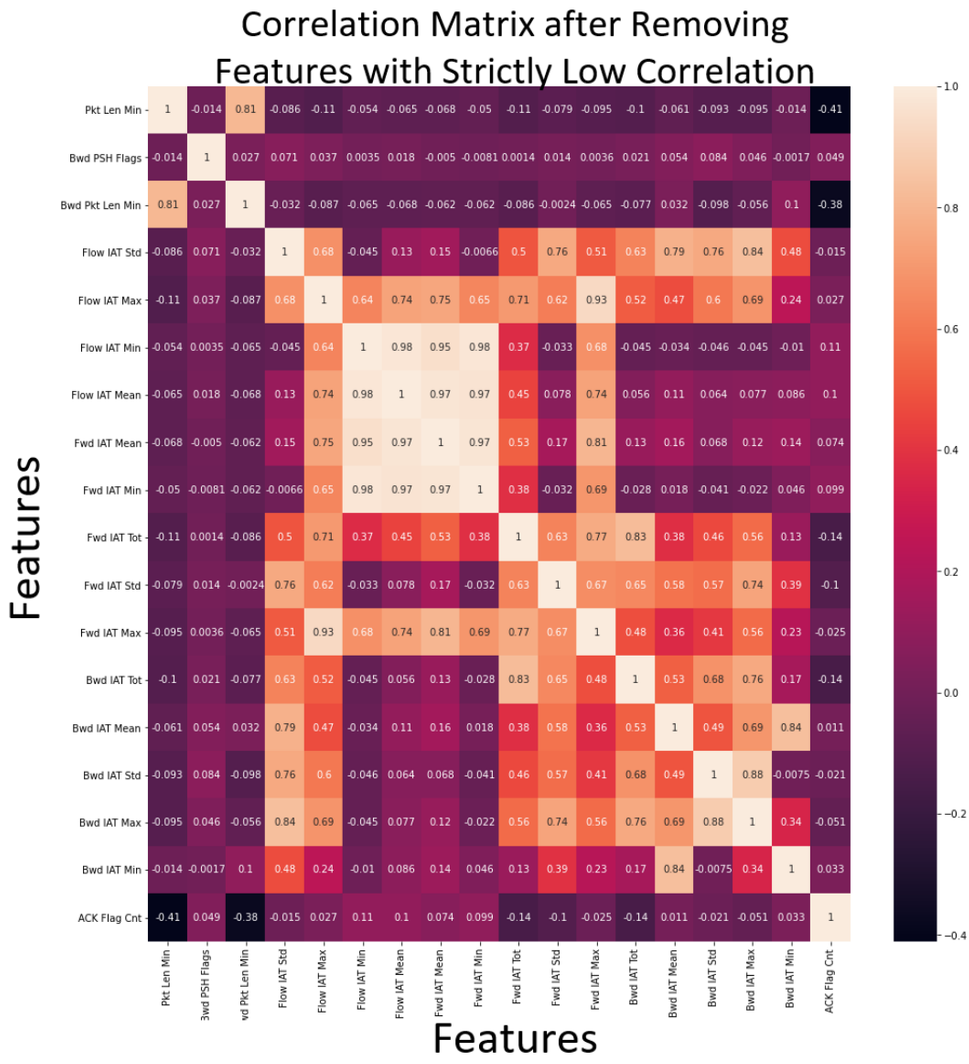


Figure 2: The correlation matrix after the dropping features which are by and largely low in correlation. The problem with low correlation features is that there is a lack of linear relationship. In the case of this work a lack in correlation indicates features that dont behave with any clear characteristic in terms of a class 'ddos' or 'benign'.

Looking deeper at the correlation matrix we reduce the total matrix to those that have a high and multiple correlations above 0.7 as that is generally cited as a having strong correlation [ref]. We split this data by looking at it from both their correlations and the scatter plot of the data. We include these per class and in total consider now seven features which contain strongly correlated features. We show the per class correlation features in Figure 1.4 we exclude the intermediate correlation matrix as it only contain information that is already viewable in both the above correlation figures. On the left of Figure 1.4 we can see the features fit very well into three categories of correlation for ddos labeled data. That is they are either very highly correlated, indicated by the white or beige color, highly correlated, indicated by the orange red color, or have low to no correlation, indicated by the dark purple or black. This type of relationship is what we are after. We want to find features that we can reliably show are correlated or not. We can clearly take this into consideration when training a model to classify ddos versus benign. The plot on the right shows something else entirely. We can see that the benign class lables are a mixture of low to very high correlation. What this shows is that the variability in normal or benign web traffic. There are some chases where this overlaps. We could conclude that these are features an attacker would try to maximize in strongly appears as benign network traffic. The stark differences between the two matrices is what sets them up as two good samples for learning. If there is a way for the model to learn high correlation from a scattering of variable correlations this model would achieve the best performance. This could be something like fuzzy cmeans which looks at the models from the dispersal of samples and there clustering centriods. The goal of this method is to classify data based on their clustering relationships but we must first consider the dataset distributions or in our case we plot the scatter plot of each dataset and consider each value independently.

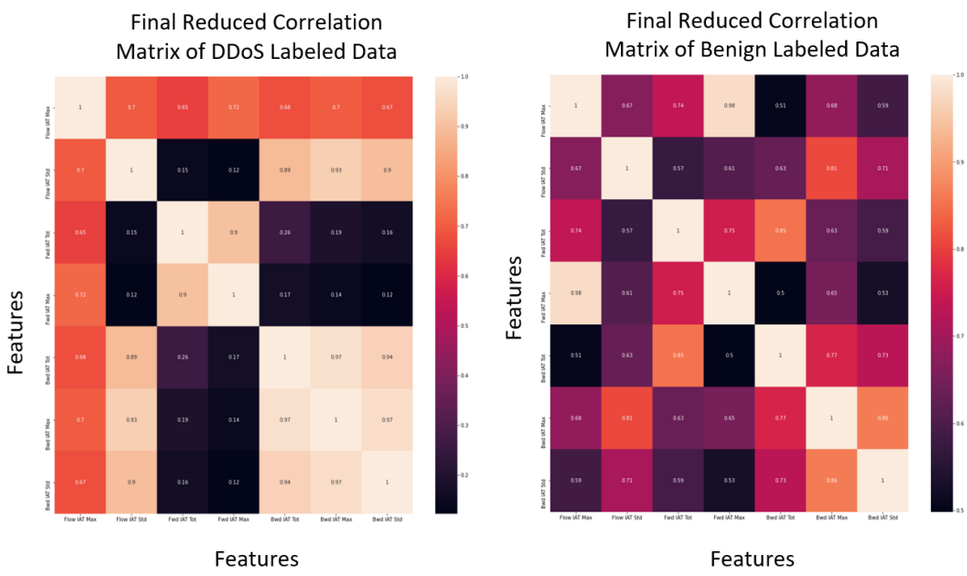


Figure 3: The correlation matricies for each class ddos and benign. Initial viewing show that ddos data is almost perfectly correlated in several features and has almost zero correlation in the remainder. The dataset begins to show actual learnable information that may be picked up by a model.

We continue by showing the scatter plots of each feature much in the same way we consider the correlation matrix for each relationship with the feature set. we show the plots for each dataset in Figure 1.4. We can now grasp that there are some features which have very little relationship as we note in the correlation matrix of Figure 1.4. These features we are unsure of but keep them as they do not need further processing at the moment. We consider the distribution or clustering of some these features, specifically those that vary from ddos to benign. One example we can clearly see is that the data in Flow IAT Std versus Fow IAT Tot for Benign network traffic has this large portion of samples in above the diagonal which are almost vertical while the DDos network traffic has this relationship in a meager sense. It appears this relationship in ddos is synthetic and again a feature of the data which is meant to showcase just how sophisticated ddos attackers may behave when initiating their attacks. The feature Flow IAT Tot is the total time between two flows. These flows are generated in the synthetic dataset but represent the number of packets transferred per second, in other words when a domain communicates with a client it sends and receive information at some nominal rate. This relationship statistically is measured using the total here. It appears like the data in the ddos class or those on the left in Figure 1.4 is trying to mimic the same behavior as benign network traffic. This does not appear to be coincidence in terms of how network traffic would look in a ddos attack. We also see that there is a strong pattern like relation ship between Fwd and Bwd IAT Tot. These two features relay the information for time between two packets being sent forward or backward respectively. This feature does not share as much similarity between both ddos and benign labeled samples. It is almost a stark difference between these two. This data would be the type that is quintessential to successfully classifying the data as ddos or benign. We could go so far as to say this type of feature is good enough by it self. We could have a simple statistical measure for this relationship that measures the difference between these and that would provide the simplest form of classification or regression between these two features for the data. That is not say it is good enough by itself but two features behave the most apparently different between all the other features.

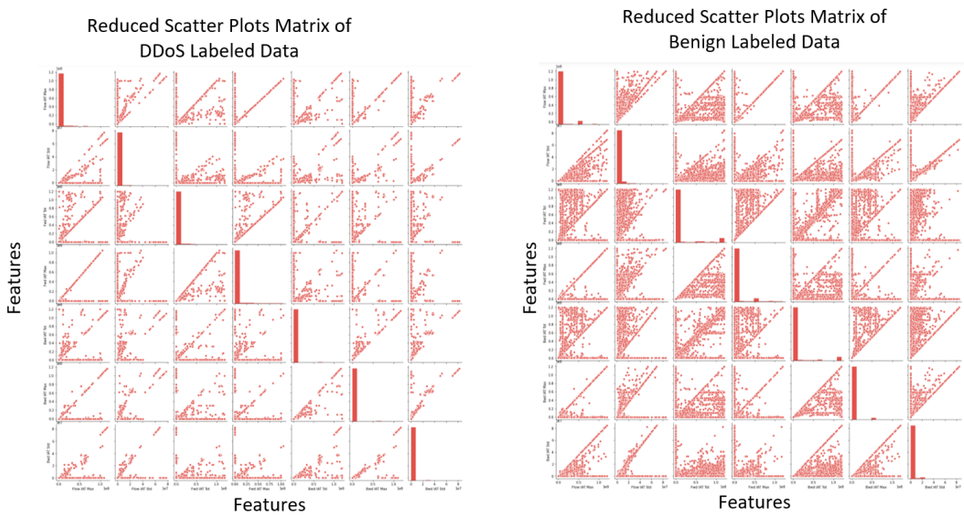


Figure 4: The scatter plots for each class ddos and benign. We view these to see if we can further reduce the dimensionality of the dataset. Note the difference the features. From left to right the ddos lableld scatter plots and right the benign plot. We can see that there is some features that almost have no real relationship. While some have very different information.

## 1.5 Learning Processes

In earnest we do not show complex methods here as they detract the facilitation of explainable models. Instead in this work we focus on simpler methods and infact choose the most explainable method as our top winner which is discussed in detail in the results section. We consider the task of classifying the attacks as ddos and benign again. The dataset has been preprocess (sliced and diced) in several ways to achieve a dataset which should comfortably learn from the data. Given the proper tuning, the developed dataset should perform well not only in highly complex models such as deep learning neural networks or ensembles of multiple learning methods. We discuss primarily simple implementations of the models: Support Vector Machines (SVM), Decision Trees, KMEANS, Artifical Neural Network (ANN) in the form of a multilayer perceptron (MLP), and Fuzzy C-Means. We facilitate a discussion on these methods and discuss the majority of their implementation details. The data selection methods and parameter tuning are also discussed. Before training at all we normalize the data using a form of standard deviation normalization. This was used because the dataset is large and simplifying the data to min max normalization may squash some of the information contained in the data. With standard deviation normalization the data follows a standard normal distribution

There are fundamentally two issues that arise when training or fitting data. In general these are (1) which samples do we provide to the model to get the best results? and (2) which parameters are best for each model to maximize metrics of concern? These two fundamental problems grow with complexity as the data grows in complexity. As we increase in dimensionality we also increase the number of samples which are needed to fully capture the inforamtion in a dataset. This issue plauges most data mining processes. In the case of this work we are not really bound by this issue as we have only seven features and still over 200,000 samples. We consider the actual issue at hand which is what percent of data is enough to fit a good model? This issue has a least a starting point where we can consider things such as the distance between d-dimensional space with d samples and n samples which we have defined here as d equal to 258,304 and n equal seven we show the equation in Equation 1 and calculate the value as 0.084. This values for D(d,n) is the relationship between distances between two points in the dataset. This distance is not so bad but it is small indiciating there is a lot of information in the datset. There are certain implications when the distance between data is so large that for example say we cannot have clear understanding of the space. When we reduce the samples we obviously lose information and this number grows smaller indicating that the complexity of the feature space is dropping [4].

$$D(d,n) = \frac{1}{2}(\frac{1}{n})^{\frac{1}{d}} \tag{1}$$

This metric is used mainly to develop the intuition for the dataset which we use in our best model which we will discus further in the Results section. For the time being we mention

this and expand on the fact that information is not as clear in higher dimensional space. That said we must decrease the sample space for the first piece in designing our models because of training and testing. The dataset was split into two groups for training and test except for the MLP which has a third group for validation, this is used within the model to check improve the weights at the end of an epoch. This also give a better understanding of the overfitting issue which typically plagues deep learning methods. The representative amount is difficult to ascertain. Depending on the dataset we could use 70% for training and 30% for testing or 80/20, 90/10 even 50/50 would make sense in some cases. Initially in this work we split it at 70% and 30% because we have so many samples. This number of samples was used for every model to test their ability. The data tended to perform very poorly with this and the classification accuracy and more important to this work F1-Score was very low. We changed the percent for training and testing iteratively until the final conclusion was that the data be split 20% for training and 80% for testing. This yielded roughly 7% increased accuracy by just shifting the portion of the dataset in one model. We use a random seed for reporducability, we were not using different portions of the data in the sense of randomly choosing the 80% of different random samples each test. We use the same seed to keep the random generator consistent This increase in accuracy may be which contain either more relevant samples which perform better on the majority of the data or less information that it does not bias the output from the model.

For the Decision Tree, SVM, and KMEANS methods we develop a grid search method which is one time consuming method for parameter tunining. We do a grid search for the MLP but the model characteristics are different and we develop this further in brief first. Since the model paramters of a deep learning model are totally configured by the user, that is to say not one shoe fits all problems. We control the paramters for dropout, number of layers, number of neurons per layer, number of training epochs, batch size, input shape of the data, activation functions for the each layer (most importantly the final output layer), learning rate and optimizer and the loss function. Several of these are developed in the equations below but due to the save of the problem and time we trained our grid search based on only a subset of these paramters. We use dropout, neurons per layer, number of layers, epochs, batch size, learning rate and loss function as tuned parameters. For both *accuracy* and *MSE* $y_i$ is the observed value and $\tilde{y}_i$ is the predicted value. We use mean square error as our loss function, seen in (3). The optimizer used in the MLP was Adaptive Moment Estimation (Adam) [5] optimizer, described in (4), with the learning rate $\eta$ can be set to a value such as 0.02, following the TFQ documentation. In (4), $\theta_{t+1}$ is the current gradient of the stochastic gradient descent (SGD) based on the previous gradient $\theta_t$,

$$accuracy = \frac{1}{n}\sum_{i=1}^{n}(y_i = \tilde{y}_i) \tag{2}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \tag{3}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \tag{4}$$

where the weight $\hat{v}_t$ and momentum $\hat{m}_t$ are defined as:

$$\begin{aligned} \hat{v}_t &= \frac{\beta_2 v_{t-1} + (1-\beta_2)g_t^2}{1-\beta_2^t} \\ \hat{m}_t &= \frac{\beta_1 v_{t-1} + (1-\beta_1)g_t}{1-\beta_1^t}. \end{aligned} \tag{5}$$

with $m_t$ and $v_t$ are estimates of the gradients' mean and variance respectively, and $\beta_1$ and $\beta_2$ are forgetting factors.

The final dense layer of the model uses *relu* as the activation function since the two classes aim to classify *(0, 1)*. The *relu* function is defined in (6) where *phi* is the current sample weight,

$$ReLU\phi(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{6}$$

The *relu* function is used because it is chooses between the label 0 and 1 which is what we are aiming for in this work. We could have used softmax and sigmoid but we are trying to binary classification. The goal of artificial neural networks is to stack several layers with

different number of neurons and parameters such that the learning function, typically some kind of stochastic function, can optimize the function to the data which is often called fitting. The issue with deep learning methods is that it is often easy to fall into pit falls leading to over fitting or incorrect optimization of the gradient function. This learning function extensively relies on the parameters mentioned in the equations above but also the back propagation method of updating weights and biases of the network. Briefly, another issue is vanishing gradients where a model may begin learning well for a time and suddenly begin to stay at a constant local position. This is due to the implementation of gradient descent methods where the gradient which is calculated and then passed backward through the back propagation method is so small that the weights do not change anymore. These issues along with their lack of interpertabilty make them difficult to use and hard to gain meaningful conclusions as aside from the the discussion presented here to shed some light on the topic they are consider black box methods. In our case we found the MLP to not be worth the fine tuning required and amount of time it would take to find a highly accurate model. We found that the majority of parameters tested resulted in vanishing gradients or over fitting. The best model is included in the results section.

The Support Vector Machine used here tries to tune parameters for the kernel function 'rbf' versus 'linear', the penalty for samples which cross the decision boundary, and the gamma which is for the kernel 'rbf'. We tune these parameters in the expectation that the model is not going to abstract very complex relationships in the data but expect that the data will contain enough information to have high classification scores. The support vector machine has a long history of solving many contemporary problems in the field of machine learning it is able to handle multiple classes and large amounts of data fairly well. It is one of the methods that should be in a sense tried and true when developing intuition about a dataset. There are many forms of support vector machines but in general the goal of the method is to classify a model by finding a boundary between the features that separates the classes. The boundary can contain overlapping samples or samples which belong on only one side typically at a cost of accuracy in training but can lead to better generalizations of the model. The model here was essentially as simple an SVM as possible. The goal with this model and the SVM and KMEANS models was to let the data do the work in a sense. Of course that is not to say the parameters were not tuned in several at temps to improve the results but after gathering clear and concise dataset after preprocessing we must assume there is a learning method that more easily classifies the data set then others. Additionally if at all possible it would be best to have a simple model over one that is complex. Having very specific design causes the capability of a model to be less explainable and often times less robust. After tuning the parameters we found that in general the results resembled the MLPs but were slightly worse and overall were not spectacular in solving this problem.

The design of the decision tree classifier we also briefly develop here. Decision tree work by creating regular logic expressions to each feature in the data set. The decision tree here using the gini index for determining the criterion of splitting a tree based on a feature. Gini index is used in the C4.5 algorithm for making decision trees. The C4.5 algorithm is designed to consider information gain ratios to determine which feature will be used for splitting. For example data that is containing continuous values may be split for values greater or less than some value. Categorical values may be split based on a similar method but the underlying proponent to making decision tree is this gini index or information gain criteria. These essentially tell us where in a continuous feature should we split the data or based on which categorical value would the tree more likely improve based on its splitting. The development of decision tree here searches based on total depth of the tree and the minimum number of leaves in a branch. The goal with the decision tree is not classify the samples provided only but to also perform well on the testing dataset. We do not want a very well defined and specific decision tree with tens of thousands of leaf nodes and with a large depth that only explains the training samples. In fact the smaller the tree the more generalize it will be in terms of variability in the testing data. We believe this has been met as we trained our best model with the decision tree classifier.

KMEANS is generally speaking the least complex in terms of setting up specific parameters to solve the classification problem. We in general split the problem into the issue of where initialize the dataset and the total number of iteration run. The most important parameter in KMEANS is the 'K' or the number of clusters to form or the number of classes, in the case of this work k is equal to two because we are solving a binary classification.

## 1.6 Results

The results here discuss the metrics in Table 1.6 and their equations for precision, recall and F1-Score are given below. Precision is calculated as a ratio of true positive (TP) and false positive (FP)

$$Precision = \frac{TP}{TP + FP}, \tag{7}$$

whereas recall is calculated as a ratio of TP and false negative (FN).

$$Recall = \frac{TP}{TP + FN}, \tag{8}$$

F1-score is calculated as the relationship (harmonic mean) between precision and recall,

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}. \tag{9}$$

The results show that the Decision Tree classifier overall was the top performer. The results we see here are in the table show that the MLP came in second followed by the SVM and last KMEANS. We will discus the results of the decision tree first and explain why it is the model we choose as the best here. We evaluate each classes results seperately in Table 1.

| Model Metrics | | | | | | | |
|---|---|---|---|---|---|---|---|
| Model | Accuracy | Precision(ddos) | Recall(ddos) | F1-Score(ddos) | Precision(Benign) | Recall(Benign) | F1-Score(Benign) |
| MLP | 0.70 | 0.83 | 0.51 | 0.63 | 0.65 | 0.90 | 0.75 |
| SVM | 0.63 | 0.93 | 0.28 | 0.43 | 0.58 | 0.98 | 0.73 |
| KMEANS | 0.45 | 0.08 | 0.01 | 0.02 | 0.47 | 0.88 | 0.61 |
| Descision Tree | 0.90 | 0.92 | 0.86 | 0.89 | 0.87 | 0.93 | 0.90 |

Table 1: Test data results from dataset provided. The models are listed in the left most column with the the metrics follow these. The results are vastly in favor of the Decision tree method. The worst model over all was KMEANS.

Our decision tree model achieves an F1-Score of 89.5 overall which is the best F1 overall which as we defined initially is the metric for being the top performer in the work here. We also note that all the metrics for this model were high. This model has a high accuracy of 90% which is not really a good metric for the highly imbalanced nature of the dataset but when considered with the dataset that was preprocessed it is good. The reason we do not elaborate on accuracy in this work is the inherent nature of accuracy for imbalanced datasets. Accuracy can imply that a model is performing better than it really is because of the imbalanced classes. If we showed results for the entire dataset this number would be very skewed as the original data is skewed largely in favor of the benign class. This one of the main reasons we had to perform so much preprocessing before we could develop any learning methods. For this model we also include the confusion matrix as it paints a complete picture with the data in the Table 1. The confusion matrix is in Figure 1.6 which shows the True positive ratio being very large for the positive ratio of DDoS samples is very clearly high, the False positive and false negative ratios being also very low and true negative or truly benign also high.

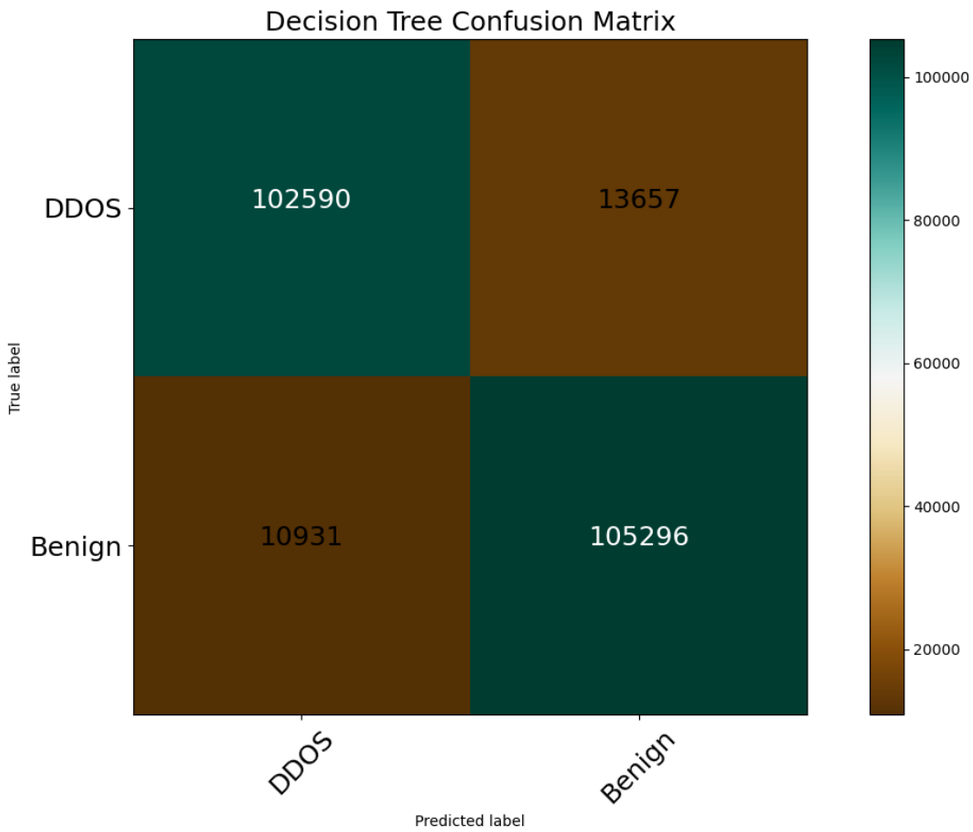


Figure 5: The confusion matrix for the Decision Tree classification model. Noting the high True Positive and high True Negative rates.

The receiver operator curve or ROC curve shows the ability of the models to correctly predict, in general they show the relation between True Positive and False Positive Rates. False Positives or type 1 errors in the case of DDoS attacks is not the most important factor to consider because we know that we are able to handle these cases through some other automated procedure in the worst case. However, having less is still better. The False positive rate here for the decision tree model is very low which is good in the case of making correct classifications of non-benign samples. This is the only model which this can be said about. The other models show very strong relationship between successful classification and false positives. This is not what you can mitigate in a system. These results would classify most network traffic as ddos attacks which is not something a business would be able to manage. The goal of the model is to remove to automatically capture these malicious web traffic contents in order to prepare and handle them. The ROC curve shows that the SVM and MLP almost share the same characteristics except in a sense inversely. The true positive rate for the SVM model is higher much earlier but the MLP catches on much later. Overal the KMEANS again looks rather poor in this evaluation. The ROC curve is just one metric we can use to show this relationship. Similar information can be gathered form the confusion matrix above for the decision tree. We include the ROC curve to combine all of these in stead of showing each confusion matrix as the results were generally poor for the other models.

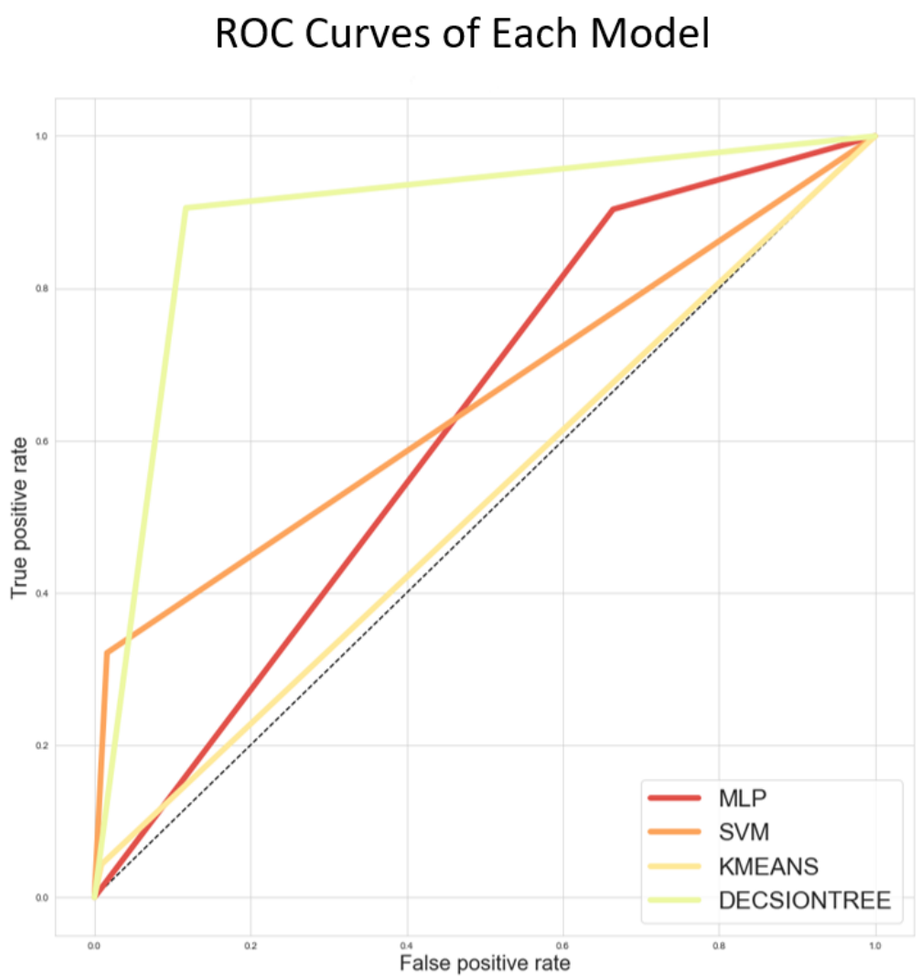


Figure 6: The ROC curve with each of the models true positive and false positve rates as the Y and X axis respectively. where the decision tree model is the clear winner the other modes struggle with the dataset. KMEANs is almost as bad as the mid point line.

## 1.7 Conclusion

In conclusion we have seen and discussed topics related to DDoS attack and developed how classification of network traffic can be learned using different learning methods. The work here focuses on the the preprocessing and learning function which were used to achieve an F1-Score of 89.5 overall using a decision tree classifier on the prepared data. We have covered several learning methods and evaluated their results using the ROC curves plots as well in our results. We also develop the intuition behind how some of these attacks are growing more and more sophisticated, albeit difficult to ascertain without diving into the data behavior and underlying principles of the data are shared between these attacks and nominal network traffic generated by everyday people. The dataset which is generated by the University of New Brunswick provides a very good source of learning about these attacks. The dataset originally beings with over eighty features however in our final dataset we end with only seven. We additionally train on a very small subset of data to generalize the learning behavior of the models and evaluate our methods on a accuracy, precision, recall, and F1-Score. We have found the best of these models to be also the most explainable. The decision tree classifier used here has a depth of 44 and 2,234 leaves which was well below/above the threshold we implemented in our parameter tuning for decision trees. Although this decision tree is our best in terms of F1-Score we also achieve incredible results a fraction of the samples, we were able to lose roughly 3% of accuracy and F1-Score while using only 1,400 samples of the 258,304 remaining after preprocessing. This result to us was very shocking as the resultant tree has a depth of 20 and 179 leaves. These results are even more simple then the larger decision tree. Due to the requirements of this work we describe the best model as the larger decision tree as it achieve the highest F1-Score albeit significantly more complex and only slightly better.

Overall we were not surprised by the outputs of the MLP. The MLP we believe could have performed better given more and more parameter tuning options but the time it takes to train this model for all the parameters which may influence the final metrics is an exhaustive

search and the parameter space is larger than the feature space. The overall F1-Score was 69 which makes it the second best of our models. This complexity is something very undesirable specifically in the case of binary classification. There are several methods which attempt to solve the largely complex datasets with high dimensionality like random forests or gradiant boosting (XGBoost being one popular method) or ensemble learning are all very good for these types of problems. It was our opinion that these methods are over kill. The dataset is not thoroughly complex after performing preprocessing and the correlation matrix for ddos data has very strong relationships and very clear lack of relationships. The benign data was very noisy but it regular network traffic should also seem noisy, there may even be a lot of benign traffic that resembles ddos network traffic due to the type of requests people either on accident or by need for testing, development, and implementation of products. Information we can see in some of these features resembles an almost mimicry of benign samples where the ddos data appears to try and conceal itself a nominal network traffic.

The other two methods SVM and KMEANS were surprising in their results as they achieved very poor scores overall. The SVM being the better of the two. The SVM model was tuned over several different configurations, but still it was only able to achieve a combined F1-Score of 0.58 which is not very good. It also does very poorly on the Recall of ddos and the precision of benign samples. These are rather alarming results if they were to be used in a system. The True positive ratio of ddos is perhaps the second most important metric for this problem as it ascertains the true classification of a ddos sample and in general is preferred when considered again false positive which can be handled in a system somewhere down the pipeline. The precision for benign is also concerning as it was not able to learn which samples were actually benign. The issue with precision in the case of this model is somewhat expected as the features we show in the correlation matrix show that benign samples had less features that were strongly correlated together than the ddos class. This may be the reason why the precision of ddos samples in the SVM is the highest of all the models even better than our best decision tree model. This model also has the highest recall for benign samples. The KMEANS model performs very poorly and yet manages to provide a recall for benign samples which is on par with the other three models. This model only achieves an F1-Score of 0.61 which very low in terms of the dataset we have preprocessed. the poor results may be to the large overlap of the data contained in the training samples. Although we tried shifting the dataset to try KMEANs on different samples within the training data specifically it still performed poorly. The remaining metrics for KMEANS are very low and combined with an F1-Score of 0.61 could almost be considered guessing. That is the model did not really learn much or fell into a local minimum very soon after training. This issue is cited in the documentation for scikit-learn and there were few options in the package for mitigating this issue.

## 1.8 Discussion on Applicable System Design

In terms of a full system design the model we have here is not totally ready to be adopted. There is a lot fine tuning we would need to solve for in order to have an F1-Score high enough that suffices in a real world application. The goal of the model developed here would be a very good starting point for beginning production testing. The dataset we have here may or may not perform as well as the real world but may represent the fundamental component for the design of a system which can strengthen enterprise security. In general this could be used for a rather small company that doesn't have massive amounts of large network traffic otherwise the manual review of the model outputs for false positive and false negative would take an overwhelming amount of time.

One such system might try to mitigate DDoS attacks by an iterative flow of parameter estimation. The Decision Tree model which we developed here would be one of these inputs the system. In general this could be an ensemble based method with either multiple models or multiple sources of network information that may in part be good for classification are included. For example their may be some simple characteristics of data that may not need learning which can decrease the complexity of the manual review once the ensemble method is applied. The ensemble method or just decision tree would be loaded online to be used in real time and outputs would then be passed along to the next portion of the system which evaluates the output against other know information, for example this could be related to time of day or number of concurrent users in the last hour. or number of unique users since the business opened. Features such as these may be able to reduce the issue of false positives. The issue of false negatives is another matter. We must consider the case when the predicitve model is not able to correctly classify DDoS like network traffic and handle this with the additional methods in the system. This may be some kind of quota which the model must meet based on historical data that has been captured and used to build the live production model. This information could be crucial in determining whether or not the model may have been hit with a DDoS attack but was not able to identify it.

Overall the system would need to be incorporated in a way where the model can be retrained quickly, easily, and frequently as the network traffic is not always the same. That is unless the input samples were weighed by a system which was designed for outlier detection such as in [1] which is designed to handle covariante shifts in data. The problem with most imbalanced datasets or live loads of information is that the parameters of methods must be finely tweaked in order achieve high performance metrics. The case is the same when considering DDoS attacks. The data here we have seen is highly imbalanced. What is more is the features in the network data may shift gradually and may not be captured successfully in the outlier detection step which means parameters will need to be tuned regularly also. All these things and several more aggregate to the problem of cyber security being a challenge frontier of problems that are often not one and done. I think the system I have design probably exists in one form or another in the world today with much more counter measures that comes from years of experience in the field of cyber security.

# References


[1] Mohsen Asghari, Daniel Sierra-Sosa, Michael Telahun, Anup Kumar, and Adel S Elmaghraby. Aggregate density-based concept drift identification for dynamic sensor data models. *Neural Computing and Applications*, pages 1–13, 2020.

[2] Wesam Bhaya and Mehdi EbadyManaa. Ddos attack detection approach using an efficient cluster analysis in large data scale. In *2017 Annual Conference on New Trends in Information & Communications Technology Applications (NTICT)*, pages 168–173. IEEE, 2017.

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. pages 108–122, 2013.

[4] Mehmed Kantardzic. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Opeyemi Osanaiye, Haibin Cai, Kim-Kwang Raymond Choo, Ali Dehghantanha, Zheng Xu, and Mqhele Dlodlo. Ensemble-based multi-filter feature selection method for ddos detection in cloud computing. *EURASIP Journal on Wireless Communications and Networking*, 2016(1):130, 2016.

[7] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[8] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[9] Raja Majid Ali Ujjan, Zeeshan Pervez, Keshav Dahal, Ali Kashif Bashir, Rao Mumtaz, and J González. Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn. *Future Generation Computer Systems*, 111:763–779, 2020.