Machine Learning and Cybersecurity: Studying network behaviour to detect anomalies

Jiawen Chen

July 25, 2018

MSc in High Performance Computing with Data Science

The University of Edinburgh

Year of Presentation: 2018

Abstract

Cybersecurity is an indispensable part of the Internet nowadays as almost everything can be connected by Internet. Individual privacy and property will face great danger if they are attacked by malware. This project aims to use the machine learning knowledge to build a classifier that can detect malware behaviour and avoid further damage to the network and its entity. The dataset used in this project is the CTU-13 dataset[1], which is NetFlow traffic from the normal, malware infected and background hosts. The classifier is built with the labelled normal and infected NetFlow, then applied to the unlabelled background NetFlow. This project follows the O'Neil & Schutt data science process[7] to process the data. The data is cleaned and processed, then aggregate based on time window to create a new dataset with the extracted features. This new dataset is the inputs of the following machine learning classification model: logistic regression, decision tree and random forest. Random forest model with an optimal threshold turns out to be the best model with an accuracy of 0.946 on the test dataset. Further application with this model is also implemented to the background traffic to detect potential malware.

Keywords: Cybersecurity, NetFlow, Data science, Machine learning, Model building

Contents

Chapter 1 Introduction 1
1.1 Background 1
1.2 Project goals 2
1.3 Research methods 3
1.4 Report structure 3
Chapter 2 Contextual Review 5
2.1 The CTU-13 dataset 5
2.1.1 Dataset introduction 5
2.1.2 Dataset format 6
2.2 Internet and malware 7
2.2.1 SPAM behaviour 8
2.2.2 DDoS behaviour 8
2.2.3 IRC behaviour 9
2.2.4 Other malicious behaviours 10
Chapter 3 Machine Learning Methodology 11
3.1 Machine learning introduction 11
3.2 Machine learning process 12
3.3 Test & training dataset 13
3.4 Model Fitting 13
3.4.1 Logistic regression 14
3.4.2 Decision tree 15
3.4.3 Random forest 16

3.5 Model Evaluation metrics 16
3.5.1 Confusion matrix 16
3.5.2 ROC curve and AUC 17
Chapter 4 Project Implementation 19
4.1 Data cleaning 19
4.1.1 Dataset fields selection 19
4.1.2 Data labelling and separation 20
4.1.3 Data format unification 20
4.2 Exploratory Data Analysis (EDA) 22
4.2.1 Protocol selection 22
4.2.2 Port selection 23
4.2.3 Duration 25
4.2.4 Total packets 27
4.3 Feature extraction 28
4.3.1 Time-window based data 28
4.3.2 Features 29
4.4 Model building 29
4.4.1 Training and test set split 30
4.4.2 Logistic regression 30
4.4.3 Decision tree 30
4.4.4 Random forest 31
4.5 Model Evaluation 31
Chapter 5 Results and Evaluation 32
5.1 Model evaluation results 32
5.1.1 Logistic regression 32
5.1.2 Decision tree 34
5.1.3 Random forest 36
5.2 Model comparison and analyse 38

5.2.1 Models with different dataset split method 38
5.2.2 Best model 38
5.3 Model application 39
Chapter 6 Conclusions 40
6.1 Review 40
6.2 Goal accomplishment 40
6.3 Future work 41
References 42

List of Tables

Table 1 Distribution of labels in the NetFlows for each scenario in the dataset.....5
Table 2 Characteristics of botnet scenarios.....6
Table 3 Amount of data on each botnet scenario.....6
Table 4 NetFlow fields (One flow).....7
Table 5 Chosen fields (data slice).....19
Table 6 Common methods for classification model[28].....30
Table 7 Commonly used classification metrics.....31
Table 8 Performance on single scenario (Logistic regression).....34
Table 9 Decision tree feature importance.....35
Table 10 Performance on single scenario (Decision tree).....36
Table 11 Random forest feature importance.....37
Table 12 Performance on single scenario (Random forest).....38
Table 13 Performance of 10 scenarios together.....39
Table 14 Application label results.....39

List of Figures

Figure 1 The Data Science process - O'Neil & Schutt 3
Figure 2 Layer model and corresponding protocol 8
Figure 3 Centralized botnet structure 9
Figure 4 Machine learning process 12
Figure 5 Sigmoid function 14
Figure 6 Confusion matrix format 16
Figure 7 ROC curve 18
Figure 8 Raw data format 21
Figure 9 Hex to Dec convert 21
Figure 10 NaN to -1 convert 21
Figure 11 Number of flows in each class 22
Figure 12 Protocol distribution 23
Figure 13 Protocol distribution bar chart 23
Figure 14 The most frequent used ports by malware (True) 24
Figure 15 The most frequent used ports by normal software (False) 24
Figure 16 Duration distribution of the malware and the normal 25
Figure 17 Log Duration box-plot 26
Figure 18 Log Duration distribution 26
Figure 19 TotBytes-TotAppByte-TotPkts 27
Figure 20 Total packets distribution 27
Figure 21 Log Total packets box-plot 28
Figure 22 Logistic regression performance (Randomly split) 32
Figure 23 Logistic regression performance (Scenario split) 33
Figure 24 Decision tree performance (Randomly split) 34
Figure 25 Decision tree performance (Scenario split) 35

Figure 26 Random forest (Randomly split) 36
Figure 27 Random forest (Scenario split) 37

Acknowledgements

First of all, I sincerely thank my dissertation supervisor, Marc Sabate Vidales of the University of Edinburgh, for giving me advice and recommendations on the research direction of my dissertation. In the process of writing the paper, I have been given a lot of guidance on the difficulties and doubts I encountered, especially many profound suggestions for improvement, and invested a lot of efforts and energy. Also, I would like to thank Ioanna Lampaki for her valuable advice and patient guidance.

Then, I would like to thank my friends, flatmates and classmates for their great support and help during these days of writing this paper the whole day in the library, which has greatly inspired me.

I also would like to thank my personal tutor Dr Mark Bull and programme officer Ben Morse for their help and care! At the same time, I would like to thank all the teachers and the students of EPCC for their hard work and help during this whole year, we have spent a perfect and unforgettable time together here in Edinburgh.

Finally, I would like to thank my family and friends back in China for their encouragement and support in completing this paper.

Chapter 1

Introduction

In this chapter, we will introduce some basic background about this project in section 1.1, in order to draw off the project goal in section 1.2. Research methods and report structure will also be briefly introduced in section 1.3 and 1.4.

1.1 Background

In the modern world, the Internet is one indispensable part of daily life, especially with the development of IOT (Internet of Things), everything can be connected to the Internet. This brings many advantages for the people but can cause cybersecurity problems as well. At best, such problems are information leakage or network congestion; at worst, they can cause heavy property loss or damage. Over the years, many countries have published official cybersecurity strategy document in the hope of raising attention to it and fighting against cybercrime[2]. Therefore, cybersecurity is a long-term task and requires continuous effort to cope with.

Cybersecurity relies on a range of defensive mechanisms, from the sophisticated intrusion detection system to the basic firewalls that can prevent or identify malicious behaviour. One common way is to monitor the traffic going through the network. Instead of trying to decode the exact content, which is nearly impossible under the complex encryption scheme, only the header of packets is extracted to analyse the host behaviour. The header of an encapsulated packet includes information such as the source and the destination IP addresses and port numbers, the internetwork layer protocol and several other information[3]. These headers information can reflect the different patterns of normal and malicious behaviour.

However, due to the huge volume of network data generated every day, it is a great challenge for engineers to manually cope with this large scale of network traffic and identify all the malicious behaviours. Current analyse tools usually need post-processing by engineers and are overly difficult to configure. In order to deal with traffic data more efficiently and precisely, a modern data science technology featuring machine learning classification models will be adopted in this project to distinguish malicious behaviours and locate infected hosts.

The project model follows the idea of big data process framework. Nowadays, big data process has become a popular topic in the industry when it comes to analysing large volumes of data. The machine learning methods like regression algorithms and classification algorithms provide us with powerful tools to deal with patterns that cannot be easily recognised by humans. With these up-to-date technologies, hopefully, a classifier can be built to identify normal hosts and malware-infected hosts.

1.2 Project goals

The network is a complicated system consists of thousands of information exchange; there are many vulnerabilities that malware can invade, infect and manipulate. Once a host is infected, it is very likely to perform some abnormal acts over the network. For example, SPAM malware will send out large numbers of junk emails to other hosts, these emails are mainly sent using the Simple Mail Transfer Protocol (SMTP) through port 25 [4], which means if one host continuously sends packets under protocol SMTP and destination port 25 during a short period, then it is under great suspicion of being infected. Apart from the SPAM, there are also many other malware exist, so one goal of this project is to use the Exploratory Data Analysis (EDA) in data science to extract these malicious behaviours for further study.

With the development of computer science, more and more real-life problems are being processed using machine learning technology. Nowadays machine learning can not only help select future directors[5] but also tell cancer cells from normal cells[6]. Back to the dataset we use, the CTU-13 dataset[1], is a dataset consists of 13 captures with different botnet, normal and background traffic. The traffic is presented in the form of NetFlows. One single NetFlow datagram represents several packets of communication between two certain hosts over the network; these packets share the same header information. The botnet and normal traffic are clearly labelled while the background traffic is not, so in this project we will use the botnet and the normal traffic to build the model and the background traffic for application. Features can be extracted from the NetFlow information and viewed as a classification model's input, because we have the label marking if the flow is sent from an infected host or not. Based on the idea of building a classifier, machine learning models like decision trees, random forests and logistics regression will be applied in the hope of finding a model that best estimates the flow and host state.

Once the model is built, it will be used to test on the unlabelled background data to classify labels and evaluate whether the corresponding host is infected or not according to the proportion of marked data.

To sum up, the goal of this project is to study and understand network behaviour and extract features of malicious and normal behaviour from the NetFlow traffic, using the CTU-13 dataset – a public annotated NetFlows dataset[1], and apply machine learning knowledge to build a classification model that can identify if a host is infected by malware or not.

1.3 Research methods

To build a machine learning model with network traffic data, we will apply a data science process framework. Many practical frameworks are used in industry and research like KDD (Knowledge Discovery in Database), SEMMA (Sample, Explore, Modify, Model and Assess), CRISP-DM (Cross Industry Standard Process for Data Mining), EMC's Data Analytics Lifecycle and O'Neil-Schutt process. Although there are so many different kinds of process, they all share the same basic routine: determine the project goal, obtain relevant data, data cleaning, data mining, analyse results and exploit knowledge. This project follows O'Neil-Schutt process detailed in Figure 1[7] as it best described the steps we need to take in this project: fetch raw data, pre-process and clean data, extract features using EDA, build ML models and exploitation.

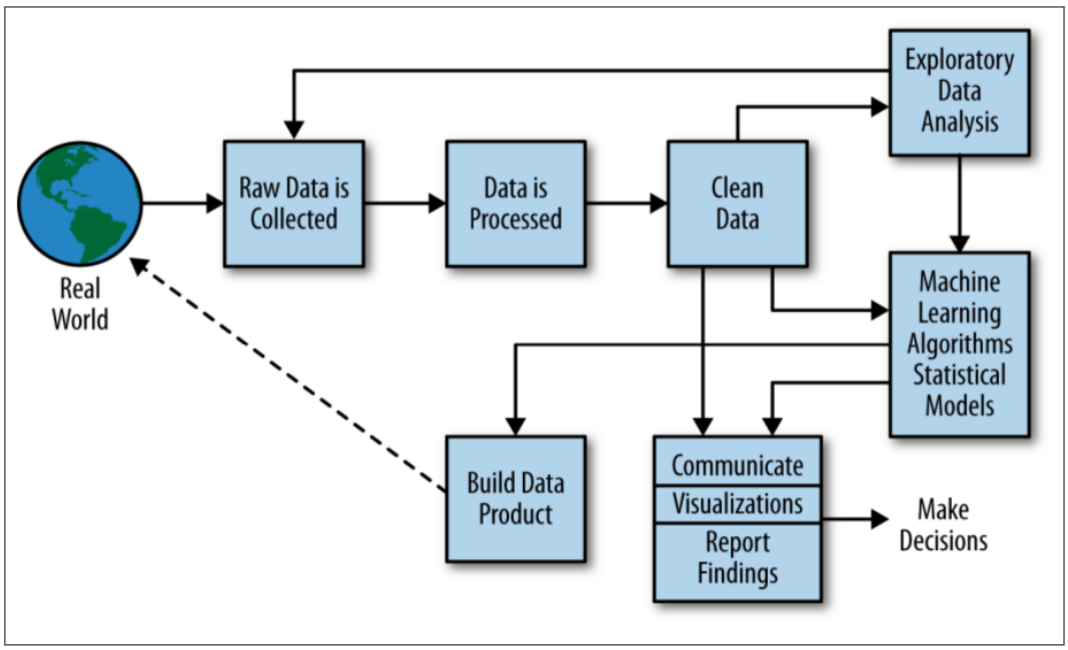


Figure 1 The Data Science process - O'Neil & Schutt

The primary developing language of this project is Python 3.6, Anaconda distribution[8]. The development environment is mainly Jupyter Notebook and Spyder IDE. The libraries used to process the data and build the models include pandas[31], numpy[10], matplotlib[9], seaborn[12] and scikitlearn[11]. Cirrus is also used when it comes to large volume of data processing.

1.4 Report structure

This chapter attempted to introduce some background information on the cybersecurity industry and the challenges engineers facing, also covered the general goal and the methods we will use in this project. Chapter 2 gives some details of the dataset used, along with the potential malware behaviour to investigate what features can be extracted. Chapter 3 is the methodology of machine learning, including machine learning introduction, model algorithm introduction and the evaluation metrics. Chapter 4 records

the implementation procedure and the experiment details. Chapter 5 is the model results and evaluation, along with the model application. Finally, chapter 6 concludes all the results and proposed the limitation and direction for future research.

Chapter 2

Contextual Review

As mentioned in section 1.3, we will follow the O'Neil-Schutt process to deal with data in this project. Usually, a data analysis project can be split into two parts, data pre-processing and model fitting. In this chapter, we will focus on the data pre-processing part by introducing the raw dataset in section 2.1, as well as some background knowledge of botnet behaviour in section 2.2.

2.1 The CTU-13 dataset

2.1.1 Dataset introduction

In order to collect enough useful information on analysis and model building, a large volume of data is needed, the CTU-13 dataset is one good choice as it has thirteen different botnet traffic captures (called scenarios) with a total of around 800,000 records of labelled normal and infected NetFlows. Besides these NetFlows that can be used to build models, there are also a large number of unlabelled background NetFlows which is perfect for the final model application. Table 1 shows the distribution of each kind of NetFlow in each scenario.

This dataset was captured in 2011 by the CTU University, Czech Republic. In each scenario, there is a specific malware executed. More information about the malware, the duration and the size of each capture is in Table 3. These malware use several different botnets and perform different actions. Table 2 shows the botnet characteristics of each scenario, more details of this will be introduced in section 2.2.

Table 1 Distribution of labels in the NetFlows for each scenario in the dataset

Scen.	Total Flows	Botnet Flows	Normal Flows	C&C Flows	Background Flows
1	2,824,636	39,933(1.41%)	30,387(1.07%)	1,026(0.03%)	2,753,290(97.47%)
2	1,808,122	18,839(1.04%)	9,120(0.5%)	2,102(0.11%)	1,778,061(98.33%)
3	4,710,638	26,759(0.56%)	116,887(2.48%)	63(0.001%)	4,566,929(96.94%)
4	1,121,076	1,719(0.15%)	25,268(2.25%)	49(0.004%)	1,094,040(97.58%)
5	129,832	695(0.53%)	4,679(3.6%)	206(1.15%)	124,252(95.7%)
6	558,919	4,431(0.79%)	7,494(1.34%)	199(0.03%)	546,795(97.83%)
7	114,077	37(0.03%)	1,677(1.47%)	26(0.02%)	112,337(98.47%)
8	2,954,230	5,052(0.17%)	72,822(2.46%)	1,074(2.4%)	2,875,282(97.32%)
9	2,753,884	179,880(6.5%)	43,340(1.57%)	5,099(0.18%)	2,525,565(91.7%)
10	1,309,791	106,315(8.11%)	15,847(1.2%)	37(0.002%)	1,187,592(90.67%)
11	107,251	8,161(7.6%)	2,718(2.53%)	3(0.002%)	96,369(89.85%)
12	325,471	2,143(0.65%)	7,628(2.34%)	25(0.007%)	315,675(96.99%)
13	1,925,149	38,791(2.01%)	31,939(1.65%)	1,202(0.06%)	1,853,217(96.26%)

Table 2 Characteristics of botnet scenarios

Table 2 – Characteristics of the botnet scenarios. (CF: ClickFraud, PS: Port Scan, FF: FastFlux, US: Compiled and controlled by us.)

Id	IRC	SPAM	CF	PS	DDoS	FF	P2P	US	HTTP	Note
1	√	√	√							
2	√	√	√							
3	√			√				√		
4	√				√			√		UDP and ICMP DDoS.
5		√		√					√	Scan web proxies.
6				√						Proprietary C&C. RDP.
7									√	Chinese hosts.
8				√						Proprietary C&C. Net-BIOS, STUN.
9	√	√	√	√						
10	√				√			√		UDP DDoS.
11	√				√			√		ICMP DDoS.
12							√			Synchronization.
13		√		√					√	Captcha. Web mail.

Table 3 Amount of data on each botnet scenario

Id	Duration(hrs)	# Packets	#NetFlows	Size	Bot	#Bots
1	6.15	71,971,482	2,824,637	52GB	Neris	1
2	4.21	71,851,300	1,808,123	60GB	Neris	1
3	66.85	167,730,395	4,710,639	121GB	Rbot	1
4	4.21	62,089,135	1,121,077	53GB	Rbot	1
5	11.63	4,481,167	129,833	37.6GB	Virut	1
6	2.18	38,764,357	558,920	30GB	Menti	1
7	0.38	7,467,139	114,078	5.8GB	Sogou	1
8	19.5	155,207,799	2,954,231	123GB	Murlo	1
9	5.18	115,415,321	2,753,885	94GB	Neris	10
10	4.75	90,389,782	1,309,792	73GB	Rbot	10
11	0.26	6,337,202	107,252	5.2GB	Rbot	3
12	1.21	13,212,268	325,472	8.3GB	NSIS.ay	3
13	16.36	50,888,256	1,925,150	34GB	Virut	1

2.1.2 Dataset format

The record of this dataset is in the form of NetFlows[13], which is a packets' sequence from a source point to a destination point performing the same instruction. As a result, one NetFlow shares the same header information of the packets it contains, like source and destination IP address, source and destination port number, etc.

The raw data captured is the pcap (packet capture) files, then they were processed by the researchers in CTU university using Argus[14], a software that can generate bidirectional NetFlow files with detail field information. The reason why we use NetFlows instead of packets is that flow-based characteristics are easier to obtain than packet-based characteristics, and NetFlows are already aggregated from packets which means it will be easier to extract patterns from[15].

Table 4 NetFlow fields (One flow)

SrcAddr	DstAddr	Proto	Sport	Dport	State
147.32.84.165	64.4.2.109	tcp	1453	80	FIN
sTos	**dTos**	**SrcWin**	**DstWin**	**sHops**	**dHops**
0	0	64240	5615	1	14
StartTime	**LastTime**	**sTtl**	**dTtl**	**TcpRtt**	**SynAck**
40770.70884	40770.70886	127	242	0.1631	0.162798
AckDat	**SrcPkts**	**DstPkts**	**SrcBytes**	**DstBytes**	**SAppBytes**
0.000315	19	27	2365	31232	1235
DAppBytes	**Dur**	**TotPkts**	**TotBytes**	**TotAppB**	**Rate**
29742	1.485464	46	33597	30977	30.29357
SrcRate	**DstRate**	**Label**			
13.610401	19.657267	flow=From-Botnet-V46-TCP-Established-HTTP			

Table 4 is all the 33 fields one NetFlow has, each of these fields records the corresponding information about this data transmission. Some fields record basic information like source & destination address (*SrcAddr, DstAddr*), communication protocol (*Proto*), source & destination port (*Sport, Dport*), packets number (*TotPkts*), etc. Others are not that commonly used like *Hops*, which means the number of bridges, routers and gateways one packet has passed from the source to the destination. This field is mainly affected by the network congestion, so it shows little relation to malware behaviour. *TTL* (Time to Live) is a field similar to *Hops*. These less important fields will be cleaned out as they contribute a little in identifying malicious behaviours.

To build a classifier, a label field is needed for identifying whether this flow is generated by an infected host. We can create this label based on the *Label* field from Table 4. Those that generated by the malware start with "From-Botnet" and those from the normal hosts are "From-Normal". Apart from these two types, others are background flows that have no clear label, so we only use data with the explicit "normal" or "botnet" label to train our model.

2.2 Internet and malware

All the information sent through the Internet follow certain routines, and these routines can be divided according to its running parts of the Internet, which is the different layers of the communication model. There are commonly two types of model, one is Open Systems Interconnection model (OSI model) with seven layers and the other is the TCP/IP model, four layers created based on a set of protocols[16]. Although the OSI model follows the standard and is a perfect theoretical model, TCP/IP model is much more widely used in industry due to its support of many protocols and applications.

As shown in Figure 2, the left part is the OSI and TCP/IP model; the right part is the corresponding protocols running on each TCP/IP model layer. Some upper layer protocols are operated based on the lower layer protocols. In the CTU-13 dataset, the recorded protocol is on the layer 2 (Internetwork Layer) and layer 3 (Transport Layer).

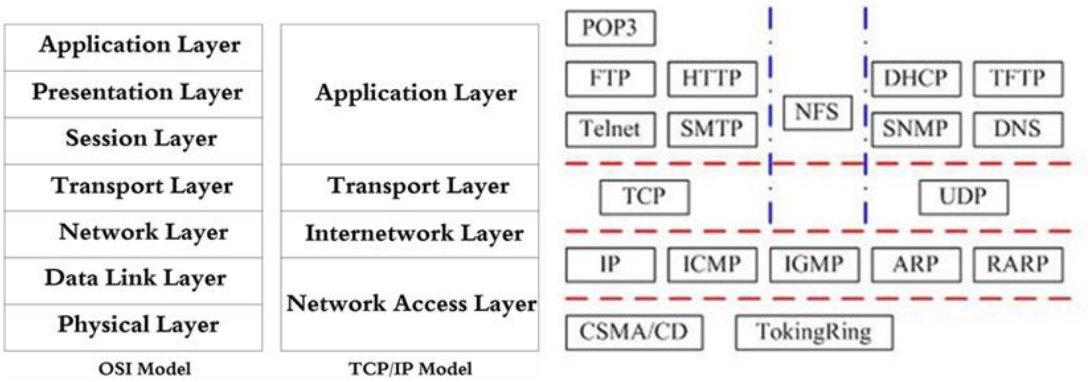


Figure 2 Layer model and corresponding protocol

The normal software sends information according to a certain protocol and keeps everything under control, while malware usually generate bulk network traffic or sends rubbish information to cause damage and create chaos. Malware has become a serious problem in the IT industry and dealing with it requires continuous effort and up-to-date technology.

SPAM and DDOS are two typical malware that can manipulate the infected host to spread rubbish information or block other hosts, even create a network outage. Other malware like Click Fraud (CF) can falsify the clicks of certain link, cause a loss to the link owner. Port Scan (PS) malware can scan the available ports of other hosts and trying to perform malicious behaviour.

Table 2 gives us the detailed malware in each scenario, including IRC (Internet Relay Chat), SPAM, CF (Click Fraud), PS (Port Scan), DDoS (Distributed Denial of Service), FF (Fast Flux) and US (Compiled and Controlled by CTU University). In this section, we are going to describe the behaviour of some typical malware.

2.2.1 SPAM behaviour

SPAM usually refers to the junk emails but here we also use SPAM to represent the malware that spreads junk emails. Usually, spam is sent through the Internet in a way exactly like normal mails, the only difference is that SPAM will continuously send the same content mail to different hosts in a certain period. Since the mail system usually uses the SMTP protocol to send emails, spams are also sent through this protocol. Similarly, the port used for SMTP is normally port 25 and port 587, specified by IANA and RFC7504[17]. Therefore, to identify SPAM behaviour, we need to focus on the protocol, the port number and the sent time difference.

2.2.2 DDoS behaviour

Distributed Denial of Service (DDoS) is a typical botnet attack. A botnet is a group of infected devices connected in a coordinated way that can perform malicious behaviour[18], each infected device is a bot, the command and control (C&C) server controls all the bots, and finally the server is controlled by the hacker, so the hacker does not control bots directly, this is the reason why DDoS attack is so hard to trace back.

Figure 3 is a typically botnet structure: centralised botnet. During the attack, the C&C server receives orders from the hacker, the server will spread the order command to all the bots connected to it. Then the bots will follow the command and act according to instruction like “zombie robots”, sending large numbers of Internet Control Message Protocol (ICMP) packets towards the target victim at the same time. These ICMP packets contain requests and require a reply, a large number of requests will use up the victim’s resources and block it from answering requests from other normal hosts.

As a result of DDOS attacking behaviour, if we want to identify one bot, we need to focus on the number of NetFlow it sent to a specific destination host in a fairly short period using ICMP protocol.

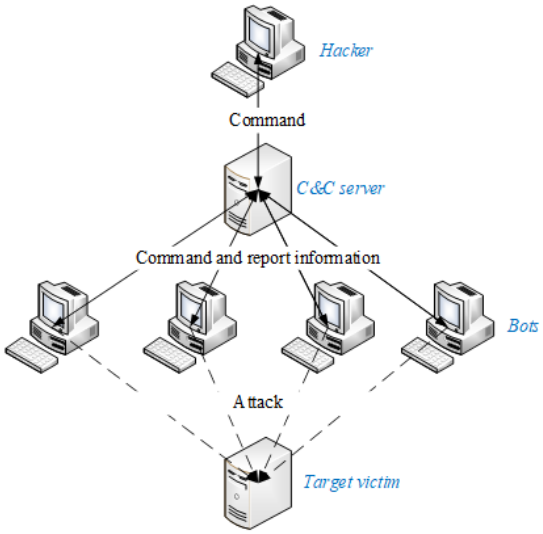


Figure 3 Centralized botnet structure

2.2.3 IRC behaviour

IRC is short for Internet Relay Chat, it is originally an online chatting application layer protocol. IRC works on a client/server model. There are many different chatting channels on several servers that clients can connect to and communicate with those in the same channel, both in groups or in private[19].

IRC is a relevantly simple and open protocol and easy to be attacked. Bots will pretend to be normal clients, sending malware to other clients or flooding channels with repeated connection requests, to block normal conversation and increase server load[20]. IRC also uses the Xabi Direct Client-to-Client (XDCC) protocol[21], which allows clients to have direct conversation with each other, no need for the traffic to go through the server. This also results in the risk of being attacked by a bot. Apart from all above attack, IRC can even perform a DDOS attack by infecting the clients in a channel and manipulate them as bots.

Therefore, a variety of attacks against IRC networks and users can be performed. There is no clear pattern of what behaviour IRC attack have. But we can still focus on the default

IRC port 6667 defined in RFC7194[22], the number of flows sent in a certain time and the packets size.

2.2.4 Other malicious behaviours

Section 2.2.1 to 2.2.3 are three top common attacks; there are also other malware exist in the CTU dataset that we cannot see their behaviour patterns clearly. One point we can focus on is the destination port because sending information through the Internet requires endpoint, which is the communication port. There are 65536 (0~65535) ports defined by Internet Assigned Numbers Authority (IANA)[23]. The first 1024 ports (0~1023) are well-known ports, they are reserved for specific processes and network service. Other ports can be used flexibly by any applications.

Here are some common well-known ports and the services run on it:

Port 22：SSH (Secure Shell) potential port scan target

Port 53：DNS (Domain Name Server)

Port 80/8080/3128/8081/9080/443: HTTP/HTTPS proxy server

Port 135：RPC (Remote Procedure Call) potential port scan target

Note that all the port numbers discussed above are the destination port number, while the source port number is randomly allocated by the system. So all the port numbers used in this paper refer to the destination port number.

The port number is one possible way of recognising malware. There is no perfect way to identify malware all out by standard behaviour, so we are going to use the statistical methods: Exploratory Data Analysis in section 4.2, to see if we can find any other distinctive features between the normal and the malware.

Chapter 3

Machine Learning Methodology

This chapter is mainly about the methodology used in machine learning area. We will focus on the models we used in during this project, includes logistic regression, decision tree and random forest. After a brief introduction on machine learning general procedure in section 3.1 and 3.2, we will provide some information on how to split the training/test set in section 3.3 and the algorithms each model used and evaluate in section 3.4 and 3.5.

3.1 Machine learning introduction

Machine learning, also called statistical learning in the statistical area, is a set of methods that can identify data pattern automatically and perform data prediction or make decisions[24]. Machine learning gives people a better way to understand the hidden meaning of data.

Generally, there are two types of machine learning method: Supervised Learning and Unsupervised Learning. A Supervised Learning model is built for predicting, or estimating, an output based on each observation of the input[25]. It is a mapping from the input predictor to the output response. Methods like linear regression, tree-based methods and Support Vector Machine (SVM) all belong to supervised learning. Unsupervised Learning, contrary to Supervised Learning, does not have the supervising output. Unsupervised Learning aims to learn the data relationship directly from the input[25]. A typical Unsupervised Learning method is clustering.

One way to characterise variables is by their nature, either continuous or categorical. Continuous variables have clear continuous numeric values while categorical variables usually fall into different classes or categories[25]. For example, continuous variables can be the age of a person or income, and categorical variables can be the binary class like (yes, no), gender (male, female) or multiclass like brand (A, B, or C). Based on the differences in variables, Supervised Learning can also be divided into two parts in regard to its output. If the response of a problem is categorical, then it belongs to classification; otherwise, it is referred to as regression[25]. Since the data this project going to process is labelled with binary class (botnet and normal), we will focus on building classification models of Supervised Learning methods.

One typical classification method is logistics regression, it is a parametric model. Parametric model means that we assume the function of the mapping from input to output follows a mathematical form that has a fixed number of parameters, like in the form of a linear model. Once we have this model (function), we can apply the training data using approaches like the ordinary least squares and maximum likelihood to fit the model, in order to get the suitable parameters for the function. The parametric model reduces the problem of estimating the arbitrary function down to the estimation of several

parameters.[25] But it also has some challenges if the chosen model does not follow the true form of function or in a situation where the observations cannot be linearly separated.

In contrast to the parametric model, there is another model called the non-parametrical model; a typical model is random forest. For non-parametrical models, we do not make assumptions about the potential function. Instead, we just fit the model using the observation to construct the relationship between the input and the output[25]. In this way, the model can fit more possible shapes than parametric models since it does not have a hypothesis relationship between the inputs and the outputs. But this also leads to a disadvantage that if we want a better model, a large amount of data is needed, much more than those of the parametric model. In this project, we are going to apply both the parametric and the non-parametrical model for comparison.

3.2 Machine learning process

The machine learning process usually involves iterative steps of splitting data, model fit, model evaluation and model application. Figure 4 is a general flow of the machine learning process used in this project.

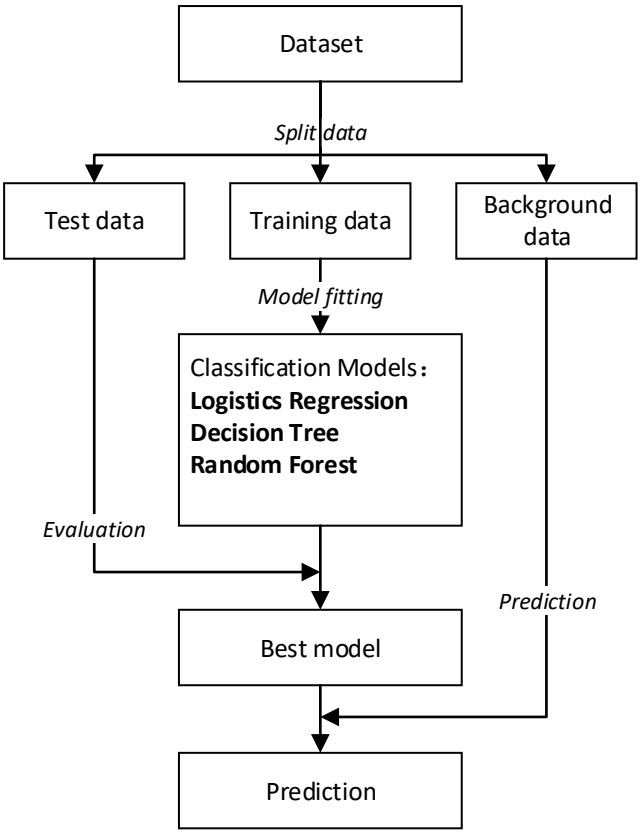


Figure 4 Machine learning process

First, the input is the processed dataset with features extracted from the raw dataset. Then for the model training and evaluation, we need to split the dataset into a training dataset and a test dataset. These datasets have complete information of the input predictor and the output response. Apart from these two datasets, we also have the background dataset

that do not have the output response, for model application. More details of the background dataset will be described in section 4.1.2. Once the training set is ready, they can be applied to the classification models. In this project, we will build the following models: logistics regression, decision tree and random forest. The test set will be applied to the trained model to evaluate their performance using criteria like accuracy, recall, receiver operator characteristic curve (ROC) and confusion matrix. Based on the criteria above, we can modify the features used and do some tuning to optimise the models. And finally, one best-performed model will be selected and applied to the background dataset to see if there exists any malicious behaviour. The general goal of the whole process is to build a model whose artificial outputs are close to the real outputs so that it can be practical for all sorts of inputs likely to be encountered in practice[26].

3.3 Test & training dataset

The training set is a part of the data that is used to train the model. In supervised learning, we give the model an input and it will return a result output, based on the comparison of the result output and the target output, combined with the specific learning algorithm, we can adjust the parameters of the model and get a model that best fits the training dataset. Things are not over yet when we have the model. Since we built the model to estimate the probability or category of future occurrence, we need to test if this model works well on the unseen dataset and how is its generalisation ability. That is where the test dataset plays its part. Test dataset is a dataset with the same probability distribution as training data, but independent of the training data. Test dataset will be used to test the model and generate the results of the evaluation metrics.

If a model fits well in both training set and test set, it is a basically good model; if the model performs good on training set but bad on test set, this is called overfitting, that happens when the model works too hard to find all the patterns in the training data, even the errors and noise patterns are picked up. On the contrary, if the performance on the test set and the training set are both bad, this is underfitting, which means the model still needs more training.

In this project, the dataset we are using has 13 scenarios; we can split them by scenario and apply as the training set and the test set. In this way, we can analyse the model generalisation ability on unseen malware. We can also concatenate them all and randomly split them to training set and test set. We will try both ways and see if there is any difference.

3.4 Model Fitting

The goal of this project is to build a classifier that can identify malware behaviour and detect infected hosts. All the input features are numeric measurements, and the output is a binary variable (detailed in Chapter 4). In this project, the following classification models were chosen: logistic regression, decision tree and random forest. The details of these models are discussed from section 3.4.1 to 3.4.3.

3.4.1 Logistic regression

Logistic regression is a model that based on the linear relationship between of one or more predictors and the target response. Unlike a classifier, logistic regression estimates the probability of the binary response and use this probability to compare with a threshold value and decide which class the response belongs to. A typical threshold value for binary response is 0.5. Logistic regression can also be generalised to classify more than two values, this model is called multinomial logistic regression. But in this project, we only discuss under the circumstance of binary values.

Logistic regression is based on linear regression, equation (1) is a common form of the linear regression model. But we cannot simply use linear regression to estimate class because the result of this equation is not limited to the (0, 1), results out of this range cannot be classified.

(1) $g(X) = \theta_0 + \theta_1 X$

In order to avoid this, the logistic function (sigmoid function) is introduced to restrict the output to (0, 1). Equation (2) is the sigmoid function and from Figure 5 we can see that this function restricts the outputs between 0 and 1 for all the input values X.

(2) $h(X) = \frac{1}{1+e^{-g(X)}}$

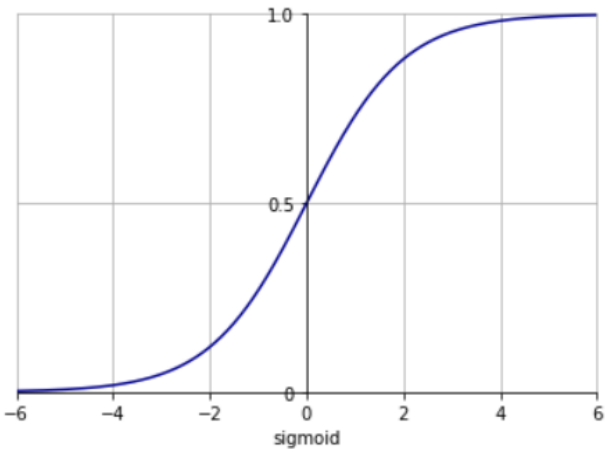


Figure 5 Sigmoid function

Once the model is defined, we will build the cost function and find the parameters to minimise it. The logistic regression cost function is based on maximum likelihood instead of the least squares used in linear regression because maximum likelihood has better statistical properties[25].

Equation (3) is the maximum likelihood function, we want to find the parameter θ that maximize this equation. To simplify the calculation, we can calculate the max log-likelihood, which is in equation (4). So, the cost function of logistic regression can be defined as equation (5). To minimise the cost function, we usually use optimization algorithms like Gradient descent, Stochastic Gradient descent, and Mini-batch Gradient descent.

(3) $$L(\theta) = \prod_{i=1}^{m} p\left(y^i \middle| x^i; \theta\right) = \prod_{i=1}^{m} \left(\hbar_\theta\left(x^i\right)\right)^{y^i} \left(1 - \hbar_\theta\left(x^i\right)\right)^{1-y^i}$$

(4) $$log(L(\theta) = \sum_{i=1}^{m} y^i \, log \, \hbar_\theta\left(x^i\right) + (1 - y^i) \, log \left(1 - \hbar_\theta\left(x^i\right)\right)$$

(5) $$Cost(\theta) = -\sum_{i=1}^{m} y^i \, log \, \hbar_\theta\left(x^i\right) + (1 - y^i) \, log \left(1 - \hbar_\theta\left(x^i\right)\right)$$

3.4.2 Decision tree

Unlike logistic regression, decision tree is a non-parametrical model, it splits the predictor space into some simple regions based on the value of input observation. Then for the classification tree, we use the mean or the mode of the training observations in the region to make estimations.

Since the model is a "tree", there are root node, internal nodes, leaves and branches. Each node is split to create new branches and new nodes until the leaf node. Classification tree will split according to a certain value of each feature to decide which feature is suitable. This value can be the classification error rate, the entropy or the Gini index. The classification error rate is the fraction of the training set in that region that do not belong to the most common class[25]. But in practice, the information gain and the Gini index are more commonly used.

The entropy is the information of the certain samples. The next feature to be selected is the one that has the most information gain, which means the entropy reduces the fastest. The final leaf entropy to be as small as possible. The information gain function is stated in equation (6). H means the entropy (equation (7)).

(6) $$IG(X, a) = H(X) - H(X|a) = H(X) - \sum_{a_i} p(a = a_i) H(X|a = a_i)$$

(7) $$H(X) = -\sum_{a_i} p_{a^i} \, ln(p_{a^i})$$

The Gini index is a replacement of entropy, it is the measure of total variance across the K classes. Similar to entropy, we also want the final Gini index to be small (equation (8)).

(8) $$Gini(p) = \sum_{k=1}^{k} p_k (1 - p_k) = 1 - \sum_{k=1}^{k} p_k^2$$

The split process will repeat for each new node until all entries in the node are in the same class, or all features are used, in this circumstance, a majority vote will be taken to decide which class the leaf belongs to.

Decision tree model is likely to be overfitting because it produces very detailed and large tree, and each feature is considered in detail, this would give good performance on the training set, but the errors are also learnt so the performance on test set may not be that good. One way to avoid overfitting is pruning. There are two ways to prune the trees, one is called Pre-Pruning and the other is Post-Pruning. The pre-pruning method can be setting a threshold before building the tree, and once the information gain is smaller than this threshold, we stop splitting. Post-Pruning is prune after the tree was built, like

calculating the entropy of the two leaves from one internal node, if it is smaller than a threshold, we merge the two leaves and use the internal node as the new leaves based on a majority class criterion. Normally Post-Pruning will have better results than Pre-Pruning.

3.4.3 Random forest

Decision tree is easy to understand but more likely to overfit. Therefore, to reduce the probability of overfitting, a more powerful classification model based on trees is built, that is random forest.

Random forest is an ensemble method that uses bagging plus randomly select features for model building. Bagging is short for bootstrap aggregating, this is an ensemble meta-algorithm that randomly select the training data based on the sampling with replacement method and then build the tree classifier. The classifier's result is determined by a majority vote of the classes predicted by each of the trees, which is the most commonly occurring class among all the predictions[25]. Therefore, random forest is randomly choosing the features used and randomly choosing the training data to build several trees and combined all the trees results together.

Because of the two randomnesses in random forest, it is not easy to run into the overfitting problem and not sensitive to noise. But it is hard to understand than the straightforward interpretation of the decision tree.

3.5 Model Evaluation metrics

There are many metrics for evaluating the classification model. For a binary classifier, here are some common metrics: accuracy, recall, specificity, precision, ROC curve and AUC. We are going to introduce all these metrics in this section.

3.5.1 Confusion matrix

Nearly all the classification metrics are based on the confusion matrix, also known as the error matrix. It is a table that compares the predicted values and actual values (standard values), it reflects the performance of an algorithm. Figure 6 is the format of a confusion matrix. Columns are the predicted values counts and rows are the actual values counts.

		Standard Value	
		TRUE	FALSE
Predict Outcome	TRUE	True Positive (TP)	False Positive (FP)
	FALSE	False Negative (FN)	True Negative (TN)

Figure 6 Confusion matrix format

- **Accuracy**

Accuracy is the fraction of correctly classified samples. This is the most commonly used metric to evaluate a model. The higher this value is, indicates the model is better.

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

- **Recall**

Recall, also known as sensitivity, it is the true positive rate (TPR). This measures the proportion of actual positive values that are correctly identified. Higher recall means fewer false negative values.

(10) $$Recall = \frac{TP}{TP+FN}$$

- **Specificity**

Specificity measures the proportion of actual negative values that are correctly identified. Higher specificity means fewer false positive values. It is the true negative rate and usually compares with sensitivity (recall).

(11) $$Specificity = \frac{TN}{TN+FP}$$

- **Precision**

Precision (positive predictive value) represents the degree to which repeated measurements under unchanged conditions show the same results[27].

(12) $$Precision = \frac{TP}{TP+FP}$$

- **F1 score**

F1 score, also known as balanced F-score or F-measure, it is the harmonic average of recall and precision, considering they have equal weight. This is a good metric when recall and precision cause conflict and hard to decide which one is better.

(13) $$F1\ score = \frac{2*recall*precision}{recall+precision}$$

3.5.2 ROC curve and AUC

ROC curve is the receiver operating characteristic curve, it is the sensitivity against (1-specificity), which is TPR against FPR (Figure 7). The point on the curve depends on the threshold probability for which class the response belongs to.

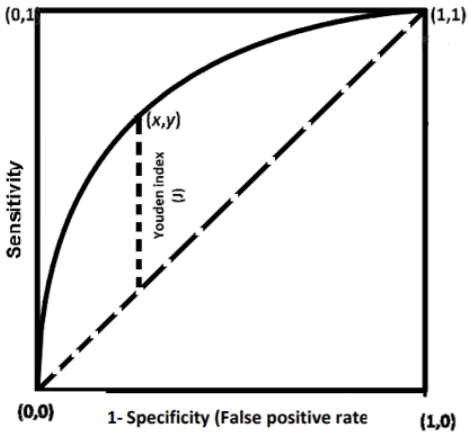


Figure 7 ROC curve

If the curve is close to point (0,1), means the performance of this model is good. This can also be measured by a value: AUC (Area under the curve), this value returns the area of the curve covers. AUC varies between 0 and 1, and over 0.7 indicates a good classifier.

To classify labels, the model will return the probability of being in each class. Therefore, we can modify the threshold of the two classes to improve the classifier performance. One of the most widely used methods to choose the threshold is the Youden index, it is a statistic that captures the performance of a classifier. Equation (14) is the Youden index. The threshold of the max Youden index will be the optimal threshold.

(14) $$Youden\ index = Sensitivity + Specificity - 1 = TPR - FPR$$

Chapter 4

Project Implementation

This chapter will describe what the main work is and how they were done in the whole project following the O'Neil & Schutt Data Science process mentioned in section 1.3. Section 4.1 to 4.3 will introduce the methods and standards we use to extract features from raw datasets, which belong to the data cleaning part. Section 4.4 will focus on applying machine learning models to the processed dataset and tuning for better classifiers. Section 4.5 is the model evaluation part.

4.1 Data cleaning

As shown in Table 2 from section 2.1.1, there is more than one botnet in each scenario, it is difficult to build a classifier on every single botnet, so we concatenate all the 13 datasets together in the hope of building one classifier that can identify all the malicious behaviours.

4.1.1 Dataset fields selection

The raw dataset consists of 13 NetFlow captures with different botnets. As stated in section 2.1.2, each NetFlow has more than 30 fields represent its detailed information, but not all of them are useful when distinguishing normal and malicious behaviour. Based on the network knowledge and the discriminators Moore et al.[29] listed in their report "Discriminators for use in flow-based classification", we chose the fields in Table 5 to investigate: *SrcAddr, DstAddr, Proto, Sport, Dport, StartTime, Dur, TotPkts, TotBytes and TotAppByte*. Besides, information from the *Label* field is used to generate the *label_1* for classification.

Table 5 Chosen fields (data slice)

	scenario	SrcAddr	DstAddr	Proto	Sport	Dport	StartTime	Dur	TotPkts	TotBytes	TotAppByte	Label	Label_1
0	8	147.32.1.20	147.32.85.7	udp	53	43512	2011-08-16 15:00:02.082443	0.000000	1	75	33	flow=From-Normal-V49-UDP-CVUT-DNS-Server	False
1	8	147.32.1.20	147.32.85.7	udp	53	50133	2011-08-16 15:00:02.082453	0.000000	1	75	33	flow=From-Normal-V49-UDP-CVUT-DNS-Server	False
2	9	147.32.1.20	147.32.85.7	udp	53	56036	2011-08-17 14:45:17.555958	0.000000	1	75	33	flow=From-Normal-V50-UDP-CVUT-DNS-Server	False

The following is the brief description of each chosen field in Table 5:

- ***Scenario***: The scenario number of this flow.
- ***SrcAddr/DstAddr***: The IP address of source/destination host.

- ***Proto***: The protocol this flow applied, like TCP, UDP, ICMP, etc.
- ***Sport/Dport***: The communication port number of source/destination host.
- ***StartTime/Dur***: The start of the flow and the duration time (Central European Summer Time).
- ***TotPkts/TotBytes/TotAppByte***: Total packets, bytes and AppBytes that sent through this flow.
- ***Label***: The label indicates from where or to where this flow is sent, there are three types of hosts: botnet, normal and background host.
- ***Label_1***: This label is used as the class of the NetFlow, if field *label* starts with "From-Botnet", then this flow will be marked as *True*; if field *label* starts with "From-Normal", this will be marked as *False*.

4.1.2 Data labelling and separation

The raw dataset includes all the botnet traffic mixed with normal traffic and background traffic. However, we will only use flows that can be labelled to train our models. The new field *Label_1* is generated for this purpose. This is a binary field labelling the flows from infected host as *True,* and *False* for flows from normal hosts. To identify normal and infected hosts, we can use the information provided in the field *Label* or the log file provided along with the dataset[1]. Only labels that contain "From-Botnet" and "From-Normal" can clearly point out which one is normal and which one is infected. Other labels contain like “To-Botnet”, “To-Normal” and “Background” should all be considered as background flows and separated for model application later. The data size for model building is around 800,000 NetFlows.

Note that the label is based on the source address, so this does not exclude the normal traffic sent from the infected host, these flows are also labelled *True*. Besides, some hosts were not infected at first, but the flows they sent before infected are also "From-Botnet", we picked out those flows manually according to the timeline provided in the dataset log file[1], and change their labels to *False* for data accuracy.

4.1.3 Data format unification

The raw dataset uses different formats to record information. For example, there are not only numerical data like *Dur* and *TotPkts*, but also string data stored the IP address in field *SrcAddr* and *DstAddr*. Besides that, we also need to deal with date data and mixed data (Figure 8). In order to clean and aggregate these raw data, we need to do some conversion and unification.

```
scenario        int64
SrcAddr        object
DstAddr        object
Proto          object
Sport          object
Dport          object
StartTime      object
Dur           float64
TotPkts         int64
TotBytes        int64
TotAppByte      int64
Label          object
Label_1          bool
dtype: object
```

Figure 8 Raw data format

Normally, the port should only include a decimal number from 0 to 65535, but in this dataset, the *Sport/Dport* columns are mixed data contain decimal numbers, hexadecimal numbers and NaN (a tiny minority of protocols do not need an endpoint port, like ARP[30]). For simplicity, we use the hex-to-dec function to convert the hexadecimal number to decimal number. From Figure 9, we can see that *Dport* of flow 1798 is changed from "0xc413" to "50195".

	index	scenario	SrcAddr	DstAddr	Proto	Sport	Dport	StartTime	Dur	TotPkts	TotBytes	TotAppByte	Label	Label_1
1798	1798	1	147.32.84.134	61.191.41.7	icmp	0x0303	0xc413	2011-08-10 11:10:12.151919	0.077727	2	960	836	flow=From-Normal-V42-Jist	False

	scenario	SrcAddr	DstAddr	Proto	Sport	Dport	StartTime	Dur	TotPkts	TotBytes	TotAppByte	Label	Label_1
1798	1	147.32.84.134	61.191.41.7	icmp	0x0303	50195.0	2011-08-10 11:10:12.151919	0.077727	2	960	836	flow=From-Normal-V42-Jist	False

Figure 9 Hex to Dec convert

For NaN, since they do not have any valid port number, we can use an invalid number to represent it. We choose "-1" to replace all the NaN port as shown in Figure 10.

	scenario	SrcAddr	DstAddr	Proto	Sport	Dport	StartTime	Dur	TotPkts	TotBytes	TotAppByte	Label	Label_1
796137	1	147.32.87.11	147.32.87.1	arp	NaN	NaN	2011/08/10 10:01:44.546115	3536.416992	8	480	256	flow=From-Normal-V42-MatLab-Server	False

	scenario	SrcAddr	DstAddr	Proto	Sport	Dport	StartTime	Dur	TotPkts	TotBytes	TotAppByte	Label	Label_1
796137	1	147.32.87.11	147.32.87.1	arp	-1	-1	2011-08-10 10:01:44.546115	3536.416992	8	480	256	flow=From-Normal-V42-MatLab-Server	False

Figure 10 NaN to -1 convert

After converting all the port records to the decimal number, we can unify them to float format.

In the raw dataset, the capture time (*StartTime*) of each flow is recorded in the string format, which is rather inconvenient when calculating the time difference for feature extraction. Since python has a large number of powerful libraries, we can use the date time covert function in the *Pandas*[31] library and convert the string to date. This date format enables us to do any time difference calculation for later feature extraction detailed in section 4.3.1.

4.2 Exploratory Data Analysis (EDA)

This section will explain some exploratory work performed on this dataset to gain an insight into the statistical distribution of the difference between normal and malicious behaviour. This analysis combined with the possible behaviour described in section 2.2, can provide us with a better understanding of what measurements is suitable for feature extraction.

Before analysing the data, we count the number of flows in each class, *True* is 443243 and *False* is 356534 (Figure 11), so the two classes are basically even.

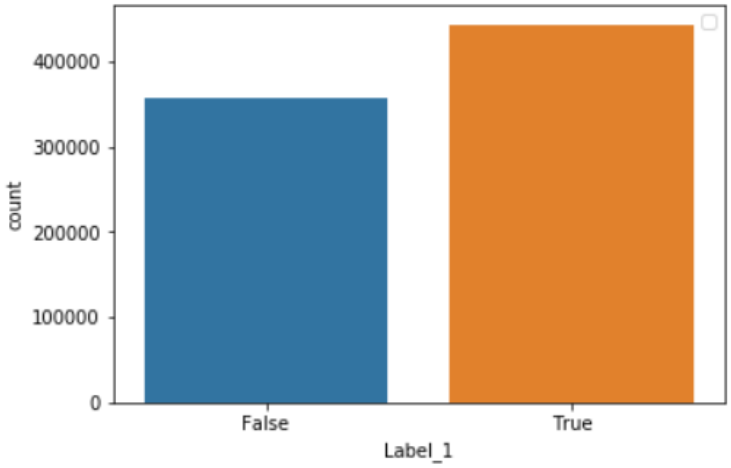


Figure 11 Number of flows in each class

4.2.1 Protocol selection

The *Proto* field records the protocol information used on layer 2 (Internetwork Layer) and layer 3 (Transport Layer) of each flow (Figure 2). The following types of protocol are used in this dataset: ARP, ICMP, IGMP, TCP, UDP and RTP. Figure 12 shows the number of each kind of protocol used in normal and infected hosts respectively. Figure 13 is a visualisation of the number distribution. Evidently, IGMP and RTP only represent one class of data, but they only have few numbers of flows to support. Otherwise, they can be a good feature to distinguish malicious behaviour. When using ICMP and TCP, flows labelled *True* outnumber those labelled *False*, while UDP gives the opposite outcome. Therefore, the number of ICMP&TCP against the number of UDP used can be an indicator of malicious behaviour.

```
In [76]: df.groupby(["Label_1","Proto"]).size()
Out[76]: Label_1  Proto
         False    arp        308
                  icmp      3307
                  igmp         2
                  tcp     124117
                  udp     228800
         True     icmp     114982
                  rtp           3
                  tcp      180004
                  udp      148254
         dtype: int64
```

Figure 12 Protocol distribution

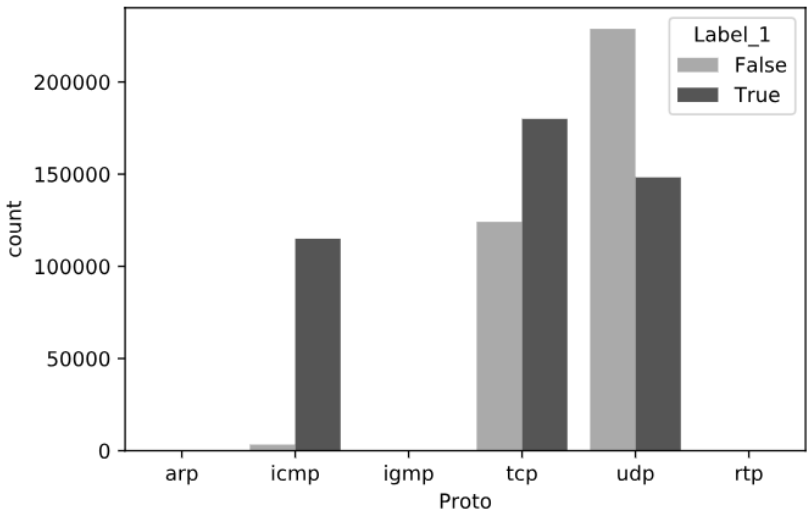


Figure 13 Protocol distribution bar chart

4.2.2 Port selection

In section 2.2.4, we introduced some ports and their common usage, but more analysis is needed to identify which port and which malware actually reflect the malicious behaviour. We count the occurrence of a certain port in each class and sort the values to get the most frequently used port in both classes (Figure 14 and Figure 15). Since we have nearly 800,000 flows in this dataset, we only take counts over 1000 into consideration.

Port numbered 53, 443, 80 are the most common ports used when surfing the Internet. There are no clear evidence showing if they are related to any malware shown in Table 2, and no big difference in the frequency of occurrence between the two classes from the statistical results. On the contrary, port numbered 25, 22, 6667, 65500, 135, 179, 3389 are more frequently appeared in malicious flows than in normal flows. It is also accord with the malicious behaviour introduced in section 2.2. This provides a good way to distinguish them. Port numbered 8977, 27015 appear a lot in normal flows, but they are not well-known ports, all the service can use these two ports, so they do not have much meaning in identifying behaviour difference.

Apart from all the ports mentioned above, there is a special one: port 0, frequently appears in the malicious flows. This port is reserved in TCP/IP networking and should not be used to send messages, so this large usage of port 0 is quite unusual. This port may be used by malware as a method to analyse the response messages that hosts generate, this response can help attackers learn more about the potential network vulnerabilities[32]. Or it may also accidentally generate by applications programmed incorrectly. Anyhow, we will count this port as a feature.

```
In [138]: df[df.Label_1==True].Dport.value_counts().sort_values()
161.0          124
7600.0         182
888.0          190
5678.0         199
8000.0         243
3128.0         426
587.0          475
4506.0         827
3389.0        1062
179.0         1425
135.0         4725
65500.0       5461
6667.0       10868
80.0         26529
22.0         26748
443.0        34268
25.0         66538
0.0         109933
53.0        145815
Name: Dport, Length: 1404, dtype: int64
```

Figure 14 The most frequent used ports by malware (True)

```
In [136]: df[df.Label_1==False].Dport.value_counts().sort_values()
27019.0        123
27025.0        144
123.0          153
33822.0        167
27018.0        170
8950.0         178
5222.0         184
27031.0        266
27017.0        286
13067.0        308
54728.0        318
27016.0        358
8618.0         718
25.0          1127
8977.0        1371
27015.0       2325
443.0        21331
80.0        100457
53.0        222644
Name: Dport, Length: 1619, dtype: int64
```

Figure 15 The most frequent used ports by normal software (False)

4.2.3 Duration

The duration of one flow varies from 0 to 3600 seconds. In order to see if there is any difference between the duration of the two classes, we plot the hist distribution of them in Figure 16. Apparently, the data is too densely distributed at the small numbers and difficult to see any larger numbers' distribution. To overcome this, we apply the log operation on the duration data. Logarithmic scales can compress large-scale data without changing its original relation and property.

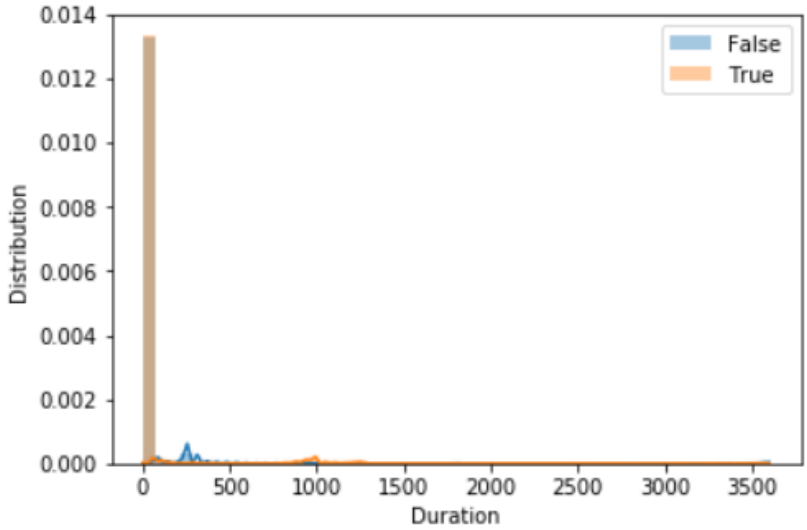


Figure 16 Duration distribution of the malware and the normal

Figure 17 is the boxplot of the log-duration data. Boxplot is a visualisation of distribution based on the five data summary: the minimum data point, the first quartile (Q1), the median (Q2), the third quartile(Q3), and the maximum data point. To calculate the minimum and maximum data points, we first need to calculate the interquartile range (IQR), it is the difference between third and first percentiles, which is IQR=Q3-Q1, so minimum data point is Q3-1.5IQR and the maximum is Q3+1.5IQR. Data points that are smaller than the minimum or larger than the maximum are outliners. They are plot as individual points. Boxplot gives an intuitive view of the data distribution without the influence of outliners[33].

From the boxplot we can see that the majority of the two classes are distributed at different levels, the Q3 of normal flows duration is nearly equal to the Q1 of malicious flows, this gives us a good indicator of pattern differences.

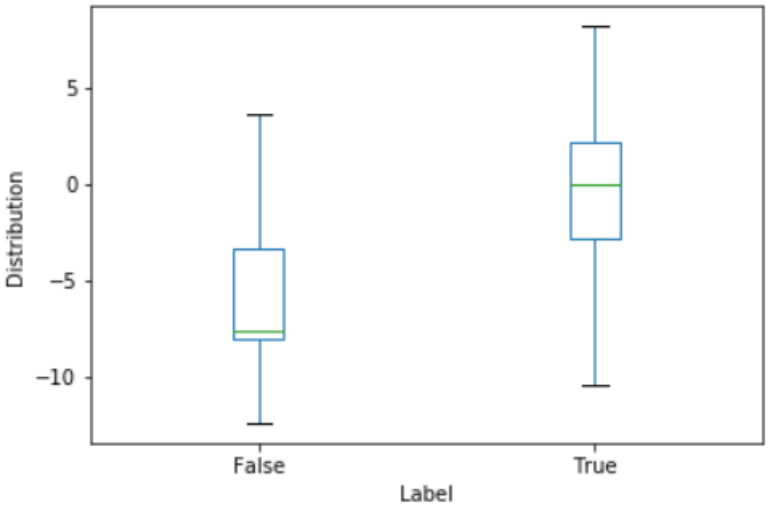


Figure 17 Log Duration box-plot

Apart from the boxplot, we also have the kdeplot of log-duration in Figure 18, which is the kernel density estimation figure. Kernel density estimation (KDE) uses a non-parametric way to estimate the probability density function of the target data sample. Equation (15) is the kernel density estimator, where *K* is the kernel function that integrates to one and *h* is the bandwidth. Like histogram, KDE can also describe the data distribution, but KDE is much more smooth and easier to spot the differences[35].

(15) $\hat{f}(y) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{y-x_i}{h})$

From Figure 18, normal flows tend to have shorter log-duration than malicious flows.

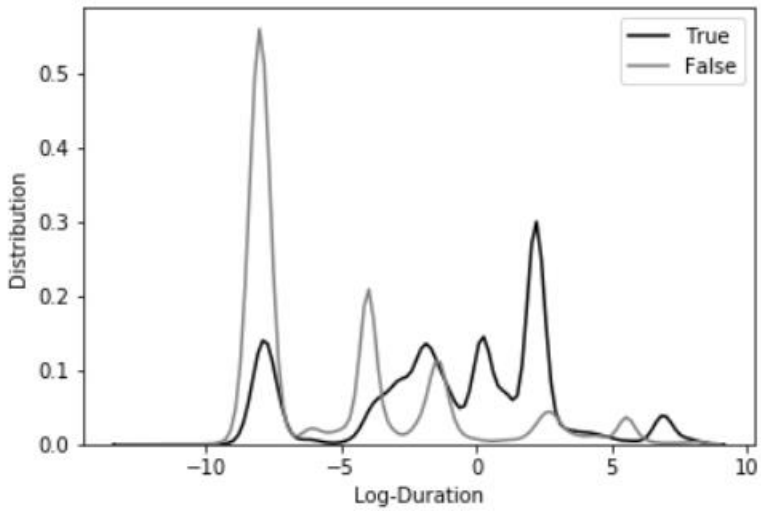


Figure 18 Log Duration distribution

4.2.4 Total packets

The reason why we kept all the fields related to flow size: *TotPkts*, *TotBytes* and *TotAppByte* at first is because we were uncertain which best represents the flow size. To analyse their relationship, we drew the scatter plots of them, the results are in Figure 19. The left figure shows a clear linear correlation between *TotBytes* and *TotAppByte*, while the right figure shows that *TotPkts* and *TotAppByte* are roughly Linearly dependent. Due to the various types of packet format sent through different protocols, there may be slight differences between the length of information one packet contains. As a result, the three fields basically share the same information, and we will use the field *TotPkts* for feature analysing.

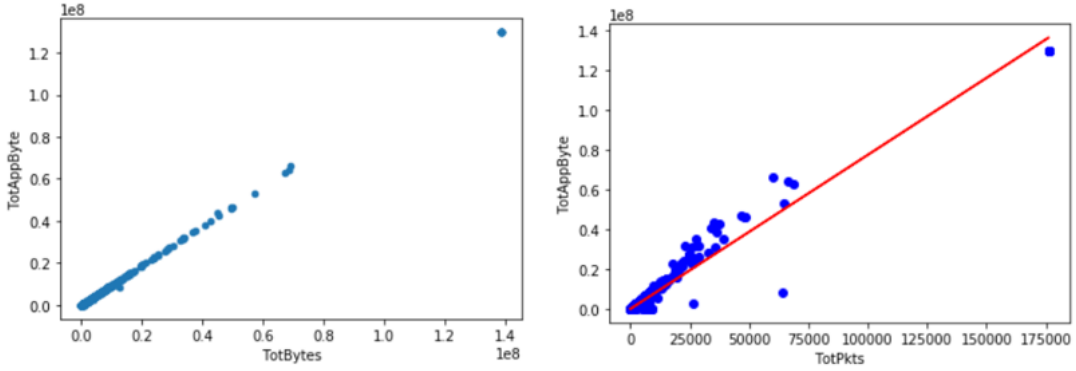


Figure 19 TotBytes-TotAppByte-TotPkts

Similar to the duration, we plot the original distribution in Figure 20, but it is hard to tell any difference, therefore we did the log conversion as well. Figure 21 is the Log-TotPkts boxplot. The medians of the two classes are similar to each other. But their interquartile ranges (IQR) do have a difference, this can be taken into consideration as a feature for model fitting.

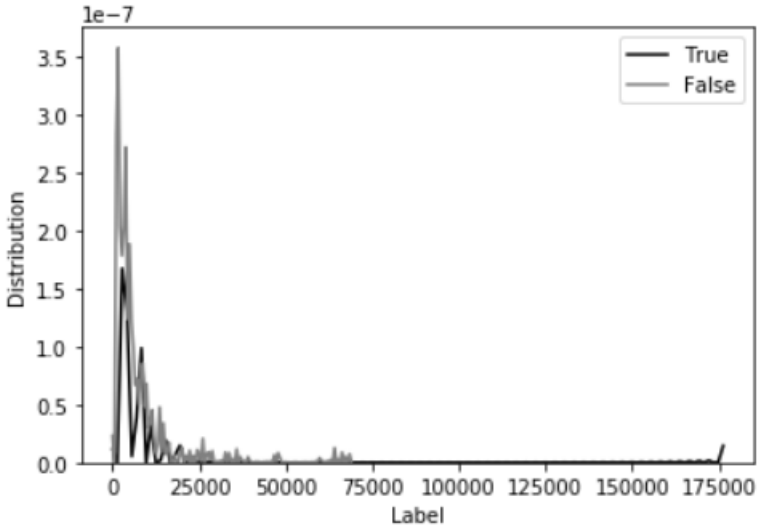


Figure 20 Total packets distribution

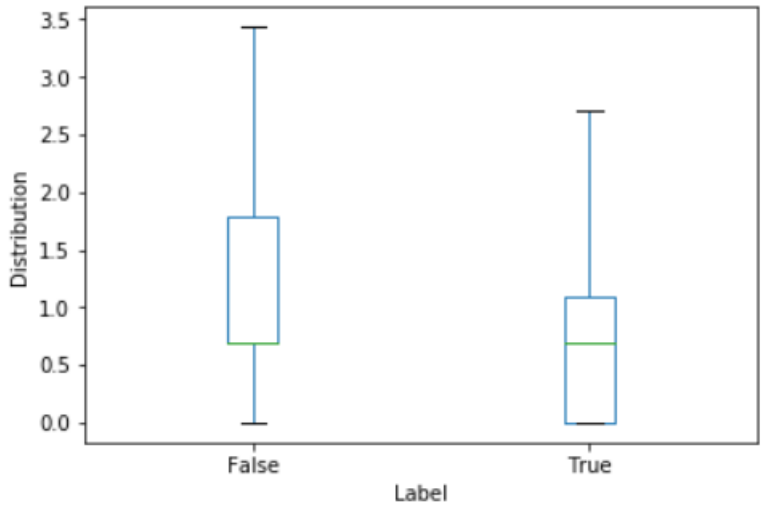


Figure 21 Log Total packets box-plot

4.3 Feature extraction

4.3.1 Time-window based data

The nature of malware is to produce abnormal traffic that can attack the vulnerable parts of the network, most of the attacks consist of thousands of repetitive flows sent in a short time, like DDoS, SPAM, PortScan, etc. This time difference character can be evaluated by counting the number of flows sent in a specific time slot, called time-window.

In this paper, we mainly analyse the malicious behaviour based on the source host (meaning same label), so if we want to use the time difference, this should be the difference between the flows sent from the same source host. First, we sort the dataset in the order of *SrcAddr* and *StartTime*. Then, based on each source host, we calculated the time difference between the current flow and the first flow sent by this source host, and this value is stored in a new column named *Delta_time.*

For the simplicity of model input, we will aggregate the flows in one time slot, so each flow needs a time-window number, flows in the same time slot have the same time-window number. An easy and quick way to calculate this number is to do the floor division of *Delta_time* using the time slot, the default time slot we use is 5 seconds. A floor division returns the integer part of the quotient, discarding any fractional result, this gives the same time-window number to flows that sent in 5 seconds. For example, flows that have time difference 2s and 4s will be allocated the same number 0 and time difference 5s will be allocated 1.

After calculating all the time-window number for each source hosts, these number will be stored in a new column named *Time_window* for aggregation and feature extraction.

The aggregated dataset has 113601 rows, with 66748 rows of label “False” and 46853 rows of label “True”, so the data basically follows a uniform distribution and has no skewed data problem.

4.3.2 Features

Since we are going to analyse the flows behaviour based on the time slot, we will need to aggregate data and extract some evident features that can represent the behaviour difference. According to the malware behaviours introduced in section 2.2 and the Exploratory Data Analysis in section 4.2, we chose the following fields as features.

- ***Proto***
  For protocol, section 4.2.1 and Figure 13 reveals that the number of flows using TCP&ICMP and those using UDP shows different patterns between the normal and infected hosts, so we will count the occurrence of protocol “TCP&ICMP” and “UDP” in one time slot as two features: *Proto_count_1* and *Proto_count_2.*
- ***DstAddr***
  Some attacks particularly target one destination host so it is necessary to count the occurrence frequency of a certain IP address and how many different destinations are reached during one time slot. This leads to the following two features: *Dstaddr_top_value_count* and *Dstaddr_category_count*.
- ***Dport***
  As explained in section 2.2.4 and 4.2.2, some ports are very likely related to malicious behaviour, because normal traffic seldom sent through these ports, we chose port numbered 25, 22, 6667, 65500, 135, 179, 3389 and 0 as the most suspicious ports and count the occurrence of them in one time slot, this creates a new feature named *Dport_count.*
  The reason to count the category of ports appeared in one slot is the same as destination address: identify any continuous attack. *Dport_category_count.*
- ***Dur***
  As the duration itself already gives us some insight into the potential patterns, we decided to keep the basic information of it; we used the mean and the standard deviation of duration as two features: *Duration_mean* and *Duration_std*.
- ***TotPkts***
  Similar to *Dur*, record the mean and the standard deviation of total packets: *Total_packet_mean* and *Total_packet_std.*
- ***Time_window***
  Count the flow number in one slot: *Flow_number*.

4.4 Model building

According to section 3.4, the following three classification models will be built to evaluate their performance: logistic regression, decision tree and random forest. The final features are listed in section 4.3.2. We mainly use the Jupyter notebook and library Scikitlearn[11] to evaluate these models. Scikitlearn is a powerful machine learning library that had many algorithms for regression, classification, clustering, etc.

4.4.1 Training and test set split

According to section 3.3, the concatenated and aggregated dataset will be split into training and test set in two methods: the first is randomly split them using the Scikitlearn method: *sklearn.model_selection.train_test_split*, we set a 70 percent training data and a 30 percent test data. The second is split them by scenario: training set is scenario 1-3, which have 51009 rows of data and the rest is test set; when evaluating the model, we will test both on the 10 test scenarios as a whole and separately on each scenario.

4.4.2 Logistic regression

In Scikitlearn, the class for building a logistic regression classifier is the *sklearn.linear_model.LogisticRegression()*. There are many parameters provided that can be tuned to get a better model. Four example, "*penalty*" can be used to specify the norm used in the penalisation, like L1 and L2 penalty; "*random_state*" can fix the seed when shuffling the data. After trying out some of the parameters, the default setting had the best result. Therefore, in the logistic regression model, we only used the "*random_state*" to fix the shuffled data.

After defining the model, we can apply the methods provided in Scikitlearn to fit and predict the labels we want. Part of the methods are in Table 6. These are four commonly used methods in scikitlearn machine learning models, including the decision tree and the random forest we are going to introduce later.

Here in logistic regression, the *predict_proba (X)* method returns the probability of being classified to a certain class, this is according to the logistic regression linear property. And the return of *predict (X)* is based on the *predict_proba (X)* value and the threshold. The default threshold for logistic regression is 0.5. In order to get better results, we use the Youden index mentioned in equation (14) to calculate the optimal threshold and apply it to the test set.

Table 6 Common methods for classification model[28]

fit (X, y[, sample_weight])	Fit the model according to the given training data.
predict (X)	Predict class labels for samples in X.
predict_proba (X)	Probability estimates.
score (X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.

4.4.3 Decision tree

Sklearn.tree.DecisionTreeClassifier() is the class for decision tree model. Important parameter includes "*criterion*": choose gini for the Gini impurity and entropy for the information gain; "*max_depth*": control the tree depth; "*min_samples_split*": control the number of samples required to split an internal node; "*random_state*": control the seed

used by the random number generator. In this project, we set the parameter as follows: *random_state=0, min_samples_split=5*. This class can also return the feature importance.

The methods for decision tree are basically the same as the logistic regression, note that although trees predict the class by the majority class in the leaf, the algorithm do return of a *predict_proba (X)* as the proportion of samples of the same class in the leaf, we can use this proportion to draw a ROC curve and find an optimal threshold as well.

4.4.4 Random forest

Random forest class is in the ensemble machine learning module library: *sklearn.ensemble.RandomForestClassifier()*. Random forest shares most of the parameters with decision tree, except that random forest can decide the number of trees used for modelling: "*n_estimators*"; it can also use the out-of-bag training data to do the validation and calculate the generalisation accuracy: "*oob_score*", which is similar to K-fold cross-validation.

In the experiment, the following parameters are chosen for they returned better results than the default setting: *n_estimators=20, random_state=0, max_features=5 and min_samples_split=5*. Same as decision tree, we can draw the ROC curve with the return of *predict_proba (X)* to find an optimal threshold.

4.5 Model Evaluation

There is a specific module "*sklearn.metrics:Metrics*" in Scikitlearn for evaluating the modules. It includes score functions, performance metrics, pairwise metrics and distance computations for regression, classification, clustering, etc[11].

According to section 3.5.2, Table 7 is the metrical methods we used in this project. Note that all the evaluation was performed on the test dataset.

Table 7 Commonly used classification metrics

metrics.accuracy_score(y_true,y_pred[, …])	Accuracy classification score.
metrics.auc(x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.confusion_matrix(y_true, y_pred[, …])	Compute confusion matrix to evaluate the accuracy of a classification
metrics.precision_score(y_true, y_pred[, …])	Compute the precision
metrics.recall_score(y_true, y_pred[, …])	Compute the recall
metrics.roc_curve(y_true, y_score[, …])	Compute Receiver operating characteristic (ROC)
metrics.f1_score(y_true, y_pred[, labels, …])	Compute the F1 score, also known as balanced F-score or F-measure

Chapter 5

Results and Evaluation

This chapter is about the results of each model and the analysis based on the results. Section 5.1 records the results of different models and different training /test data split methods. Then we compare the three models results and try to find one best model in section 5.2. Finally, section 5.3 is the model application part by applying the best model on the unlabelled background data to see how it responds.

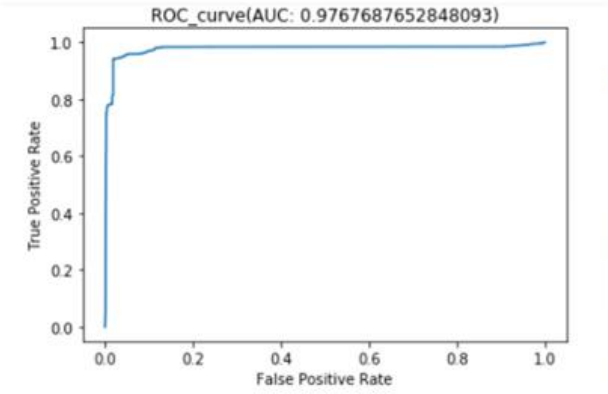
5.1 Model evaluation results

As stated in section 4.4, the dataset is split into training set and test set in two ways to evaluate the models, one is randomly split, and the other is scenario-based split. Therefore, we will have two sets of training data to train the model and three sets of test data. With each model, three evaluation results are provided, one for the randomly split test dataset, one for the test set of scenario 4 to 13 as a whole and one for the test set of each scenario individually with the optimal threshold, note this threshold is calculated using the whole scenario 4 to 13 as test set. We are going to compare the outcomes and see how the models perform respectively.

5.1.1 Logistic regression

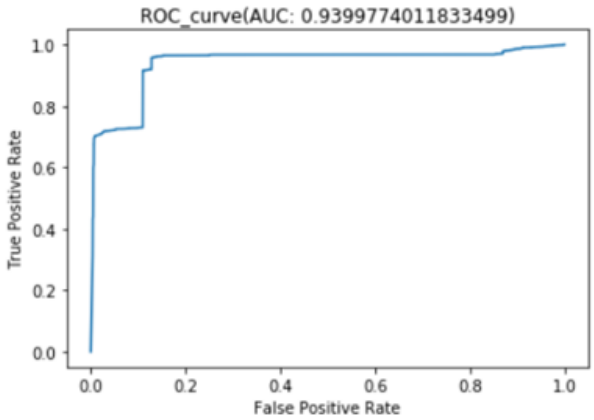
Figure 22 is the performance of the randomly split set logistic regression model, the performance results include the roc curve and the corresponding AUC, the calculated optimal threshold and its the confusion matrix, along with the accuracy, the recall and the precision of the default and the optimal threshold.

Figure 23 is the performance of scenario-based split set logistic regression model and Table 8 is the performance of each scenario.




```
Optimal threshold: 0.3749412683983224
AUC:0.977
Confusion_matrix is:
 [[19805   129]
 [ 3122 11025]]
Accuracy with default threshold:0.905
Accuracy with optimal threshold:0.965
Precision with default threshold:0.988
Precision with optimal threshold:0.973
Recall with default threshold:0.779
Recall with optimal threshold:0.941
F1 score with default threshold:0.872
F1 score with optimal threshold:0.957
```

Figure 22 Logistic regression performance (Randomly split)




```
Optimal threshold: 0.01501470713760896
AUC:0.940
Confusion_matrix is:
 [[29988   250]
 [ 9702 22652]]
Accuracy with default threshold:0.841
Accuracy with optimal threshold:0.916
Precision with default threshold:0.989
Precision with optimal threshold:0.889
Recall with default threshold:0.700
Recall with optimal threshold:0.958
F1 score with default threshold:0.820
F1 score with optimal threshold:0.922
```

Figure 23 Logistic regression performance (Scenario split)

As can be seen, the performance of the model built on the randomly split training set is pretty good compared to that built on the first three scenarios. In practice, a classifier will not be this good without tuning, so there must be something wrong with the first way of splitting the data randomly, we will explain more of this in section 5.2.1. When using the second way to test the models, the results are more reasonable. We can see that after changing the threshold from 0.5 to the optimal threshold 0.015, the accuracy and the recall improved a bit but the precision decreased, more observations are identified as True now.

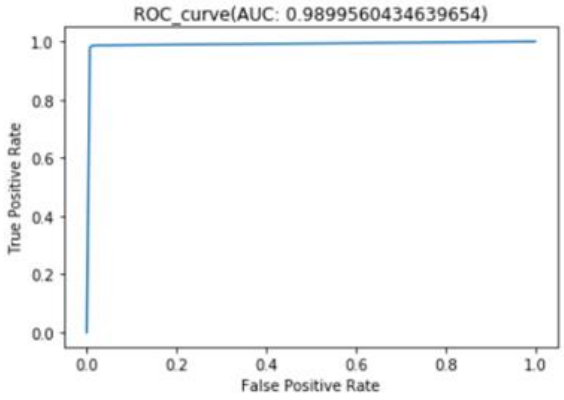
We use this model with optimal threshold to evaluate the performance of test set scenario 4 to 13 individually, the results are in Table 8. The “size” column is the number of input time-window and the “True number” column is the number of *True* time-window inputs (label as *True*) in this scenario. It shows that on scenarios with imbalanced classes, like scenario 4 and 7, where only less than 10 percent of the data is in the “True” class, the logistic regression model does not have a good performance as those with more balanced classes like scenario 13. Particularly on scenario 7, the recall is 0.4 and the precision is only 0.09, which means the model mistakenly classify many normal observations to *True*. This may due to the 5-second time slot is too big for a short period captured in scenario 7, result in not have efficient information indicating any malware pattern in one time-window. Also, the malware in scenario 7 uses the HTTP protocol, which the model may not be able to learn to identify from the first three scenarios.

Table 8 Performance on single scenario (Logistic regression)

	Scenario	Accuracy	Recall	Precision	F1 score	Size	'True' number
0	4	0.806231	0.918919	0.339623	0.495948	3210	333
1	5	0.869640	0.960976	0.743396	0.838298	583	205
2	6	0.927134	0.995745	0.842737	0.912874	1839	705
3	7	0.801843	0.400000	0.097561	0.156863	217	10
4	8	0.903536	0.990865	0.610798	0.755738	12357	1861
5	9	0.947039	0.980951	0.955364	0.967988	19486	15906
6	10	0.833262	0.882497	0.711396	0.787763	4660	1634
7	11	0.881633	0.765957	0.666667	0.712871	245	47
8	12	0.709709	0.408922	0.440000	0.423892	1030	269
9	13	0.945268	0.944571	0.963530	0.953957	18965	11384

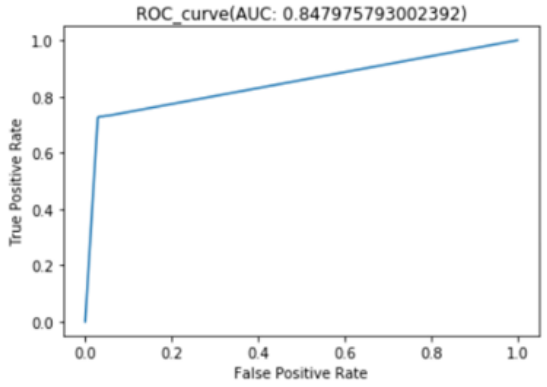
5.1.2 Decision tree

Figure 24 and Figure 25 is the corresponding performance of the decision tree model using the two different splitting methods. Same as logistic regression, randomly split dataset model still has an abnormally good performance. For the scenario-based splitting model, the optimal threshold is still 0.5, so the default threshold is already the best. The accuracy and the precision are pretty high, but the recall is not that good, many infected inputs are labelled as *False*, they are not detected by the model.




```
Optimal threshold: 0.3333333333333333
AUC:0.990
Confusion_matrix is:
 [[19738   196]
 [  233 13914]]
Accuracy with default threshold:0.988
Accuracy with optimal threshold:0.987
Precision with default threshold:0.989
Precision with optimal threshold:0.986
Recall with default threshold:0.981
Recall with optimal threshold:0.984
F1 score with default threshold:0.985
F1 score with optimal threshold:0.985
```

Figure 24 Decision tree performance (Randomly split)




```
Optimal threshold: 0.5
AUC:0.848
Confusion_matrix is:
 [[29348   890]
 [ 8874 23480]]
Accuracy with default threshold:0.844
Accuracy with optimal threshold:0.844
Precision with default threshold:0.963
Precision with optimal threshold:0.963
Recall with default threshold:0.726
Recall with optimal threshold:0.726
F1 score with default threshold:0.828
F1 score with optimal threshold:0.828
```

Figure 25 Decision tree performance (Scenario split)

Apart from all the metrics used in logistic regression, we can also list the importance of each feature to see which feature plays a more important role. Table 9 is the feature importance in this decision tree model, from it we can see that some features have extremely high importance like "Dport_count", "Duration_mean" and "Total_packet_mean" while some have barely any importance, this attribute can be used to select and reduce the features used in the model.

Table 9 Decision tree feature importance

Proto_count1	7.24E-04
Proto_count2	2.64E-03
Dstaddr_top_value_count	7.63E-03
Dstaddr_category_count	2.10E-03
Dport_count	9.50E-01
Dport_category_count	6.24E-04
Duration_mean	1.14E-02
Duration_std	1.93E-03
Total_packet_mean	2.18E-02
Total_packet_std	7.59E-04
Flow_number	7.06E-05

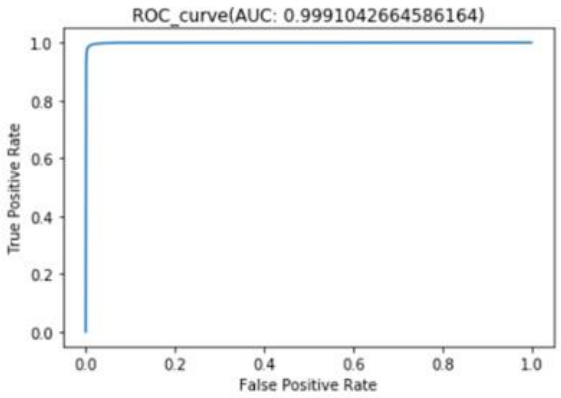
Table 10 is the performance of each test scenario. Apparently, the performance of scenario 13 is pretty bad. It only correctly predicts half of the data. The recall is only 0.29 while the precision is high. This indicates that the model is not good at telling the infected data, it mainly marks the data as normal. One possibility for that is there are too much malware performing at the same time, so in one time slot exists different patterns that are difficult for the model to tell apart.

Table 10 Performance on single scenario (Decision tree)

	Scenario	Accuracy	Recall	Precision	F1 score	Size	'True' number
0	4	0.908100	0.942943	0.532203	0.680390	3210	333
1	5	0.909091	0.804878	0.926966	0.861619	583	205
2	6	0.982599	0.987234	0.968011	0.977528	1839	705
3	7	0.953917	0.200000	0.500000	0.285714	217	10
4	8	0.971109	0.946803	0.872277	0.908013	12357	1861
5	9	0.972904	0.975795	0.990871	0.983275	19486	15906
6	10	0.940343	0.881885	0.944299	0.912025	4660	1634
7	11	0.951020	0.829787	0.906977	0.866667	245	47
8	12	0.827184	0.468401	0.782609	0.586047	1030	269
9	13	0.577063	0.299895	0.985281	0.459829	18965	11384

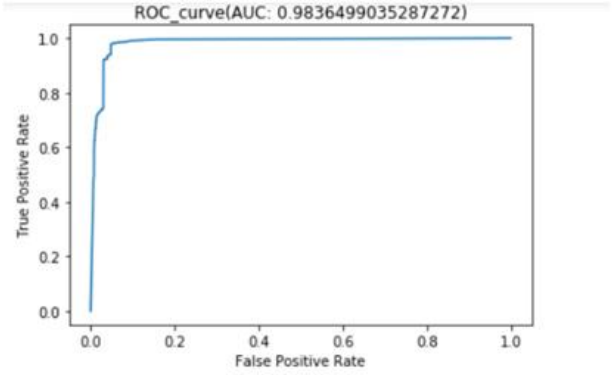
5.1.3 Random forest

Same as logistic regression and decision tree, Figure 26 gives a 99 percent high accuracy on the randomly split dataset. While on the scenario-based splitting model (Figure 27 and Table 12), with default threshold, the recall is still relatively low, so we calculated the optimal threshold for the random forest. It optimises the accuracy to 0.916, the recall to 0.941 and the precision to 0.954, which is a pretty good result.




```
Optimal threshold: 0.2770833333333333
AUC:0.999
Confusion_matrix is:
 [[19711   223]
 [  114 14033]]
Accuracy with default threshold:0.990
Accuracy with optimal threshold:0.990
Precision with default threshold:0.991
Precision with optimal threshold:0.984
Recall with default threshold:0.985
Recall with optimal threshold:0.992
F1 score with default threshold:0.988
F1 score with optimal threshold:0.988
```

Figure 26 Random forest (Randomly split)




```
Optimal threshold: 0.21444444444444444
AUC:0.984
Confusion_matrix is:
 [[28782  1456]
 [ 1914 30440]]
Accuracy with default threshold:0.850
Accuracy with optimal threshold:0.946
Precision with default threshold:0.975
Precision with optimal threshold:0.954
Recall with default threshold:0.729
Recall with optimal threshold:0.941
F1 score with default threshold:0.834
F1 score with optimal threshold:0.948
```

Figure 27 Random forest (Scenario split)

Table 11 shows the feature importance of random forest. Compared with the decision tree, the importance is much more balanced, but we can still see that features "Proto_nor_count", "Dport_count" and "Duration_mean" are more important than others.

Table 11 Random forest feature importance

Proto_count1	6.38E-03
Proto_count2	2.11E-01
Dstaddr_top_value_count	1.50E-01
Dstaddr_category_count	1.82E-02
Dport_count	3.47E-01
Dport_category_count	3.13E-03
Duration_mean	2.02E-01
Duration_std	1.01E-02
Total_packet_mean	1.72E-02
Total_packet_std	3.42E-02
Flow_number	7.33E-04

Table 12 is the single scenario performance using the random forest model with the optimal threshold. Compared with the decision tree and logistic regression, some scenarios have significant improvement. For example, scenario 7 and 13, all their performances are improved. Although for the scenario 7, the precision is still not that good probably due to the reason explained in section 5.1.1.

Table 12 Performance on single scenario (Random forest)

	Scenario	Accuracy	Recall	Precision	F1 score	Size	'True' number
0	4	0.898442	0.993994	0.505344	0.670040	3210	333
1	5	0.948542	0.946341	0.910798	0.928230	583	205
2	6	0.973355	0.988652	0.944444	0.966043	1839	705
3	7	0.944700	0.600000	0.428571	0.500000	217	10
4	8	0.956624	0.965073	0.792236	0.870155	12357	1861
5	9	0.976188	0.986420	0.984440	0.985429	19486	15906
6	10	0.935837	0.954100	0.874369	0.912496	4660	1634
7	11	0.938776	0.893617	0.807692	0.848485	245	47
8	12	0.882524	0.754647	0.786822	0.770398	1030	269
9	13	0.919958	0.871574	0.994388	0.928939	18965	11384

5.2 Model comparison and analyse

5.2.1 Models with different dataset split method

As the evaluation results shown in section 5.1, models built with the randomly split training set and test with the randomly split test set all have the "perfect" performance, which is abnormally high for first built models. Therefore, something must be wrong with the data.

From the definition of the training set and test set explained in section 3.3, the test set must be independent of the training set. The raw dataset is aggregated by time-window, which means if the malware performed an attack longer than one time slot, then this attack pattern will be split into two time slots, and based on the randomly split algorithm, there are chances that one will be in training set and the other in the test set. This results in the independence of training set and test set; therefore, the evaluation results of these models cannot be generalised to other malware.

While with the second method, we tested the model using malware that do not appear in the training set to evaluate its generalisation ability, and the performance turned out to be more reasonable. Therefore, we will compare the performance of models using the second method to find the best model.

5.2.2 Best model

Table 13 is the performance of scenarios 4 to13 together on each model trained with the first 3 scenarios. For all the models, we evaluated the performance using both the default and the optimal threshold. In case of the conflict of comparing recall and precision, we use column F1_score alternatively. Apparently, the optimal models have better outcomes. The optimal random forest model has the best accuracy, but its recall and precision are not the best, so we need the F1 score to make a choice. Since the optimal random forest has the highest accuracy and F1 score, it is the best model that fit our project goal. We will use the optimal random forest models for further application.

Table 13 Performance of 10 scenarios together

	Accuracy	Recall	Precision	F1_score
LGR(default threshold)	0.841	0.7	0.989	0.82
LGR(optimal threshold)	0.916	0.958	0.889	0.922
DT(default threshold)	0.844	0.726	0.963	0.828
DT(optimal threshold)	0.844	0.726	0.963	0.828
RF(default threshold)	0.85	0.729	0.975	0.834
RF(optimal threshold)	0.946	0.941	0.954	0.948

In this project, the features we chose are more like non-linear related. Back to logistic regression, it is a linear model so it will not perform well if the features are not linearly dependent. However, the experiment turns out that the non-linear model decision tree also cannot provide a better prediction. Random forest is based on the decision tree, it is also a non-linear model and helps prevent overfitting and has a good generalisation ability to detect malware that has not seen before, the experiment also gives good evidence to prove it. We can draw a conclusion that random forest indeed is a good model for malware detecting.

5.3 Model application

Due to the limitation of time and computer storage capacity, we will only choose one scenario's background traffic for analysis. We applied the same data cleaning and feature extraction procedures to the background traffic of scenario 5, except that these input observations have no labels. After predicting the labels of all the observations, we calculate the corresponding probability as a malware identification metric.

Table 14 is part of the comparison result for the source address in the background traffic. The column "number" is the number of all the observations from a certain address and the column "count" is the times that observation from the certain source address being marked as True (infected) by our model. Then we calculate the malware infected proportion using the "count" divide by "number" and store this in the column "prop".

Table 14 Application label results

	Src_addr	number	count	prop
0	147.32.84.229	362	357	0.986188
1	147.32.84.59	362	183	0.505525
2	147.32.84.138	361	4	0.011080
3	147.32.85.89	361	2	0.005540

For example, we can see that 357 out of 362 observations from address 147.32.84.229 are marked as *True*, which is a pretty high proportion so it has a great risk of being infected. While address 147.32.84.89, it only has 2 out of 361 marked as *True*, so its risk of infection is relatively low. However, we do not have the exact labels of these observations so the results may not be that reliable and further network analysis is needed.

Chapter 6

Conclusions

This is the final chapter summarising the whole project by looking back on what we have done in section 6.1. Then we will analyse the goal accomplishment in section 6.2 followed by some improvement points in section 6.3.

6.1 Review

In this paper, we realised a data analytical project with the O'Neil & Schutt data science process. A classifier was built based on the NetFlow traffic of CTU-13 dataset to analyse the malware behaviour and identify the infected hosts.

We first introduced the background of network, Cybersecurity and machine learning related knowledge in chapter 1, following the proposal of this project and some basic implementation methods. In chapter 2, there are some conceptual introduction of the dataset used and the characteristic of the malware and the Internet. As for the machine learning methods, chapter 3 describes the whole process of applying a machine learning model, including the splitting of data, model fitting and model choosing. Chapter 4 and 5 are the practical application and analysis of the whole process. After cleaning and extracting features from the raw dataset, the features were used to fit in the three chosen classification model: logistic regression, decision tree and random forest. Under the classifier evaluation metrics, we selected the best model: the optimal threshold random forest model, and applied it to the background data to detect malware. And the final chapter 6 is for summary and future investigating.

6.2 Goal accomplishment

As stated in section 1.2, the goal of this project is to study the network behaviour and extract features of normal and malicious behaviour from the NetFlow traffic, then apply machine learning knowledge to build a classification model that can identify infected hosts. Generally, this can be separated into three parts: study the network behaviour, feature engineering and classification model building and application.

For the network behaviour, we studied the OSI and TCP/IP model for Internet structure in Chapter 2. Some of the most common malware like SPAM, DDoS and IRC are also analysed. However, these are the prior learned knowledge, not from the model's prediction. The application of the background traffic needs further analysis to see if there are any special patterns behaviours. Due to the limitation of the project time, this part can be conduct in the future work.

To extract useful features for the model building, we analysed the raw data, combining with the network background knowledge, eleven features are selected. These features are

based on the basic NetFlow information like IP address and protocol that can best indicate the differences between the normal and the infected. In section 5.1, we had the feature importance returned from the model, apparently some features do not play an important part and should be removed, so there is still room for the feature improvement.

The final model we built is the random forest classification algorithm, with an optimal threshold, it can reach an accuracy of 0.946, a recall of 0.941 and a precision of 0.954 on test set of scenario 4 to 13. This model also performs better than logistic regression and decision tree on each scenario individually. And for the application, the model does identify some malicious behaviour, but we are not sure if that is truly a malware or not, further analysis is needed. One thing this model did not accomplish is the identification of attacking behaviour and normally behaviour, this may need to change the model we use.

6.3 Future work

Although we successfully built a classifier and identified some malicious pattern, there are still many works we can do to improve this classifier.

First is the feature selection, according to the feature importance, some modification can be done to choose features that better represent the behaviour pattern. Another improvement is the parameter tuning, in this project, we just randomly chose the parameters based on the background knowledge and the outcome, but more formal and professional way like grid search should be conduct alternatively. We can also try other classification models like SVM (Support Vector Machine) to improve the result. Finally, in order to test the model, we can set up a similar environment performing malware behaviour to evaluate the model and apply in practise.

References

[1] The CTU-13 Dataset. A Labeled Dataset with Botnet, Normal and Background traffic. (n.d.). Retrieved August 2, 2018, from http://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html

[2] A. Klimburg (Ed.). 2012 National cyber security framework manual, NATO CCD COE Publications

[3] Leiba, B., "Update to Internet Message Format to Allow Group Syntax in the "From:" and "Sender:" Header Fields", RFC 6854, DOI 10.17487/RFC6854, March 2013, <https://www.rfc-editor.org/info/rfc6854>.

[4] Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC 2821, DOI 10.17487/RFC2821, April 2001, <https://www.rfc-editor.org/info/rfc2821>.

[5] Erel, I., Stern, L. H., Tan, C., & Weisbach, M. S. (2018). Selecting Directors Using Machine Learning (Working Paper No. 24435). National Bureau of Economic Research.

[6] Lindqvist, N., & Price, T. (2018). Evaluation of Feature Selection Methods for Machine Learning Classification of Breast Cancer (Dissertation)

[7] Schutt, R., & O'Neil, C. (2013). Doing Data Science: Straight Talk from the Frontline. O'Reilly Media, Inc.

[8] :: Anaconda Cloud. (n.d.). Retrieved August 23, 2018, from https://anaconda.org/

[9] Matplotlib: Python plotting — Matplotlib 2.2.3 documentation. (n.d.). Retrieved August 23, 2018, from https://matplotlib.org/

[10] NumPy — NumPy. (n.d.). Retrieved August 23, 2018, from http://www.numpy.org/

[11] scikit-learn: machine learning in Python — scikit-learn 0.19.2 documentation. (n.d.). Retrieved August 23, 2018, from http://scikit-learn.org/stable/

[12] seaborn: statistical data visualization — seaborn 0.9.0 documentation. (n.d.). Retrieved August 23, 2018, from http://seaborn.pydata.org/#

[13] Hofstede, R., Čeleda, P., Trammell, B., Drago, I., Sadre, R., Sperotto, A., & Pras, A. (2014). Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. IEEE Communications Surveys Tutorials, 16(4), 2037–2064. https://doi.org/10.1109/COMST.2014.2321898

[14] Argus - NSMWiki. (n.d.). Retrieved August 2, 2018, from http://nsmwiki.org/Argus#Introduction

[15] Velan, P., Medkova, J., Jirsik, T., & Celeda, P. (2016). Network traffic characterisation using flow-based statistics. Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP, 907-912.

[16] Alani, M. M. (2014). TCP/IP Model. In Guide to OSI and TCP/IP Models (pp. 19–50). Springer, Cham. https://doi.org/10.1007/978-3-319-05152-9_3

[17] Klensin, J., "SMTP 521 and 556 Reply Codes", RFC 7504, DOI 10.17487/RFC7504, June 2015, <https://www.rfc-editor.org/info/rfc7504>.

[18] What is a Botnet? - Definition from Techopedia. (n.d.). Retrieved August 6, 2018, from https://www.techopedia.com/definition/384/botnet

[19] Oikarinen, J. and D. Reed, "Internet Relay Chat Protocol", RFC 1459, DOI 10.17487/RFC1459, May 1993, <https://www.rfc-editor.org/info/rfc1459>.

[20] Meyer, R., & Cukier, M. (2006). Assessing the Attack Threat due to IRC Channels. In International Conference on Dependable Systems and Networks (DSN'06) (pp. 467–472). https://doi.org/10.1109/DSN.2006.12

[21] Wang, Wallace (2004-10-25). "Instant Messaging and Online Chat Rooms: Downloading files from IRC". Steal this File Sharing Book (1st ed.). San Francisco, California: No Starch Press. p. 66. ISBN 1-59327-050-X.

[22] Hartmann, R., "Default Port for Internet Relay Chat (IRC) via TLS/SSL", RFC 7194, DOI 10.17487/RFC7194, August 2014, <https://www.rfc-editor.org/info/rfc7194>.

[23] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <https://www.rfc-editor.org/info/rfc6335>.

[24] Murphy, K. P. (2012). Machine Learning : A Probabilistic Perspective. Cambridge, Mass: The MIT Press.

[25] James G., Witten D., Hastie T., Tibshirani R. (2013) Statistical Learning. In: An Introduction to Statistical Learning. Springer Texts in Statistics, vol 103. Springer, New York, NY

[26] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning : Data mining, inference, and prediction (Second ed., Springer series in statistics). New York: Springer.

[27] Taylor, J. R. (1997). Introduction To Error Analysis: The Study of Uncertainties in Physical Measurements. University Science Books.

[28] sklearn.linear_model.LogisticRegression — scikit-learn 0.19.2 documentation. (n.d.). Retrieved August 18, 2018, from http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[29] Moore, A., Crogan, M., Moore, A. W., Mary, Q., Zuev, D., Zuev, D., & Crogan, M. L. (2005). Discriminators for use in flow-based classification.

[30] Plummer, D., "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", STD 37, RFC 826, DOI 10.17487/RFC0826, November

1982, <https://www.rfc-editor.org/info/rfc826>.

[31] Python Data Analysis Library — pandas: Python Data Analysis Library. (n.d.). Retrieved August 12, 2018, from http://pandas.pydata.org/

[32] Mitchell, B. (n.d.). Why Is the Port 0 Special on TCP and UDP Networks? Retrieved August 12, 2018, from https://www.lifewire.com/port-0-in-tcp-and-udp-818145

[33] Williamson, D., Parker, R., & Kendrick, J. (1989). The box plot: A simple visual method to interpret data. Annals of Internal Medicine, 110(11), 916-21.

[34] Rosenblatt, M. (1956). Remarks on Some Nonparametric Estimates of a Density Function. The Annals of Mathematical Statistics, 27(3), 832–837. https://doi.org/10.1214/aoms/1177728190

[35] Rosenblatt, M. (1956). Remarks on Some Nonparametric Estimates of a Density Function. The Annals of Mathematical Statistics, 27(3), 832–837. https://doi.org/10.1214/aoms/1177728190