Final Report

Identifying the best machine learning model for web-based attack

Cardiff University School of Computer Science and Informatics

Author: Alice Edwards

Supervisor: Amir Javed

Moderator: Liam Turner

One Semester Individual Project – 40 credits

CM3203

Abstract

This paper studies how to identify malicious network traffic from a web application. This involves studying the common vulnerabilities that web applications have and the way in which they can be abused in order to compromise the application's security.

To achieve this, I performed multiple attacks on vulnerable web applications, and built a machine learning classifier to categorise network traffic in order to identify attacks, so that a service administrator may maintain their application.

My results identified Trees as the best method for classifying these datasets, and in particular highlighted the J48 tree as effective. This meant that my IDS (Intrusion Detection System) had a precision, recall and f-measure of over a relatively high score of 0.8. However, this may change with different datasets.

From a network administration perspective this study shows the importance of maintaining web applications to minimise vulnerabilities. This is built upon with the use of an IDS system to watch traffic at all times, as not all vulnerabilities can be purged.

Acknowledgements

I would like to thank my supervisor Amir Javed for his invaluable help throughout this project.

I would also like to thank Eirini Anthi for taking on a secondary supervisor role, and providing an expert view on penetration testing, as well as advice when I got stuck.

Finally, I would like to thank Caitlin Burns for showing me their Kaggle account and sharing some machine learning work they had completed in the past in order to guide my own work.

Table of Contents

Table of Figures

1. Introduction

1.1 Web Attacks

Web Applications often have vulnerabilities that mean users with malicious intent can compromise the application's security. According to a recent study, 44% of web applications are vulnerable to data leakage and security problems [1].

The five most common web attacks are XSS, SQL injection, DOS, File Path Traversal and Command Injection [2]. These have been around for years, and yet it is still common that they are not fully protected against. Therefore, I will focus on these five primarily in the project. However, this is not to say that there are not many other attacks types; and new attacks are constantly in development.

Aside from ensuring everything is up to date and patches are regularly used, there are three types of protection [2]: 1) Vulnerability scanning and security testing, 2) Web Application Firewalls (WAFs), and 3) Secure Development Training (SDT).

This project focuses on vulnerability scanning in the form of a network traffic classifier.

1.2 Penetration Testing

Penetration testing (pen testing) uses simulated cyber-attacks against the computer system to check for exploitable vulnerabilities [3]. Pen testing is commonly used in web applications to augment a WAF.

In the context of this paper, penetration testing has been used to carry out common attacks on vulnerable web applications. From penetration testing I can gather the network traffic for malicious activity (where an attacker is attempting to exploit a vulnerability) and compare it to benign network traffic (the expected traffic for this web application.)

1.3 Machine Learning

Machine Learning algorithms use statistics to find patterns in vast amounts of data [4]. Machine learning has 3 categories: supervised, unsupervised and reinforcement.

As I want my algorithm to identify attacks by name, I will use labelled data, thus will be using supervised machine learning.

1.4 Why I am building a classifier

A classifier allows data to be grouped by a label (as opposed to clustering in unsupervised learning) [5]. In this context of finding attacks, this is useful to separate the traffic from what

are expected outputs and what are not, so that cybersecurity can be used to patch vulnerabilities.

Using a classifier means that a cybersecurity system can analyse patterns and use the learning from this to prevent similar attacks in the future and respond to changing behaviour [6]. This means that a cybersecurity team can be more proactive in preventing attacks and respond to them in real time. If my classifier is deployed to continuously monitor network traffic it can identify a potential attack as it is taking place, and thus allow a cybersecurity team to stop it.

This means an organisation can prioritise its time, thus making cybersecurity less expensive and more effective.

I will primarily use a binary classifier to split traffic into malicious or benign. From this I will then use a different classifier to split data by attack category.

1.5 The Project

This project looks at what vulnerabilities can exist in web apps. Then, understanding the ways vulnerabilities can be abused – to culminate in using machine learning techniques to identify where this abuse is taking place. My classifier can be deployed as an IDS to catch attacks in progress and learn from them.

2. Background

2.1 Similar Studies

I began my background research by looking into what similar studies already existed. To do this, I primarily used Google Scholar to find research papers on similar projects. When this well ran dry due to lack of similar content, I started looking from a normal google search for any similar applications, whether they had a research paper or not.

2.1.1 SQL-IDS [7]

One study I found was a SQL-IDS. This study focussed only on identifying SQL-injection attacks, and instead of looking at network traffic it studied the SQL requests to ensure they matched the expected format. It uses a specification-based methodology. They found it particularly effective as it meant no changes to the web application or database schema. Instead they just added a verification of the SQL statement, with lexical analysis. Their future work suggests looking at other types of injection attacks, such as XSS, suggesting that they would look at the network traffic like in this study to identify correct traffic.

2.1.2 Extending Web Application IDS Interface: Visualising Intrusions in Geographic and Web Space [8]

This study had little information available without paying to view it. However, I was able to establish that it was looking into making IDS information more comprehensible to those responsible to stop attacks, rather than a normal text-based approach. My assumption is that this would break down attacks into their categories, in a similar way to the intended secondary aim of my project. However, this study seems to be more aesthetic than technical, and would likely use an already existing IDS, only changing its output. As such, my project is still very important, though in future iterations it may benefit from a more graphical output as guided by this study.

2.1.3 A Closer Look at Intrusion Detection Systems for Web Applications [9]

This paper talks about how IDS is a known methodology for detecting attacks on network systems, but it is still relatively immature in monitoring and detecting web-based attacks. The general overview of this article pronounces the need for IDS approaches specifically designed for web applications, due to their difference to generic network attacks. Thus, showing a need for my own project.

2.1.4 Applying Blocking Measures Progressively to Malicious Network Traffic [10]

This study explores the idea of applying blocking measures based on an application like mine detecting anomalies and using a loop to test whether these measures get rid of the anomaly. This shows ways in which my application could be used aside from just notifying administrators of an attack, allowing automated blocking in the future. In effect, this would turn the IDS into an IPS (Intrusion Prevention System), thereby proving multiple applications for this project, thus making it worthwhile.

2.1.5 Detecting Malicious Web Links and Identifying their Attack Type [11]

I found this paper that studies malicious URLs and detecting them. This is similar to my study in that it attempts to classify attacks. However, it looks specifically at attacks orchestrated via URL links. They use machine learning to detect and categorise attacks, which is the same as my intended method. They performed two machine learning algorithms: SVM (support vector machine), and RAkEL (random *k*-labelsets ensemble algorithm) and ML-kNN (K-nearest neighbours for multi-label data). This means using a combination of generative and discriminative models, as SVM is Generative and RAkEL and ML-kNN are discriminative. Using the combination of these two is something I took into consideration for my own development process.

2.1.6 Techniques for Identifying and Managing Potentially Harmful Web Traffic [12]

This paper looks at a spectrum of techniques for identifying attacks from web traffic. It aims to identify traffic in coordination with passing traffic through a firewall. One of the ways it suggests doing this is by parsing the traffic request attributes and assigning it with a threat rating based on threat profiles. This appears to fall under the same branch of machine learning as my own work but assigns a rating rather than detecting and labelling an attack. As such this would not help identify ways in which to fix the attacks, and suggests that they would be using unsupervised learning, rather than labelled data. This would be less precise than my model but allows for more future growth.

2.1.7 Conclusion from Similar Studies

These studies show that there is little focus from Intrusion Detection Systems on web apps, even though they are one of the

most attacked areas of computing. Therefore, demonstrating the need for my project in filling this gap in the market. They also demonstrate ways in which the system can be used in the future.

These studies range from being developed in 2003 until 2020, showing a sparse timeline of developments in this area. In particular the study demonstrating the immaturity of this area (A closer look at intrusion detection systems for web applications) was published as recently as 2018 [9], highlighting the fact that there is still a lack of work in this area.

84% of all cyber-attacks are happening on the application layer [13], as said in 2015. My classifier address this by focusing on attacks of the application layer, allowing for more precision in detecting them than a generic classifier for all possible cyber-attacks.

As such, my study is relevant today, and up to date. There is still a need to help identify issues in web applications in order to ensure a safer, more private, web.

2.2 Training

To begin my project, I first had to learn how to do penetration testing. I did this primarily by using OWASP's WebGoat [14], and Security Shepard [15] training program, from the OWASP virtual machine. These applications gave me lessons that introduced me to the world of attacks on web applications. Some of the exercises were harder than others where I had to look up walkthrough videos on YouTube to help me understand what I needed to do. I particularly used Jim Jet Wee's YouTube channel as he had a series of instructional videos on WebGoat [16].

3. Tools and methods

3.1 My Planned Approach

My initial plan was to develop my own vulnerable web app, using OWASP's top 10 web app vulnerabilities [17]. From this I was to perform penetration testing on my web app and capture the network traffic. Ideally, I would have collected data for the five most common attacks, and normal network traffic, in large enough volumes to classify the results. I was going to use Python to build a classifier.

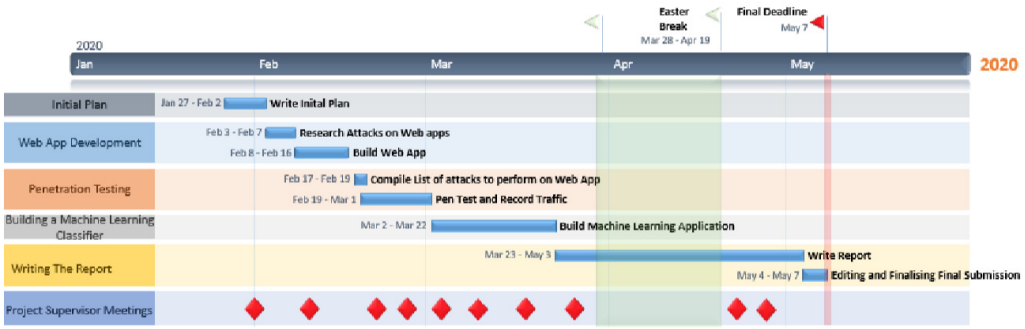

Figure 1: Gantt Chart from my Initial Plan

I did not follow the initial time plan, or set objectives as closely as originally intended, and the project was given a one-week extension for the final deadline due to the outbreak of Covid-19.

3.2 The Reality of my Approach

In the end my time management approach looked more like this:

Week	Task	Subtask	
1 27/01 – 02/02	Initial Plan	Writing Initial Plan	
2 03/02 – 09/02	Research	Look at similar studies and different attacks	
3 10/02 – 16/02	Penetration Testing	Set up virtual machine & training program	
4 17/02 -23/02		Start Penetration Testing	
5 24/02 – 01/03		Collate network traffic	
6 02/03 – 08/03			
7 09/03 – 15/03	Machine Learning	Learn machine learning on Kaggle & try and use my network traffic to fit to model	
8 16/03 – 22/03			**Covid-19 starts**

9 23/03 – 29/03		Find new data sets & test different machine learning models	**officially effecting everyday life**
Easter 1 28/03 – 03/04			
Easter 2 04/04 – 10/04			
Easter 3 11/04 – 17/04	Report Writing	Draft 1	
10 18/04 – 24/04			
11 25/04 – 01/05		Draft 2	
12 02/05 – 08/05			
13 09/05 – 13/05		Final Report	

Figure 2: Time Management Review

As you can see, I took a waterfall model approach with completing each section before going on to the next.

3.2.1 Penetration Testing

In order to perform my penetration testing I had to run web applications from a virtual machine so as to not expose these deliberately vulnerable web apps to the outside world [18]. To do this I installed Virtual Box [19] to run my virtual machine.

I used OWASP's Broken Web Applications Virtual Machine [20] instead of creating my own vulnerable web application. This was with the mindset that it came with a training program to allow me to learn how to best conduct different attacks and more web apps were included rather than just one, which should have provided me with more network traffic for my machine learning.

I also opted to use ZAP (Zed Attack Proxy by OWASP) [21] as it is a free alternative to the BURP suite [18]. I used this primarily for its HUD (heads-up display) and the ease of use in being able to intercept traffic and alter it in order to perform attacks. Furthermore, it had lots of documentation, including a web series showing how to use its features [22].

In order to record traffic, I used Wireshark [23], configured to record any traffic on my virtual machine. I opted to use Wireshark due to my previous experience with the application in other university modules and I am competent using its interface to capture traffic.

3.2.2 Machine Learning

With machine learning I was not sure where to start, and so following some advice I completed an introductory course on Kaggle [24]. This taught me how to build a machine learning model in Python using scikit-learn [25] and pandas [26]. From this I built a basic random forests classifier in Python [appendix 1] and used the network traffic I had gathered from my penetration testing.

However, with this approach I was unable to find any significant data patterns, and so on the advice of my supervisor I then downloaded Weka [27]. Weka was great at quickly comparing the different types of classifiers in order to find the best model for my dataset. However, with the data from my pen testing I later found there was just not enough data and had to take a different approach.

My secondary supervisor advised due to time constraints that instead of trying to build a drastically bigger dataset I should try to use existing datasets available online. Initially it was hard to find a dataset that was only about Web Applications.

I looked at around 20 different datasets. This included some datasets that looked promising such as the HTTP CSIC Torpeda 2012 dataset (which is referenced in appendix 1, as I used it with my python implementation). This dataset had the labels I was looking for with the majority of the attacks I performed on WebGoat, missing only DoS attacks. However, its data was unusable – the attributes included were not useful at all in classifying the data. Other datasets had similar problems. I took about five days of valuable time trying to find an optimal dataset. In the end I selected two to use.

Eventually I decided to use the NSL-KDD dataset [28] and the UNSW-NB15 dataset [29]. The KDD dataset had labelled data to distinguish between malicious and benign network traffic, whereas the UNSW dataset distinguished between attack types.

I was searching for a dataset that would have labelled data for the five most common attacks, especially XSS. However, I could not find a dataset with the relevant information. I needed to build a classifier that also featured these attacks as labels.

For the final part of my project I had intended to use the KDD dataset to train and create my model and my UNSW dataset to test it (changing its labels to show whether it was malicious

or benign rather than the breakdown of attacks). However, they only had five attributes in common, and thus there was not enough compatibility between the datasets to perform this.

4. Deliverables from selected approach

4.1 Background Objectives

My first background objective was to establish an understanding of the different types of attacks on network traffic. To demonstrate this, I produced a list of possible attack vectors and how they work (this list is not exhaustive):

Attack	**Description**	**Vulnerability abused**
1. XSS – Cross-site Scripting	A client-side injection attack. Allows a user to run JavaScript on the page [30].	Where a webpage uses unsanitized user input in its output.
2. SQL Injection	Attacks data driven applications – using object that allow user input to alter a SQL query, to alter the purpose of the query [31].	Typically, where the user input allows string literal escape characters.
3. DOS – Denial of Service	The perpetrator seeks to make a machine or network resource unavailable to its intended users. Floods the targeted machine/resource to overload the systems with superfluous requests [32].	No specific vulnerability other than lack of Intrusion Prevention Systems (IPS) and firewalls to deny traffic coming from same ports or IP addresses.
4. Broken Authentication	Attackers can use automated tools with password lists and dictionary attacks [33]	Lack of automated threat or credential stuffing protections. Not using multi-factor authentication. Application timeouts are not set properly.
5. XXE – XML External Entities	An attacker uses the ability to	If an application

	upload XML to extract data, execute a remote request from the server, scan internal systems, perform a denial-of-service attack and more [34].	accepts XML directly, or XML uploads, or instances into XML documents which is when parsed by an XML processor, it is vulnerable.
6. Sensitive Data Exposure	Executing Man-in-the-middle attacks or stealing clear text off a server in transit [35].	Any data transmitted in clear text, old /weak cryptographic algorithms, weak crypto keys no encryption and lack of verifying server certificates are all vulnerabilities that can allow these attacks.
7. Broken Access Control	Users use privileged functions they should not have access to [36].	Vulnerabilities are where the users can elevate their privilege.
8. Insecure Deserialization	Leads to remote execution attacks. 2 types of attacks: Object / data structure related attacks and data tampering attacks [37]	Applications are vulnerable if they deserialize hostile or tampered objects supplied by an attacker
9. Vulnerable Components	Components typically run with the same privileges as the application itself, so can be used as a backdoor attack. Works particularly in IoT devices [38].	All components and software need to be kept up to date to avoid vulnerabilities.
10. CSRF – Cross Site Request Forgery	An attacker makes users perform actions they do not	Sessions are handled only by cookies. Can be

	intend to perform [39].	stopped by using CSRF tokens.

Figure 3: Research on Attacks and Vulnerabilities

My second background objective was to understand the vulnerabilities a web application can have, and why it makes them insecure. This is included in the table above.

4.2 Primary Aims

My first primary aim was to build a web application with vulnerabilities. I decided not to do this because I felt I could be more effective with the main focuses of the project – penetration testing and machine learning. In theory, by using different web application rather than just one, I could create a more effective machine classifier, and learn more about web attacks.

My second primary aim was to record and document attacks on the web app. I did perform and record attacks that I ran on the virtual machine web apps rather than one I had developed [appendix 2]. However, I only recorded XSS attacks, as I was trying to get my machine classifier to work with these attacks before recording any others. But I have completed all the attacks in the lessons on WebGoat, and SecurityShephard.

The third primary aim was to code a network classifier, which I achieved in Python [appendix 1], although it was not particularly successful due to the data, I provided it with.

Therefore, for my fourth aim of categorising benign and malicious data I used Weka, to experiment with many models and find my ideal one [appendix 3].

4.3 Secondary Aim

I fulfilled my secondary aim of identifying what kind of attack is taking place via a different dataset with different attributes, and labels. However, I used the same methodology as I did previously when discriminating between malicious and benign. [appendix 4]

5. Design and Results of Experiments

5.1 Classifier Models

My experiments consisted of using Weka to find the best learning technique for my classifier to sort the different network traffic. To do this I tested 38 different classifier models, these divided into seven subcategories.

Initially I did some research into what classifier model would best suit my project. To do this I looked at three different websites. This research suggested that Naïve Bayes would not be suitable as it is typically used for binary classification [40].

Contrastingly I found research suggesting that CART would be particularly apt for my application [41]. However, CART (classification and regression trees) was not an option on Weka. Another site [42] came with the following flowchart:

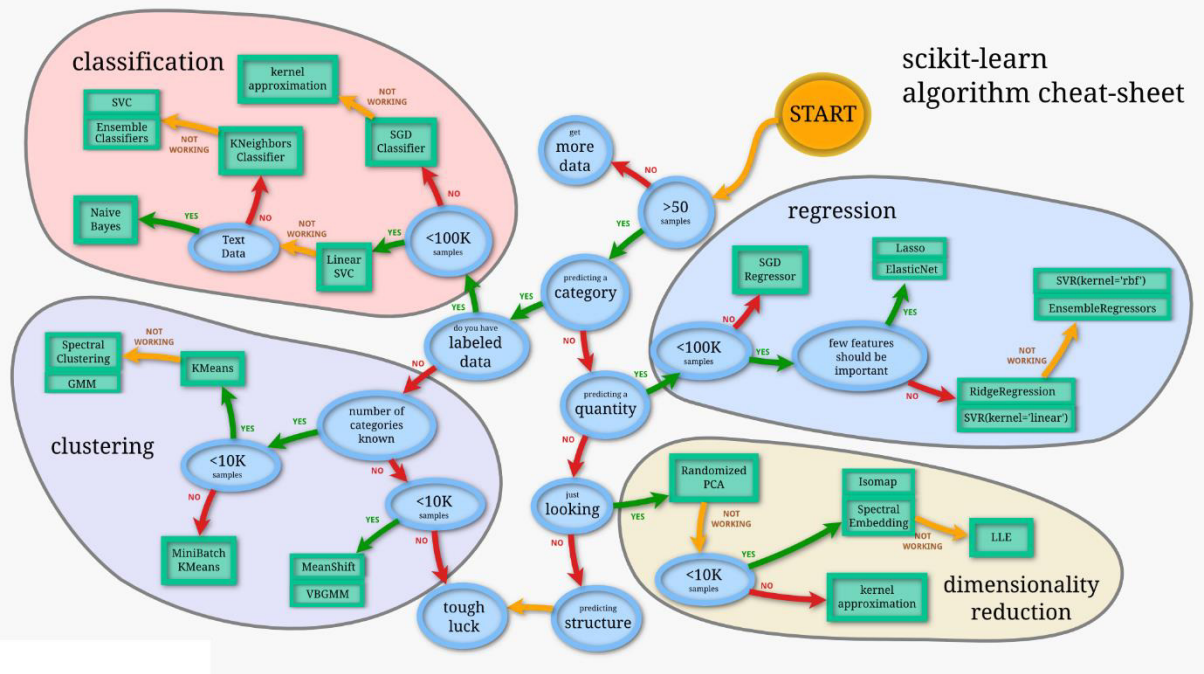

Figure 4: scikit-learn algorithm cheat-sheet

This suggests that since I have more than 100,000 samples, with labelled data, that an SGD classifier would be most appropriate, or failing this kernel approximation.

After using this research to understand the different classifier models, I decided to experiment with Weka's classification. The top five algorithms for classification in Weka are meant to be logistic regression, Naïve Bayes, decision tress, k-nearest neighbours, and support vector machines [43]. These also come in the top six common classification algorithms on another site [44]. With this in mind I looked first at the suggested and named classifiers, and then decided to test every classifier Weka would allow me to.

5.1.1 Bayes

A Bayesian classifier is built on the idea that the job of a class is to predict the values of features for members of that class [45]. As such Bayes' rule can be used to predict classes given some of the values that are features of the class. Therefore Bayesian classifiers are generative models.

Bayes rule states that:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

5.1.1.1 Naïve Bayes

Naïve Bayes methods are a set of methods based on applying Bayes' theorem with the assumption of conditional independence between every pair of features given the value of the class variable [46]. There are three main types of naïve Bayes varying by the assumptions they make regarding the distribution of $P(x_i|y)$:

- Gaussian Naïve Bayes – the likelihood of features is assumed to be Gaussian: $P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$
- Multinomial Naïve Bayes – used for multinomially distributed data
- Complement Naïve Bayes – an adaption of the standard multinomial naïve Bayes algorithm that is particularly suited for imbalanced data sets.

Weka describes Naïve Bayes by stating 'numeric estimator precision values are chosen based on analysis of the training data. For this reason, the classifier is not an updatable classifier.'. Its capabilities are binary class, missing class values and nominal class. In this case I will be using it for nominal classes.

5.1.1.2 Bayes Net

A Bayes Net or Bayesian Network classifier is one which assumes strong (naïve) independence assumptions based on Bayes' Theorem [47].

It assumes that a feature being present in a class is unrelated to any other features being present. It assigns a probability to every event of interest.

5.1.1.3 Naïve Bayes Updatable

This classifier works similarly to naïve Bayes, but it can be updated, with training data changing as more data becomes available making it a more dynamic model.

5.1.2 Functions

These classifier models all depend on one key function. Machine learning functions let you work with your data set in different stages of the data analysis process [48].

5.1.2.1 SGD

SGD (stochastic gradient descent) is an estimator that implements linear models with SGD learning: the gradient of the loss is estimated, each sample at a time, and the model is updated along the way with a decreasing strength schedule / learning rate [49].

The model parameters are shrunk towards the zero-vector using the squared Euclidean norm L2, the absolute norm L1 or a combination of both (elastic net).

5.1.2.2 Logistic

Logistic Regression is a classifier that uses statistics. It uses a linear regression equation to produce discrete binary outputs [50].

The activation function for logistic regression is:

$h(x) = \theta(\sum_{i=0}^{d} w_i x_i)$

Where w_i = weights/coefficients, h = hypothesis set that selected classifier brings and θ = sigmoid/logistic function.

As I will be using it in a supervised classifier, it helps to converge uncertain posterior values with a differentiable decision function.

5.1.2.3 SGD Text

SGD Text implements stochastic gradient descent for machine learning, but operates directly, and only on string attributes. Other types of input are accepted but ignored for training and classification [51]. This helps with some fields of my datasets but completely disregards others and so it will be interesting to see how accurate it is. I believe it would be more suited for classifying documents in the semantic web, taking where words are commonly used and sorting them into what they relate to. However, I wish to see how it affects my data.

5.1.2.4 Simple Logistic

The simple logistic regression classifiers use simple regression functions as base learners. These are used with LogitBoost to iterate through multiple times to find an optimal classification. The optimal number of iterations to perform is cross validated, leading to automatic attribute selection [52].

5.1.2.5 SMO

SMO uses John Platt's sequential minimal optimisation algorithm to train a support vector classifier. It globally replaces all missing values and transforms nominal attributes into binary ones. It also normalises all attributes [53].

As this is a multi-class problem it solves it using pairwise-classification.

5.1.2.6 Voted Perceptron

Voted Perceptron as a classifier is an implementation of the voted perceptron algorithm by Freund and Schapire [54]. It globally replaces any missing values (of which I have a few) and transforms nominal attributes (of which I have roughly three in each dataset), into binary ones.

The algorithm itself takes advantage of data that are linearly separable with large margins [55]. It is comparable to SVM and is said to have a similar accuracy but is simpler and more efficient.

5.1.3 Lazy

Lazy learning is a where generalising training data waits until a query is made. This means that data set can be continuously updated with new entries [56] that stop training data from being rendered obsolete.

5.1.3.1 IbK

Ib stands for instance based, and the K shows that the number can be adjusted. This is really a k-nearest neighbours' algorithm, where K > 1 [57]. In this context it selects the value of K based on cross validation [58].

5.1.3.2 Kstar

K* is an instance-based classifier. The class of a test instance is based upon the class of similar training instances as determined by some similarity function. It differs from

other instance-based learners like IbK as it uses an entropy-based distance function to calculate similarity [59].

5.1.3.3 LWL

Locally weighted learning uses an instance-based algorithm to assign instance weights. When used for classification it often used Naïve Bayes to classify the data [61].

5.1.4 Meta

In meta-learning automated learning algorithms are applied on metadata about machine learning experiments [61]. The main goal is to use metadata to understand how automatic learning can become flexible and thus improve the performance of existing algorithms. As such these classifiers tend to take longer, as they can iterate through classifying multiple times.

5.1.4.1 Ada Boost M1

AdaBoost is designed to boost a nominal classifier. It often drastically improves performance but can only be used on nominal problems and can overfit [62]. It is short for adaptive boosting and is most effective using weak learners. The final equation for classification can be represented as:

$$F(x) = sign(\sum_{m=1}^{M} \theta_m f_m(x))$$

Where f_m stands for the m_θ weak classifier and θ_m is the corresponding weight. It is the weighted combination of M weak classifiers.

5.1.4.2 Attribute Selected Classifier

The dimensionality of both training and test data is reduced by attribute selection before being passed on to a classifier [63].

5.1.4.3 Classification Via Regression

Classes are binarized and one regression model is used for each class value [64].

5.1.4.4 Filtered Classifier

This is a class for running a classifier on data that has been passed through a filter. The classifier and filter alike have their structure based exclusively on the training data. Test instances will be processed by the filter without changing their structure. Therefore, it only functions where weights /

attribute weights are equal. As such, when unequal weights are present the instances and / or attributes are resampled with replacements based on the weights before they are passed to the filter or classifier as appropriate [65].

5.1.4.5 Iterative Classifier Optimizer

The iterative classifier optimiser chooses the best number of iterations for an iterative classifier – such as LogitBoost (as seen below) – using cross-validation or a percentage split evaluation. This produces a choice that is the optimal number of iterations [66].

5.1.4.6 Logit Boost

Logit Boost is a class for performing additive logistic regression. It performs classification using a regression scheme as a base learner and can handle multi-class problems [67].

5.1.4.7 Multi Class Classifier

This is a metaclassifier for handling multi-class datasets with 2-class classifiers. This classifier is also capable of applying error correcting output codes for increased accuracy. If the base classifier cannot handle instance weights, and they are not uniform, the data will be resampled with replacements based on the weights before being passed to the base classifier, similarly to the filtered classifier [68].

5.1.4.8 Multi Scheme

This class is used to select a classifier from among several using cross validation or the performance on the training data [69].

5.1.4.9 Random Committee

Random Committee is a class for building an ensemble of base classifiers. Each base classifier is built using a different random number of seed based on the same data. The final prediction is an average of the predictions generated by each individual base classifier [70].

5.1.4.10 Randomizable Filtered Classifier

Randomizable filtered classifier is a class for running any classifier on data that has been passed through any filter. The structure is based solely on the training data for both the classifier and the filter, and the test instances will be

processed without changing their structure [71]. This is similar to the filtered classifier but allows for randomising.

5.1.4.11 Random Sub space

Random subspace constructs a decision tree-based classifier that maintains the highest accuracy on the training data and improves on generalisation accuracy as it grows in complexity. The classifier consists of multiple trees constructed systematically by pseudo-randomly selecting subsets of components of the feature vector, that is, trees constructed in randomly chosen subspaces [72].

5.1.4.12 Stacking

Stacking combines several classifiers [73]. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model. The base level often consists of different leaning algorithms making it heterogenous more often than not [74].

5.1.4.13 Vote

The voting classifier works as a wrapper for a set of different classifiers that are trained and evaluated in parallel, in order to exploit the different peculiarities of each algorithm [75]. Data sets are trained using different algorithms and then ensembled to predict the final output. This is based on majority voting, using one of two strategies:

- Hard voting/ majority voting: the class that received the highest number of votes will be chosen
- Soft voting: the probability vector for each predicted class for every classifier are summed up and averaged. The winning class is the one corresponding to the highest value. This only works if the classifiers are well calibrated, and thus I will not be using this method.

5.1.4.14 Weighted Instances Handler Wrapper

This method utilises a generic wrapper around any classifier to enable weighted instances support. It uses resampling with weights if the base classifier is not implementing the provided interface, and there are instance weights other than 1.0 present. By default, the training data is passed through to the base classifier if it can handle the instance weights. However, it is possible to force the use of resampling with weights as well [76].

5.1.5 Miscellaneous

Miscellaneous is a very small section of classifiers on Weka. These are the classifiers that are not rule, tree, function or meta – based. Nor are they to do with Bayes' theorem.

5.1.5.1 Input Mapped Classifier

In general, all classifiers take an input and map it to a class. The input mapped classifier is a wrapper classifier that deals with training and testing data that are incompatible by building a mapping between the training data that a classifier has been built with and the test data's structure. Attributes in the model that are not found in the incoming instances receive missing values, as do incoming nominal attribute values that the classifier has not seen before [77].

This would be relevant if I managed to build a model with one of my two datasets and test it with the other. However, this was not possible as there just was not enough compatible attributes. I tested it on my other datasets regardless as they allowed it.

5.1.6 Rules

Rule-based machine learning identifies, learns, or evolves rules [78]. This makes these models discriminative.

5.1.6.1 PART

PART uses partial trees to generate a decision list that is shown in the output. This is the set of rules that classification is then based on [79]. It is a separate and conquer method where it makes a rule, removes the instances covered by the rule and continues making rules for the remaining instances. Rules are made via building partial trees and reading off the rule for the largest leaf.

Due to it being called PART I initially assumed it was related to CART, which was recommended for my style of machine learning, but was not available on Weka. As I could not find any documentation on what PART stands for – unless it is just for partial tree – I can neither prove nor disprove this assumption. However, it does not seem to mention regression in any of its documentation and so it is probably not connected.

5.1.6.2 Decision Table

Decision Table in Weka is a class for building and using a simple decision table majority classifier [80].

5.1.6.3 Jrip

Jrip is also known as Ripper. It implements a propositional rule learner: Repeated Incremental Pruning to Produce Error Reduction.

It consists of 2 main stages: building stage and optimizing stage. The building stage can be broken down into a growing phase and a prune phase, used in repletion until the description length of the ruleset and example is significantly larger than any previous description length, or there are no positive examples, or the error rate >= 50% [81].

5.1.6.4 OneR

OneR is short for one rule. It generates one rule for each predictor in the data, then selects the rule with the smallest total error as its 'one rule'. To create a rule for a predictor a frequency table for each predictor against the target is generated [82]. Described by Weka as using the minimum error attribute for prediction, discretising numeric attributes [83].

5.1.7 Trees

Tree-based machine learning is similar to rule based, as it divides data into classes based on how it fits into certain rules. This means it is also discriminative in its models.

5.1.7.1 Random Forest

Random forest is an extension of bagging; it creates a forest of random trees [84]. It is a meta estimator that fits a number of decision tree classifiers on various sub samples from the dataset and uses averaging to improve predictive accuracy and control overfitting [85]. It has a good reputation as being one of the more accurate models and therefore was one of the first models I looked at.

5.1.7.2 Decision Stump

A decision stump is a decision tree which uses a single attribute for splitting. For discrete attributes this generally means that the tree is constructed of only an interior node. If the attribute is numerical the tree may be more complex [86]. My data consists mainly of numerical data and so this may slow the algorithm down, but it is a good model to avoid overfitting. It is usually used as a weak learner in conjunction with a boosting algorithm, so I do not expect much from it by itself.

5.1.7.3 Hoeffding Tree

This model is an incremental, anytime decision tree induction algorithm that can learn from massive data streams – assuming that the distribution, generating examples, does not change overtime. This would be particularly useful for my model if I can generate some code to automatically format the network traffic for the model and thus categorise the traffic in real-time.

Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This is based on the Hoeffding bound which quantifies the number of operations needed to estimate some statistic within a prescribed precision. One key benefit of this model is that it has sound guarantees of performance [87].

5.1.7.3 J48

J48 is a Java implementation of the C4.5 algorithm. It produces a decision tree based on information theory. This assumes the best attribute to split on is the attribute with the greatest information gain [88]. It is said to be one of the best machine learning algorithms to examine data categorically and continuously [89], which should make it ideal if I can deploy my model in real-time to categorise network traffic.

5.1.7.4 LMT

LMT is a classifier for building Logistic Model Trees, which are classification trees with logistic regression functions at their leaves. The algorithm can deal with binary and multiclass-target variables [90]. Here I will be using it for both, depending on the dataset that I use, making it ideal. It can also deal with numeric and nominal attributes, of which I have both.

5.1.7.5 Random Tree

This model constructs a tree that considers K randomly chosen attributes at each node and performs no pruning. It has an option to allow estimation of class probabilities based on a hold-out set (backfitting) [91]. However, I did not look at this stage into any of the options other than Weka’s default settings.

5.1.7.6 REP Tree

REP is a fast decision tree learner. It builds a decision tree using information gain and prunes it using reduced-error

pruning with backfitting. It only soft values for numeric attributes once [92], which seeing as I have many numeric attributes may not be ideal.

5.2 The Data

5.2.1 Comparing the training data set to the testing data set

With the KDD dataset I had a separate testing and training file. Therefore, I have compared the two files to see how similar they are by checking the mean and standard deviation of each of the numerical attributes.

	Testing set averages	Testing Set standard devn.	Training set averages	Training Set standard devn.	Diff. between averages	Diff. between standard devns.
duration	218.8591	1407.177	287.1447	2604.515	68.28557	1197.339
src_bytes	10395.45	472786.4	45566.74	5870331	35171.29	5397545
dst_bytes	2056.019	21219.3	19779.11	4021269	17723.1	4000050
land	0.000311	0.017619	0.000198	0.014086	0.000112	0.003533
wrong_fragment	0.008428	0.142599	0.022687	0.25353	0.014259	0.110931
urgent	0.00071	0.036473	0.000111	0.014366	0.000599	0.022107
hot	0.105394	0.928428	0.204409	2.149968	0.099015	1.22154
num_failed_logins	0.021647	0.150328	0.001222	0.045239	0.020424	0.105089
logged_in	0.442202	0.496659	0.395736	0.48901	0.046466	0.007649
num_compromised	0.119899	7.269597	0.27925	23.94204	0.159351	16.67245
root_shell	0.00244	0.049334	0.001342	0.036603	0.001098	0.012731
su_attempted	0.000266	0.02106	0.001103	0.045154	0.000837	0.024094
num_root	0.114665	8.041614	0.302192	24.39962	0.187527	16.358
num_file_creations	0.008738	0.676842	0.012669	0.483935	0.003931	0.192907
num_shells	0.001153	0.048014	0.000413	0.022181	0.000741	0.025833
num_access_files	0.003549	0.067829	0.004096	0.09937	0.000547	0.03154
num_outbound_cmds	0	0	0	0	0	0
is_host_login	0.000488	0.022084	7.94E-06	0.002817	0.00048	0.019267
is_guest_login	0.028433	0.166211	0.009423	0.096612	0.019011	0.069599
count	79.02834	128.5392	84.10755	114.5086	5.07921	14.03064
srv_count	31.12438	89.06253	27.73789	72.63584	3.386491	16.42669
serror_rate	0.102924	0.295367	0.284485	0.446456	0.181561	0.151089
srv_serror_rate	0.103635	0.298332	0.282485	0.447022	0.17885	0.148691
rerror_rate	0.238463	0.416118	0.119958	0.320436	0.118505	0.095682
srv_rerror_rate	0.235179	0.416215	0.121183	0.323647	0.113995	0.092568
same_srv_rate	0.740345	0.412496	0.660928	0.439623	0.079417	0.027127
diff_srv_rate	0.094074	0.259138	0.063053	0.180314	0.031021	0.078823
srv_diff_host_rate	0.09811	0.253545	0.097322	0.25983	0.000789	0.006285
dst_host_count	193.8694	94.03566	182.1489	99.20621	11.72047	5.17055
dst_host_srv_count	140.7505	111.784	115.653	110.7027	25.09753	1.081231

dst_host_same_srv_rate	0.608722	0.435688	0.521242	0.448949	0.08748	0.013261
dst_host_diff_srv_rate	0.09054	0.220717	0.082951	0.188922	0.007589	0.031795
dst_host_same_src_port_rate	0.132261	0.306268	0.148379	0.308997	0.016118	0.002729
dst_host_srv_diff_host_rate	0.019638	0.085394	0.032542	0.112564	0.012904	0.02717
dst_host_serror_rate	0.097814	0.273139	0.284452	0.444784	0.186639	0.171645
dst_host_srv_serror_rate	0.099426	0.281866	0.278485	0.445669	0.179059	0.163803
dst_host_rerror_rate	0.233385	0.387229	0.118832	0.306557	0.114553	0.080671
dst_host_srv_rerror_rate	0.226683	0.400875	0.12024	0.319459	0.106443	0.081415
AVERAGE	345.239	13048.93	1738.082	260385.8	1394.998	247338.5
STDEV	1707.502	76672.15	7974.424	1139642	6318.537	1075642

Figure 5: Comparing KDD Testing and Training Dataset Values

It was only through this that I noticed that num_outbound_cmds was, in this dataset, a pointless attribute as in neither testing nor training did it have any values. This might change with other data if it is used in the future, but without adding data to the training set this will do nothing to classify the traffic.

I was unable to do this with the UNSW dataset as I was having to use Weka's splitting function on one file and so was unable to compare the mean and standard deviations as I do not know what values would be sorted into each purpose.

5.2.2 Comparing Attributes

I then compared the attributes of the datasets using Pearson's Correlation Coefficient. The issue with this was that Pearson's only works on numerical data, but the labels for my data were only nominal.

5.2.2.1 KDD Dataset

Therefore, for the KDD dataset I made the labels binary, 1 for anomalous data, and 0 for normal data. For the three other nominal attributes I found their most common values and used those as binary values for comparison in Pearson's. While this does not show the impact of the attribute as whole it gives a guidance on how important the attribute is.

To directly compare the different coefficient values (to find the most important attributes for classification) I found the absolute values of the coefficients, as whether it is a negative or positive correlation does not matter in this context – only the magnitude does.

In order to interpret the results, I used Deborah J. Rumsey's guidelines [93]. These state that around 0.5 is a moderate relationship, around 0.3 is a weak relationship, around 0.7 is a strong relationship, 1 is absolute, and 0 demonstrates no relation between the arrays. I have indicated these interpretations using colours in the chart below:

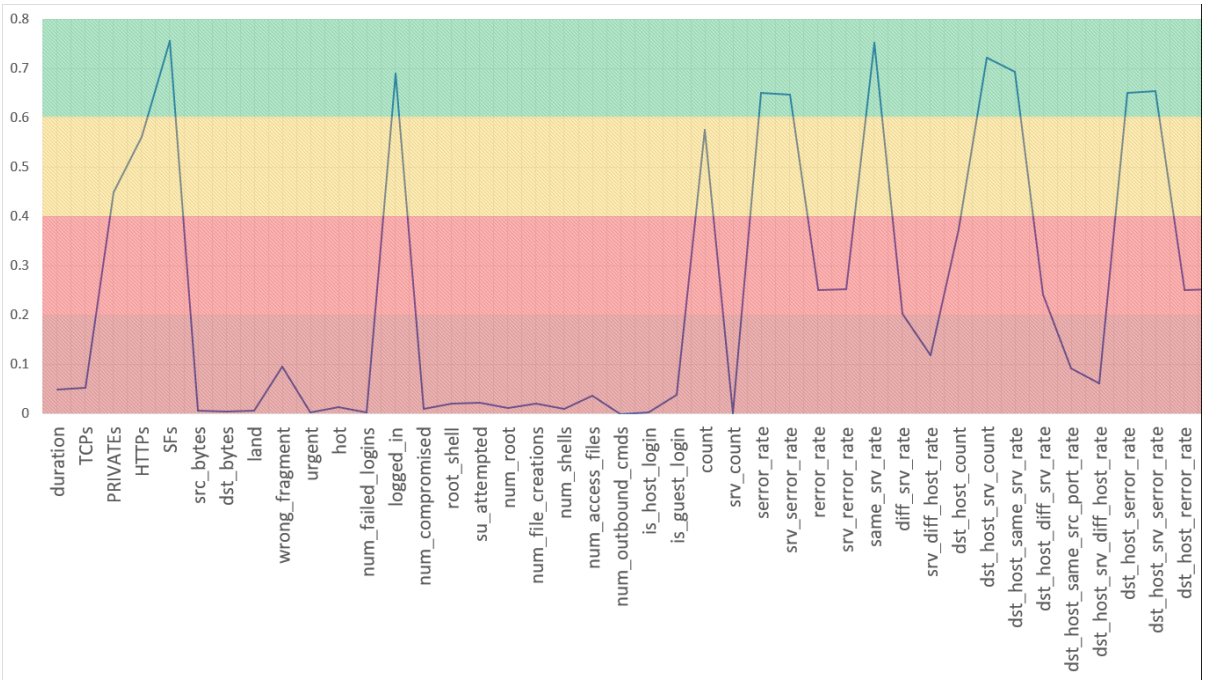

Figure 6: KDD Magnitude of Pearson's Correlation

This indicates that the SF flag has a strong correlation to whether the data is benign or malicious, along with the values of logged_in, serror_rate, srv_rerror_rate, same_srv_rate, dst_host_srv_count, dst_host_same_srv_rate, dst_host_serror_rate, dst_host_srv_serror_rate.

Therefore, I would presume the flag attribute, and the others mentioned above are the essential attributes for the machine learning program to discriminate between malicious and benign network traffic.

5.2.2.2 UNSW Dataset

The issue with the UNSW dataset is that the label attribute was both nominal and had ten different values. I could not assign them numerical values of 0 to 9 as this would suggest that some values were closer to another than to others and bias the results. Instead I looked at each type of attack individually: using binary values to show for each record whether it is a selected type of attack. This meant finding

ten Pearson Correlation Coefficient values for each other attribute.

On top of this I had another 3 nominal attributes. For these I took the binary value of whether or not they were roughly the modal attribute or not. For the protocol attribute around 46% of the records were TCP, and around 36% were UDP. As such, I looked at these two values individually. For the service attribute the majority of the records (54%) had a null value, and therefore I tested how much of an impact a value there had on its classification. Finally, the state attribute was composed of 47% INT and 43% FIN values, and so I tested the presence of these as binary variables.

These results were rather interesting as it was astonishing how low the correlation was for all attributes for some of the attack types. This is shown below:

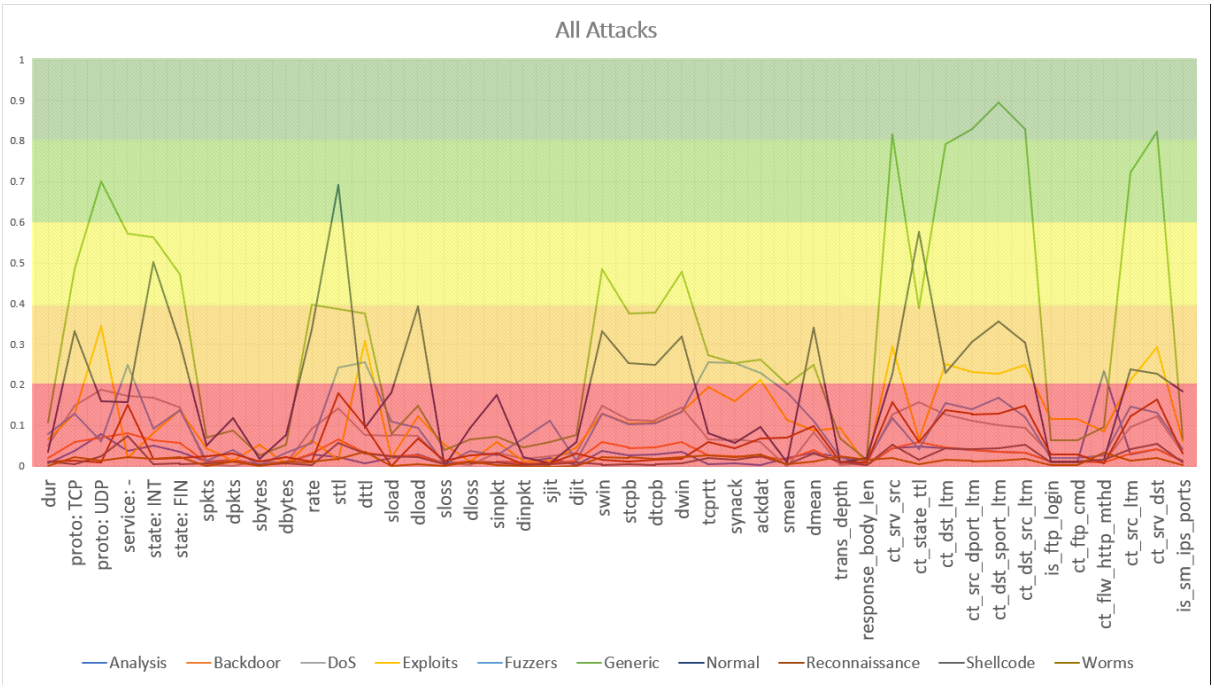

Figure 7: UNSW Magnitude of Pearson's Correlation for All Attacks

This shows that only normal and generic had any attributes that had a strong correlation to their label. It therefore shows that it is the combination of attributes together that leads to them being labelled by the machine learning techniques rather than a particular attribute.

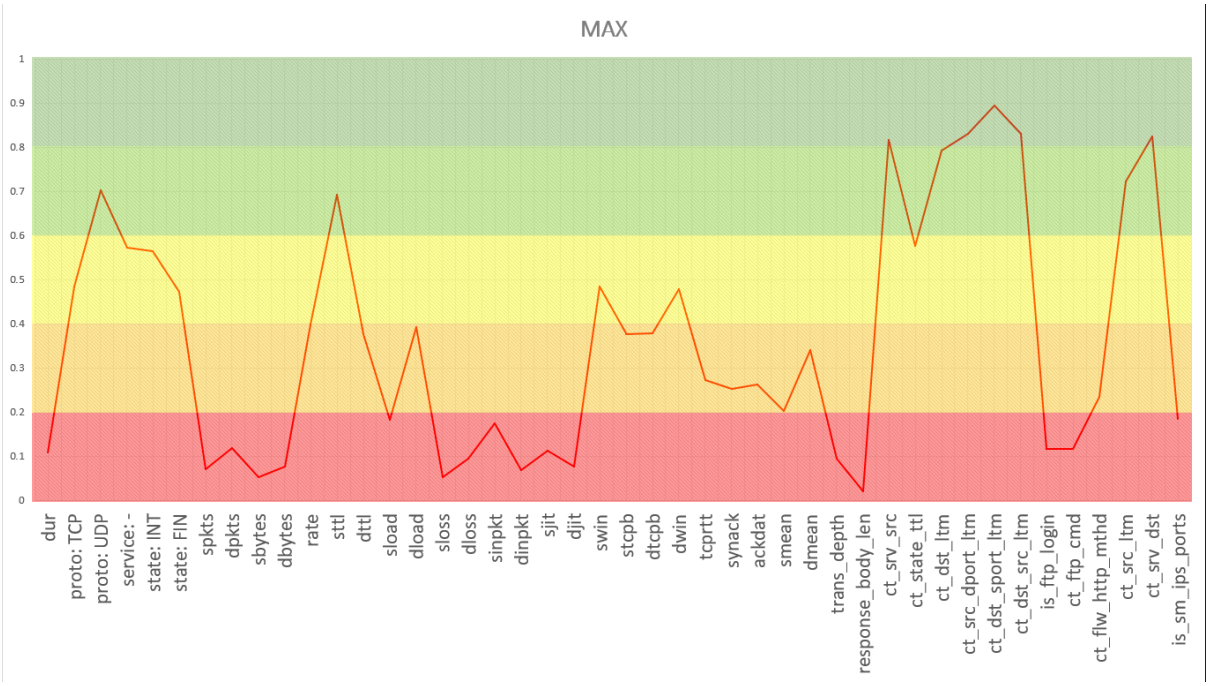

Figure 8: UNSW Maximum Magnitude of Pearson's Correlation for All Attacks

Therefore, I looked at the maximum correlation each attribute has to any attack, noting the attributes that are particularly low are the duration, source packets, destination packets, source bytes, destination bytes, source loss, destination loss, etcs.

One thing to note is that the way in which I have used Pearson's takes a similar approach to Naïve Bayes, as clearly some of these attributes will be related to each other, but I have disregarded this fact for finding only how they relate to the labels. Furthermore, one reason for the scores being so low may be because of the relation to a single attack rather than a group of them. The nominal data makes this dataset particularly hard to model using Pearson's.

This is most clearly demonstrated when I take the mean value from the attacks for all attributes:

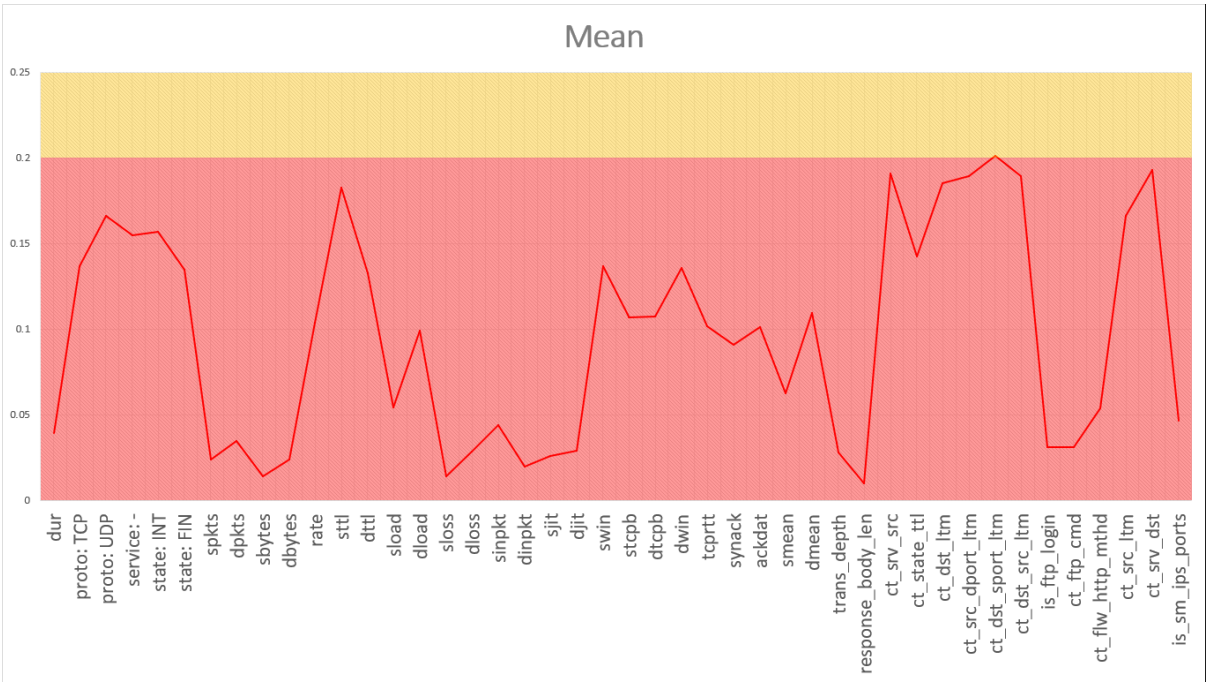

Figure 9: UNSW Mean Magnitude of Pearson's Correlation for All Attacks

This shows a very weak correlation with all attributes, with only one attribute just making it closer to being fairly weak (ct_dst_sport_Itm – the number of rows of the same destination ip address and source port number in 100 rows) rather than very weak.

However, it is still interesting to look at which attribute has the strongest correlation to each attack on an individual basis, even though most of these correlations are very weak.

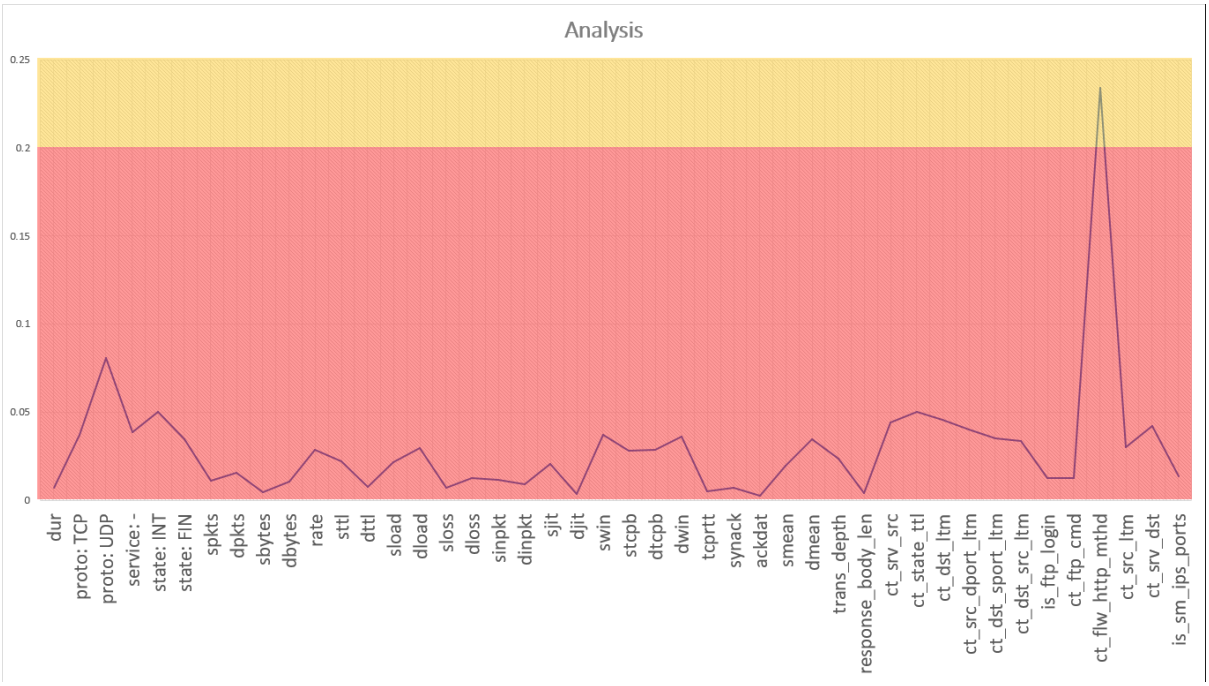

Figure 10: UNSW Magnitude of Pearson's Correlation for Analysis Attack

An analysis attack uses active reconnaissance, scanning a network in some manner but not exploiting vulnerabilities. This consists of attacks such as port scans, vulnerability scans, spam files and footprinting [94].

For the Analysis attack the ct_flw_http_mthd (number of methods such as Get and Post in http service) attribute stands out significantly from the rest of the attributes, and though it is still a weak correlation, the difference from the rest of the attributes is noteworthy. This will be because in attacks such as port scans the system will send GET requests to find port numbers of various machines.

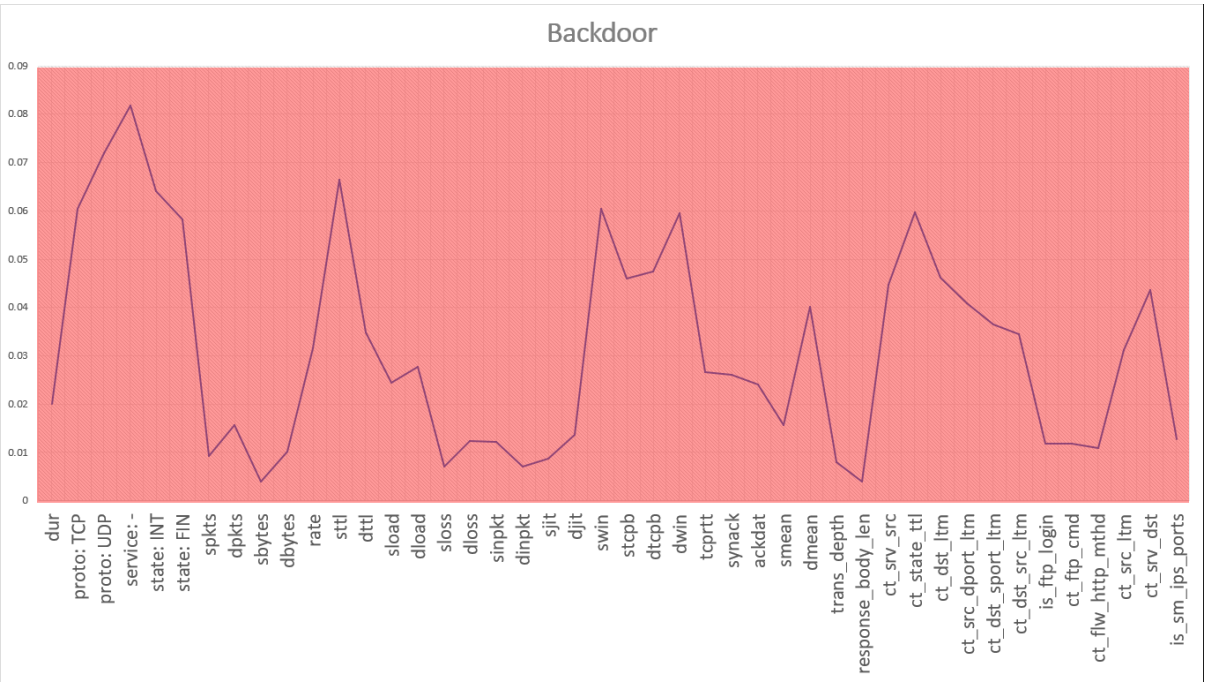

Figure 11: UNSW Magnitude of Pearson's Correlation for Backdoor Attack

A backdoor attack utilizes a technique where attackers use a legitimate system portal to gain illicit access. Backdoors use malicious software to install themselves in a computer system and provide remote access to attackers as part of an exploit [94].

With the backdoor attack every Pearson value is less than 0.09, which is very weak, and could mean that the attributes have nothing to do with the classification. However, there are still some clear differences from one attribute to the next; with the absence of a service specified topping the charts. Perhaps suggesting if I tried some of the variations of attacks this would have a higher correlation, but there are very few occurrences of these and so at the time this seemed unimportant.

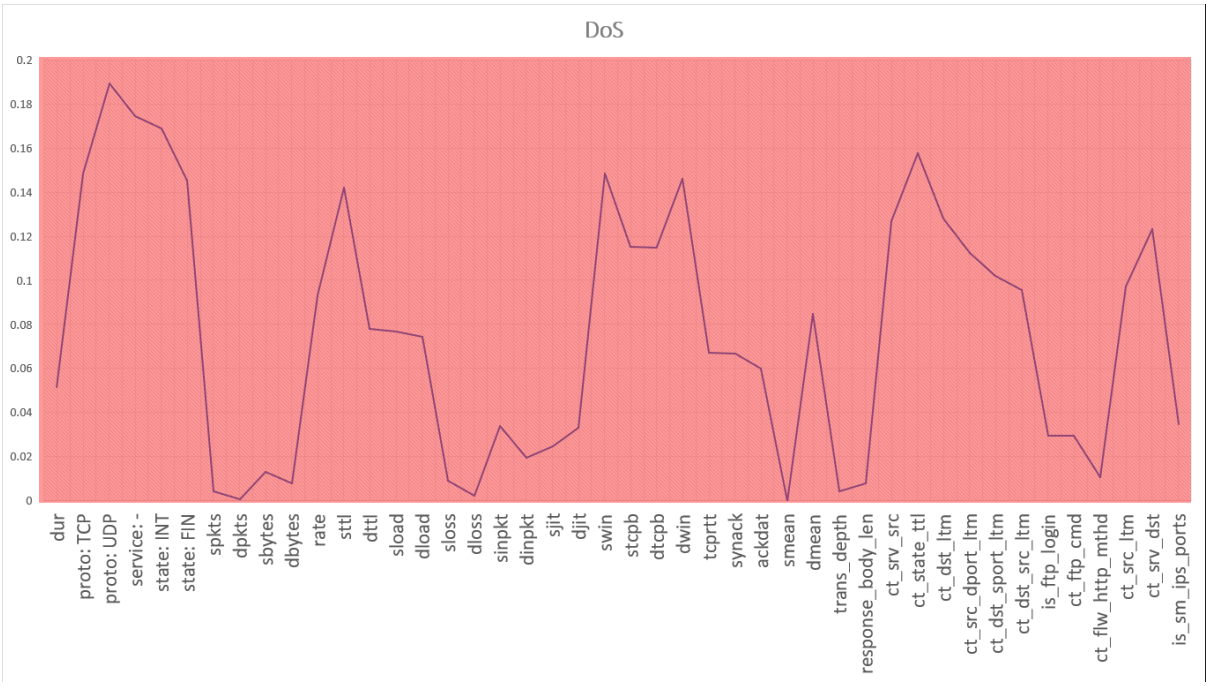

Figure 12: UNSW Magnitude of Pearson's Correlation for DoS Attack

DoS attacks are described earlier in my attack portfolio. They compromise a machine with several illegitimate connection requests to make the network resources unavailable to its intended users [94]. These a particularly hard to detect and prevent but they can be stopped when in progress and thus particularly important for this system to detect.

DoS still had a very weak correlation for each attribute, but the magnitude of its correlation did prevail over the Backdoor attack; with the highest attribute scoring a magnitude of 0.19 for UDP being the protocol. Again, this is a nominal attribute where I did not test all values and thus may be something to look into.

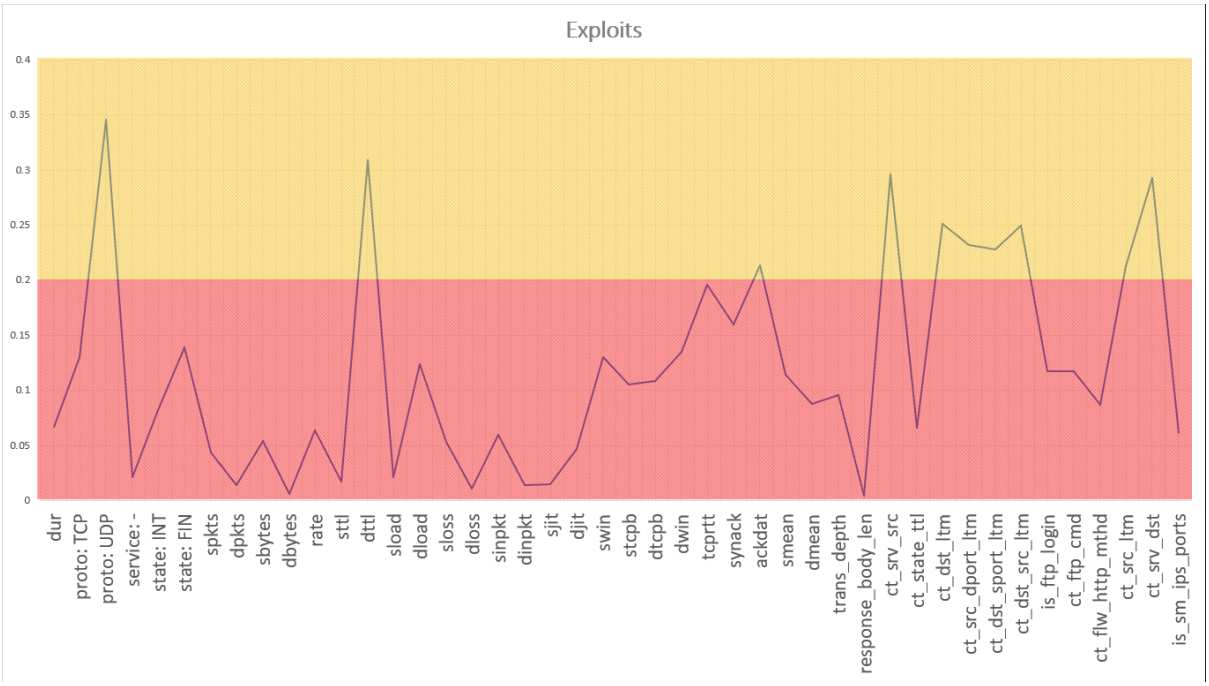

Figure 13: UNSW Magnitude of Pearson's Correlation for Exploits Attack

Exploit attacks are generally achieved by targeting and compromising known vulnerabilities which exist in operating systems [94].

Exploits has some moderately weak results rather than just very weak. The highest result once more coming from the protocol attribute with the value UDP. This makes me very curious about these nominal values. There were also another 8 attributes that had a value closer to 0.3 than 0.

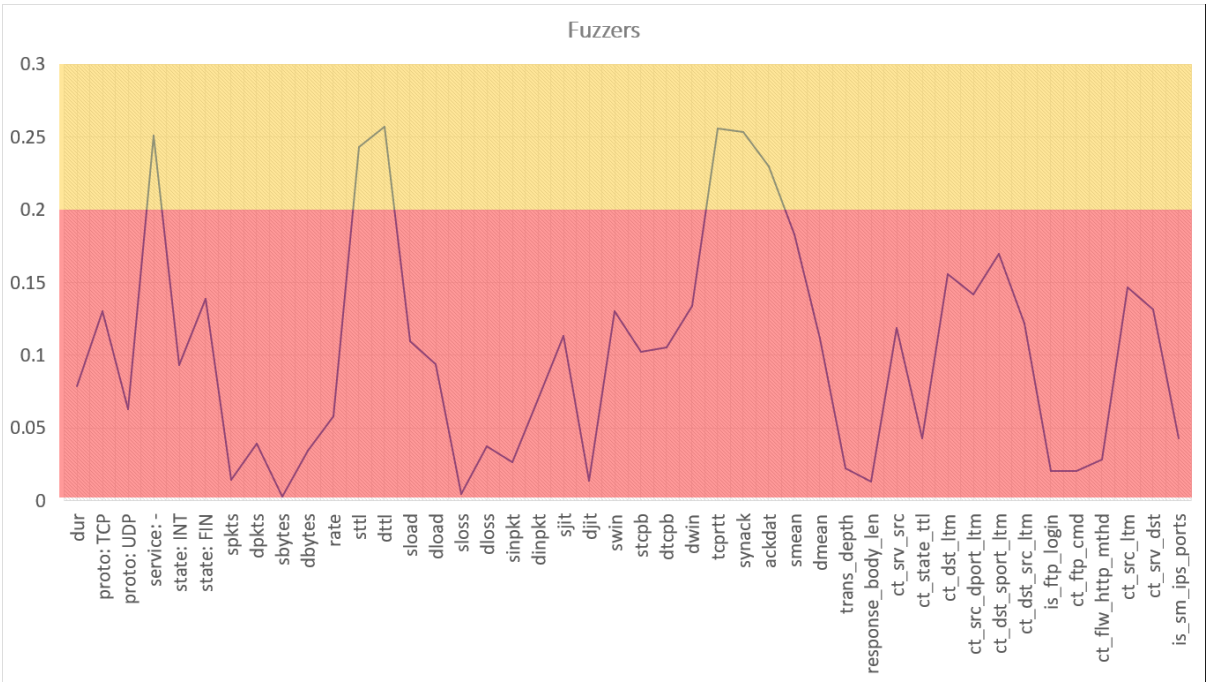

Figure 14: UNSW Magnitude of Pearson's Correlation for Fuzzers Attack

Using Fuzzers is an attack that uses massive amounts of randomised data (fuzz) to trigger a failure of a network or make an attempt to crash important servers on a network [94].

The Fuzzers attack luckily had some values that were not in the very weak category. Despite the correlation still being weak this gave some hope to the interpretation of the data. Of the six attributes that had a value above 0.2 again one of them was a nominal attribute: the value of the service. But the largest correlation was from dttl (destination to source time live). I cannot fathom how the 'fuzz' would impact this and so it is interesting to note.

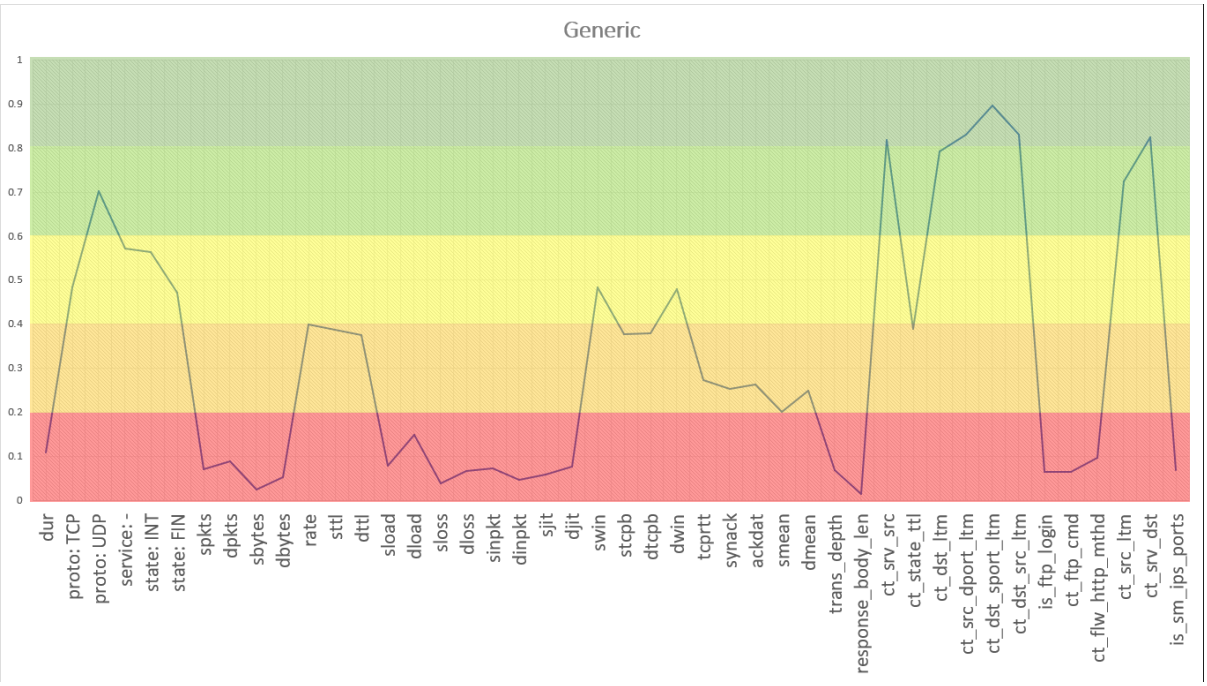

Figure 15: UNSW Magnitude of Pearson's Correlation for Generic Attack

Generic attacks are based on cyphers, essentially performing a collision attack on the secret key generated by the cryptographic principles. This can be applied to block, stream, and message authentication code (MAC) cyphers [94].

These attacks were one of the only results that showed positive results with some of the attributes having a very strong correlation. The very strong correlation attributes were ct_srv_dst, ct_dst_src_ltm, ct_dst_sport_ltm, ct_src_dport_ltm, and ct_srv_src – all numeric attributes that are all counts of rows that had something in common within 100 rows [94]. This is likely to the brute-force nature of the attack in attempting to find the secret key.

But another value that was reasonably high in correlation that I want to note is once again the nominal attribute of

protocol, with whether it has the value of UDP or not. If I had a way to evaluate the nominal values without altering them to binary, I may find some stronger correlations.

However, it is interesting to note that Generic is the only actual attack that shows strong correlations to attributes, the only other label showing a strong correlation is the benign data (normal). Thus, generic clearly stands out against other attacks types.

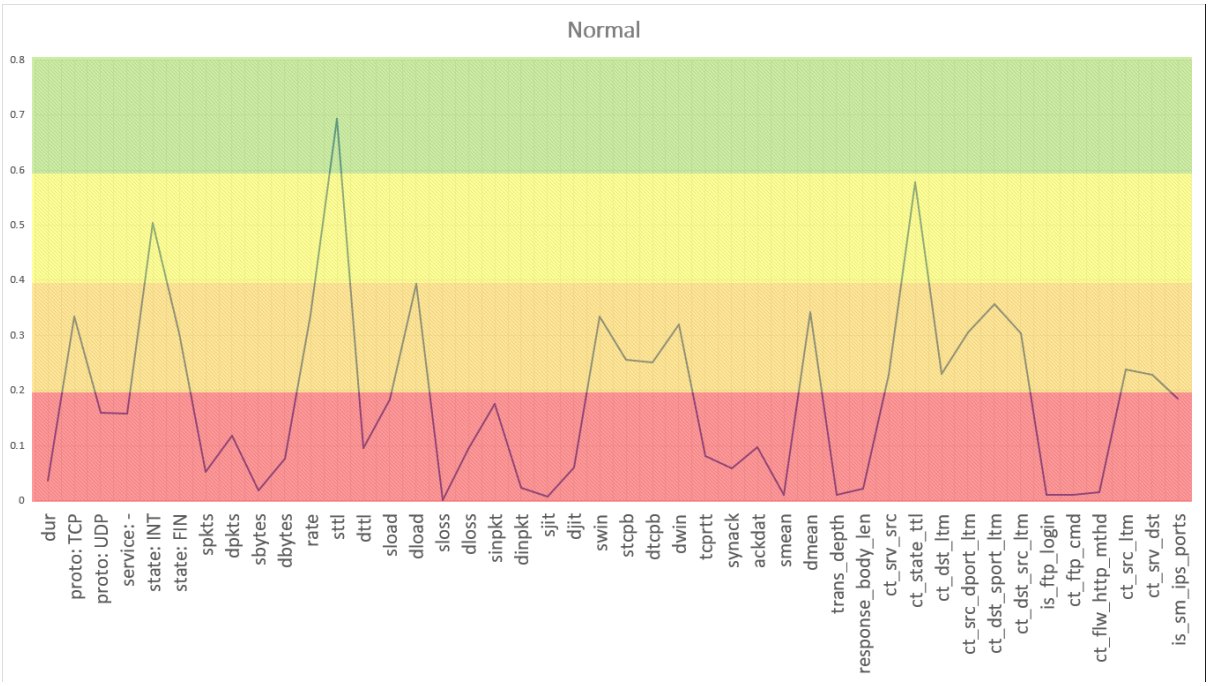

Figure 16: UNSW Magnitude of Pearson's Correlation for Normal Traffic

The normal/ benign data has one strong correlation from sttl, (the source to destination time to live). It has only 2 moderately strong attribute to label correlations: the state being INT or not, and ct_state_ttl (the number of each state according to values of sttl and dttl). These 3 are all clearly related.

This shows the importance of at least one of these attributes in identifying normal data from malicious data in this dataset. Unfortunately, this does not compare to any attributes in the KDD dataset that I can identify, and so I cannot check for similarities.

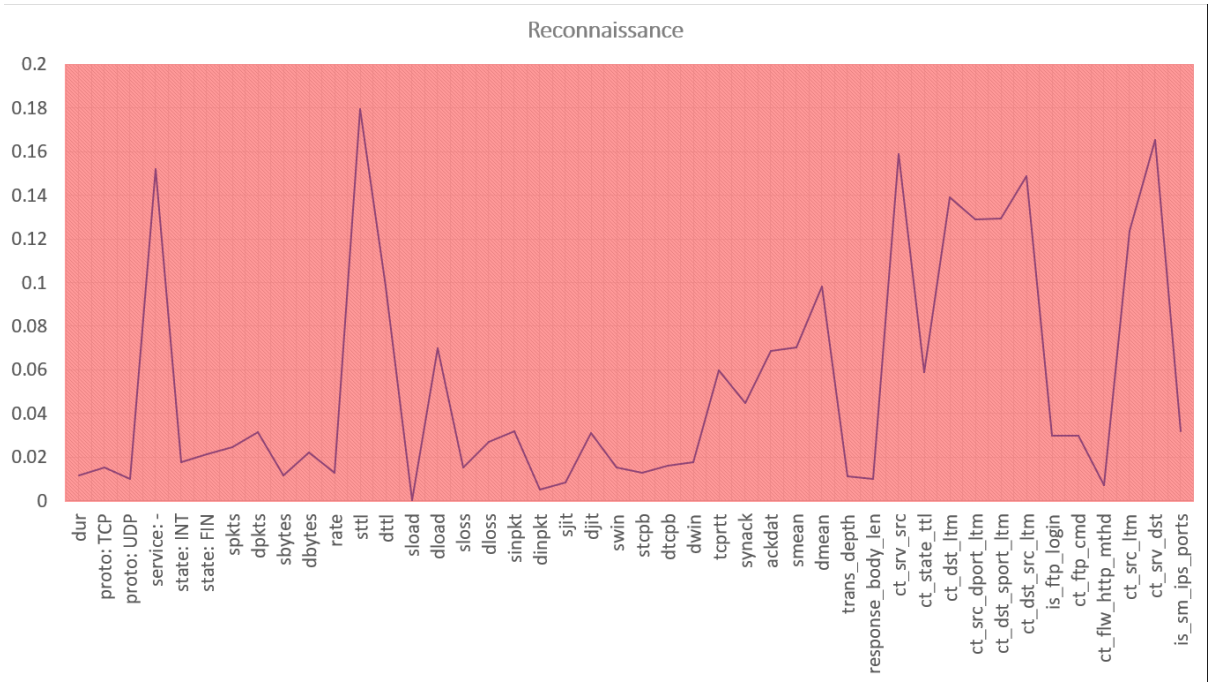

Figure 17: UNSW Magnitude of Pearson's Correlation for Reconnaissance Attack

Reconnaissance attacks in this context are passive reconnaissance (analysis attacks are not included; they are active reconnaissance). They collect preliminary information about any public network or target host and use exploit techniques to penetrate by leveraging the gathered information [94].

Unfortunately this is another attack with only very weak correlations, which means they probably do not signify anything, but I do note that once again one of the peaks comes from a nominal attribute – service – suggesting that maybe the inclusion of all the values of this might make a difference.

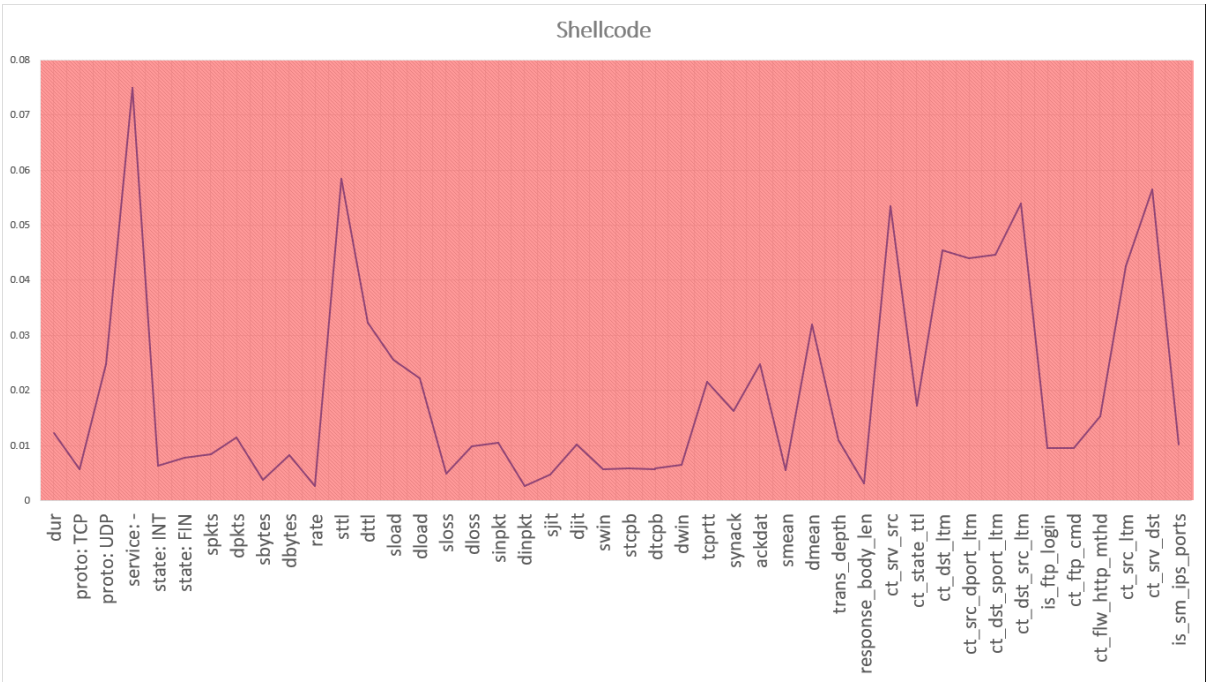

Figure 18: UNSW Magnitude of Pearson's Correlation for Shellcode Attack

Shellcode attacks are a subset of the exploit attacks and thus we would expect similar patterns between the two. It utilises a small piece of code as a payload of an attack. The malicious code is injected into an active application and compromises / gains access to a victim's computer. Typically, this starts a command shell to control the compromised machine [94].

In terms of correlation, the shellcode attack is similar to reconnaissance, with the data having a very weak correlations to the attack, but again the service type is a clear peak in this data. This may mean nothing, as it is a very weak correlation, but once again leads me to question whether other values of the nominal attribute might make a bigger difference. It varies from exploit attacks in correlation – despite being a subset – showing why it is classified separately from other exploit attacks as its correlations are much weaker.

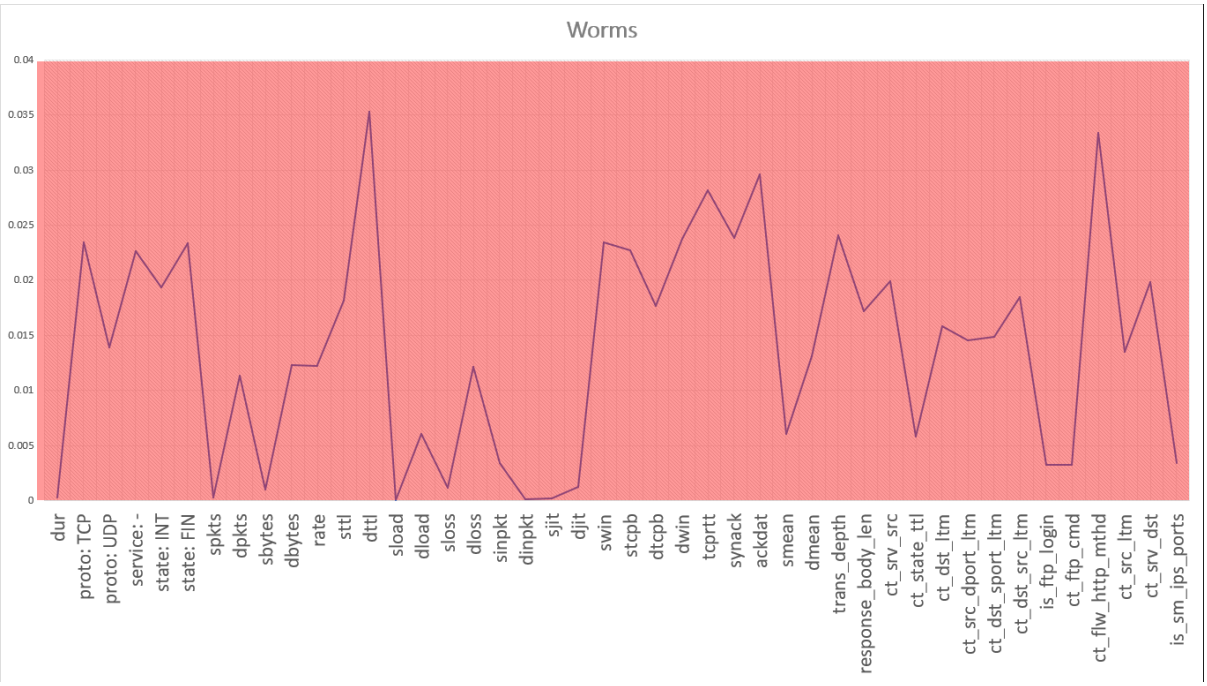

Figure 19: UNSW Magnitude of Pearson's Correlation for Worms Attack

A Worm is a malicious attack which spreads through network propagation and infects a larger network than other attacks, typically much quicker. It can also infect computers and turn them into zombies or bots to do the attackers bidding – normally to perform distributed attacks through the formation of botnets [94].

The worms attack has probably the weakest correlation of all the attack types, comparable only to analysis and backdoor attacks in how weak it is. As such, there is very little I can say about it. However, this does highlight that it does stand out against normal traffic, and so even if it is hard to classify itself, we can tell that it is malicious to detect it and stop it from propagating further.

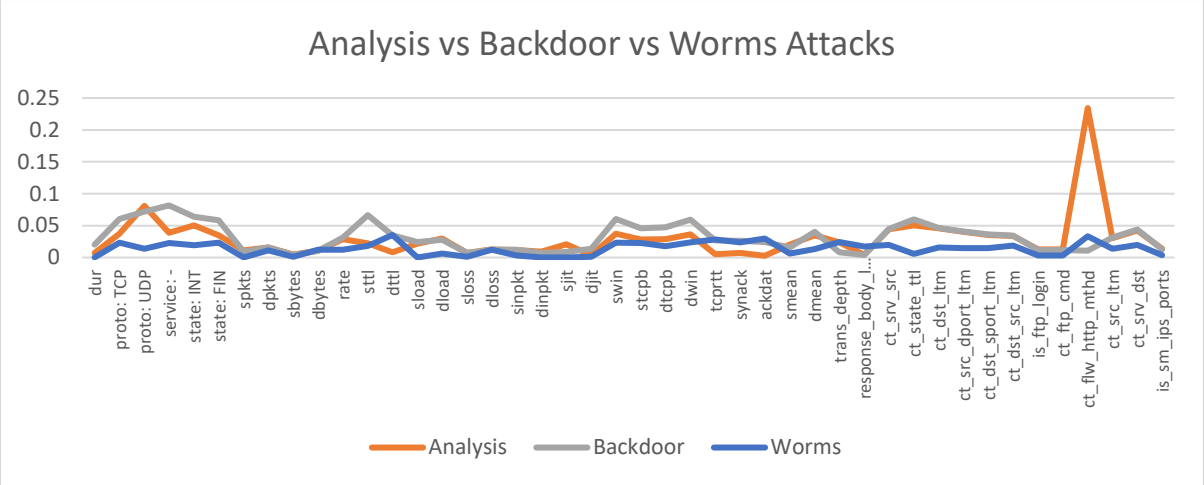

Figure 20: UNSW Magnitude of Pearson's Correlation for Analysis, Backdoor, Worms Attack

I decided to compare the three attacks types with overall weakest correlations to visualise just how insignificant the majority of their correlations are.

5.3 Results of Machine Learning

When I ran the 38 different classifier models, I took note of their precision, recall f-measure, and time to build in order to find the best classifier for my data.

5.3.1 KDD Dataset

With the KDD dataset this was easy enough, using the training set to train the model and the testing set to test it. These are the results I got:

	Classifier Name	Time to build model	Precision	Recall	F-measure	Time to test model
function	SGD	68.96	0.805	0.759	0.758	1.22
tree	Random Forest	133.28	0.805	0.759	0.758	2.58
bayes	Naïve Bayes	1.57	0.809	0.761	0.759	1.65
meta	AdaBoost M1	41.41	0.834	0.784	0.783	0.49
rules	PART	87.67	0.856	0.813	0.812	0.69
bayes	Bayes Net	9.52	0.822	0.744	0.739	1.32
bayes	Naïve Bayes Updatable	1.84	0.809	0.761	0.759	2.52
function	Logistic	114.84	0.804	0.756	0.754	0.46
function	SGD Text	12.58	?	0.431	?	0.28
function	Simple Logistic	140.37	0.798	0.746	0.743	0.44
function	SMO	1658.74	0.802	0.754	0.752	0.4
function	Voted Perceptron	25.45	0.286	0.412	0.335	69.39
lazy	Ibk	0.05	0.841	0.794	0.792	280.9
lazy	Kstar	0.02	0.837	0.777	0.774	18722.21

lazy	LWL	0.08	timeout	timeout	timeout	timeout
meta	Attribute Selected Classifier	15.19	0.826	0.762	0.758	0.3
meta	Classification via Regression	67.37	0.808	0.77	0.769	0.72
meta	Filtered Classifier	8.38	0.826	0.762	0.758	0.48
meta	Iterative Classifier Optimizer	177.03	0.797	0.747	0.745	1.13
meta	Logit Boost	20.17	0.797	0.747	0.745	0.48
meta	Multi-Class Classifier	84.73	0.804	0.756	0.754	0.43
meta	Multi-Scheme	0.19	?	0.431	?	0.25
meta	Random Committee	12.55	0.849	0.8	0.798	0.4
meta	Randomizable Filtered Classifier	0.54	0.775	0.72	0.716	91.84
meta	Random Sub Space	34.32	0.816	0.779	0.778	0.37
meta	Stacking	0.49	?	0.431	?	0.17
meta	Vote	0.02	?	0.431	?	0.14
meta	Weighted Instances Handler Wrapper	0.02	?	0.431	?	0.11
misc	Input Mapped Classifier	0.02	?	0.431	?	0.43
rules	Decision Table	119.32	0.814	0.726	0.718	0.27
rules	Jrip	176.82	0.836	0.774	0.771	0.24
rules	OneR	1	0.851	0.814	0.814	0.16
tree	Decision Stump	1.98	0.841	0.8	0.799	0.16
tree	Hoeffding Tree	3.55	0.812	0.772	0.771	0.37
tree	**J48**	**25.01**	**0.858**	**0.815**	**0.815**	**0.2**
tree	**LMT**	**698.69**	**0.86**	**0.823**	**0.823**	**0.49**
tree	Random Tree	1.7	0.837	0.814	0.814	0.63
tree	REP Tree	7.62	0.835	0.815	0.816	0.36

Figure 21: Classifier performance for KDD dataset

This has the values colour coded on a scale from green for the best values to red for the worst values.

It is clear that the best model for precision, recall and f-measure is the LMT tree, closely followed by the J48 tree. However, the LMT tree is significantly slower than the J48 tree, taking a lot longer to build and to test the model.

As their values for precision, recall and f-measure are so close, and both give good results, I would suggest that the best model for this data is the J48 model. This allows the model to be rolled out and potentially analyse traffic as it happens, to allow quick response by a cyber forensics team / first responder.

With this issue, speed is particularly important to be able to detect malicious traffic as early as possible before it causes too much trouble. However, its precision is also vital to ensure as few false positives as possible – as to not waste a cyber forensics team's time when they could be focused on other aspects – and to minimise false negatives as to not allow malicious traffic to break through the system.

I also find it interesting that the majority of the best results come from tree models – with some exceptions. This shows the way the attributes work together to best build a model.

5.3.2 UNSW Dataset

The UNSW dataset had some issues with compatibility between the provided training and testing set. It also was too large for Weka to handle on my laptop. As such I used just the testing dataset provided, as it was the larger of the two, and used Weka's split function to split the data 66% to 34%. These are the results I got:

	Classifier Name	Time to build model	Precision	Recall	F-measure	Time to test model
function	SGD	n/a	n/a	n/a	n/a	n/a
tree	**Random Forest**	**108.99**	**0.868**	**0.873**	**0.868**	**2.88**
bayes	Naïve Bayes	2.81	0.722	0.517	0.567	17.93
meta	AdaBoost M1	12.89	?	0.533	?	0.43
rules	PART	8455.4	0.816	0.82	0.811	3.05
bayes	Bayes Net	17.34	0.831	0.721	0.752	2.46
bayes	Naïve Bayes Updatable	4.41	0.722	0.517	0.567	11.66
function	Logistic	timeout	timeout	timeout	timeout	timeout
function	SGD Text	n/a	n/a	n/a	n/a	n/a
function	Simple Logistic	timeout	timeout	timeout	timeout	timeout
function	SMO	timeout	timeout	timeout	timeout	timeout
function	Voted Perceptron	timeout	timeout	timeout	timeout	timeout
lazy	Ibk	0.08	0.753	0.765	0.755	1097.98
lazy	Kstar	0.56	timeout	timeout	timeout	timeout

lazy	LWL	0.03	timeout	timeout	timeout	timeout
meta	Attribute Selected Classifier	41.78	0.822	0.822	0.799	0.2
meta	Classification via Regression	n/a	n/a	n/a	n/a	n/a
meta	Filtered Classifier	15.11	0.833	0.828	0.804	0.62
meta	Iterative Classifier Optimizer	timeout	timeout	timeout	timeout	timeout
meta	Logit Boost	248.54	?	0.785	?	0.39
meta	Multi-Class Classifier	2905.69	0.778	0.775	0.759	2.93
meta	Multi-Scheme	0.73	?	0.319	?	0.51
meta	Random Committee	33.56	0.817	0.823	0.812	2.16
meta	Randomizable Filtered Classifier	1.74	0.659	0.668	0.66	373.29
meta	Random Sub Space	NEM	NEM	NEM	NEM	NEM
meta	Stacking	1.54	?	0.319	?	0.9
meta	Vote	0.05	?	0.319	?	0.48
meta	Weighted Instances Handler Wrapper	0.02	?	0.319	?	0.28
misc	Input Mapped Classifier	0.02	?	0.319	?	0.49
rules	Decision Table	318.09	0.795	0.806	0.782	1.19
rules	Jrip	NEM	NEM	NEM	NEM	NEM
rules	OneR	3.68	?	0.767	?	0.22
tree	Decision Stump	4.7	?	0.533	?	0.27
tree	Hoeffding Tree	18.37	?	0.687	?	4.83
tree	J48	116.79	0.831	0.832	0.811	0.99
tree	LMT	NEM	NEM	NEM	NEM	NEM
tree	Random Tree	4.85	0.793	0.799	0.791	1
tree	REP Tree	26.97	0.816	0.824	0.806	0.5

Figure 22: Classifier performance for UNSW dataset

As you can see a lot of the models did not work on this dataset. NEM cells represent where the heap ran out of memory due to the limit on the amount I could allocate on my laptop, and many of the models outputted a '?' as the values, being unable to calculate them. As such the choices for models for this data set were more limited.

However, despite the loose correlation seen for this data set I did get some reasonable results. Most notably from the Random Forest Tree. It was not as quick as I would have preferred, but it computed in what was still a reasonable amount of time for its precision of 0.868, recall of 0.873 and f-measure of 0.868.

With more heap space I might have been able to find a better model, but this dataset had 10 different labels to classify to, and thus needed more memory to separate the records.

5.3.3 Comparison between Datasets

The J48 tree that prevailed for the KDD dataset was not as strong for the UNSW dataset, and the LMT tree would not even compute in the allocated memory space. Comparatively, the Random Forest Tree does not stand out initially in the KDD dataset. Therefore, to find the best model for the combination of the two datasets (even though the current attributes make them incompatible), I took the average for each measure between the two models for the models that had values for both datasets:

	Classifier Name	Time to Build model	Precision	Recall	F-measure	Time to Test Model
tree	Random Forest	121.135	0.8365	0.816	0.813	2.73
bayes	Naïve Bayes	2.19	0.7655	0.639	0.663	9.79
rules	PART	4271.535	0.836	0.8165	0.8115	1.87
bayes	Bayes Net	13.43	0.8265	0.7325	0.7455	1.89
bayes	Naïve Bayes Updatable	3.125	0.7655	0.639	0.663	7.09
lazy	Ibk	0.065	0.797	0.7795	0.7735	689.44
meta	Attribute Selected Classifier	28.485	0.824	0.792	0.7785	0.25
meta	Filtered Classifier	11.745	0.8295	0.795	0.781	0.55
meta	Multi-Class Classifier	1495.21	0.791	0.7655	0.7565	1.68
meta	Random Committee	23.055	0.833	0.8115	0.805	1.28
meta	Randomizable Filtered Classifier	1.14	0.717	0.694	0.688	232.565
tree	J48	70.9	0.8445	0.8235	0.813	0.595
tree	Random Tree	3.275	0.815	0.8065	0.8025	0.815
tree	REP Tree	17.295	0.8255	0.8195	0.811	0.43

Figure 23: Median Classifier Performance for both Datasets

This shows J48 and Random Forest, along with PART as the three best classifiers for the average of the datasets in terms of precision, recall and f-measure. But unfortunately, PART has a

significantly longer time to build the model, making it non-optimal for the implementation.

OF Random Forest and J48, J48 has the best times, and just about better scores for precision, recall and f-measure as well. Indicating it might be the best model for both (based on a median measure of average).

Looking instead at a mean average between the two datasets I get these results:

	Classifier Name	Time to Build model	Precision	Recall	F-measure	Time to Test Model
Tree	Random Forest	120.5246	0.835907	0.814007	0.811137	2.725876
Bayes	Naïve Bayes	2.100405	0.764263	0.627246	0.656013	5.439164
Rules	PART	860.979	0.835761	0.816492	0.8115	1.450689
Bayes	Bayes Net	12.84822	0.826488	0.73241	0.745472	1.801999
Bayes	Naïve Bayes Updatable	2.848579	0.764263	0.627246	0.656013	5.420627
Lazy	Ibk	0.063246	0.795785	0.779365	0.773279	555.3581
Meta	Attribute Selected Classifier	25.19203	0.823998	0.791432	0.77823	0.244949
Meta	Filtered Classifier	11.25264	0.829493	0.794315	0.780661	0.545527
Meta	MultiClassClassifier	496.1846	0.790893	0.765441	0.756496	1.122453
Meta	RandomCommittee	20.52262	0.832846	0.811419	0.80497	0.929516
Meta	RandomizableFilteredClassifer	0.96933	0.71465	0.693513	0.68743	185.1566
Tree	J48	54.04552	0.844392	0.823456	0.812998	0.444972
Tree	RandomTree	2.871411	0.814703	0.806465	0.802418	0.793725
Tree	REP Tree	14.33567	0.825445	0.819488	0.810985	0.424264

Figure 24: Mean Classifier Performance for both Datasets

This supports the findings from the median average, thus indicating that J48 would be the best choice in future work for combining the two datasets, if there is a way to find more compatible attributes.

Another interesting thing to note is while the KDD dataset has good precision, recall and f-measure values for a large proportion of models, which is a sharp contrast from the UNSW mode, its best results are reasonably lower than that of the UNSW model. This is particularly interesting seeing as the Pearson Correlation indicated that it might be difficult to classify the data for the UNSW dataset as so many of the coefficients were so low. Therefore, there was definitely more

to the data than comparing one attribute at a time to the individual label indicated.

6. Conclusion and Future Work

6.1 Conclusion

Overall, this project was a lot more complicated than I initially considered due to a number of reasons, a main one being the vast volume of data that machine learning requires. Many times, during the project my laptop struggled with the requirements it needed, sometimes even smelling of burning. Despite this I was able to run the majority of the classifiers.

This study indicates that J48 trees are the best at classifying network traffic overall – particularly for whether it is benign or malicious. It also demonstrates that the source to destination time to live is an important attribute, that should be present if you wish to classify this data. Along with: logged_in, serror_rate, srv_rerror_rate, same_srv_rate, dst_host_srv_count, dst_host_same_srv_rate, dst_host_serror_rate and dst_host_srv_serror_rate. Although the relation between some of these attributes must be considered also.

The penetration testing part of the project went the smoothest. It was very interesting bypassing different safety measures a site had in place, where they had not got complete security but had clearly considered it. I should have learnt how to make a script to perform the attacks for me; instead, I manually attacked each web app myself. Whilst this meant I did not have enough data for building my machine learning model it did allow more in depth learning of each attack and for me to see different ways a web app may respond to allow malicious activity to take place.

My machine learning model works reasonably well but the different attributes between datasets unfortunately led to compatibility issues. This signifies the issues of using external datasets rather than collating my own. It also means I do not have the know how to collate these attributes from recorded traffic. It would clearly need another script due to the vast volume of data. However, using these datasets meant I had more than enough data to build an accurate model – despite the UNSW correlation measures.

Despite the challenges I faced, I was still able to achieve the majority of my objectives in some capacity, the only exception was building my own web app as I deemed this of less significance.

6.2 Future Work

To improve my classifier, I would build a script to perform my own attacks and translate the network data into a format for the classifier. This would allow me to run it on live network data, having a script to feed the data into the classifier, and therefore,

identifying ongoing attacks, and being able to deploy first responders to stop them.

Furthermore, to allow my classifier to work for unknown attacks or be future proof, it would have to be unsupervised (thus work through clustering, not classifying). This would mean that the attacks would not be labelled but would spot new activity that did not fit in any of the pre-existing classes. The issue with this would be that a security team would then have to identify the attack type themselves, losing valuable time in stopping the ongoing attack.

Therefore, in future work I would like to make this classifier semi supervised. This would mean that in training, it would take both labelled and unlabelled data [95]. Hopefully, from this the machine would label test data where it can, but where this is not possible, sort it into an appropriate cluster. This would mean that known attacks would be identified for response, and new, unidentified attacks would still be flagged rather than falling through the gaps.

Another aspect I would like to do with implementing my data is to first classify it into malicious or benign and then take the malicious data and classify it into attack types. This would act similarly to the semi-supervised approach. The classifier was in general quicker for classifying between the two labels – normal or anomalous – than the ten attack labels. Therefore, this would guarantee a more efficient classification to identify malicious traffic and reduce the amount of data for the slower attack type classifier to sort through.

This layered approach would not only be more efficient but also should be easier to display the results in a graphical way. This is because normal data is filtered out allowing a clean breakdown of actual attacks. Seeing this breakdown would allow security teams to see where they might have a vulnerability.

Finally, in future iterations of this project I would like to implement the attacks names to be the attacks like XSS that I have studied, rather than the nine labels provided in the UNSW dataset. This is particularly useful for exploit attacks, where known vulnerabilities are used, as more detailed attack labels will allow for easier identification of the vulnerabilities.

7. Reflection on Learning

7.1 What I have learnt

During the progression of this project I have learnt lots of abstract skills such as how to do a large project, how to deal with adverse circumstances (as described below) and developed my time management, to keep on schedule even when things do not go to plan, or plans change.

Through this, I have learnt how to look at the bigger picture despite focusing on individual tasks at a time, to ensure that despite my perfectionist tendencies, I did not take too long on any one aspect of the project.

I think I initially spent too long on my penetration testing, trying to explore as much as possible, but only looked at the manual aspects of this. I then took too long to ask for additional guidance when I came to the machine learning part of the project, as I was not aware just how much data I needed until many weeks into working on this – making time management extremely difficult. Therefore, I have learnt to ask for help as soon as possible when it is needed, and to ensure I understand what I need to achieve before I begin undertaking a project.

On a more technical note, I have learnt about penetration testing, machine learning and web applications. As I am particularly interested in penetration testing, I found this a very interesting subject to study.

I am now very adept at performing XSS attacks in particular, along with various other attacks on web apps. This has taught me both about the attacks themselves, and vulnerabilities – allowing me to now (with some ease) identify vulnerabilities that can be exploited by an attacker. This skill will help with many aspects of my future career, whether it be in forensics – identifying attacks after the fact and stopping them in progress – or security – preventing them from happening in the first place.

Furthermore, I have a much more thorough understanding of machine learning. I was a complete novice in this area at the beginning of this project – having only studied the theory of artificial intelligence previously. Now I have run 38 different models of machine classifiers – and I understand each of them to at least a basic level.

Another topic which was rather new to me was statistical analysis – particularly the use of Pearson’s Correlation

Coefficient. I can now apply this to a large volume of data as seen in my results.

There is still lots of room for me to learn more, but this project has clearly been very educational in many aspects.

7.2 Challenges

Throughout the process I faced multiple challenges.

One of the biggest challenges was the fact that I could not collect enough data from my own manual penetration testing, nor could I work out how to get it in the right form to best use its data. I had not realised that it would have been better to use a script rather than manually performing each attack individually. I now know this would have been a better approach, but this realisation came too late in the process. I overcame this by using the penetration testing to learn about the attacks and how to perform them and then using online datasets to actually build my classifier.

Another challenge was the outbreak of Covid-19. This meant I had to run everything from my laptop as other resources, like the university's labs and libraries were closed. This was particularly hard in building the machine classifier as my laptop has limited processing power, and it also struggled with heap size. Despite this, by leaving my laptop running overnight on a few occasions, I was able to test most classifier models. The heap memory was still however an issue in some cases (as seen in the results table labelled as 'NEM'). I did increase my heap size as much as I could in environment variables, but it was more than my laptop could handle.

A related issue I found was that the UNSW dataset was too large for my laptop to deal with in Weka, and thus instead of using the training and testing set as appropriate, I used the testing set, and used Weka to split it 66% to 34%. This could have affected the accuracy, but there was enough data, and the split meant that the data used for training was not used for testing.

Another challenge was the large size of the datasets. This meant that when performing analysis on the data on Excel it crashed a few times. This was particularly the case for using graphs, and so I had to copy the data I needed to create a graph onto a separate spreadsheet to all Excel to process the volume of data. On one occasion Excel deleted a day's work –

despite having saved it throughout – due to the excessive amount of data, and so this was an import problem to overcome.

7.3 Supervisor Meetings

I had intended to have seven meetings with my supervisor in the development stage of the project, as specified in my initial report. As things transpired, I only had 6 meetings. This was also over a longer period of time than intended as the development of the machine learning took longer than allowed for in my initial Gantt chart (figure 1).

One of the issues I did experience with my supervisor meetings was that only two of them, during the development of the project, were face to face, with the rest conducted over Skype. Whilst this still allowed for communication, it was not quite to the same standard as face to face, and some of the issues with the project were not picked up on until later than intended. This could not be avoided due to my supervisor travelling at the start of term time, and then the outbreak of Covid-19 and the national lockdown.

Glossary

Heap	A Heap is a specialised tree-based data structure – composed of a complete binary tree. A heap's memory is allocated dynamically at runtime, and typically holds program data.
Parsing	Parsing is seen as syntactic analysis of input code, separating it into its component parts.
Credential stuffing	Credential stuffing is a cyber-attack that uses automated injection of breached username/password pairs (often from a data breach) in order to fraudulently gain access to user accounts.
Man-in-the-middle attacks	A Man-in-the-middle attack is where an attacker intercepts communication between two parties and relies the data, possibly altering it.
Kernel approximation	A kernel is the central part of an operating system, a kernel approximation uses functions to approximate the feature mappings that correspond to certain kernels.
Semantic web	The semantic web is an extension of the World Wide Web in which data in web pages is structures and tagged in a way that can be read directly by computers.

Table of Abbreviations

IDS	Intrusion Detection System
XSS	Cross-Site Scripting
SQL	Search Query Language
DOS	Denial of Service
WAFs	Web Application Firewalls
SDT	Secure Development Training
Pen Testing	Penetration Testing
Web App	Web Application
ZAP	Zed Attack Proxy
HUD	Heads-up Display
IoT	Internet of Things
IPS	Intrusion Prevention System
CART	Classification and Regression Trees
SGD	Stochastic Gradient Descent
LWL	Locally Weighted Learning
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
ML-kNN	Multi-label k-Nearest Neighbour
RAkEL	Random *k*-labelsets Ensemble

Appendices

Appendix 1: Python Code

```
#Dataset ref: CSIC 2010 HTTP Dataset in CSV Format (for Weka Analysis)
#             CSIC 2010 HTTP Dataset in CSV Format (for Weka Analysis) (2018). Available
at: https://petescully.co.uk/research/csic-2010-http-dataset-in-csv-format-for-Weka-
analysis/ (Accessed: 5 May 2020).

#imports needed
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Path of the file to read
data_file_path = r'C:\Users\Alice\OneDrive\Documents\Year 3\Final Year
Project\Datasets\HTTP CSIC Torpeda 2012 dataset\combined.csv'
testData = pd.read_csv(data_file_path)

# Create target object and call it y
y = testData.type

# Create X
features = ['method', 'protocol', 'path', 'headers', 'query', 'body']
train_x = pd.get_dummies(testData[features])

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
from sklearn.ensemble import RandomForestClassifier
clf_rf = RandomForestClassifier(n_estimators=50)

#fit model
clf_rf.fit(train_x, train_y)
RandomForestClassifier()

# Make validation predictions
val_predictions = xlf_rf.predict(val_X)

#check accuracy of predictions
correct = 0
j = 0
for i in val_predictions:
       if i == val_y.get(j):
              correct +=1
              print(j)
       j += 1
print(str(correct) + '/' + str(j))
```

Appendix 2: Wireshark Packets

See Wireshark packets folder

Appendix 3: KDD Model

KDD model and result buffer stored separately, see files

Appendix 4: UNSW Model

UNSW model and result buffer stored separately, see files

Appendix 5: KDD Dataset

Can be found in Dataset folder.

Raw dataset available at: **https://www.unb.ca/cic/datasets/nsl.html**

Appendix 6: UNSW Dataset
Can be found in dataset folder.

Raw dataset available at: **https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/**

References

[1]	Your Web Applications Are More Vulnerable Than You Think ***Your Web Applications Are More Vulnerable Than You Think*** **(2018). Available at: https://securityintelligence.com/your-web-applications-are-more-vulnerable-than-you-think/ (Accessed: 18 April 2020).**
[2]	5 Most Common Web Application Attacks (And 3 Security Recommendations) – MSSP Alert ***5 Most Common Web Application Attacks (And 3 Security Recommendations) – MSSP Alert*** **(2018). Available at: https://www.msspalert.com/cybersecurity-breaches-and-attacks/5-most-common-web-application-attacks/ (Accessed: 18 April 2020).**
[3]	What is Penetration Testing | Step-By-Step Process & Methods | Imperva ***What is Penetration Testing | Step-By-Step Process & Methods | Imperva*** **(2020). Available at: https://www.imperva.com/learn/application-security/penetration-testing/ (Accessed: 18 April 2020).**
[4]	What is machine learning? ***What is machine learning?*** **(2020). Available at: https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/ (Accessed: 19 April 2020).**
[5]	Machine Learning for Cybersecurity 101 ***Machine Learning for Cybersecurity 101*** **(2019). Available at: https://towardsdatascience.com/machine-learning-for-cybersecurity-101-7822b802790b (Accessed: 19 April 2020).**
[6]	Perlman, A. and Perkowski, M. **Perlman, A. and Perkowski, M. (2019)** ***The Growing Role of Machine Learning in Cybersecurity – SecurityRoundTable.org, SecurityRoundTable.org*****. Available at: https://www.securityroundtable.org/the-growing-role-of-machine-learning-in-cybersecurity/ (Accessed: 19 April 2020).**
[7]	SQL-IDS | Proceedings of the 2008 ACM symposium on Applied computing ***SQL-IDS | Proceedings of the 2008 ACM symposium on Applied computing*** **(2020). Available at: https://dl.acm.org/doi/abs/10.1145/1363686.1364201 (Accessed: 25 April 2020).**
[8]	T. T. Dang and T. K. Dang **T. T. Dang and T. K. Dang (2015) "Extending Web Application IDS Interface: Visualizing Intrusions in Geographic and Web Space,"** ***2015 International Conference on Advanced Computing and Applications (ACOMP),*** **Ho Chi Minh City, 2015, pp. 28-34.**
[9]	Agarwal, N. and Hussain, S. Z. **Agarwal, N. and Hussain, S. (2018) "A Closer Look at Intrusion Detection System for Web Applications",** ***Security and Communication Networks*****, 2018, pp. 1-27. Doi: 10.1155/2018/9601357.**
[10]	US7308716B2 – Applying blocking measures progressively to malicious network traffic – Google Patents ***US7308716B2 – Applying blocking measures progressively to malicious network traffic – Google Patents*** **(2003). Available at: https://patents.google.com/patent/US7308716B2/en (Accessed: 26 April 2020).**
[11]	Hyunsang Choi Korea, Bin B. Zhu, Heejo Lee

	(2020) *Gauss.ececs.uc.edu*. Hyunsang Choi Korea, Heejo Lee Korea University, Bin B. Zhu Microsoft Research Asia, Available at: http://gauss.ececs.uc.edu/Courses/c5155/pdf/webapps.pdf (Accessed: 26 April 2020).
[12]	materials, M., methods, A. and application, C. **materials, m., methods, a. and application, c. (2006) *US20070186282A1 – Techniques for identifying and managing potentially harmful web traffic – Google Patents*, *Patents.google.com*. Available at: https://patents.google.com/patent/US20070186282A1/en (Accessed: 26 April 2020).**
[13]	Clark, T. **Clark, T. (2020) *SAP BrandVoice: Most Cyber Attacks Occur From This Common Vulnerability*, *Forbes*. Available at: https://www.forbes.com/sites/sap/2015/03/10/most-cyber-attacks-occur-from-this-common-vulnerability/#57dfaeeb7454 (Accessed: 12 May 2020).**
[14]	OWASP WebGoat ***OWASP WebGoat* (2020). Available at: https://owasp.org/www-project-webgoat/ (Accessed: 20 April 2020).**
[15]	OWASP Security Shepherd ***OWASP Security Shepherd* (2020). Available at: https://owasp.org/www-project-security-shepherd/ (Accessed: 20 April 2020).**
[16]	Lim Jet Wee ***Lim Jet Wee* (2020). Available at: https://www.youtube.com/user/limjetwee (Accessed: 25 April 2020).**
[17]	OWASP Top Ten ***OWASP Top Ten* (2020). Available at: https://owasp.org/www-project-top-ten/ (Accessed: 22 April 2020).**
[18]	Porup, J. **Porup, J. (2020) *Learn to play defense by hacking these broken web apps*, *CSO Online*. Available at: https://www.csoonline.com/article/3319521/learn-to-play-defense-by-hacking-these-broken-web-apps.html (Accessed: 20 April 2020).**
[19]	Oracle VM VirtualBox ***Oracle VM VirtualBox* (2020). Available at: https://www.virtualbox.org/ (Accessed: 20 April 2020).**
[21]	OWASP Broken Web Applications Project ***OWASP Broken Web Applications Project* (2020). Available at: https://sourceforge.net/projects/owaspbwa/ (Accessed: 20 April 2020)**
[22]	OWASP ZAP ***OWASP ZAP* (2020). Available at: https://owasp.org/www-project-zap/ (Accessed: 20 April 2020).**
[23]	Inc., S. **Inc., S. (2020) *ZAP in Ten, Alldaydevops.com*. Available at: https://www.alldaydevops.com/zap-in-ten (Accessed: 20 April 2020).**
[24]	Wireshark · Download ***Wireshark · Download* (2020). Available at: https://www.wireshark.org/download.html (Accessed: 20 April 2020).**
[25]	Learn Intro to Machine Learning Tutorials ***Learn Intro to Machine Learning Tutorials* (2020). Available at: https://www.kaggle.com/learn/intro-to-machine-learning (Accessed: 20 April 2020).**
[26]	scikit-learn: machine learning in Python — scikit-learn 0.22.2 documentation ***scikit-learn: machine learning in Python — scikit-learn 0.22.2 documentation* (2020). Available at: https://scikit-learn.org/stable/ (Accessed: 20 April 2020).**
[27]	pandas – Python Data Analysis Library

	***pandas – Python Data Analysis Library* (2020). Available at: https://pandas.pydata.org/ (Accessed: 20 April 2020).**
[28]	Weka ***Weka* (2020). Available at: https://sourceforge.net/projects/Weka/ (Accessed: 20 April 2020).**
[29]	NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB ***NSL-KDD | Datasets | Research | Canadian Institute for Cybersecurity | UNB* (2020). Available at: https://www.unb.ca/cic/datasets/nsl.html (Accessed: 20 April 2020).**
[30]	The UNSW-NB15 data set description ***The UNSW-NB15 data set description* (2018). Available at: https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/ (Accessed: 20 April 2020)**
[31]	What is Cross-site Scripting and How Can You Fix it? ***What is Cross-site Scripting and How Can You Fix it?* (2020). Available at: https://www.acunetix.com/websitesecurity/cross-site-scripting/ (Accessed: 23 April 2020).**
[32]	SQL injection ***SQL injection* (2020). Available at: https://en.wikipedia.org/wiki/SQL_injection (Accessed: 23 April 2020)**
[33]	DOS ***DOS* (2020). Available at: https://en.wikipedia.org/wiki/DOS (Accessed: 23 April 2020).**
[34]	A2:2017-Broken Authentication | OWASP ***A2:2017-Broken Authentication | OWASP* (2020). Available at: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication (Accessed: 23 April 2020).**
[35]	A4:2017-XML External Entities (XXE) | OWASP ***A4:2017-XML External Entities (XXE) | OWASP* (2020). Available at: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A4-XML_External_Entities_(XXE) (Accessed: 23 April 2020).**
[36]	A3:2017-Sensitive Data Exposure | OWASP ***A3:2017-Sensitive Data Exposure | OWASP* (2020). Available at: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure (Accessed: 23 April 2020).**
[37]	A5:2017-Broken Access Control | OWASP ***A5:2017-Broken Access Control | OWASP* (2020). Available at: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control (Accessed: 23 April 2020).**
[38]	A8:2017-Insecure Deserialization | OWASP ***A8:2017-Insecure Deserialization | OWASP* (2020). Available at: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A8-Insecure_Deserialization (Accessed: 23 April 2020).**
[39]	A9:2017-Using Components with Known Vulnerabilities | OWASP ***A9:2017-Using Components with Known Vulnerabilities | OWASP* (2020). Available at: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities (Accessed: 23 April 2020).**
[40]	Academy, W. **Academy, W. (2020) *What is CSRF (Cross-site request forgery)? Tutorial & Examples | Web Security Academy, Portswigger.net*. Available at: https://portswigger.net/web-security/csrf (Accessed: 23 April 2020).**
[41]	The Top 10 Machine Learning Algorithms for ML Beginners **The Top 10 Machine Learning Algorithms for ML Beginners (2019). Available at: https://www.dataquest.io/blog/top-10-machine-learning-algorithms-for-beginners/ (Accessed: 27 April 2020).**
[42]	How to choose machine learning algorithms?

	How to choose machine learning algorithms? (2018). Available at: https://medium.com/@aravanshad/how-to-choose-machine-learning-algorithms-9a92a448e0df (Accessed: 27 April 2020).
[43]	Brownlee, J. **Brownlee, J. (2016) How To Use Classification Machine Learning Algorithms in Weka, Machine Learning Mastery. Available at: https://machinelearningmastery.com/use-classification-machine-learning-algorithms-Weka/?trackid=sp-006 (Accessed: 27 April 2020).**
[44]	Types of Machine Learning Algorithms You Should Know **Types of Machine Learning Algorithms You Should Know (2017). Available at: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861 (Accessed: 27 April 2020).**
[45]	Artificial Intelligence – foundations of computational agents – 7.3.3 Bayesian Classifiers **Artificial Intelligence – foundations of computational agents – 7.3.3 Bayesian Classifiers (2020). Available at: https://artint.info/html/ArtInt_181.html (Accessed: 27 April 2020).**
[46]	1.9. Naïve Bayes — scikit-learn 0.22.2 documentation **1.9. Naïve Bayes — scikit-learn 0.22.2 documentation (2020). Available at: https://scikit-learn.org/stable/modules/naive_bayes.html (Accessed: 28 April 2020).**
[47]	What is Bayesian Network Classifier? **What is Bayesian Network Classifier? (2018). Available at: https://medium.com/@analyttica/what-is-bayesian-network-classifier-4d2771f91f63 (Accessed: 28 April 2020).**
[48]	Machine Learning Functions **Machine Learning Functions (2020). Available at: https://www.vertica.com/docs/9.2.x/HTML/Content/Authoring/SQLReferenceManual/Functions/MachineLearning/_MLFunctions.htm (Accessed: 27 April 2020).**
[49]	sklearn.linear_model.SGDClassifier — scikit-learn 0.22.2 documentation **sklearn.linear_model.SGDClassifier — scikit-learn 0.22.2 documentation (2020). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html (Accessed: 28 April 2020).**
[50]	LOGISTIC REGRESSION CLASSIFIER **LOGISTIC REGRESSION CLASSIFIER (2019). Available at: https://towardsdatascience.com/logistic-regression-classifier-8583e0c3cf9 (Accessed: 28 April 2020).**
[51]	SGDText **SGDText (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/functions/SGDText.html (Accessed: 28 April 2020).**
[52]	SimpleLogistic **SimpleLogistic (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/functions/SimpleLogistic.html (Accessed: 28 April 2020).**
[53]	SMO **SMO (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/functions/SMO.html (Accessed: 28 April 2020).**
[54]	VotedPerceptron **VotedPerceptron (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/functions/VotedPerceptron.html (Accessed: 28 April 2020).**
[55]	Voted Perceptron – Cohen Courses **Voted Perceptron – Cohen Courses (2020). Available at: http://curtis.ml.cmu.edu/w/courses/index.php/Voted_Perceptron (Accessed: 28 April 2020).**

[56]	Lazy learning **Lazy learning (2020). Available at: https://en.wikipedia.org/wiki/Lazy_learning (Accessed: 27 April 2020)**
[57]	Aha, D.W, Kibler, D. and Albert, M.K **Aha, D.W, Kibler, D. and Albert, M.K, 1991. *Instance-Based Learning Algorithms*. Machine Learning. Boston: Kluwer Academic Publishers.**
[58]	IBk **Ibk (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/lazy/IBk.html (Accessed: 28 April 2020).**
[59]	Kstar **Kstar (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/lazy/KStar.html (Accessed: 28 April 2020).**
[60]	LWL **LWL (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/lazy/LWL.html (Accessed: 28 April 2020).**
[61]	Meta learning (computer science) **Meta learning (computer science) (2010). Available at: https://en.wikipedia.org/wiki/Meta_learning_(computer_science) (Accessed: 27 April 2020).**
[62]	AdaBoostM1 **AdaBoostM1 (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/AdaBoostM1.html (Accessed: 28 April 2020).**
[63]	AttributeSelectedClassifier **AttributeSelectedClassifier (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/AttributeSelectedClassifier.html (Accessed: 28 April 2020).**
[64]	ClassificationViaRegression **ClassificationViaRegression (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/ClassificationViaRegression.html (Accessed: 28 April 2020).**
[65]	FilteredClassifier **FilteredClassifier (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/FilteredClassifier.html (Accessed: 28 April 2020).**
[66]	IterativeClassifierOptimizer **IterativeClassifierOptimizer (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/IterativeClassifierOptimizer.html (Accessed: 28 April 2020).**
[67]	LogitBoost **LogitBoost (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/LogitBoost.html (Accessed: 28 April 2020).**
[68]	MultiClassClassifier **MultiClassClassifier (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/MultiClassClassifier.html (Accessed: 28 April 2020).**
[69]	MultiScheme **MultiScheme (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/MultiScheme.html (Accessed: 29 April 2020).**
[70]	RandomCommittee **RandomCommittee (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/RandomCommittee.html (Accessed: 29 April 2020).**

[71]	RandomizableFilteredClassifier **RandomizableFilteredClassifier (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/RandomizableFilteredClassifier.html (Accessed: 29 April 2020).**
[72]	RandomSubSpace **RandomSubSpace (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/RandomSubSpace.html (Accessed: 29 April 2020).**
[73]	Stacking **Stacking (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/Stacking.html (Accessed: 29 April 2020).**
[74]	Ensemble Learning to Improve Machine Learning Results **Ensemble Learning to Improve Machine Learning Results (2019). Available at: https://blog.statsbot.co/ensemble-learning-d1dcd548e936 (Accessed: 29 April 2020).**
[75]	Voting Classifier **Voting Classifier (2019). Available at: https://medium.com/@sanchitamangale12/voting-classifier-1be10db6d7a5 (Accessed: 29 April 2020).**
[76]	WeightedInstancesHandlerWrapper **WeightedInstancesHandlerWrapper (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/meta/WeightedInstancesHandlerWrapper.html (Accessed: 29 April 2020).**
[77]	InputMappedClassifier **InputMappedClassifier (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/misc/InputMappedClassifier.html (Accessed: 29 April 2020).**
[78]	Rule-based machine learning **Rule-based machine learning (2020). Available at: https://en.wikipedia.org/wiki/Rule-based_machine_learning (Accessed: 27 April 2020).**
[79]	Generating good decision rules – More Data Mining with Weka **Generating good decision rules – More Data Mining with Weka (2020). Available at: https://www.futurelearn.com/courses/more-data-mining-with-Weka/0/steps/29123 (Accessed: 29 April 2020).**
[80]	DecisionTable **DecisionTable (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/rules/DecisionTable.html (Accessed: 29 April 2020).**
[81]	Jrip **Jrip (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/rules/JRip.html (Accessed: 29 April 2020).**
[82]	OneR **OneR (2020). Available at: https://www.saedsayad.com/oner.htm (Accessed: 29 April 2020).**
[83]	OneR **OneR (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/rules/OneR.html (Accessed: 29 April 2020).**
[84]	RandomForest **RandomForest (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/trees/RandomForest.html (Accessed: 29 April 2020).**
[85]	3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.2 documentation

	3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.2 documentation (2020). Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (Accessed: 29 April 2020).
[86]	Webb, G. I., Fürnkranz, J., Fürnkranz, J., Fürnkranz, J., Hinton, G., Sammut, C., Sander, J., Vlachos, M., The, Y. W., Yang, Y., Mladeni, D., Brank, J., Grobelnik, M., Zhao, Y., Karypis, G., Craw, S., Puterman, M. L. and Patrick, J. **Webb, G. et al. (2011) "Decision Stump", Encyclopedia of Machine Learning, pp. 262-263. Doi: 10.1007/978-0-387-30164-8_202.**
[87]	HoeffdingTree **HoeffdingTree (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/trees/HoeffdingTree.html (Accessed: 29 April 2020).**
[88]	Machine Learning – (C4.5|J48) algorithm [Gerardnico – The Data Blog] **Machine Learning – (C4.5|J48) algorithm [Gerardnico – The Data Blog] (2020). Available at: https://gerardnico.com/data_mining/c4.5 (Accessed: 29 April 2020).**
[89]	Saravanan N, Gayathri V **Saravanan N, et al. (2018) "Performance and Classification Evaluation of J48 Algorithm and Kendall's Based J48 Algorithm (KNJ48)", International Journal of Computational Intelligence and Informatics, Vol. 7: No. 4. Pp 1. Available at: https://www.periyaruniversity.ac.in/ijcii/issue/marnew/2_mar_18.pdf (Accessed: 29 April 2020).**
[90]	LMT **LMT (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/trees/LMT.html (Accessed: 29 April 2020).**
[91]	RandomTree **RandomTree (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/trees/RandomTree.html (Accessed: 29 April 2020).**
[92]	REPTree **REPTree (2020). Available at: https://Weka.sourceforge.io/doc.dev/Weka/classifiers/trees/REPTree.html (Accessed: 29 April 2020).**
[93]	How to Interpret a Correlation Coefficient r – dummies **How to Interpret a Correlation Coefficient r – Deborah J. Rumsey (2020). Available at: https://www.dummies.com/education/math/statistics/how-to-interpret-a-correlation-coefficient-r/ (Accessed: 3 May 2020).**
[94]	Bagui, S., Kalaimannan, E., Bagui, S., Nandi, D. and Pinto, A. **Bagui, S. et al. (2019) "Using machine learning techniques to identify rare cyber-attacks on the UNSW-NB15 dataset", Security and Privacy, 2(6). doi: 10.1002/spy2.91.**
[95]	Semi-supervised learning ***Semi-supervised learning* (2020). Available at: https://en.wikipedia.org/wiki/Semi-supervised_learning (Accessed: 20 April 2020).**