Final Report
CM3203 - One Semester Individual Project (40 Credits)

Title:

Malware Detection from IoT Devices Using Machine Learning algorithms

Author: Bedoor Bahassan

Supervised by: Amir Javed

Moderated by: Moderator: Dr Daniel J. Finnegan

School of Computer Science and Informatics Cardiff University

A Final Year Project Submitted for the Degree of

BSc Computer Science with Security and Forensics.

May 2020

Abstract:

Nowadays, the Internet of things (IoT) refers to the billions of devices, which are connected into networks to deliver advanced and intelligent services are spreading between government, industry, academic, and private parties. With Machine Learning techniques taking centre stage in today's computer technology, significant efforts are made to integrate machine learning into the art and science of malware detection. Since Machin Learning approaches are effective technique and easy to implement, they are used on the purpose of reducing malware activities.

The main purpose of this project is to classify malware network traffic of IoT devices from a given dataset into either malicious or benign and to investigate the performances of different Machine Learning classification algorithms with evaluation metric to identify the best machine learning techniques.

The model is trained with a given data set, depends on the most popular Machine Learning classification algorithms that have successfully implemented in malware problems; which are, Random Forest, Support Vector Machine in three different variations and Logistic Regression. Feature Selection was maintained throughout the project, and the whole project was implemented using Python programming language.

Acknowledgements

First, Alhamdulillah, I praise almighty Allah for keeping in a good health that was necessary for completing this project during an unexpected time and for his guidance throughout my life.

I would like to take this opportunity to express my gratitude to the individuals who have supported me throughout the whole project.

A massive thank you to my supervisor *Amir Javed* for his invaluable support and continuous encouragement to keep me in the right direction throughout the whole project.

I will always be thankful to my parents for their endless support and for believing in my abilities to let me pursue my dream. Another thanks to my brother and sister, *Badr* and *Bedar* whose love and support have always sustained.

I would also like to thank all of my friends for the amazing memory we shared and for making me feel home away from my home.

Lastly, I would like to thank my sponsor company *Saudi Aramco* for giving me this opportunity that helps me achieve my goals.

Table of Contents

Chapter 1: Introduction 9

1.1 Preface 9

1.2 Project Aim and Scope 10

1.3 Intended Audience 10

1.4 Report Structure 11

Chapter 2: Background Research 12

2.1 Artificial Intelligence and Machine Learning 12

2.1.1 Supervised vs Unsupervised learning 14

2.2 Existing Solutions 15

2.3 Machine Learning approach to malware detection 16

2.3.1 Random Forest Classifier 16

2.3.2 Support Vector Machines 17

2.3.3Logistic Regression: 18

2.4 Learning Evaluation 19

2.4.1 Confusion Matrix 19

2.4.2 performance Indicator: 20

2.5 Python Machine Learning Libraries: 22

2.6 Research Questions 23

Chapter 3: Specification and Design 24

3.1 Software Requirement Specification 24

3.1.2 Functional Requirements 24

3.1.3 Non-Functional Requirements 25

3.2 System Plan: 26

3.3 System Design: 26

3.4 Development Methodology: 28

Chapter 4 Implementation: 30

4.1 Project Structure 30

4.2 Dataset used **30**

4.3 Data Preparation & Processing **31**

4.3.1 Data ingestion 31

4.3.2 Data description 32

4.3.2 Exploratory Data Analysis 33

4.3.3 Data cleaning and transformation 40

4.3.4 Train and Test Split 42

4.3.5 Feature Importance and Feature Selection 43

4.3.6 Training and Evaluating the Model: 44

4.4 Classifiers **46**

4.4.1 Implementing Random Forest Classifier 46

4.4.2 Implementing Support Vector Machines Classifier 47

4.4.3 Implementing Logistic Regression Classifier 47

Chapter 5: System Testing **49**

5.1 Test case **49**

Chapter 6: Results and Discussion **56**

6.1 Classifier Performance **56**

6.1.1 Variable importance assessment 56

6.1.2 Accuracy of the classifiers 59

6.1.3 Precision of the classifiers 59

6.1.4 Recall value of the classifiers 60

6.1.5 F1-Score of the classifiers 61

6.1.6 AUC of the classifiers 62

6.1.7 Classifiers Performance Summary 63

6.2 Limitations **64**

Chapter 7: Future Work **65**

Chapter 8: Conclusion **66**

Chapter 9: Reflection on Learning **67**

Bibliography **68**

List of Figures

Figure 2.1: Machine Learning Flow – 12

Figure 2.2: Process of Classification Model -13

Figure 2.3: Working model of a supervised learning – 14

Figure 2.4: Voting mechanism in Random Forest algorithm - 16

Figure 2.5: Support Vector Machine – 17

Figure 2.6: Shape of sigmoid function -18

Figure 2.7: Confusion matrix for binary classifier -19

Figure 2.8: The area under the ROC curve (shown in grey) -21

Figure 3.1: User Case Diagram- 26

Figure 3.2: Activity Diagram of Software System – 27

Figure 4.1: Reading and Converting Bro log File to Pandas Data Frame – 30

Figure 4.2: Data composition by traffic type (label) -34

Figure 4.3: Data type of the bro log columns -34

Figure 4.4: Distribution of duration -35

Figure 4.5: Distribution of original byte (orig_bytes) – 36

Figure 4.6: Distribution of respond bytes (resp_bytes) -36

Figure 4.7: Distribution of original packets transmitted (orig_pkts) -37

Figure 4.8: Distribution of respond packet (resp_pkts) – 37

Figure 4.9: Distribution of original IP bytes (origin_ip_bytes) – 38

Figure 4.10: Distribution of respond IP bytes (resp_ip_bytes) -38

Figure 4.11: Constituents of the label column – 40

Figure 4.12: Replacing - with 0 – 40

Figure 4.13: Label encoding to convert Benign to 0 and Malicious to 1 – 40

Figure 4.14: Conversion of continuous variables- 40

Figure 4.15 Time Conversion -41

Figure 4.17: Generating train-test and Test Set Split – 42

Figure 4.18: Feature importance using Random Forest algorithm – 42

Figure 4.19: Scikit Learn's Recursive Feature Elimination – 43

Figure 4.20: Generating performance metrics and storing in dataframe – 44

Figure 4.21: Implementation of Random Forest Classifier – 45

Figure 4.22: Implementation of SVC with linear kernel – 46

Figure 4.23: Implementation of the logistic regression classifier

Figure 5.1: Variable importance from Random Forest Classifier – 56

Figure 5.2: Recursive Feature Elimination output – 56

Figure 5.3: Comparison of accuracy scores by classifiers – 58

Figure 5.4: Comparison of precision scores by classifiers – 59

Figure 5.5: Comparison of recall score across model – 60

Figure 5.6: Comparison of F1-Score by classifiers – 61

Figure 5.7: Comparison of AUC score across models – 62

Figure 5.8: confusion matrix plots for top three model - 63

List of Tables

Table 4.1 Columns Description of all Bro Log File – 31

Table 4.2 Columns Description of used feature in Bro Log File - 33

Table 4.3: Blank Pandas Data Frame to store performance metrics – 44

Table 5.1: Loading data set (TC-0) -48

Table 5.2: Loading and parsing the data set (TC-1) – 49

Table 5.3:Exploring Data Analysis step to explore and visualize data set variables (TC-2)– 49

Table 5.4: Pre-processing the data to prepare it for the classifiers (TC-3) – 50

Table 5.5: Creating, training, testing Random Forest (TC-4) -50

Table 5.6: Creating, training, testing Support Vector Machine (Linear kernel) (TC-5) – 51

Table 5.7: Creating, training, testing Support Vector Machine (RBF kernel) (TC-6) – 51

Table 5.8: Creating, training, testing Support Vector Machine (Sigmoid kernel) (TC-7) – 52

Table 5.9: Creating, training, testing Logistic Regression (TC-8) – 52

able 5:10: Feature selection step for Random Forest (TC-9) – 53

Table 5.11: Feature Selection step for Logistic Regression (TC-10) – 53

Table 5.12: Calculating the results of all classifiers (TC-11) – 54

Table 5.1: Summary of performance metrics for all models - 62

Chapter 1: Introduction

1.1 Preface

Malware is a malicious software and is a threat to digital devices and cybersecurity. It is developed by cyber attackers to intrude or cause damage to the network system, usually without the victim's knowledge. There are multiple different types of malware include viruses, spyware, ransomware, and Trojan horses [31]. According to Norton, a leading anti-malware software developer, a malware attack is when hackers design malicious software that can be installed in victims' devices without their knowledge. The main reason for that is to obtain access to private information or to destroy the device, mainly for financial gain. Malware attacks can happen on all kinds of machines and operating systems, including Microsoft Windows, macOS, Android, and iOS [ibid]. New research from leading market analysts, Juniper Research indicates that the expenses of data breach in 2020 will overreach $150 million [23]. In 2012, Saudi Aramco oil company exposed to one of the biggest cyberattack that affected on 30,000 of its Windows based machines, the process of recovery from the harm took around two weeks [4].

Cyber-attacks is increasing year on year, therefore it is needed to avoid attacks by increasing the security of the network base and data information which could be achieved by using protected devices; such as, antivirus services, antispam devices, firewalls, IPS (Intrusion Prevention System), IDS (Intrusion Detection System). Anti-malware is software that protects digital devices and computer networks from malwares, by detecting malicious networks that are trying to interact with the devices and thereby preventing any intrusion by the malware. Anti-malware software requires a fast and accurate mechanism to detect malware in real-time. The market of Antivirus system is worth over $37 billion and achieved a high success by disclosing more than 350,000 bits of malware each day [22]. Machine Learning algorithms, with its advancement and its adoption in many aspects including computer science, has become widely popular in detecting abnormal network traffic with higher degrees of accuracy and adaptability in numerous situation and environment [33]. the development and application of machine learning techniques for malware detection will add a high security value due to its ability to maintain pace with malware evolution. Machine Learning powered antimalware tools will assist in detecting modern malware attacks and produce better scanning engines [17].

This report covers the necessary exploration of network traffic data followed by discussion of the application of various Machine Learning classification algorithm in detecting malware from amongst a pool of malicious and benign network traffic data that captured from real IoT devices.

1.2 Project Aim and Scope

The main purpose of this project is to detect malware network traffic for IoT devices from a given dataset and to investigate the performances of different Machine Learning classification algorithms with evaluation metric to identify the best machine learning techniques.

The project scope focuses on implementing and examine the performances of different Machine Learning Classification algorithms which can classify the Network traffic in training and testing set into either malicious or benign. The approach used in this project is implementing three different types of Machine learning algorithms with three different variations of one type. They are Random Forest, Support Vector Machine in three different variations (Linear kernel, RBF kernel, Sigmoid kernel) and Logistic Regression. All Machine Learning algorithms have been chosen based on their popularity and their successful used in malware detections. The data set used is a given data set captured from real IoT devices.

Since python is an essential and the most preferred language for machine learning task, All the implementation of the project will use python as programming language and will be implemented and tested on a MacOS device. More information will be disused later within implementation section.

1.3 Intended Audience

The intended audience and beneficiaries from this project are people who are interested or planning to be involved in research in the field of security application approaches build by Machine Learning algorithms especially in the field of malware detection on IoT devices. Moreover, it is beneficial for people who are interested in the application of statistical and mathematical models in security analysis.

1.4 Report Structure

This report comprises of six sections marked as chapters as follow:

• **Chapter 1:** introduces the project, its aims and objectives and the audience who may find it interesting.
• **Chapter 2:** provides a short background of Machine Learning and various Machine Learning approach utilised for malware detection and measures for evaluating Machine Learning classifiers.
• **Chapter 3:** discusses the requirements and design of the provided solution.
• **Chapter 4:** provides an explanation on the implementation of various Machine Learning algorithms on using the specific dataset down to the coding level.
• **Chapter 5**: presents the test cases that were undertaken to test the implemented solution.
• **Chapter 6:** discusses the results of the Machine Learning algorithms utilised to solve the problem along with the Limitations of the approach taken.
• **Chapter 7:** discuss potential future work that could be undertaken to improve the project
• **Chapter 8:** concludes the main findings of the project.
• **Chapter 9:** discuss the reflection on Learning gained from completing this project.

Chapter 2: Background Research

2.1 Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a field of science that makes it possible for computer systems to learn from experience and perform human-like decision making tasks [1]. It simulates human intelligence processes for a wide array of activities, especially with vast size information. It is purpose built to process the information much faster than human capabilities and with a high degree of efficiency and accuracy [ibid].

Machine learning is a branch of Artificial Intelligence which comprises of diverse methods of data analysis that automates analytical model building, and it is one of the most active technique to achieve AI [ibid]. In 1959, the first scientist on the field of computer gaming and artificial intelligence, Arthur Samuel described Machine Learning as the "field of study that gives computers the ability to learn without being explicitly programmed" (Samuel 1959). He indicated that computers will learn from experience and will reduce the need for detailed programming effort [ibid]. Nowadays, the popularity of machine learning increased, and it became important in the area of extraction information from a large data set. The main purpose of machine learning is to predict future events that are unseen to the computer. Machine Learning approaches achieved by the collection of algorithms which utilised statistics and mathematics to find patterns in massive amounts of data where data can be numbers, words, image or click [44], Machine learning is the process that powers a wide range of today's services, as most industries in multiple fields are relying on machine learning technologies; for instance, financial services, health care, transportation, gas and oil, government, and retails [ibid]. Machine learning has the ability to offer for the organisation the opportunity to obtain leverage over other competitor companies, as different goals have been used, for example; in recommendation systems, most known companies like Netflix, YouTube, and Spotify use machine learning to predicts utility of items for every user. Moreover, in search engines like Google, machine learning has various aspects as it uses for pattern detection to help identifying spam or duplicated contents, also use to identify new signals [35]. Whereas voice assistant devices like Siri and Alexa, machine learning is the reason for the continues improvement in the ability of voice-activated user interface [18].

Machine learning has a classification approach that extracts information from training datasets to use it for obtaining the categorisation of unseen data, as shown in Figure 2.1 [13].

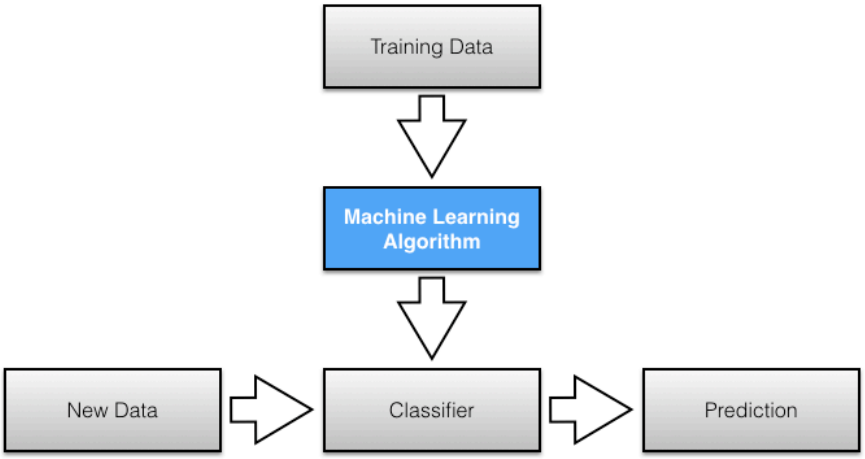

Figure 2.1: Machine Learning Flow

Classifiers are a specific family of Machine Learning algorithms that learns from the input data and then implements rules to classify observations into groups. It works by identifying specific features that help discriminate between observations. The process of the classification algorithm is to input the data and split it into two groups, one called training dataset and the other called validation/test dataset [44]. The training dataset is the one that is used by the algorithm to "learn" about the data, which is generally comprised of multiple variables, also called features. As a next step, classifiers use all or some of the features to understand the association amongst the variables; after that, rules will be generated to classify observations. The process of the classifier model is shown in figure 2.2.

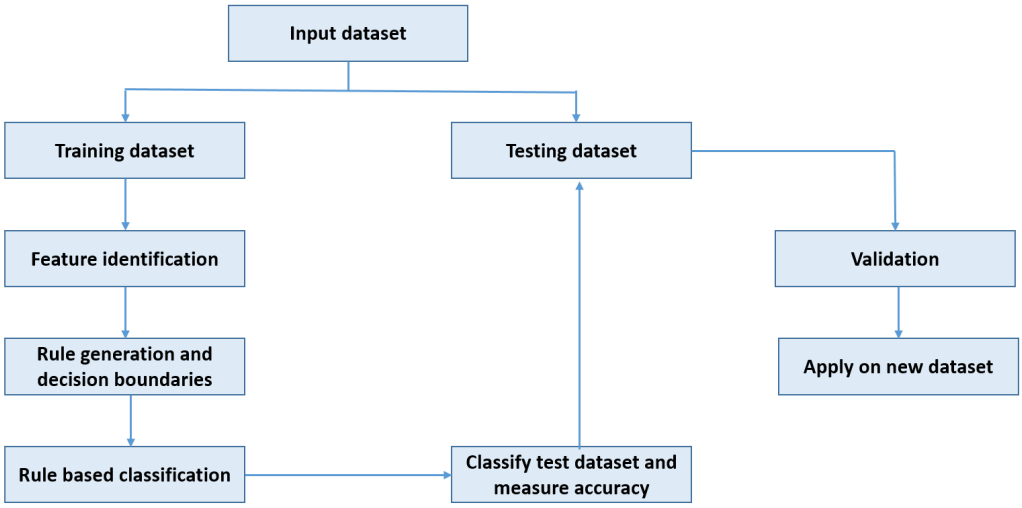

Figure 2.2: Process of Classification Model

There are multiple Machine Learning classifiers exist, some of the most popular Machine Learning classifiers that is going to use in this project will be addressed in the following sections.

2.1.1 Supervised vs Unsupervised learning

There are several types of Machine Learning, Supervised and Unsupervised Machine Learning are mostly used based on the availability of the observations which are already labelled (classified) [26]. In a supervised learning model, the algorithm discovers the pattern and learn on a labelled data set, giving an answer key that algorithm can use to evaluate the accuracy on training data which is illustrated in Figure 2.3 [36]. Therefore, classification is a typical example of supervised machine learning. In contrast, unsupervised model provides unlabelled data that the algorithm tries to understand, by extracting features and patterns by itself. Example of a labelled dataset would be data containing network traffic information with each traffic marked as “benign” or “malware”. This dataset is suitable for a supervised Machine Learning algorithm, say a classifier, which learns the features that associate strongly with a particular class; therefore, it gains the necessary knowledge to classify a new dataset that is not labelled [39]. On the other hand, an example of a dataset suitable for unsupervised learning would be a similar dataset without explicitly labelling the observations as “benign” or “malware”. An unsupervised Machine Learning classifier can read this type of dataset and then classify the observations into some reasonable groups. It leaves the researcher with the task to further analyse the groups to understand what all features segregates the data into groups. For

the purpose of this project, Supervised Machine Learning will be utilised by applying classification methods [ibid].

.

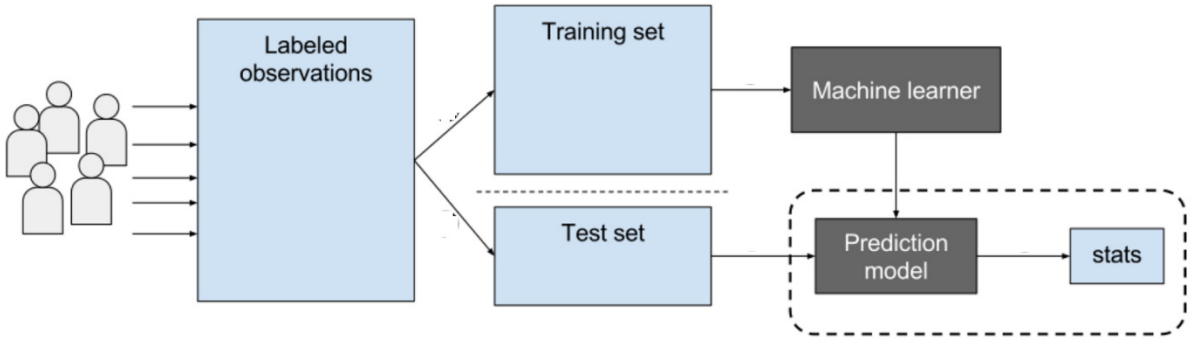

Figure 2.3: Working model of a supervised learning

2.2 Existing Solutions

Various malware detection techniques have been into existence for a long time; however, increasing diversification of malware and its subtypes has made it necessary to devise newer methods for better efficacy. In 2014, Ahmad and Salleh, has exhaustively dealt with botnets (malware), their history and issues facing rapid growth of botnet techniques [25]. They have discussed the use of mathematical algorithms for anomaly detection which involves Decision Trees, Clustering, Correlations and Neural Networks to create an artificial immune system around computer networks to prevent any malicious attacks from botnets [ibid]. Combinations of these statistical and machine learning algorithms created a hierarchical alarm signal to detect suspicious activities across the network. Similarly, in 2019, Lim, Kim, Hong and Han published a paper on Payload-Based Traffic Classification Using Multi-Layer LSTM in Software Defined Networks [28]. They proposed a traffic classification system by utilising a deep learning model in software defined networks. The authors trained two deep learning models: 1) the multi-layer long short-term memory (LSTM) model and 2) the combination of convolutional neural network and single-layer LSTM models, to execute network traffic classification. They also executed a model tuning approach to find the perfect hyper-parameters of the two deep learning models. They showed the advantage of the multi-layer LSTM model for network packet classification [ibid]. Moreover, Machine learning model has been widely used in other aspects of malware detection such as web-attacks. In 2014, research by ElBachirElMoussaid and Toumanari had successfully improved web application security by

utilising intrusion detection systems and scanners based on machine learning and artificial intelligence approaches [15].

Based on the success of earlier work, it would be used to build a machine learning model to detect malware network traffic from IoT devices. This project has considered various techniques explored by researchers in the past. However, the approach of this thesis is to solve the problem by implementing the most popular classification model for detecting malware traffic from a different group of machine learning algorithms (Tree → Random Forest, Kernel → Support Vector Machine, Function → Logistic Regression). Also, it considers three different variations of Support Vector Machine, which are, Linear kernel, RBF kernel, Sigmoid kernel. Another difference is that the developed solution will include classifiers performances evaluation with the combined result and will focus on comparing the performance of the given classifiers to decide the best model produced. Moreover, Random Forest classification was firstly chosen among the rest classifiers, due to the studies it proves the suitability in multiple fields such as malware detection, and it also prevents overfitting.

2.3 Machine Learning approach to malware detection

Malware detection is considered as a binary classification task, where in this project, Malicious network traffics are treated as (1), and benign network traffics as (0). Therefore, Machine Learning techniques and statistical approaches are utilised to build classifiers that will be trained using labelled training data that consist of network traffic captures packets and labels. As recommended by Hyo-Sik Ham and Mi-Jung Choi, some of the most popular Machine Learning classification algorithms that are used for malware detection are Random Forest, Support Vector Machine, Logistic Regressing [21]. Each one of the classifiers mentioned above is further discussed in the upcoming sections.

2.3.1 Random Forest Classifier

Random Forest is a supervised learning algorithm which uses decision tree-based classification as well as regression algorithms to evaluate a massive number of decision trees (known as forest) generated randomly from the data [7]. It takes the predictions from various trees and then selects the best by means of a voting mechanism, as shown in Figure 2.4 [27]. Random Forest algorithm also generates variable importance. There are two ways to the importance ofeach variable in the random forest. Firstly, by calculating how much the accuracy decreases

when the variable excluded. Secondly, by measuring the decrease of Gini impurity when a variable chose to divide a node [ibid]. After training the Random Forest classifier, it can predict the labels of a new dataset which has the same set of features as the training data set [ibid]. A well calibrated model can have a high degree of accuracy in generating predictions on the new dataset.

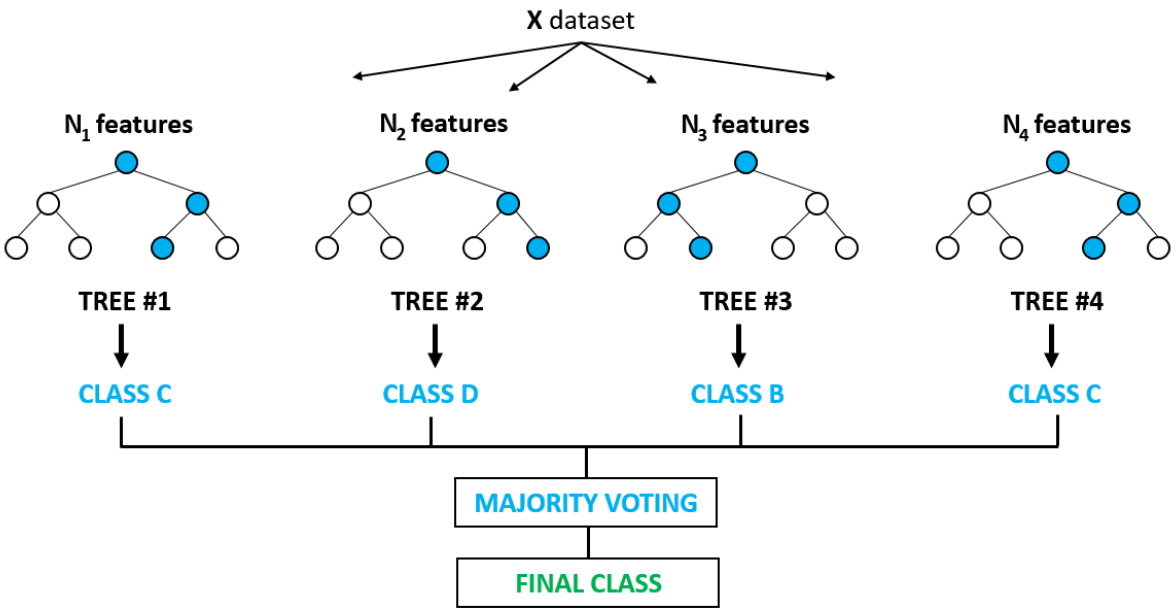

Figure 2.4: Voting mechanism in Random Forest algorithm

2.3.2 Support Vector Machines

Support vector machines are a strong classification algorithm extensively used for classification problem in wide fields and achieved successful rate [24]. The main task of a Support Vector Machine is separating the data into groups using a boundary called a hyperplane. Hyperplane can be linear or non-linear depending upon how the data is distributed in the dimensional space [29]. In a binary scenario, where the labelled dataset contains two classes; for example, benign and malicious, the algorithm generates a boundary to separate the two from each other. Data that are really close to the hyperplane are called support functions, they represent the hard-to-classify cases and they have a direct bearing on the optimal location of the hyperplane. Optimum location of a hyperplane is determined by minimising the distance of the points from the hyperplane; this distance is called margin. A good margin is one where

this separation is larger for both the classes. The reason behind choosing the hyperplane with the largest margin is because it makes the classification right for testing data that are close, but not similar to the training data [ibid]. Figure 2.5 depicts the hyperplane and margin in simple terms.

There are different mathematical sets of Support Vector Machine algorithms that are defined as the kernel. The purpose of the kernel is to take data as input and convert it into the required form. [41] Different SVM uses different kernel functions; an example of kernel used in this project are, Linear kernel, Radial Basis Function kernel (RBF), Sigmoid kernel [ibid].

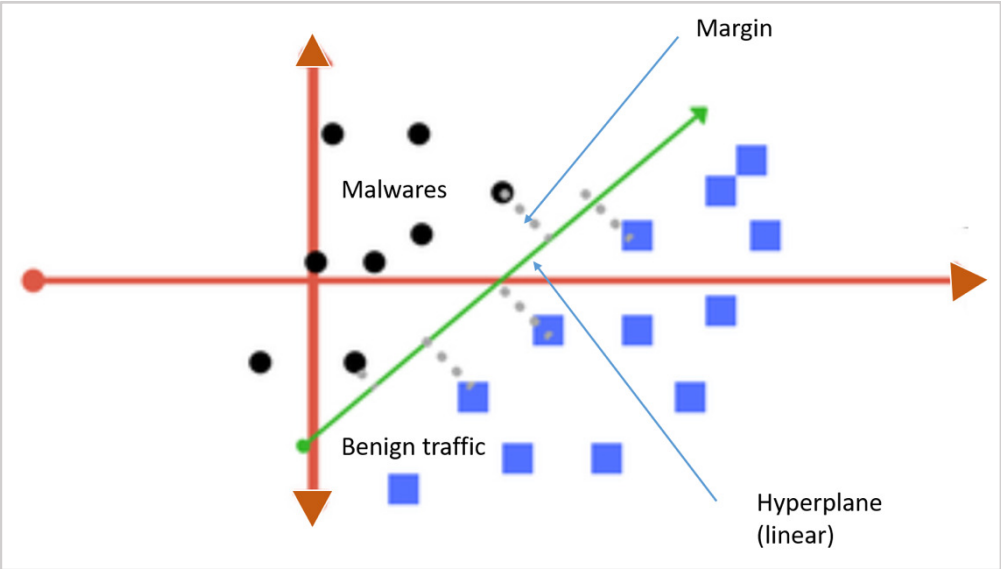

Figure 2.5: Support Vector Machine

2.3.3Logistic Regression:

Logistic regression is a suitable regression analysis to handle when the dependent variable is dichotomous (binary) [44]. It is also a supervised learning model as it needs a pre-labelled outcome variable which takes binary values like 1 and 0 or Yes and No. Logistic regression is essentially a linear regression of odd ratios. An odds ratio is a statistic that quantifies the strength of the association between two events, A and B. Events are packet lengths >10 and <=10, or origin bytes > 50 bytes and origin bytes <=50 bytes [ibid]. Logistic regression uses a

sigmoid function to map the independent variables with the dependent variable via the odds ratio, as shown in Figure 2.6 [27].

.

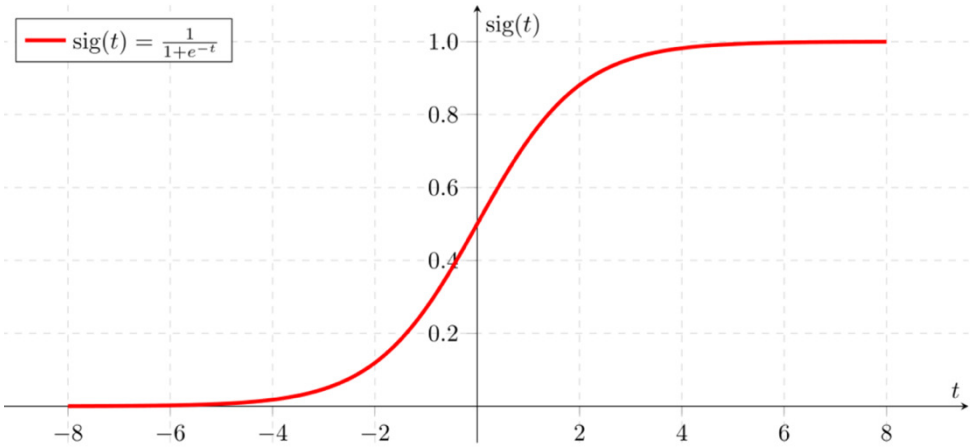

Figure 2.6: Shape of sigmoid function

2.4 Learning Evaluation

This section will include the evaluation technique for evaluation the performance of machine learning model.

2.4.1 Confusion Matrix

There are various formal metrics and tools to assess model performance. A confusion matrix is a table that indicates how successful is the classification model by summarising the prediction results. Confusion matrix outcomes are:

- True positive (TP) is a result of the model when it correctly predicts the positive class
- False positive (FP) is a result of the model when it incorrectly predicts the positive class.
- True negative (TN) is a result of the model when it correctly predicts the negative class
- False negative (FN)mis a result of the model when it incorrectly predicts the negative class

The above four measures are reviewed in a table that shows how successful is the classification model's in predicting the outcome; that is, the correlation between the label and the model's classification [44]. The label on one axis of a confusion matrix table is model prediction and the other axis is the actual label. The size of the confusion matrix table is NxN, where N

indicates the number of classes; for example, in a binary classification problem, N=2 [ibid]. Figure 2.7 shows the confusion matrix for a binary classifier.

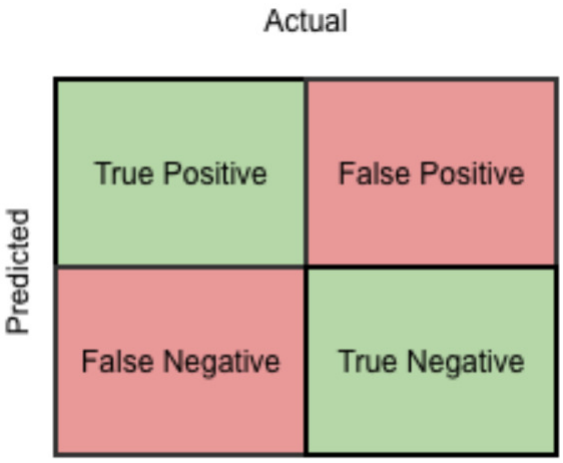

Figure 2.7: Confusion matrix for binary classifier

2.4.2 performance Indicator:

A good performance indicator from the confusion metrics will rely on having small numbers in the off-diagonal element and large numbers in the diagonal elements. The confusion metrics are used by different evaluation metrics; for example, Accuracy, Recall, Precision, F1-score and AUC to measure performance [8]. The evaluation metrics are described individually in the following list:

Accuracy: indicates the total percentage of correct prediction and it is given by the number of correct classification example divided by the total number of classified examples. In the term of the confusion metrics, it is given by:

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

the best accuracy value is 1 and the worst is 0 [ibid].

Precision: defined as the number of correct predictions returned by the machine learning model. It is the ratio of correct positive predictions to the overall number of positive examples in the dataset and it can be calculated by confusion metrics with the help of following formula:

$$Precision = \frac{TP}{TP + FN}$$

Recall: defined as the number of classifiers to find all the positive instances. It is the ratio of correct positive predictions to the overall number of positive examples in the dataset and it can be calculated by confusion metrics with the help of following formula:

$$Recall = \frac{TP}{TP + FN}$$

F1 score, also known as F-score or F-measure. It is the average of the combination of Precision and Recall. Thus, this score takes into account both false positives and false negative. Intuitively, F1 is more useful than accuracy with an uneven class distribution. In other words, accuracy works better if false positives and false negatives cost the same. If they are different, it's more accurate to consider both Precision and Recall [10]. It can be calculated by confusion metrics x with the help of following formula:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

AUC is the area under the ROC curve:

ROC is a graph representing the performance of a classification model in all thresholds. It uses a combination of true positive rate as recall and false positive rate (the proportion of negative example predicted incorrectly). More area under the ROC curve (AUC), the better the classifier. Figure 2.8 illustrates the ROC curve and the AUC [8].

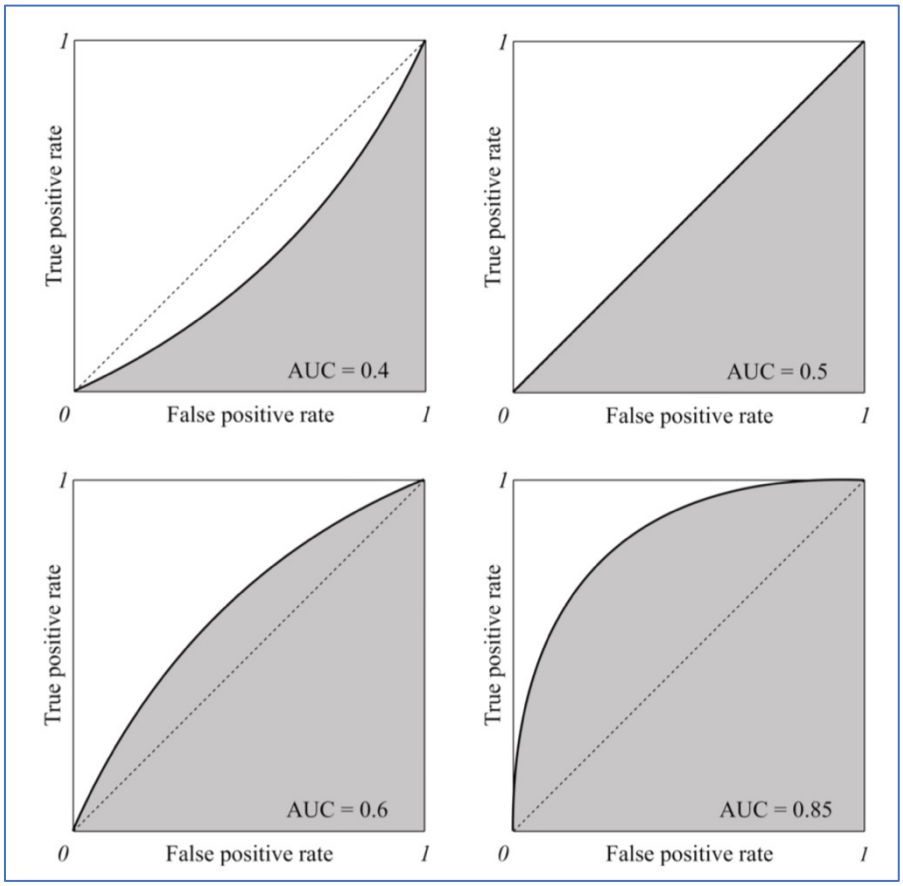

Figure 2.8: The area under the ROC curve (shown in grey)

2.5 Python Machine Learning Libraries:

This section will cover the tool used for the development of the project. Since python programming language will be used throughout the implementation step of this project, research on supporting libraries for Machine Learning had to be taken [23]. Python is considering as a high-level programming language that grants user to focus on the important function rather than programming tasks. The simple syntax rules make it easier to keep the code readable and maintainable. Moreover, different open sources specified for machine learning libraries are included with Python, which will be utilised in essential areas of the project involving in the data analysis process, prototyping and evaluation [ibid]. There are several libraries used in the project; such as Panda, NumPy, Scikit-learn, Seaborn, Matblotlip libraries, and Brothon as a package.

Pandas is a Python library used mainly for data manipulation and analysis. It employed before the data set is developed for training. Pandas make tasks with time series and structured data smooth for machine learning programmers [37]. Panda will also be used to create a panda data

frame with a tabular data structure (rows and columns). NumPy is another essential package that provides useful features and offers speedy computation and execution of complicated functions working on arrays. Also, it handles multi-dimensional data and provides mathematical operation. On the purposes of this project, Scikit-learn will be utilised, which is an actively used machine learning library for Python. It includes simple integration with different machine learning programming libraries like NumPy and Pandas to provide an easy and useful system [ibid]. Also, it includes essential and helpful methods for machine learning; for example, splitting data and measuring performance. Moreover, Brothon package was used to process and analyse a Bro IDS data [45]. Finally, Seaborn and Matplotlib both libraries used to visualise attractive graphs and to apply for statistical purposes [42].

2.6 Research Questions

Aims

The main purpose of this project is to classify malware network traffic of IoT devices from a given dataset into either malicious or benign and to investigate the performances of different Machine Learning classification algorithms with evaluation metric to identify the best machine learning techniques.

Research questions

To demonstrate the achievement of the stated aims, this project will explore common techniques employed to detect malware traffic by implementing the most appropriate Machine Learning classification algorithms and applying suitable evaluation metrics to evaluate the classifiers and to decide the best model.

Chapter 3: Specification and Design

For a successful constructed design, it is essential to select and design the main components that need to be considered when implementing the solution. Therefore, this chapter of the report will explain the design of the implemented solution to be developed. Moreover, software requirements specification and initial plan of the design will be discussed. Also, it will review the system design and the methodology of this project.

3.1 Software Requirement Specification

Software requirement specification considers as a foundation of every system design. It helps to reduce cost and time of the project as well as it helps in evaluating the success of the result. In this project, the Software Requirements Specification contains functional and non-functional requirements illustrate as follow:

3.1.2 Functional Requirements

The most important functional requirements are defined in the following list,

Data Type: The System must take network traffic data from the bro log file as an input with their labels and classify them into malicious or benign.

- Bro log files which is a specific file format "conn.log" obtained from the Zeek software are required for the system for training and testing the classifiers (more details of Bro log file will be included in chapter 4). Therefore, the system will get the data from the user input by specifying the directory of the data file.

Data flows through the system: The system must pre-process the given raw network traffic.

- Once the data import is successfully completed, which contains categorical (object), numeric (int64), and datetime64 type of values, pre-process step will take place to output a clean data with only numeric values to prepare the data for the classifiers. The next step will take place is feature selection to choose suitable features for each classifier.

Algorithms: The system must provide three different classification algorithms with three different variations of one of the provided classifiers.

- The system will have in total of five different classification algorithms. The classifiers available are Random Forest, SVM Linear Kernel, SVM RBF Kernel, SVM Sigmoid Kernel and Logistic Regression.

Evaluation: The system must allow evaluating trained classifiers to identify the best performing model.

- The system will present the results from calculating the Accuracy, Precision, Recall, F1 score and AUC of the Classifier for each training and test set.

3.1.3 Non-Functional Requirements

The non-functional requirements are defined in the following list:

• **Reliability:**

-The system will have no errors and will be available to use all the time. The performance indicators that will be provided are accuracy, precision, recall, F1 score, AUC and confusion matrices which are also indicators of reliability.

• **Usability:**

- The project will implement three models to be used for classification.
- The introduced models will be trained with pre-labelled collected data.
- These models will be evaluated using a test set from the collected data.

• **Speed:**

- Pre-processing the datasets, creating the classifiers, testing and evaluating the classifiers should be reasonably quick.

• **Size:**

- The size of the system will not exceed 1000 megabytes; hence the system will have a small impact on computer memory since the data set used are vast.

• **Re-usability:**

- The implementation of the project is divided with multiple modules that can be reusable in other projects or systems.

-All models are taking column features from a panda data frame, and most type of files like CSV, Pcap and Bro log files can be saved as a panda data. Therefore, these models can be used in a different project with a different type of files.

3.2 System Plan:

The produced system will be implemented based on the given data set from the Stratosphere Laboratory [40]. The data set called CTU-IoT-Malware-Capture-1-1 contains 539465 Malicious network traffics out of 1008740 IoT network traffics. The system will mainly classify these networks traffic depends on its labelled, 0 for benign and 1 for malicious traffic. The implementation of the project will be done using Python Programming Language and the script will be written in one Jupyter notebook file called MLForClassifiyingNetTraffic.ipynb This file will consist of sections that work sequentially to feed into the next or acts as a standalone module to be reused elsewhere in the program. There are five sections within MLForClassifiyingNetTraffic.ipynb file and they are as follow:

1- loading Bro log file: read the file from the directory
2- Exploratory Data Analysis (EDA): which is an approach to analyse data sets and conclude their main attributes.
3- pre-process data which prepare data for the machine learning classifiers
4- machine learning models (classifiers) contains three main models Random Forest, Support Vector Machine, Logistic Regression and with three different variations of SVM which are SVM Linear Kernel, SVM RBF Kernel, SVM Sigmoid Kernel.
5- Evaluation which contains the critical role in providing performance indicators

3.3 System Design:

In order to understand the structure and behaviour of the developing solution, a model of the Unified Modelling Language (UML) diagram is used. This approach helps to visualise the architecture design of the implemented code and understand its functions. The two diagrams

used are user case diagram and activity diagram. Also, the provided diagrams can be used for future work to implement the solution delivered as a software system that allows user interaction.

Figure 3.1 shows a user case diagram that represents the possible implemented models that used as a solution for this project. The five available options involve three different classifiers model with three different variations of one of the models. All classifiers are as follow, Random Forest, SVM Linear Kernel, SVM RBF Kernel, SVM Sigmoid Kernel, and Logistic Regression.

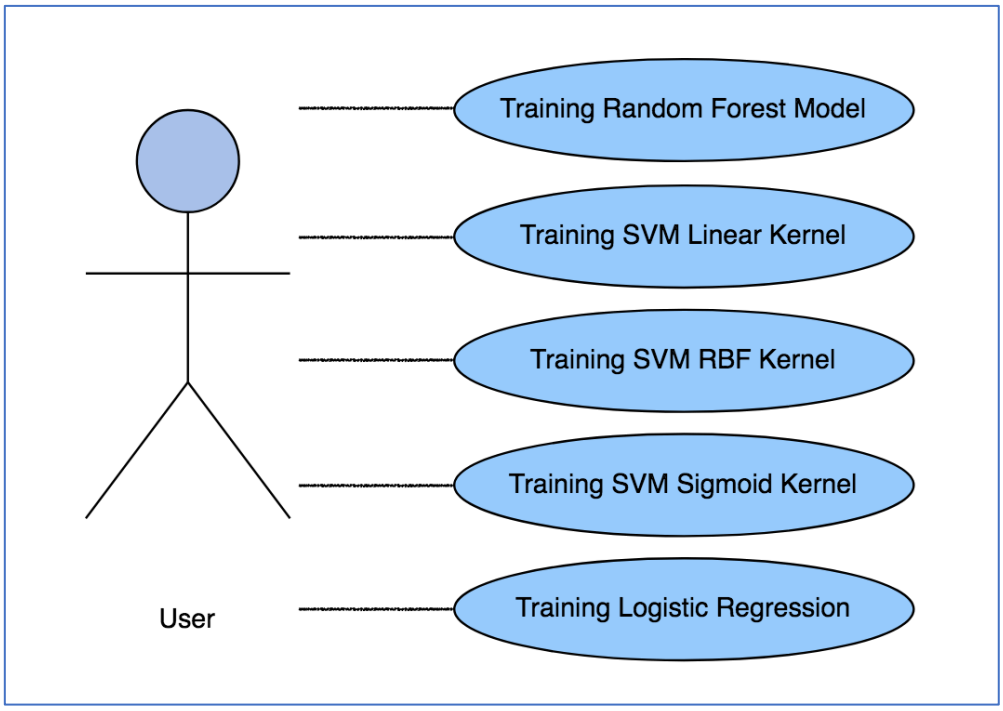

Figure 3.1: User Case Diagram

Moreover, figure 3.2 illustrates the approach includes behaviour aspects of the implemented solution as an activity diagram. The flowchart below contains the steps taken after loading the data to the script, which are; read data, exploring data (EDA), and pre-process for the classifiers. Pre-process data involves multiple steps which are; split the data into training and test set, feature selection for each model depends on the model type and classify the data as malicious or benign by using available classifying models. Finally, after predicting the test data, the evaluation metrics process will take place by using different approaches to calculate the result and compare it with the outcome of different classifiers.

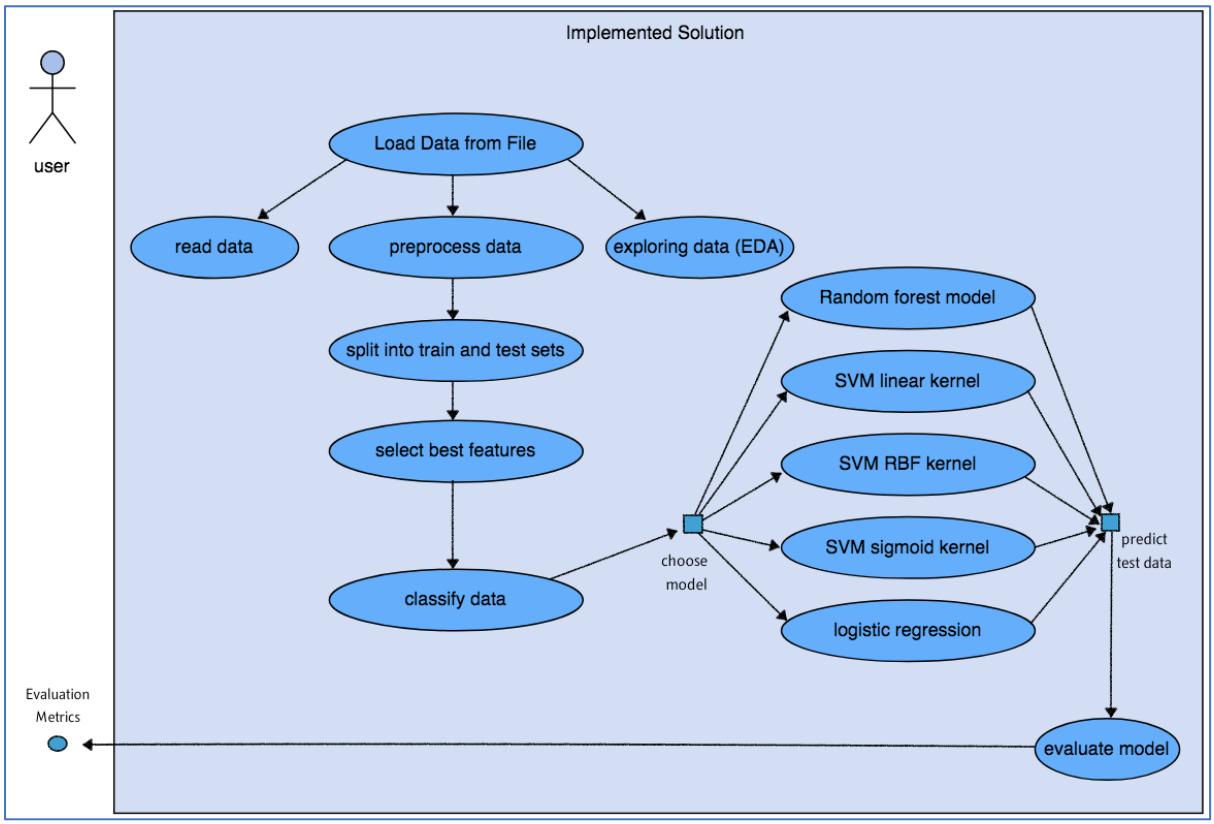

Figure 3.2: Activity Diagram of Software System

3.4 Development Methodology:

Due to the constrained time, where the solution framework must be created and tested, a development strategy must be followed to deal with the development and evaluating to ensure that all deliveries were achieved by the end of the project and to deliver optimal project result. In this project, Agile Software Development strategy has been decided to be the appropriate technique and was utilised during implementation and testing stage. With this strategy, the task could be broken down into smaller task called iteration. Whenever each iteration completed, the work product display to the customer. There are various reasons for adopting Agile methodology over other techniques in this project. Firstly, Agile allows requirement changes at the start or during the plan as the project is on moderate to high risk of changes due to the high dependency on the data set provided. Agile is a suitable method because it will allow editing, modifying or even changing the requirements during the project without losing the entire task. Secondly, a working software will be conveyed considerably quicker and successive iterations can be achieved frequently and display to the customer at a consistent pace which will gain customer satisfaction. Finally, feedback will be received from the client and can modify the project in the next iterations. Although Agile is a more cooperative

approach with a team, planning it was easier for me with a single developer. I followed the methodology by breaking down assigned tasks, arranging the task based on the priority, considering feedback that received from the supervisor on each meeting, and finally by setting the planned objectives and adapt to the changing of the work to achieve the task within the given frame time. Following is the list of four iterations that have been used in order to accomplish this project. Each iteration involves goals and deliverables.

- Iteration one: loading, parsing and exploring the data set provided. The deliverable from this iteration was pre-processed dataset that can be used in classification models.

- Iteration two: focus on implementing the three classifiers and with three variations of one of these classifiers and allow them to be trained and tested with the previous data set. The deliverable of this iteration was to produce in a total of five classifiers that are able to be trained and tested.

- Iteration three: focused on selecting features for model training. The deliverable from this iteration was selecting the features by using suitable feature selection methods; for example, feature importance and recursive feature elimination functions.

- Iteration four: focused on creating a range of different evaluation to calculate the result of each classifier and decide the best model. The deliverable from this iteration was comparing evaluation metrics for every machine model.

Chapter 4 Implementation:

This chapter will explain the implementation part along with the code.

4.1 Project Structure

To make the malware detection framework easy to debug and reader-friendly, one modular structure has been implemented which contain multiple sections as discussed in previously. **Python** version 3.7 is used, since python is a powerful programming language and accessible most developers use it as the primary language for data science project [23]. It allows developers to focus on the machine learning aspect rather than programming the task due to its ability to build complex task quickly [ibid]. Moreover, program script is written in **Jupyter** notebook as it is a handy and flexible tool, where all documentation, execution of the code, observation of output and visualization of the result can execute in one file. Also, each section works independently; thus, this has the ability to test a specific block without running the code from the start as it will ease the access to particular functions.

The six sections demonstrated and discussed as follow:

1. Reading Bro log file: load the file from the path directory
2. Exploratory Data Analysis (EDA) which is an approach to analyse data sets and conclude their main attributes.
3. pre-process data which prepare data for the machine learning classifiers
4. machine learning models (classifiers) contains three models; Random Forest, Support Vector Machine, which consists of three variations and Logistic Regression. Also, it includes feature selections.
5. The Evaluation which contain the important rule in providing performance indicators

4.2 Dataset used

The dataset used in the analysis is labelled malware botnet dataset consisting of a mix of benign and malicious network traffic data. The data set is downloaded from the Stratosphere Laboratory, AIC group, FEL, CTU University Czech Republic [40]. The IOT-23 dataset contains 20 captures of malware executed in IOT devices. Each file consists of the original

packet capture file of network traffic in .pcap format which is converted in Bro log file (Netflow format) by the providers using Zeek/Bro software [ibid]. Zeek software is an open-source network traffic analyser uses for security matter to detect all traffic on a link in-depth to hunt any suspicious activity. Additionally, Zeek features a wide range of traffic analysis tasks not only on the security domain; like performance measurements and use in troubleshooting [11]. Suspicious files are manually detected by analysts and are labelled malicious. The Bro log file used in this analysis is under "CTU-IoT-Malware-Capture-1-1" folder and it consists of 1008739 network traffic and with size of 148.3 MB.

4.3 Data Preparation & Processing

This section details the process of ingesting the dataset from physical memory of a computer to Python environment for processing and analysis.

4.3.1 Data ingestion

The first step towards data analysis is fulfilled by importing the dataset from the sources into the analysis tool. The dataset used for analysis is in a specific file format called "conn.log" which is obtained from the Zeek software. Specific Python package named "Brothon" is used to parse the bro log files. The Brothon package supports the ingestion, processing, and analysis of Bro IDS data with Python. BroLogReadere( ) is a specific function within Brothon package it will take the conn.log file as an argument, and it will be converted to a data frame using panda for further analysis. The code snippet used to read the conn.log file and to convert it to a data frame is shown in Figure 4.1.

```
# Reading file-  the bro labeled file
# pip install brothon[all] # install only for the first time
import pandas as pd
from brothon import bro_log_reader

# # Reading the bro log file
reader = bro_log_reader.BroLogReader(r'/Users/bedoorbahassan/Desktop/CTU-IoT-Malware-Capture-1-1/bro/conn.log.labeled')
bro_df = pd.DataFrame(reader.readrows())
```

```
Successfully monitoring /Users/bedoorbahassan/Desktop/CTU-IoT-Malware-Capture-1-1/bro/conn.log.labeled...
```

Figure 4.1: Reading and Converting Bro log File to Pandas Data Frame

4.3.2 Data description

For the purpose of this project, the Network traffic data set was given with labelled contains 21 columns of data with each column consisting of a header and 1008748 rows. Each column of the log file contains valuable information about the captured network traffic. Description of the columns of ***Bro log file*** is summarised in Table 4.1.

Field	Type	Description
ts	time	Timestamp
uid	string	Unique ID of Connection
id.orig_h	addr	Originating endpoint's IP address (AKA ORIG)
id.orig_p	port	Originating endpoint's TCP/UDP port (or ICMP code)
id.resp_h	addr	Responding endpoint's IP address (AKA RESP)
id.resp_p	port	Responding endpoint's TCP/UDP port (or ICMP code)
proto	transport_proto	Transport layer protocol of connection
service	string	Dynamically detected application protocol if any
duration	interval	Time of last packet seen – time of first packet seen
orig_bytes	count	Originator payload bytes; from sequence numbers if TCP
resp_bytes	count	Responder payload bytes; from sequence numbers if TCP
conn_state	string	Connection state (see conn.log:conn_state table)
local_orig	bool	If conn originated locally T; if remotely F.
local_resp	bool	If conn responded locally T; if remotely F.
missed_bytes	count	Number of missing bytes in content gaps
history	string	Connection state history (see conn.log:history table)
orig_pkts	count	Number of ORIG packets
orig_ip_bytes	count	Number of ORIG IP bytes (via IP total_length header field)
resp_pkts	count	Number of RESP packets
resp_ip_bytes	count	Number of RESP IP bytes (via IP total_length header field)
tunnel_parents label detailed label	set	If tunneled connection UID of encapsulating parent (s) labelled as Benign or Malicious after physical examination

Table 4.1 Columns Description of all Bro Log File

To take into consideration, not all 21 columns are used in the analysis because some of them are not related and some of them contain no entry. According to Eirini Anthi study on Intrusion Detection, feature that represented identifying properties were removed; for example, source IP addresses, time [14]. Therefore, in this project elimination process of some columns is needed before starting the implementation. Removed features are listed below with the specifying the reason:

- ***Ts***→ the timestamp of each network traffic is not related, while duration is related
- ***Uid***→ Unique ID Used for identifying
- ***Id.org_h, id.org_p, id.resp_h, and id.resp_p*** → They all categorical values used for identifying
- ***Service, local_orig, local_resp, missed_bytes*** → have no entry for all rows
- ***History*** → it contains categorical values (return 13 types of the state history of connection). Also, it is not related.
- ***tunnel_parents label detailed label*** → this column contains malicious and benign as a string and it has converted to a new column called mal_flag which contains 1,0. (more detail will be disuses in data cleaning and pre-processing section).

There are in total of nine features remaining for the analysis. Table 4.2 lists all the remaining feature with more details about their functionalities [12]. All remained features have been cleaned and pre-processed to be ready for classifiers; detailed steps will be illustrated in the upcoming section (4.3.3 data cleaning and pre-processing).

Feature	Column Name	Functionality
Protocol	proto	The transport layer protocol.
Duration	duration	Time period of the connection lasted
Original byte	orig_bytes	The amount of payload bytes that have been received from the originator
Responded byte	Resp_bytes	The amount of payload bytes that have been received from the responder
Conn State	conn_state	Possible values of connection state. Refer to [12] for more detailed information
Original Packet	orig_pkts	The number of packets that have been received from the originator
Respond Packet	Resp_pkts	The number of packets that have been received from the responder
Original IP Byte	orig_ip_bytes	Number of IP level bytes that have been received from the originator
Respond IP Byte	resp_ip_bytes	Number of IP level bytes that have been received from the responder

Table 4.2 Columns Description of used feature in Bro Log File

4.3.2 Exploratory Data Analysis

Exploratory Data Analysis is a numerical detective work to gain a more in-depth understanding of the data using tables and charts drawn from the data [20]. The need of EDA was essential due to the lack of information about how the data set was labelled since it labelled manually by the developers. This section will discuss and show several summary statistical and graphical representation of variables.

- Sample size: Number of records in the dataset is 1008748 with 2 distinct labels – Benign and Malicious comprising 46.5% and 53.5% of rows respectively. Composition of the data by labels is shown in Figure 4.2.

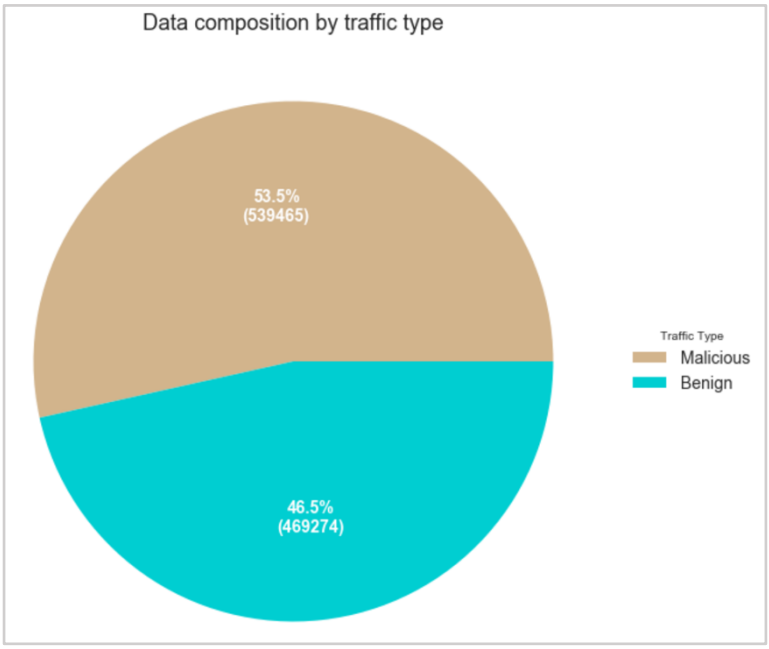

Figure 4.2: Data composition by traffic type (label)

- Variable type (categorical or numeric) identification: Python's Pandas' info method is called to get the data type information for the imported Bro log file. Figure 4.3 in Data Description section summaries the data type of the raw file.

- For Validation purpose, the number of null values was checked to make sure there are no missing values. Figure 4.3 shows all columns in Bro Log file have 1008748 non-null values.

```
# View- of the columns
clean_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1008740 entries, 0 to 1008747
Data columns (total 23 columns):
ts                                          1008740 non-null datetime64[ns]
uid                                         1008740 non-null object
id.orig_h                                   1008740 non-null object
id.orig_p                                   1008740 non-null int64
id.resp_h                                   1008740 non-null object
id.resp_p                                   1008740 non-null int64
proto                                       1008740 non-null object
service                                     1008740 non-null object
duration                                    1008740 non-null timedelta64[ns]
orig_bytes                                  1008740 non-null object
resp_bytes                                  1008740 non-null object
conn_state                                  1008740 non-null object
local_orig                                  1008740 non-null object
local_resp                                  1008740 non-null object
missed_bytes                                1008740 non-null int64
history                                     1008740 non-null object
orig_pkts                                   1008740 non-null int64
orig_ip_bytes                               1008740 non-null int64
resp_pkts                                   1008740 non-null int64
resp_ip_bytes                               1008740 non-null int64
tunnel_parents   label   detailed-label     1008740 non-null object
traffic_type                                1008740 non-null object
mal_flag                                    1008740 non-null int64
dtypes: datetime64[ns](1), int64(8), object(13), timedelta64[ns](1)
memory usage: 184.7+ MB
```

Figure 4.3: Data type of the bro log columns

- Distribution of the variables: In this section, a frequency-based approach has been utilized to understand the distribution of a variable across the spectrum of values it takes.

1. Distribution of variable duration: the duration of most of the network traffic has been 0, and the ones with non-zero durations are mostly centred around 2-3 seconds. Thus, the variable is highly skewed towards the right with peaks at 2-3 seconds. Moreover, a higher proportion of Benign traffics are of non-zero duration than Malicious. However, most of the malicious traffics have duration in the range of 2-3 seconds. The distribution is shown in Figure 4.4.

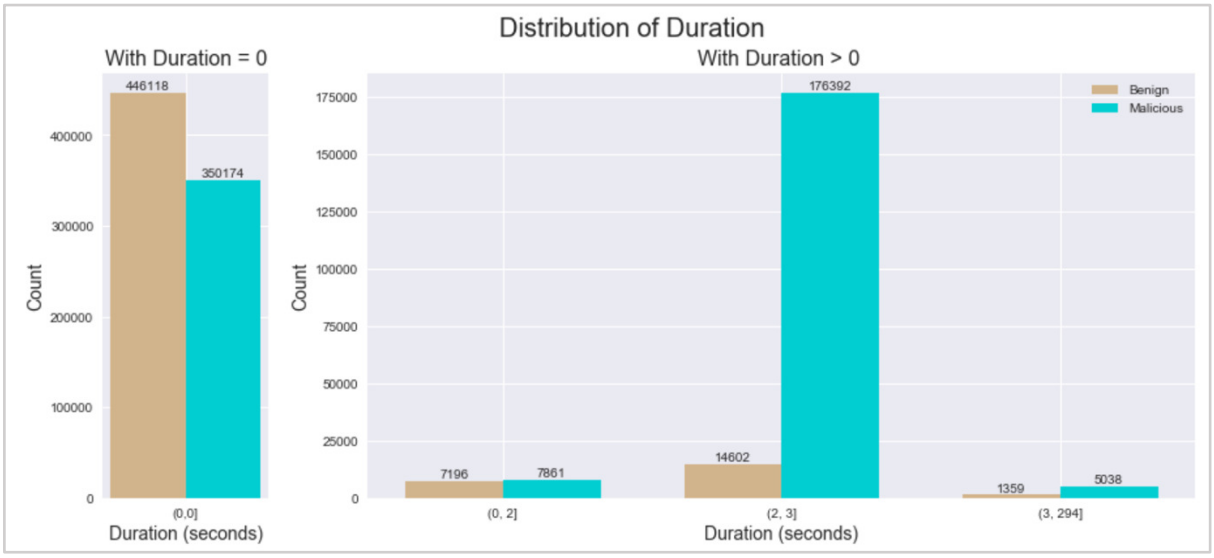

Figure 4.4: Distribution of duration

2. Distribution of original bytes transmitted from the source: Around 90% of the source transmitted 0 bytes. Among sources transmitting greater than 0 bytes, fairly equal proportion of traffic transmitted between 1 and 200 bytes as shown in Figure 4.5.

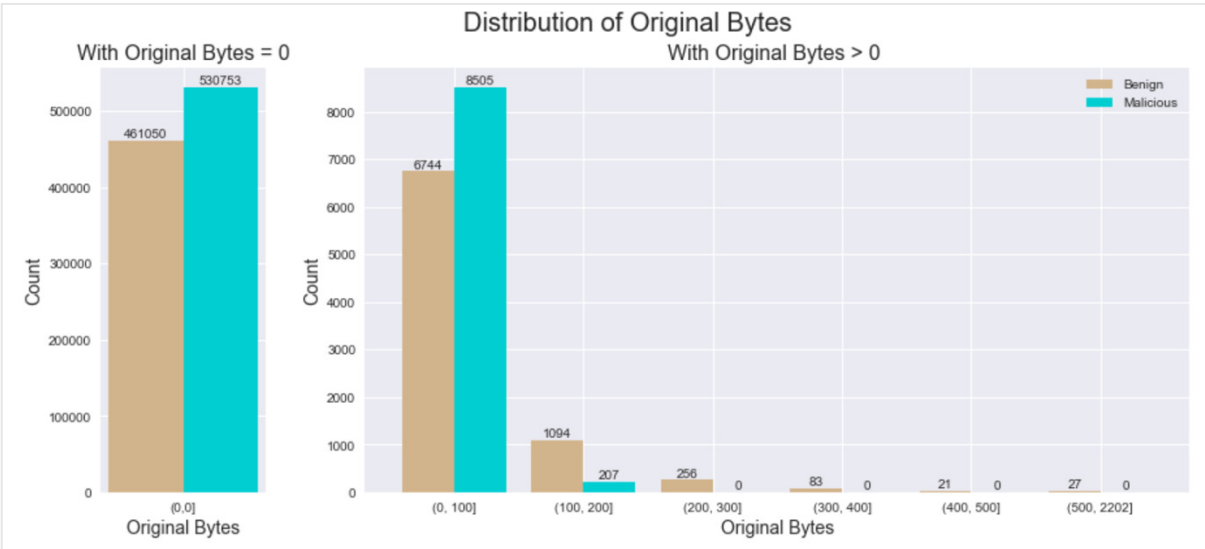

Figure 4.5: Distribution of original byte (orig_bytes)

3. Distribution of respond bytes: in most of the traffic's respond bytes are 0. However, among traffic respond bytes greater than 0, benign traffic witnessed respond bytes between 1 and 50 while malicious traffics are with respond bytes ranging between 1 and 8991 bytes, as shown in Figure 4.6.

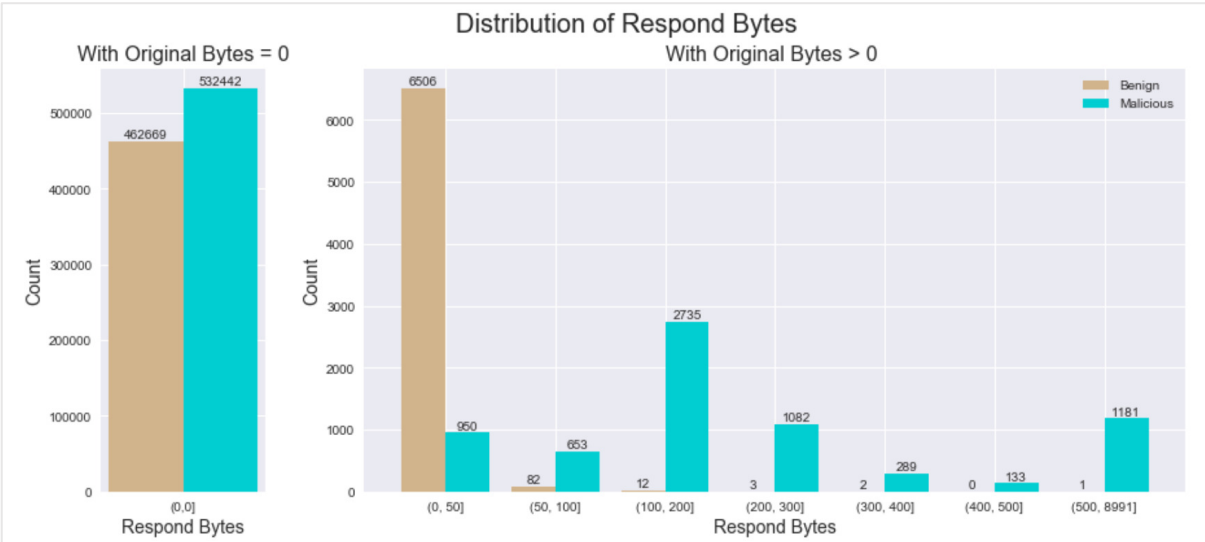

Figure 4.6: Distribution of respond bytes (resp_bytes)

4. Distribution of original packets: Most of the traffic transmitted between 0 to 1 packet, however, traffic transmitted between 2 to 60 packets are mostly malicious. The distribution of original packets is shown in Figure 4.7.

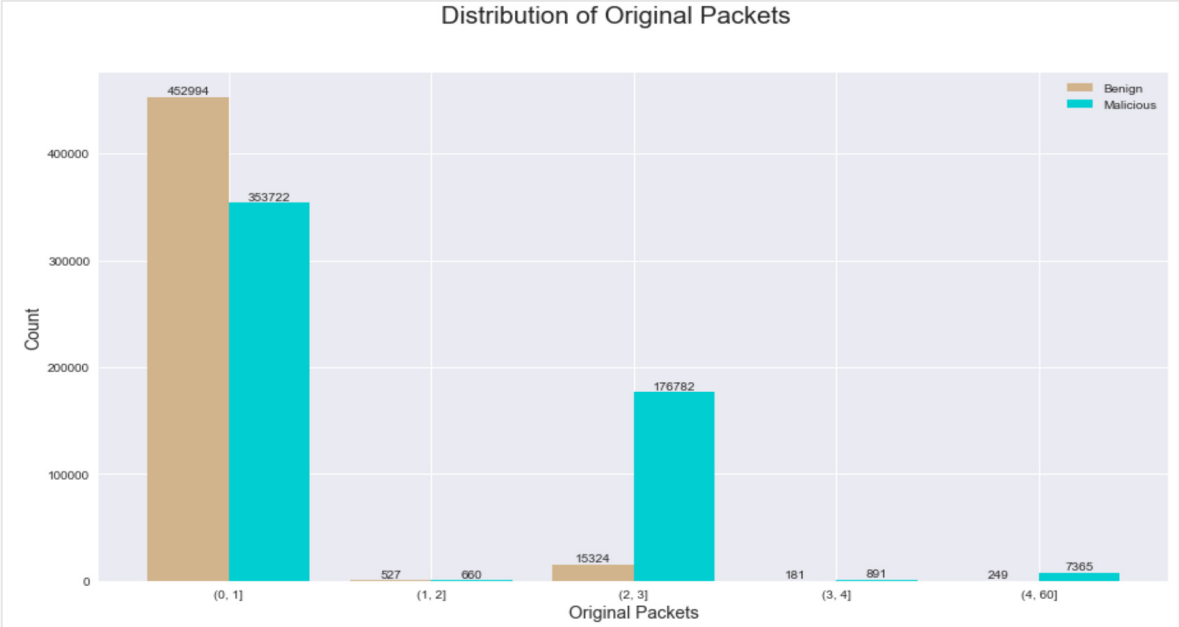

Figure 4.7: Distribution of original packets transmitted (orig_pkts)

5. Distribution of respond packets: Among traffic with non-zero respond packets, benign traffic had single respond packets while malicious traffics have respond bytes between 1 and 75. The distribution of respond packets is shown in Figure 4.8.

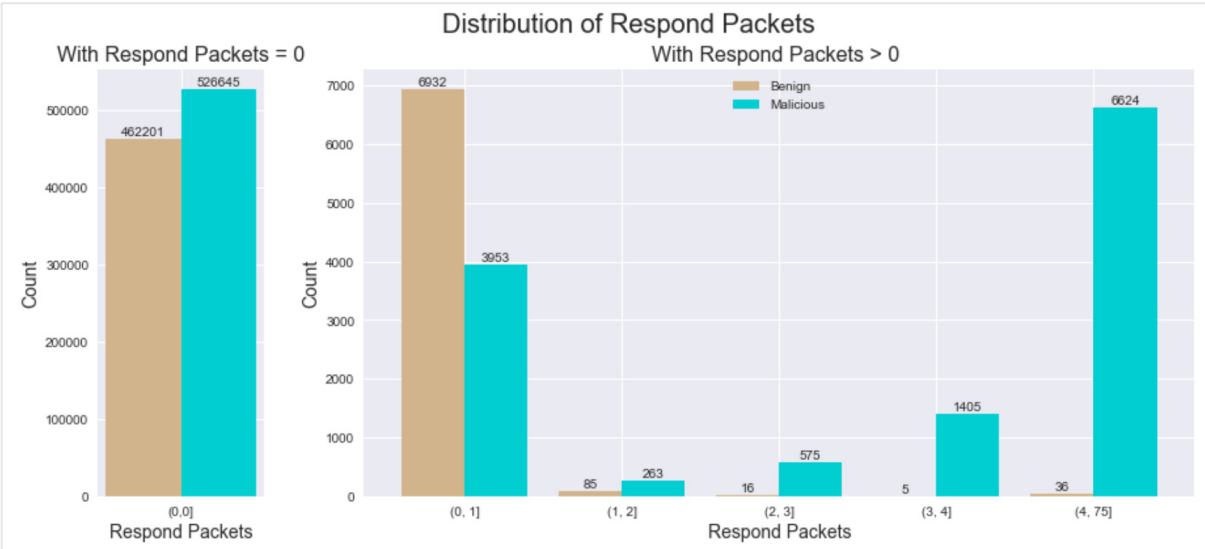

Figure 4.8: Distribution of respond packet (resp_pkts)

6. Distribution of original IP bytes: most of the benign traffic carries original IP bytes between 25 to 50 bytes whereas most of the malicious traffic carries original IP bytes of 50 to 75 bytes followed by bytes in the range of 175 to 200. The distribution of original IP bytes is shown in Figure 4.9.

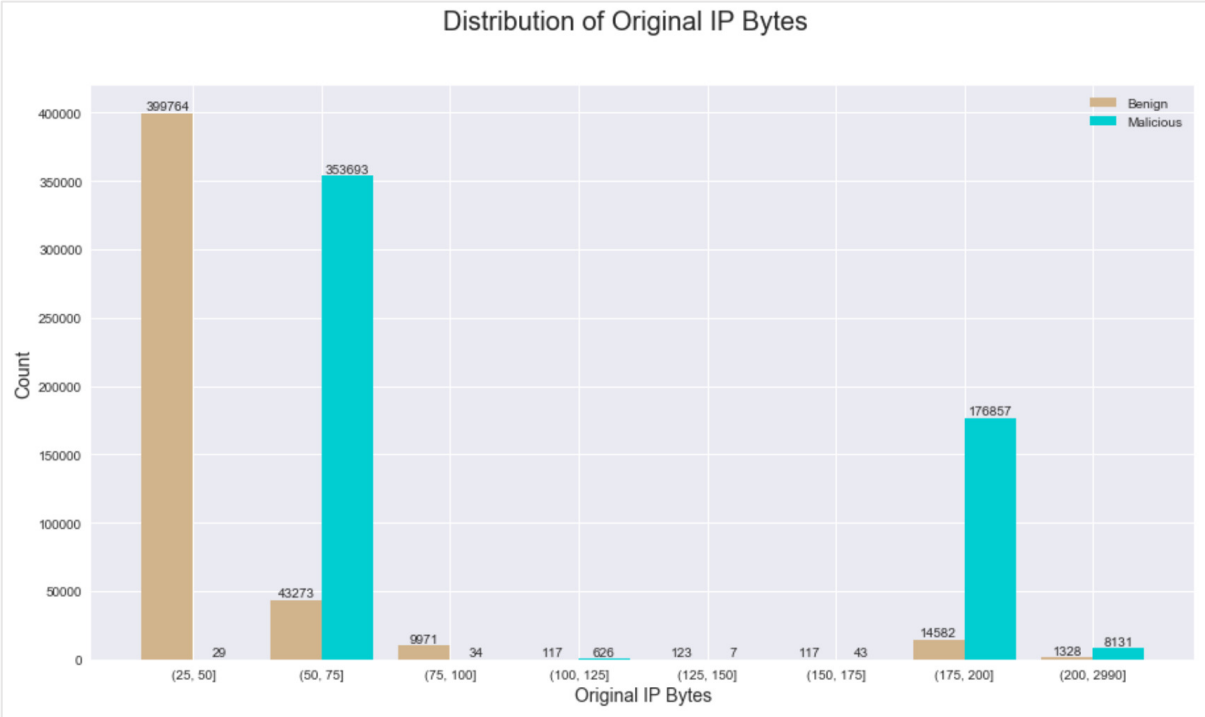

Figure 4.9: Distribution of original IP bytes (origin_ip_bytes)

7. Distribution of respond IP bytes: almost 90% of data (containing both benign and malicious traffic) bears 0 respond IP bytes, a small portion of malicious traffic has respond bytes between 0 and 50 and between 200 and 9415. Benign traffic with non-zero respond IP bytes have range between 50 and 100. The distribution of respond IP bytes is sown in Figure 4.10.

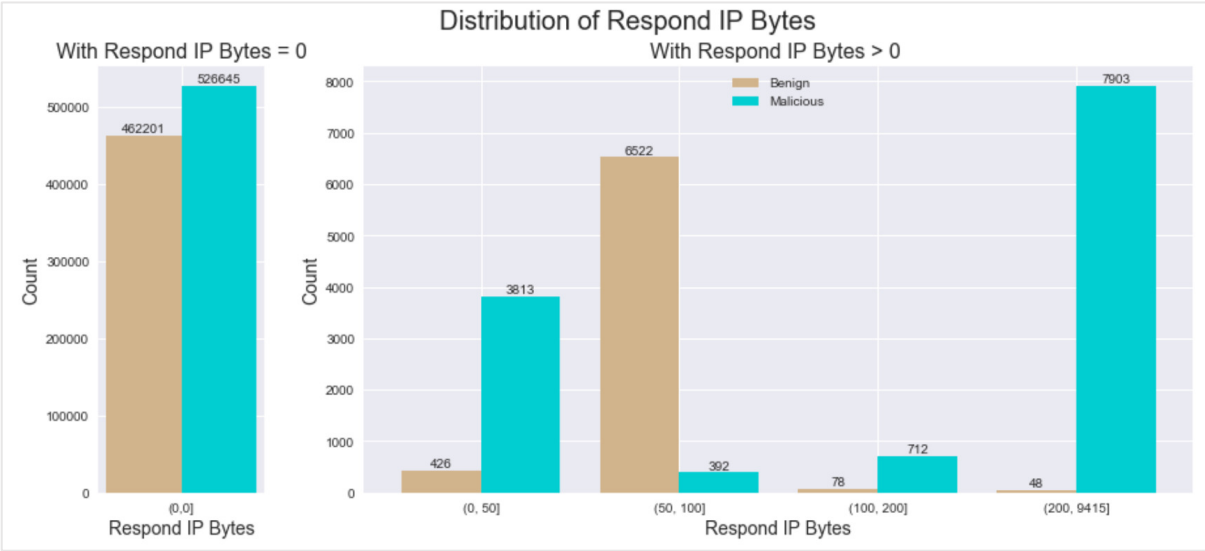

Figure 4.10: Distribution of respond IP bytes (resp_ip_bytes)

The conclusion from EDA:

Summarising the observations from various stages of the EDA leads us to draw the following conclusion:

- Data counts shown in Figure 4.3 shows that there are no missing value in the dataset across all the columns
- Dataset contains a mix of categorical and numeric (continuous and discrete) variables
- Most of numeric variables are skewed towards the right (higher value, away from 0)
- Distribution of the numeric variables, like duration, origin_ip_bytes, resp_ip_bytes, resp_pkts, orig_pkts indicates that the data is non-normal and is not likely to follow any standard theoretical distribution
- Distribution of the numeric variables (duration, origin_ip_bytes, resp_ip_bytes, resp_pkts, orig_pkts) shows that malicious traffic and benign traffic has clear distinction across all numeric features of the dataset and hence could be employed in training the Machine Learning classifiers.

4.3.3 Data cleaning and pre-processing

An integral part of data analysis is to perform data cleaning and transformation to ensure the data is ready for further use. The prime objective of the project is to utilize the dataset for Machine Learning algorithms to be able to detect the labels accurately so that it can be used in real-time malware detection. In that regard, it is important to validate the accuracy of the data, de-duplicate columns (if any), handle and treat blank values, reformat values to useful data format and check the threshold values of the numeric fields. Data cleaning and transformation steps are discussed here.

- Cleaning label column: The columns contains three types of labels as shown in Figure 4.11, Data label "(empty) Malicious C&C" is heavily underrepresented and this project is not considering different types of malicious file; therefore, it has been removed from the dataset.

```
# Check the frequency - of the individual labels in the bro file
bro_df['tunnel_parents   label   detailed-label'].value_counts()
```

```
(empty)   Malicious   PartOfAHorizontalPortScan    539465
(empty)   Benign   -                               469275
(empty)   Malicious   C&C                               8
Name: tunnel_parents   label   detailed-label, dtype: int64
```

Figure 4.11: Constituents of the label column

- Cleaning columns "orig_bytes" and "resp_bytes": these two columns contained "-" in place of 0. As per convention, these columns are numeric in nature (integer); Thus, "-" needs to be replaced by 0 to preserve the integrity of the column in Pandas data frame. The code shows in Figure 4.12.

```
# Replacing - with 0
clean_df["orig_bytes"].replace({"-": 0}, inplace=True)
clean_df["resp_bytes"].replace({"-": 0}, inplace=True)
```

Figure 4.12: Replacing - with 0

- Data transformation: there are three different data transformation instances in this project.

1- Label column is converted from string "Benign" and "Malicious" to 0 and 1. This conversion process achieved by using labelEncoder( ) function, which converts labels into numeric from as shown in figure 4.13. This function is available with SciKit-learn library in Python [37].

```
# Using label encoding to code the  d Benign as 0. New flag variable is named mal_flag
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
clean_df['mal_flag']= le.fit_transform(clean_df['tunnel_parents   label   detailed-label'])
```

Figure 4.13: Label encoding to convert Benign to 0 and Malicious to 1

2- Continues numeric fields are converted to group data (quantization) using quantile-based partitioning (binning) to increase the efficiency of the classification algorithms such as Random Forest, Support Vector Machine and Logistic Regression [5]. Figure 4.14 shows the conversion of continues numeric variables.

```
# Convert variables - Converting continious variables to bin numerical data into groups (Normalizing)
clean_df['duration_cat'] = pd.qcut(clean_df['duration'].rank(method='first'), q=20, labels=False )
clean_df['orig_bytes_cat'] = pd.qcut(clean_df['orig_bytes'].rank(method='first'), q=10, labels=False )
clean_df['resp_bytes_cat'] = pd.qcut(clean_df['resp_bytes'].rank(method='first'), q=10, labels=False )
clean_df['orig_pkts_cat'] = pd.qcut(clean_df['orig_pkts'].rank(method='first'), q=10, labels=False )
clean_df['orig_ip_bytes_cat'] = pd.qcut(clean_df['orig_ip_bytes'].rank(method='first'), q=10, labels=False )
```

Figure 4.14: Conversion of continuous variables

3- Duration appears in timedelta (different from a reference time) format in Bro log files; therefore, it has converted into comprehendible format such as seconds as shown in Figure 4.15.

```
# converted time from timedelta to comprehendible format such as seconds.
clean_df['duration']=clean_df['duration'] / np.timedelta64(1, 's')
```

Figure 4.15 Time Conversion

4- Creating dummies variable to convert categorical values to numeric by using get_dummies() function, which is one of OneHotEncoding methods in python. This function takes an input with type object(categorical) and returns a vector for each unique value of the categorical column. OneHotEncoding method is used for the features '***Proto" and "conn_state"*** as shown in Figure 4.16

```
# Dummy variables-
#Creating dummy variables for the categorical variable conn_state and save in new dataframe named clean_df2
clean_df2=pd.get_dummies(clean_df, columns=['proto','conn_state'])
```

Figure 4.16 Dummies variables Conversion

4.3.4 Train and Test Split

Machine Learning is built on the premise that it trains (learns the pattern) itself on a dataset known as training dataset and then makes predictions on another dataset, commonly known as test dataset. The training dataset is usually more extensive than the test dataset, so that it satisfies the statistical test and avoid fluctuations of sampling. Both the training and the test dataset should have similar statistical properties. Train-test split is done to ensure that the model works with considerable accuracy even on unseen data.

There are three Machine Learning algorithms with three different variations of one model in this project, a test dataset with 25% size of the sample is chosen for all the models. Train-test split is achieved by Sci-Kit learn's train_test_split method from model_selection package. The code snipped to obtain this task is shown in Figure 4.17.

```
# Import train_test_split function
from sklearn.model_selection import train_test_split
# Split dataset into features and labels
y=clean_df2['mal_flag'] # Labels
X=clean_df2[['proto_tcp','proto_udp','orig_ip_bytes_cat','orig_pkts_cat','duration_cat','proto_icmp','conn_state_OTH',
             'conn_state_S0']]  # Features

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25) # 75% training and 25% test
```

Figure 4.17: Generating train-test and Test Set Split

4.3.5 Feature Importance and Feature Selection

The process of scoring the features to understand its relative importance amongst a large number of features is known as feature importance. It ensures that the dimensions (features) are reduced without losing any significant information from the data. Feature selection enables the classifiers model to perform faster training, reduces complexity, eases the interpretation, and improve accuracy.

This project involves two different feature importance and selection algorithms, one using Random Forest algorithm and the other using recursive feature selection.

Sci-kit learn's Random Forest has a property called feature_importances_ function which used for Tree-based classifiers. It returns an n-dimensional array with normalized importance score, the position of an item in the array follows the index of the feature list [3]. Random Forest based feature selection is shown in Figure 4.18. Also, in the upcoming chapter "result and discussion" the result of this process will be discussed in more details.

```
# Run the Random Forest Classification model to get variable importance first
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)
y_pred=clf.predict(X_test)

# Print the variable importance
import pandas as pd
feature_imp = pd.Series(clf.feature_importances_, index=['duration_cat','orig_bytes_cat','resp_bytes_cat','orig_pkts_cat',
                                                         'orig_ip_bytes_cat','resp_pkts_cat','resp_ip_bytes_cat','proto_icmp',
                                                         'proto_tcp','proto_udp','conn_state_OTH','conn_state_REJ',
                                                         'conn_state_RSTO','conn_state_RSTOS0','conn_state_RSTR',
                                                         'conn_state_RSTRH','conn_state_S0','conn_state_S1','conn_state_S2',
                                                         'conn_state_SF','conn_state_SH']).sort_values(ascending=False)
```

Figure 4.18: Feature importance using Random Forest algorithm

Another powerful feature selection by feature elimination process is Recursive Feature Elimination (RFE). It uses an estimator that allocates weights to features (in this case coefficients of logistic regression), the objective of recursive feature elimination algorithm is to choose set of features by iteratively considering smaller and smaller sets of features.

According to Scikit Learn, first, the estimator is trained on the full set of features, and essential features are concerned. Then, the least important features are dropped (pruned) from the full set of features. This process will be repeated recursively on the pruned set till wanted number of features to choose is finally reached, which is either passed as a number of features to be selected or determined based on a threshold value [6]. Scikit Learn's RFE module generates feature ranking as an array where rank i corresponds to the ranking position of the i-th variable. All chosen features are assigned rank 1. Figure 4.19 shows the implementation of RFE feature extraction process used for building the logistic regression classifier in the project.

```
# Feature selection - Recursive Feature Elimination for variable selection for Logistic Regression
from sklearn.feature_selection import RFE
array=clean_df.values
X = array[:,0:20]
Y = array[:,21]
# feature extraction
model = LogisticRegression(solver='lbfgs',max_iter=1000)
rfe = RFE(model, 5)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

Figure 4.19: Scikit Learn's Recursive Feature Elimination

4.3.6 Training and Evaluating the Model:

In most data science projects, multiple Machine Learning algorithms are tested to identify the one which fits the given data the best and remains fairly stable over repeated implementations of the algorithm. The repetition process increases the model's generalisation and reduces bias. In this project, three separate Machine Learning algorithms have been implemented, each trained on the clean dataset with the most important feature set specially created for the algorithm.

This project revolves around classifying network traffic into two classes Benign and Malicious using supervised learning classifiers; therefore, the performance of the models is evaluated by comparing performance metrics for each classifier. There are in total of five performance metrics have been considered which are (1) Accuracy, (2) Precision (3) Recall (4) F1-Score and (5) Area Under the Curve (AUC). All performance metrics were utilised using Scikit Learn's package with base class "metrics" function, which generates different performance metrics functions named as shown below [Scikit]:

- sklearn.metrics.accuracy_score( ): In classification, this function calculate subset accuracy

- sklearn.metrics.precision_score ( ) : Computes the precision, the capability of the classifier to not label a negative sample as positive.
- sklearn.metrics.recall_score ( ): Computes the recall, the capability of the model to find all the positive samples.
- sklearn.metrics.f1_score ( ): Compute the F1 score. The F1 score is the average of precision and recall. The relative participation of precision and recall to the F1 score are equal.
- metrics.roc_auc_score ( ): calculates Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

These model performance metrics are generated for each algorithm and stored in a dataset. An empty Pandas data frame is created to store the specific metric per model as shown in Table 4.3 The dataset has 11 columns. One column to save the model name and the remaining are to store the metric described above, one score for test dataset and one for train dataset.

	Model name	Accuracy - Train	Accuracy - Test	Precision - Train	Precision - Test	Recall - Train	Recall - Test	F1-Score - Train	F1-Score - Test	AUC - Train	AUC - Test
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 4.3: Blank Pandas Data Frame to store performance metrics

Implementation of Scikit Learn's metric function to generate performance metrics of the classifiers and storing it in the Pandas dataframe is shown in Figure 4.20.

.

```
#Import scikit-learn metrics module to calculate model accuracy
from sklearn import metrics
# Generating the performance metrics and storung in a dataframe perf_metric
perf_metric.iloc[0,0]='Random Forest Classifier'
perf_metric.iloc[0,1]=metrics.accuracy_score(y_train,y_pred_train)
perf_metric.iloc[0,2]=metrics.accuracy_score(y_test,y_pred)
perf_metric.iloc[0,3]=metrics.precision_score(y_train,y_pred_train,average='binary',pos_label=1)
perf_metric.iloc[0,4]=metrics.precision_score(y_test,y_pred,average='binary',pos_label=1)
perf_metric.iloc[0,5]=metrics.recall_score(y_train,y_pred_train,average='binary',pos_label=1)
perf_metric.iloc[0,6]=metrics.recall_score(y_test,y_pred,average='binary',pos_label=1)
perf_metric.iloc[0,7]=metrics.f1_score(y_train,y_pred_train,average='binary',pos_label=1)
perf_metric.iloc[0,8]=metrics.f1_score(y_test,y_pred,average='binary',pos_label=1)
perf_metric.iloc[0,9]=metrics.roc_auc_score(y_train,y_pred_train)
perf_metric.iloc[0,10]=metrics.roc_auc_score(y_test,y_pred)
```

Figure 4.20: Generating performance metrics and storing in dataframe

4.4 Classifiers

In this project, three different classifiers have been implemented with three different variations of one of the classifiers. Each module is independent of the other. All Machine Learning algorithms have been chosen based on their popularity and their successful used in malware detections.

The classifier functions are detailed as follows:

1. RandomForestClassifier is imported from sklearn.ensemble module
2. SVC is imported from sklearn.svm module (Linear kernel, RBF kernel, Sigmoid kernel)
3. LogisticRegression is imported from sklearn.linear_model module

4.4.1 Implementing Random Forest Classifier

Random Forest algorithm is the supervised learning classifier used to classify network traffic into Benign or Malicious. A random forest is a meta-estimator that fits a multiple of decision tree classifiers on different sub-samples of the data set and use the average to increase the prediction of accuracy score and reduce overfitting [7]. To implement random forest model, RandomForestClassifier() is used, which is an ensemble technique in Scikit-learn that takes a parameter called n_estimator.

Hyperparameters for Random Forest Classifier used in the project are n_estimators = 100, which is also the default to represent the tree numbers [34]. The model is fitted on training dataset and predictions are generated for the test as well as train dataset exclusively to test the performance across train and test dataset. Complete implementation of the steps is shown in Figure 4.21.

```
# Running the final Random Forest Classification after removing unimportant variables
import time
start= time.process_time()

from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

# prediction on the training set
y_pred_train=clf.predict(X_train)

# prediction on test set
y_pred=clf.predict(X_test)
```

Figure 4.21: Implementation of Random Forest Classifier

4.4.2 Implementing Support Vector Machines Classifier

Support Vector Machines Classifier (SVC) is the second type of supervised machine Learning model utilised in the project. The most important hyper parameter for SVC is the kernel selection which determines the shape of the decision boundary [30]. Given the data structure which comprises mostly of categorical data, it is important to try non-linear kernels like **Sigmoid** and **Gaussian (RBF)** kernel in addition to **linear** kernel. Thus, the hyper parameter tuning trailed in this project depends entirely on selecting the kernels. All other hyper parameters are left to their default values.

The model is fitted on training dataset and predictions are generated for test as well as train dataset exclusively to test the performance across train and test dataset. The implementation of linear kernel is shown in Figure 4.22, for the other two kernel's the implementation is the same with the only changing in (kernel='linear').

```
# Running the final Random Forest Classification after removing unimportant variables
import time
start= time.process_time()

from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

# prediction on the training set
y_pred_train=clf.predict(X_train)

# prediction on test set
y_pred=clf.predict(X_test)
```

Figure 4.22: Implementation of SVC with linear kernel

4.4.3 Implementing Logistic Regression Classifier

Logistic Regression Classifier is the third type of supervised machine Learning model utilised in the project. To implement Logistic Regression model, LogisticRegression() is used, which is a linear model technique in Scikit-learn [37]. Since the dataset does not suffer from class imbalance, and the dataset is large enough to rule out fluctuations of sampling, there is no need for specific class weights to be assigned. Therefore, for hyper parameters, default settings are allowed. The implementation of the logistic regression classification is shown in Figure 4.23. Similar to the other classifiers, the model is fitted on training dataset and predictions are generated for the test as well as train dataset exclusively to test the performance across train and test dataset.

```python
# Fitting the final logistic regression model
from sklearn.linear_model import LogisticRegression
#create an instance and fit the model
logmodel = LogisticRegression()
logmodel.fit(X_train, y_train)

# prediction on the training set
y_pred_train=logmodel.predict(X_train)

# Generating the prediction accuracy
Predictions = logmodel.predict(X_test)
```

Figure 4.23: Implementation of the logistic regression classifier

Chapter 5: System Testing

This chapter includes test cases that were utilised to guarantee that the solution provided satisfies the requirements and to demonstrate how it works as intended. All the test cases presented have been carried out on Mac operation system device version 10.12.6 in **Anaconda** navigator using **jupyter** notebook.6.01 and **Python** version is 3.7.4

5.1 Test case

Test Case ID: TC-0	Loading data set			
Precondition: None				
Test case step: 2				
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**	**Comment**
1	In ***Reading file cell*** Input the path of the file directory (any kind of file)	a message from Jupiter appears "successfully monitoring'	**Fail**	A message 'error' appears since it entered a .pcap file
2	In ***Reading file cell*** Input the path directory of a new bro log file to read it and save it to a data fame	a message from Jupiter appears "successfully monitoring'	**Pass**	the script accepts only bro log file format
Comment: None				
Related test: None				

Table 5.1: Loading data set (TC-0)

Test Case ID: TC-1	Loading and parsing the data set		
Precondition: pro log file			
Test case step: 4			
Step No	**Procedure**	**Expected Response**	**Pass/ Fail**
1	In ***Reading file cell*** Input the path of the file directory to read it and save it to a data fame	a message from Jupiter appears "successfully monitoring'	**pass**
2	Run ***Check data frame cell*** to see what data is in the file	Few lines of the data frame shown with name of columns and row	**Pass**
3	Run ***Check the frequency cell*** to see the number of malicious and benign data	The number of malicious and benign row in the data are shown	**Pass**
4	Run ***save to csv cell*** with writing the path directory of the new csv	A csv file created and save in the directory provided	**Pass**
Comment: all these cells will be included in the first section of Jupiter file. Section name is #Section 1: Reading bro log file			
Related test: None			

Table 5.2: Loading and parsing the data set (TC-1)

Test Case ID: TC-2	Exploring Data Analysis step to explore and visualize data set variables		
Precondition: an existing data set that was previously saved to a data frame by running TC-1			
Test case step: 3			
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**
1	Run ***Label code cell*** to add a column to the data frame that has value 1 for malicious and 0 for Benign	A new Column added to the data frame and by running ***view cell*** all variables in the data frame is shown with the new columns	**Pass**
2	Run ***Replacing cell*** to change value with (-) to a number 0 in the specified columns	Values (-) is changed to 0	**Pass**
3	Run each cell under ***(Start of EDA) section*** to see the distribution of each variable in the data frame	A bar graph shown the distribution of each column in the data set	**Pass**
Comment: all these cells will be included in the second section of Jupiter file. Section name is #Section 2: Exploratory Data Analysis (EDA).			
Related test: TC-1			

Table 5.3: Exploring Data Analysis step to explore and visualize data set variables (TC-2)

Test Case ID: TC-3	Pre-processing the data to prepare it for the classifiers		
Precondition: an existing data set that was previously saved to a data frame by running TC-1, and the first two steps from TC-2			
Test case step: 3			
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**
1	Run ***drop the column cell*** with entering variable's names that you want to drop	All specified variables are removed from the data frame. To check, run ***view cell,*** all specified variables are removed from the data frame	**Pass**
2	Run ***Convert variables cell*** to bin numerical data into groups (Normalizing), with entering variable's names that you want to convert	New columns added for each specified variable with normalized values. To check this run the ***view cell,*** all variables are shown	**Pass**
3	Run ***Dummy variables cell*** to convert categorical variables to a form of data that can be provided to ML algorithms. Enter variable's names that contain categorical values	New columns added to the data frame for each categorical variable. To check this run the ***view cell,*** all variables are shown	**Pass**
Comment: all cells will be included in the third section of Jupiter file. Section name is #Section 3: pre-processing step			
Related test: TC-1, TC-2			

Table 5.4: Pre-processing the data to prepare it for the classifiers (TC-3)

Test Case ID: TC-4	Creating, training, testing **Random Forest**			
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, and pre-processing step TC-3				
Test case step: 1				
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**	**Comment**
1	Running all cells under ***Model 1: Random Forest Classifier section***	No error message appears	**Pass**	All results will be shown later in (TC-11)
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, under Model 1: Random Forest.				
Related test: TC-1, TC-2, and TC-3				

Table 5.5: Creating, training, testing Random Forest (TC-4)

Test Case ID: TC-5	Creating, training, testing **Support Vector Machine (Linear kernel)**			
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, and pre-processing step TC-3				
Test case step: 1				
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**	**Comment**
1	Running all cells under ***Model 2: Support Vector Machin – Linear kernel section***	No error message appears	**Pass**	All results will be shown later in (TC-11)
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, under Model 2: Support Vector Machine – Linear kernel.				
Related test: TC-1, TC-2, and TC-3				

Table 5.6: Creating, training, testing Support Vector Machine (Linear kernel) (TC-5)

Test Case ID: TC-6	Creating, training, testing **Support Vector Machine (RBF kernel)**			
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, and pre-processing step TC-3				
Test case step: 1				
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**	**Comment**
1	Running all cells under ***Model 2: Support Vector Machin – RBF kernel section***	No error message appears	**Pass**	All results will be shown later in (TC-11)
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, under Model 3: Support Vector Machine – RBF kernel.				
Related test: TC-1, TC-2, and TC-3				

Table 5.7: Creating, training, testing Support Vector Machine (RBF kernel) (TC-6)

Test Case ID: TC-7	Creating, training, testing **Support Vector Machine (Sigmoid kernel)**			
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, and pre-processing step TC-3				
Test case step: 1				
Step No	**Procedure**	**Expected Response**	**Pass/ Fail**	**Comment**
1	Running all cells under ***Model 2: Support Vector Machin – Sigmoid kernel section***	No error message appears	**Pass**	All results will be shown later in (TC-11)
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, under Model 4: Support Vector Machine – Sigmoid kernel.				
Related test: TC-1, TC-2, and TC-3				

Table 5.8: Creating, training, testing Support Vector Machine (Sigmoid kernel) (TC-7)

Test Case ID: TC-8	Creating, training, testing **Logistic Regression**			
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, and pre-processing step TC-3				
Test case step: 1				
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**	**Comment**
1	Running all cells under ***Model 3: Logistic Regression***	No error message appears	**Pass**	All results will be shown later in (TC-11)
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, under Model 5: Logistic Regression.				
Related test: TC-1, TC-2, and TC-3				

Table 5.9: Creating, training, testing Logistic Regression (TC-8)

Test Case ID: TC-9	Feature Selection step for **Random Forest**		
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, pre-processing step TC-3, and Random Forest model TC-4			
Test case step: 1			
Step No	**Procedure**	**Expected Response**	**Pass/ Fail**
1	Running ***feature selection cells*** within ***Model 3: Logistic Regression***	A graph shown to visualize variable importance	**Pass**
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, within Model 1: Random Forest.			
Related test: TC-1, TC-2, TC-3, and TC-4			

Table 5:10: Feature selection step for Random Forest (TC-9)

Test Case ID: TC-10	Feature Selection step for **Logistic Regression**		
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, pre-processing step TC-3, and Logistic Regression model TC-8			
Test case step: 1			
Step No	**Procedure**	**Expected Response**	**Pass/ Fail**
1	Running two cells ***feature selection*** within ***Model 3: Logistic Regression***	Number of important features appear with the selected feature and ranking	**Pass**
Comment: all cells will be included in the 4th section of Jupiter file. Section name is #Section 4: ML Models, within Model 5: Logistic Regression.			
Related test: TC-1, TC-2, TC-3, TC-8			

Table 5.11: Feature Selection step for Logistic Regression (TC-10)

Test Case ID: TC-11	calculating the results of all classifiers		
Precondition: an existing data set that was previously saved to a data frame by running TC-1, the first two steps from TC-2, pre-processing step TC-3, Finally, all ML models step TC-4, TC-5, TC_6, TC-7, and TC,8			
Test case step: 7			
Step No	**Procedure:**	**Expected Response**	**Pass/ Fail**
1	Run ***performance metric table cell*** to see accuracy results for all classifiers	Table shows with accuracy scores for all classifiers	**Pass**
2	Run ***accuracy plot cell*** to see accuracy results for all classifiers in a graph	A bar graph shows with accuracy scores for all classifiers	**Pass**
3	Run ***Precision plot cell*** to see accuracy results for all classifiers in a graph	A bar graph shows with precision scores for all classifiers	**Pass**
4	Run ***Recall plot cell*** to see recall results for all classifiers in a graph	A bar graph shows with recall scores for all classifiers	**Pass**
5	Run ***F1-score plot cell*** to see F1-score results for all classifiers in a graph	A bar graph shows with F1-score for all classifiers	**Pass**
6	Run ***AUC plot cell*** to see AUC score results for all classifiers in a graph	A bar graph shows with AUC scores for all classifiers	**Pass**
7	Run ***Confusion Metrics cell*** to see confusion metrics results for all classifiers	A confusion metrics plot shows	**Pass**
Comment: all these cells will be included in the last section of Jupiter file. Section name is #Section 5: Evaluation.			
Related test: TC-1, TC-2, TC-3, TC-4, TC-5, TC-6, TC-7, and TC-8.			

Table 5.12: Calculating the results of all classifiers (TC-11)

Chapter 6: Results and Discussion

Successful execution of the three machine learning classifiers with three different variations of one of the classifiers enables us to analyse the output of the models critically. The end result of classification algorithms is to predict the labels; therefore, the performance measures are analysed and compared amongst the models. In addition to the assessment of the final outcome, feature importance is also analysed and compared. These two aspects have been shown and discussed in this section.

6.1 Classifier Performance

This section exhibits the importance attributed to each input variables as determined by non-parametric Random Forest algorithm and by parametric Logistic Regression followed by evaluation of performance metrics for each classifier. Performance metrics considered are accuracy, recall, precision, F1-Score and AUC. Classification accuracy and misclassification cases are summarised by way of confusion matrix.

6.1.1 Variable importance assessment

Variable importance refers to the extent to which a model relies upon a variable to make accurate predictions. The more a model relies on a variable to make predictions, the more important it is for the model. There are two algorithms in the project where variable importance generated.

1) Random Forest based variable importance used for variable selection in the final Random Forest model and Support Vector Machines
2) Logistic Regression based Recursive Feature Elimination used for variable selection in final Logistic Regression model

Random Forest is a forest of decision trees. In the purpose of this project, Default Scikit-learn's feature importance is used which calculated as the decrease in node impurity (Gini impurity) weighted by the probability of reaching that node [16]. As discussed in section 2.2.1, Gini impurity is the probability of wrongly classifying a random item according to the distribution of the class labels of the dataset [9]. The decrease in Gini indicates higher variable importance

and higher the value, the more important is the feature. Figure 5.1 shows the variable importance chart as per Random Forest Classifier.

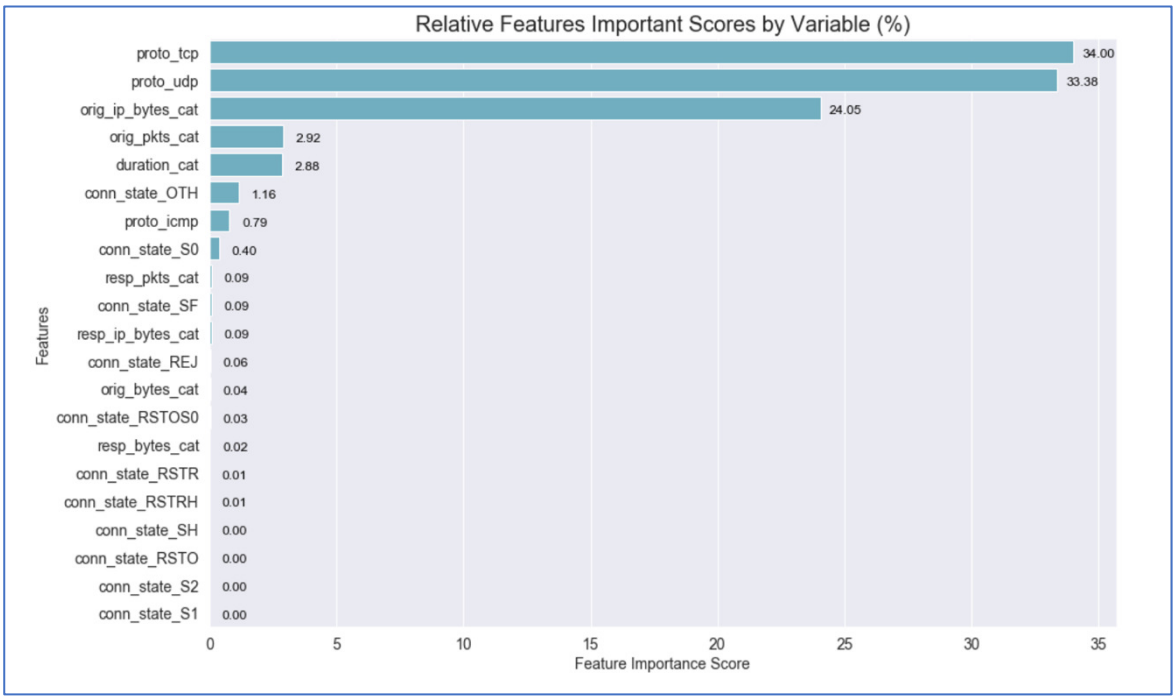

Figure 5.1: Variable importance from Random Forest Classifier

Moreover, logistic regression uses dominance analysis where importance is defined as a quantitative comparison between pairs of predictors [2]. The importance of predictor increases than the other predictor, if it has more contribution to the prediction of the response variable in all possible subset models; in other words, all possible combinations of predictors and that predictor called dominant predictor [ibid]. Figure 5.2 shows the output generated by recursive feature elimination with logistic Regression.

```
Num Features: 5
Selected Features: [False False  True False False False False  True  True  True False False
  True False False False False False False False]
Feature Ranking: [ 3 13  1 15  6  7  4  1  1  1 12 14  1  8  2  5 11 16  9 10]
```

Figure 5.2: Recursive Feature Elimination output

Output format

Feature importance using Random Forest Classifier generates relative importance scores where all variables added to the new Random Forest classifier up to 1 or 100% score (refer to Figure 5.1). For example, *proto_tcp, proto_udp. orig_ip_bytes_cat, orig_pkts_cat, duration_cat, proto_icmp, conn_state_OTH,* and *conn_state_S0 which is* scores 0.47 so it considered as an important feature to Random Forest Classifier. While all the remaining variables like,

conn_state_SF, resp_ip_bytes_cat, conn_state_REJ, resp_pkts_cat, conn_state_RSTOS0, orig_bytes_cat, resp_bytes_cat, conn_state_RSTRH, conn_state_RSTR, conn_state_SH, conn_state_S2, conn_state_s1, and conn_state_RSTO removed where all variables score less than 1% according to the graph generated by feature importance function.

In logistic regression, using Recursive feature elimination RFE function from Scikit Learn package generates feature ranking; for instance, ranking_i corresponds to the ranking position of the ith feature. All chosen features are estimated best features without any order, and all are assigned as rank 1 without more specification of which one ranks higher than others.

Comparison of the result

Variables such as *orig_pkts, proto_tcp, proto_udp, conn_state_OTH, conn_state_RSTOS0* and *duration_sec* are considered of high importance in both the models but ***orig_ip_bytes*** is considered third most important variable as per Random Forest but in recursive feature elimination using logistic regression it ranks as the least importance(rank 16th). This difference can be explained using the distribution of orig_ip_bytes (as shown in 4.3.2 point 6), malicious traffic has highly skewed distribution, all towards the left around the range of 25-50 bytes per traffic whereas huge majority of benign traffic are in the adjacent byte range of 50-75. Since huge majority of the traffic for malicious and benign are in short range, the variable is not considered to be an important one with the power to discriminate between malicious and benign traffic.

6.1.2 Accuracy of the classifiers

Accuracy is defined as the fraction of correct predictions amongst the total number of cases. The scores of classifiers' accuracy for train and test data set is shown in Figure 5.3.

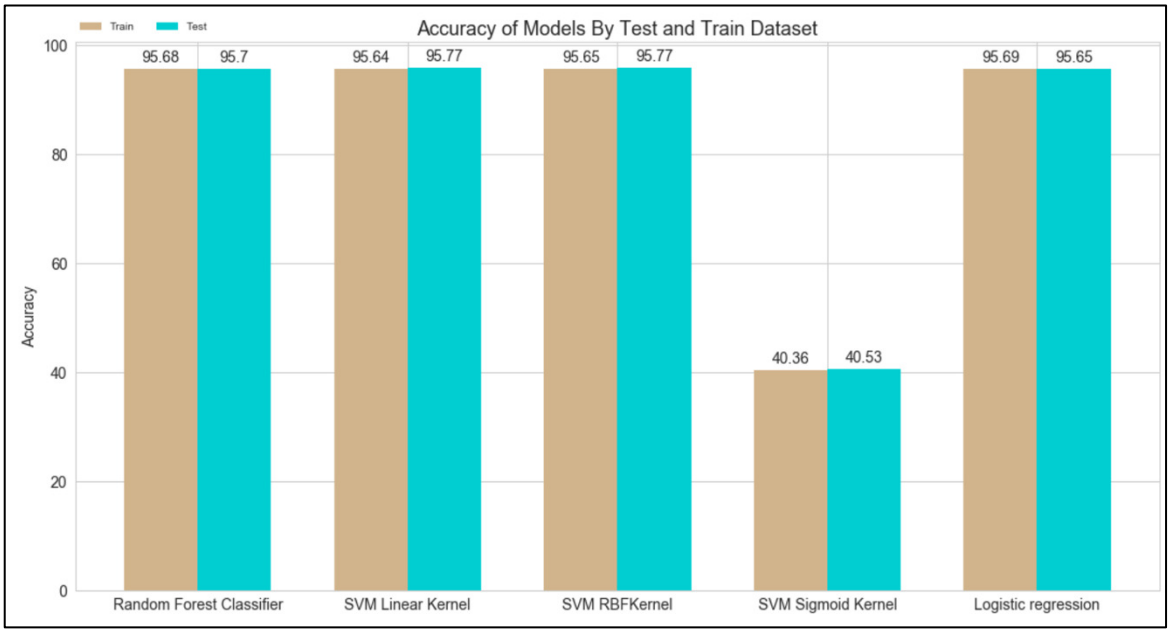

Figure 5.3: Comparison of accuracy scores by classifiers

The value of accuracy for all models except SVM Sigmoid Kernel are very similar to each other and achieved more than 95% for both train and test dataset. SVM Linear Kernel model performs better than the comparable models. While SVM Sigmoid Kernel performed abysmally and achieved an accuracy of around 40% in the train as well as in the test dataset. Although Sigmoid kernel is widely used, its properties are not fully studied [29].

6.1.3 Precision of the classifiers

Precision, as discussed in previously, is the ability to not label positive sample as negative and it calculated as the number of true positives divided by the total number of true positives and false positives. The result 0.0 for no precision and 1.0, which is the highest value for full or perfect precision. Analysis of precision score suggests that although most models achieved high precision scores for detecting malicious network traffic for the selected dataset, SVM linear kernel slightly outperformed other classifiers in the test set. Thus, high precision here

means a lower chance of labelling a benign file as malicious. Figure 5.4 compares the precision scores of the classifier models.

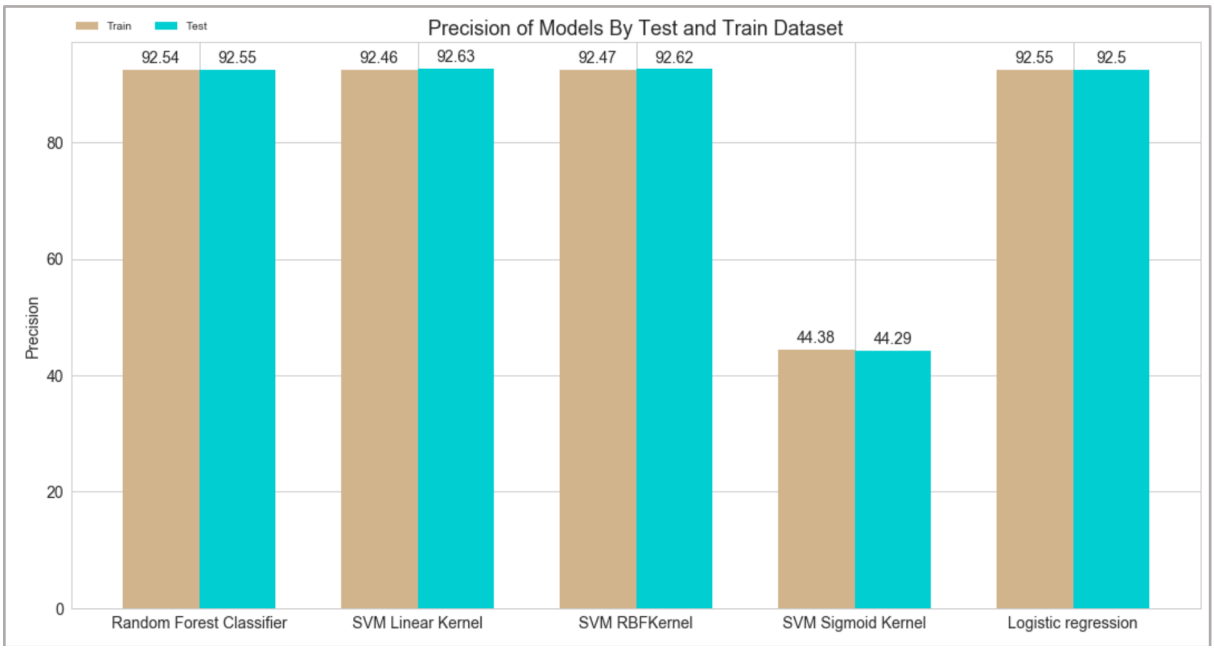

Figure 5.4: Comparison of precision scores by classifiers

6.1.4 Recall value of the classifiers

The recall, as discussed previously, is the ability of the classifier to predict all the positive samples in the dataset. Analysis of the recall scores suggests that all the models except SVM Sigmoid Kernel are doing well in classifying actual positive as positive in the chosen data set especially SVM linear kernel and SVM RBF kernel both achieved the full score in the test set. In the context of the project, positive labels are used to flag malicious traffic and hence high recall means that chances of not flagging an actual malicious traffic as malicious is very low. Figure 5.5 compares recall score across classification model.

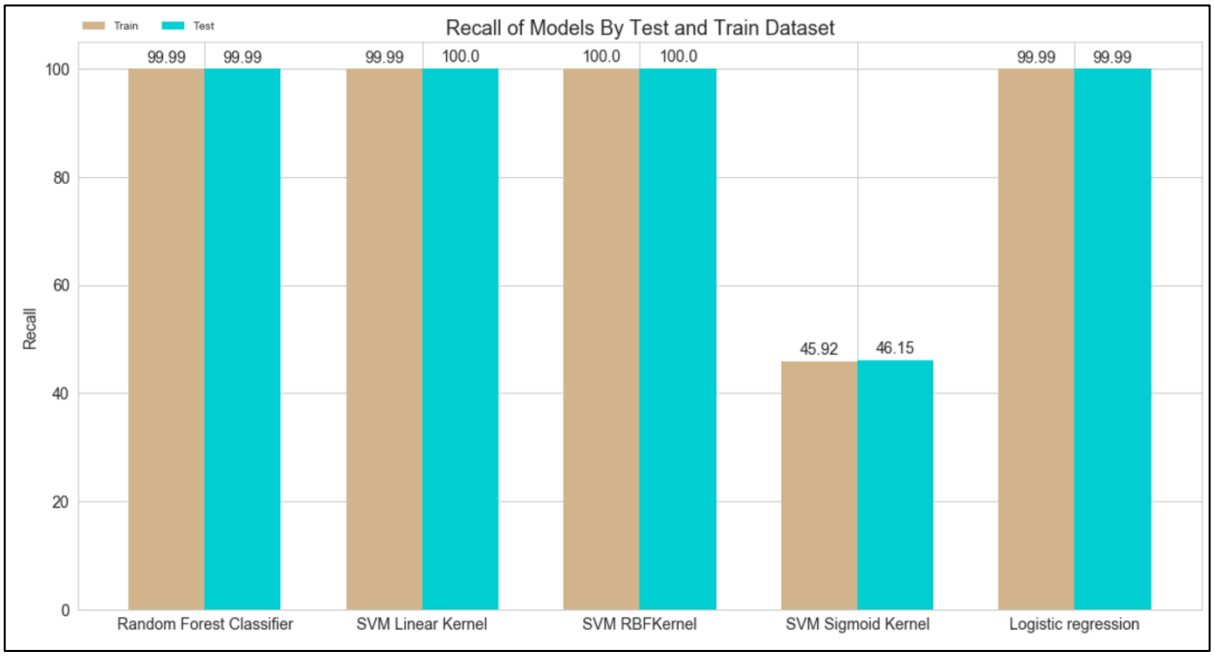

Figure 5.5: Comparison of recall score across model

6.1.5 F1-Score of the classifiers

Thus F-Score is a combined value between precision and recall, where the better value of F1 is 1, and the poor value is 0. The contribution of precision and recall to the F1 score are equal. Higher F1-Score means the higher chance of predicting actual positives as positives and low probability of predicting real negative as positive. All models except SVM Sigmoid Kernel has relatively high F1-Score (around 96%). Amongst models with higher F1-Score, performance on the test dataset is better than on training dataset. Figure 5.6 compares F1-Score among various classifier model.

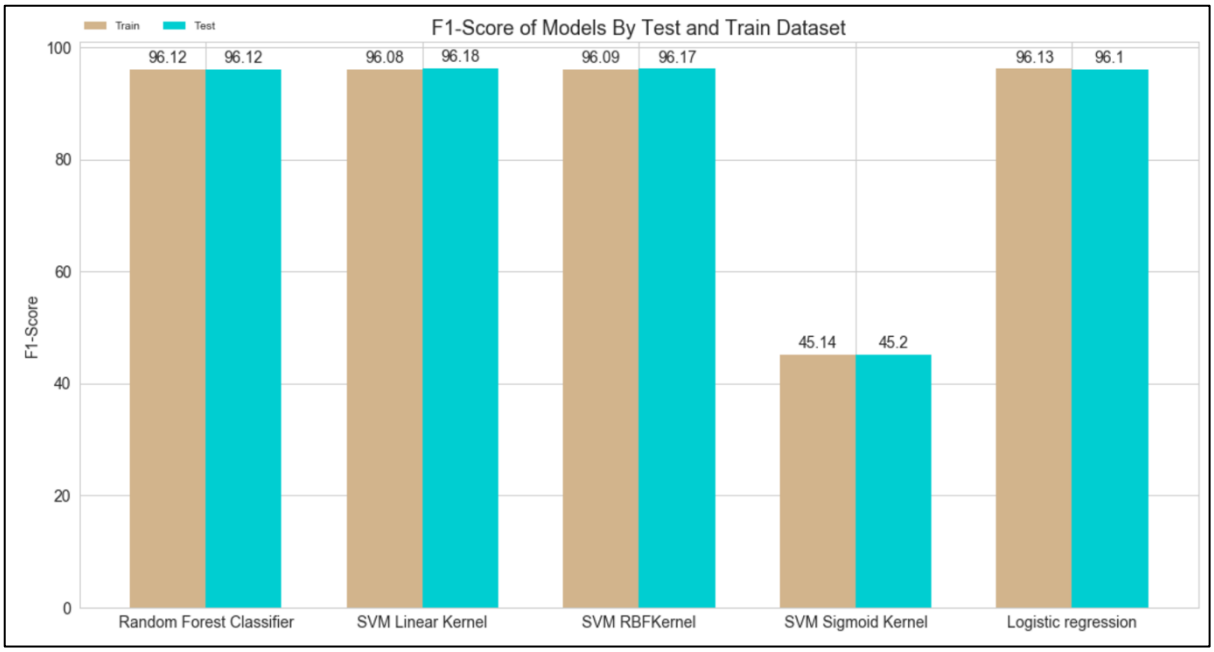

Figure 5.6: Comparison of F1-Score by classifiers

In the purpose of this project, F1-Score considers as more important than accuracy because accuracy contributed by a large number of True Negatives and True Positive and both are not as important as F1-Score. False Negatives and False Positives are important. An example of False Negative is when a file is malicious, and the classifier indicates as not malicious, this could have a negative consequence [38].

6.1.6 AUC of the classifiers

AUC, as discussed previously, is the area under the ROC curve. This curve plots two parameters which are True Positive Rate and False Positive Rate [8]. Therefore, ROC is the curve and AUC is the measure of separability, which shows the capability of the classifier to differentiating between classes. Higher the AUC score is 1, which represent better classifier. Although all the models (except SVM Sigmoid Kernel) has similar and high degrees of AUC score, SVM Liner Kernel marginally outperformed other models in this regard. Figure 5.7 compares the AUC score across all the models.

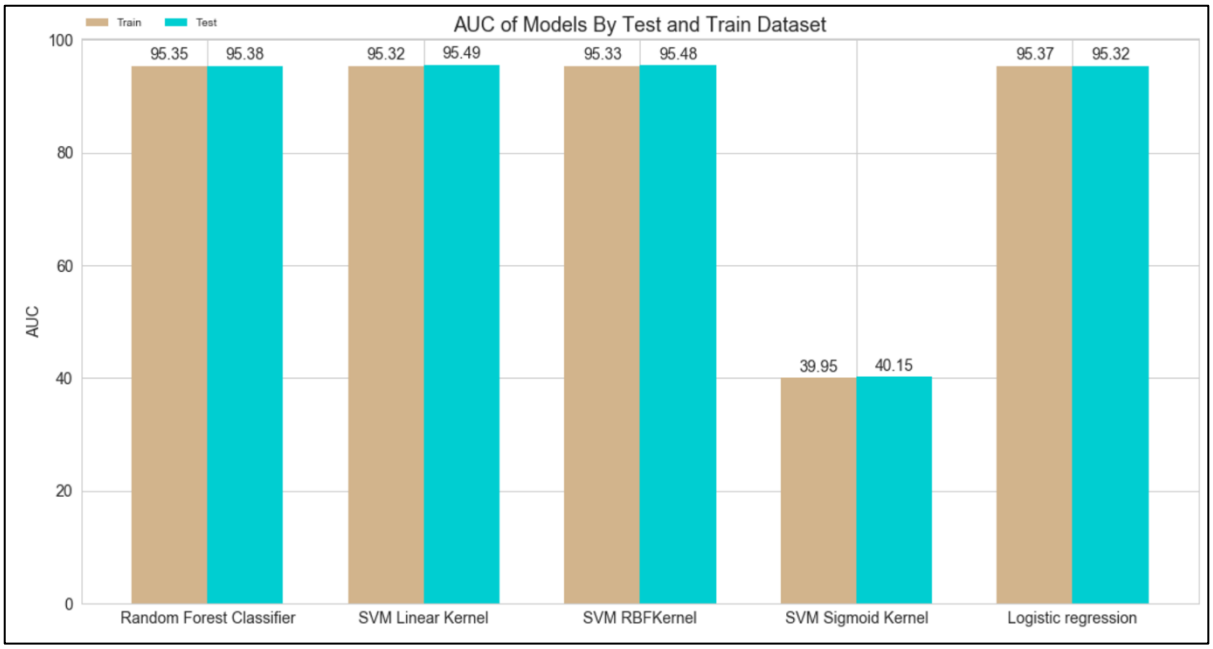

Figure 5.7: Comparison of AUC score across models

6.1.7 Classifiers Performance Summary

Standalone comparison of performance metrics for the all models have been considered in the preceding section. In this section, all the performance metrics are combined together at one place to identify the best performing model as shown in Table 5.1.

	Model name	Accuracy - Train	Accuracy - Test	Precision - Train	Precision - Test	Recall - Train	Recall - Test	F1-Score - Train	F1-Score - Test	AUC - Train	AUC - Test
0	Random Forest Classifier	0.956955	0.956484	0.925537	0.924973	0.999911	0.999874	0.961287	0.960966	0.953773	0.953146
1	SVM Linear Kernel	0.95564	0.960111	0.923313	0.930592	0.999851	1	0.960059	0.964048	0.952494	0.957126
2	SVM RBFKernel	0.955667	0.959952	0.923239	0.930335	1	1	0.960088	0.963911	0.952512	0.956955
3	SVM Sigmoid Kernel	0.383218	0.386836	0.421372	0.425174	0.419931	0.416222	0.42065	0.42065	0.380606	0.384637
4	Logistic regression	0.955614	0.960111	0.923232	0.930592	0.999901	1	0.960038	0.964048	0.952462	0.957126

Table 5.1: Summary of performance metrics for all models

Analysis of Table 5.1 shows that SVM Linear Kernel model outperforms all others. The train set and test set performance metric are also better than others in most cases.

Visual inspection of three confusion matrix as shown in Figure 5.8 for Random Forest classifier, best fit SVM model – the linear kernel and Logistic Regression classifier, also shows that although the False Positive count is slightly higher than others, false negative rate is 0. False Negative is costlier and more important than false positive in Malware detection as it is dangerous to indicates malicious files as benign; hence, SVM Linear Kernel performs satisfactorily.

In addition, SVM Linear Kernel model can be considered as the best model; therefore, it considered as the most appropriate algorithm to solve malware detection problems.

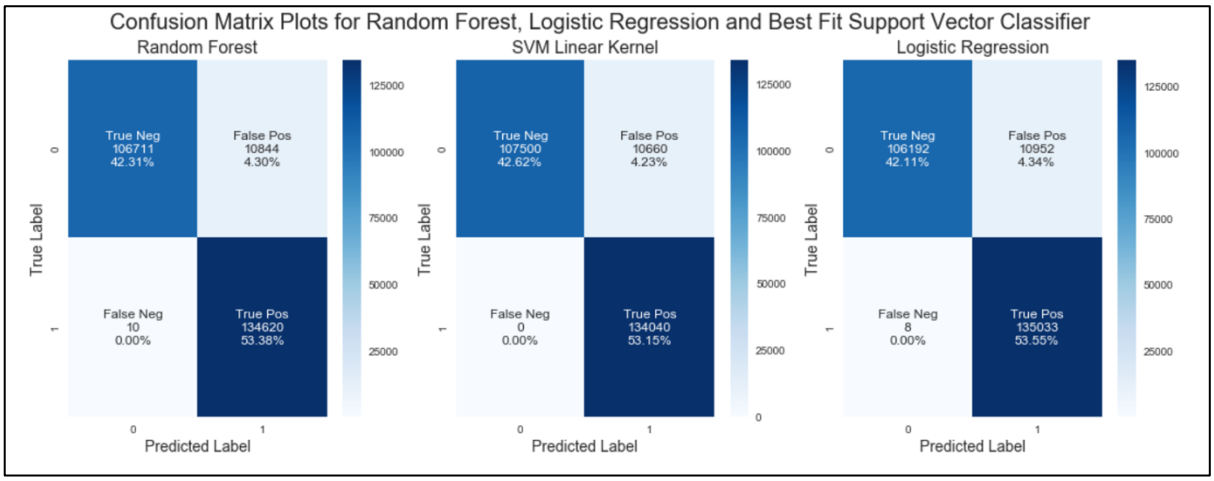

Figure 5.8: confusion matrix plots for top three model

6.2 Limitations

There are some constraints in achieving this project that will be discussed in this section. One of the main challenges is the sudden occurrence of coronavirus COVID-19 pandemic and its consequences of lockdown, difficulty of travelling and evacuation of international students have affected on my health, time, and effort toward achieving this project. Another challenge faced in this project of malware detection domain was the lack of information on how labelling proses has been made. Also, since the data set labelled manually by a human analyst, improper or erroneously labelling might occur, which leads to incorrect processing of the output result and significant threat to the entire scheme. Secondly, recency of the training dataset is also a strong consideration. Malware evolves every day; hence practical use of the prototype can be assessed only on a real time basis. Thirdly, malware is one of many types, as such, even within the malware, there is enormous variation across different attributes considered during the modelling exercise. In statistical term, malware detection is a multi-class classification problem rather than a binary classification. Additionally, malicious IoT network traffic is increasing over time and became more complex. Therefore, the process might be hard to predict by human experts as the behaviours of malware are constantly changing. Although challenges are occurred with malware detection projects, many solutions were applied and utilised as it is an interesting field for researching.

Chapter 7: Future Work

Due to the limited time allowed for this project, there are different facts of the project that could be addressed. First aspect revolves around machine learning implementations such as use of a training dataset that contains multi-class labels so that the predictions are more specific and can be extended to classify not only malicious and benign network traffic but also to categorise the type of attack occur. Moreover, testing other family of algorithms like deep learning, Natural Language Processing on the content or an ensemble model are in scope. This will help in further investigating and examining machine learning classifiers approach for detecting malicious network traffic for IoT devices. Another aspect is to develop the solution as a software system to allow interaction with users with a nicely built user interface. This can be achieved using web technologies such as, HTML with Java script to create a graphical user interface that allow interaction with users. Moreover, implementing the solution as multiple classes by breaking the code script into smaller parts, rather than writing the whole script in one as this will ease the process of building the software system. 9Final aspect is to provide a summary of all classifiers performances to the user and save the result for further analysis.

Chapter 8: Conclusion

In conclusion, throughout this report, multiple tests were performed on three different classifiers with three different variations of one of the implemented classifiers, in order to evaluate and determine the best model that can classify malicious and benign network traffic using machine learning approach. This problem was solved by implementing Random Forest, SVM Linear Kernel, SVM RBF Kernel, SVM Sigmoid Kernel, and finally Logistic Regression. All classifiers evaluated using the performance metrics indicators, including accuracy, precision, recall, F1-Score, and AUC for each classifier.

According to the results mentioned, high scores achieved for all Machine Learning models except SVM Sigmoid Kernel classifier in different experiments. Whereas, SVM Linear kernel achieved slightly the highest among all classifiers. Appropriate choice of algorithm for this specific type of the problem, systematic approach of data pre-processing, feature selection and the use of large dataset were the key to a successful implementation of Machine Learning algorithms. Furthermore, multiple algorithms need to be tested to ensure that the chosen model fares the best among other relevant models.

Chapter 9: Reflection on Learning

Throughout this project, I realised that working on the final year project was completely different than other university projects, as doing my graduation project required large amount of commitment and hard working. One of the greatest lessons I learned, is the importance of writing the report alongside with the time of implementing the solution as in my case, I started writing the report after finishing the implementation part. This cause a struggle in writing the report, since most ideas were not fresh anymore and could not be remembered as I had to go back over things I already did before and that cause a waste of the time where I believe I could use it in something useful. Moreover, my knowledge expanded in different aspects while doing this project. Firstly, in Machine learning field I gained knowledge about various techniques in machine learning as a classifier. I understand the algorithms that have been used in this solution along with familiarising myself with the libraries implemented, constructing models, pre-processing, evaluating the training and testing data, finally producing a complete piece of a report for the entire solution. Overall, I gained learning experience from undertaking this project, I have the chance to improve my coding skills and to gain confidence. Also, to develop valuable skills; for example, analysing, problem-solving, research skills, overcome obstacles and I believe all skills that have learnt from this project will be utilised in my career department which involve security aspects.

Bibliography

[1] Awad, M. and Khanna, R., 2015. *Efficient Learning Machines*. 1st ed. Berkeley, CA: Springer Nature.

[2] Azen, R. and Traxel, N., 2009. Using Dominance Analysis to Determine Predictor Importance in Logistic Regression. *Journal of Educational and Behavioral Statistics*, 34(3), pp.319-347.

[3] Beaulieu, K. and Dalisay, D., 2020. *Machine Learning Mastery*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com [Accessed 1 May 2020].

[4] Bronk, C. and Tikk-Ringa, E., 2020. The Cyber Attack on Saudi Aramco. *Survival Global Politics and Strategy*, [online] 55, 2013(2), pp.81-96 |. Available at:https://www.tandfonline.com/doi/abs/10.1080/00396338.2013.784468?journalCode=tsur20 [Accessed 12 February 2020].

[5] Bowne-Anderson, H., 2018. *Feature Engineering With Kaggle Tutorial*. [online] DataCamp Community. Available at:https://www.datacamp.com/community/tutorials/feature-engineering-kaggle [Accessed 2 May 2020].

[6] Brownlee, J., 2020. *Feature Selection For Machine Learning In Python*. [online] Machine Learning Mastery. Available at:https://machinelearningmastery.com/feature-selection-machine-learning-python/ [Accessed 1 May 2020].

[7] Brown G. 2009. Ensemble learning. C. Sammut, G. Webb (Eds.) Encyclopedia of Machine Learning, Springer [Online]. Available at: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_252 [Accessed: 7 May. 2020]

[8] Burkov, A., 2019. *The Hundred-Page Machine Learning Book*. 1st ed. Quebec City, Canada: Andriy Burkov, pp.13-20.

[9] Chio, C. and Freeman, D., 2018. *Machine Learning And Security : Protecting Systems With Data And Algorithms*. 1st ed. Beijing, China: O'Reilly.

[10] C. Parker, "An Analysis of Performance Measures for Binary Classifiers," *2011 IEEE 11th International Conference on Data Mining*, Vancouver,BC, 2011, pp. 517-526, doi: 10.1109/ICDM.2011.21.

[11] Docs.zeek.org. 2020. *Introduction — Zeek User Manual V3.1.3*. [online] Available at: https://docs.zeek.org/en/current/intro/ [Accessed 2 June 2020].

[12] Docs.zeek.org. 2020. *Base/Protocols/Conn/Main.Zeek — Zeek User Manual V3.1.3*. [online] Available at: https://docs.zeek.org/en/current/scripts/base/protocols/conn/main.zeek.html[Accessed 1 May 2020].

[13] Dr. Sebastian Raschka. 2014. *Predictive Modeling, Supervised Machine Learning, And Pattern Classification*. [online] Available at: https://sebastianraschka.com/Articles/2014_intro_supervised_learning.html [Accessed 6 May 2020].

[14] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos and P. Burnap, "A Supervised Intrusion Detection System for Smart Home IoT Devices," in *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042-9053, Oct. 2019, doi: 10.1109/JIOT.2019.2926365.

[15] ElBachirElMoussaid, N. and Toumanari, A., 2014. Web Application Attacks Detection: A Survey and Classification. *International Journal of Computer Applications*, [online] 103(12), pp.1-6. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.800.3515&rep=rep1&type=pdf [Accessed 4 May 2020].

[
[16] Eryk Lewinson, E., 2019. *Explaining Feature Importance By Example Of A Random Forest*. [online] Towards Data Science. Available at: https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e [Accessed 12 May 2020].

[17] Gibert, D., Mateu, C., Planes, J., The rise of machine learning for detection and classification of malware: Research developments, trends and challenges, Journal of Network and Computer Applications (2020), doi: https://doi.org/10.1016/j.jnca.2019.102526.

[18] Hao, K., 2018. *What Is Machine Learning?*. [online] MIT Technology Review. Available at: https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/ [Accessed 9 May 2020].

[19] Hassan N.A., Hijazi R. (2018) The Evolution of Open Source Intelligence. In: Open Source Intelligence Methods and Tools. Apress, Berkeley, CA

[20] Ho Yu, C., 2010. Exploratory data analysis in the context of data mining and resampling. *International Journal of Psychological Research*, [online] 3(1), pp.9-22. Available at: https://www.researchgate.net/publication/50946368_Exploratory_data_analysis_in_the_context_of_data_mining_and_resampling [Accessed 15 April 2020].

[21] Hyo-Sik Ham and Mi-Jung Choi, "Analysis of Android malware detection performance using machine learning classifiers," *2013 International Conference on ICT Convergence (ICTC)*, Jeju, 2013, pp. 490-495, doi: 10.1109/ICTC.2013.6675404.

[22] Jovanović, B., 2020. *Virus Alert: Antivirus Statistics And Trends In 2020 | Dataprot*. [online] DataProt. Available at: https://dataprot.net/statistics/antivirus-statistics/ [Accessed 4 June 2020].

[23] Juniper research. (2020). *Cybercrime will Cost Businesses Over $2 Trillion by 2019*. [online] Available at: https://www.juniperresearch.com/press/press-releases/cybercrime-cost-businesses-over-2trillion-by-2019 [Accessed 1 Feb. 2020].

[24] J. Yu, H. Lee, M. S. Kim, and D. Park, "Traffic flooding attack detection with SNMP MIB using SVM," *Comput.Commun.*, vol. 31, no. 17, pp. 4212–4219, Oct. 2008.

[25] Karim, A., Ali Shah, S., Salleh, R. and Shamshirband, D., 2015. Mobile Botnet Attacks – an Emerging Threat: Classification, Review and Open Issues. *KSII Transactions on Internet and Information Systems*, [online] 9(4). Available at: https://www.researchgate.net/publication/273125126_Mobile_Botnet_Attacks_-_an_Emerging_Threat_Classification_Review_and_Open_Issues/references [Accessed 10 April 2020].

[26] Kantardzic, M. (2011). *DATA MINING Concepts, Models, Methods, and Algorithms*. [Online]. Available at: https://doc.lagout.org/Others/Data%20Mining/Data%20Mining_%20Concepts%2C%20Models%2C%20Methods%2C%20and%20Algorithms%20%282nd%20ed.%29%20%5BKantardzic%202011-08-16%5D.pdf [Accessed: 03-04-2020].

[27] Kirasich, Kaitlin; Smith, Trace; and Sadler, Bivin (2018) "Random Forest vs Logistic Regression: Binary Classi cation for Heterogeneous Datasets," SMU Data Science Review: Vol. 1 : No. 3 , Article 9. Available at: https://scholar.smu.edu/cgi/viewcontent.cgi?article=1041&context=datasciencereview [Accessed: 7 May. 2020]

[28] Lim, Kim, Kim, Hong and Han, 2019. Payload-Based Traffic Classification Using Multi-Layer LSTM in Software Defined Networks. *Applied Sciences*, [online] 9(12), p.2550. Available at: https://www.researchgate.net/publication/333939459_Payload-Based_Traffic_Classification_Using_Multi-Layer_LSTM_in_Software_Defined_Networks [Accessed 2 May 2020].

[29] Lin, H. and Lin, C., n.d. *A Study On Sigmoid Kernels For SVM And The Training Of Non-PSD Kernels By SMO-Type Methods*. Taipei 106, Taiwan: National Taiwan University.

[30] Liu, Z. and Xu, H., 2014. Kernel Parameter Selection for Support Vector Machine Classification. *Journal of Algorithms & Computational Technology*, [online] 8(2), pp.163-177. Available at: https://www.researchgate.net/publication/273366480_Kernel_Parameter_Selection_for_Support_Vector_Machine_Classification [Accessed 9 March 2020].

[31] Porter, K., 2020. *Malware Attacks: What You Need To Know*. [online] Us.norton.com. Available at: https://us.norton.com/internetsecurity-malware-malware-101-how-do-i-get-malware-complex-attacks.html[Accessed 27 May 2020].

[23] Raschka, S. and Mirjalili, V., 2017. *Python Machine Learning - Second Edition*. 2nd ed. Birmingham: Packt Publishing, p.XIV.

[33] Rieck K., Holz T., Willems C., Düssel P., Laskov P. (2008) Learning and Classification of Malware Behavior. In: Zamboni D. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA 2008. Lecture Notes in Computer Science, vol 5137. Springer, Berlin, Heidelberg

[34] Rogel-Salazar, J., 2017. *Data Science And Analytics With Python*. Boca Raton, FL: Taylor & Francis Group.

[35] Rowe, K., 2018. *How Search Engines Use Machine Learning: 9 Things We Know For Sure*. [online] Search Engine Journal. Available at: https://www.searchenginejournal.com/how-search-engines-use-machine-learning/224451/#close
[Accessed 4 May 2020].

[36] SALIAN, I., 2018. *NVIDIA Blog: Supervised Vs. Unsupervised Learning*. [online] The Official NVIDIA Blog. Available at: https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/ [Accessed 1 June 2020].

[37] Scikit-learn (2017). [Online]. Available at: https://scikit-learn.org/stable/index.html [Accessed: 04-02-2020].

[38] Shung, K., 2018. *Accuracy, Precision, Recall Or F1?*. [online] Medium. Available at: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9 [Accessed 18 May 2020].

[39] Shwartz, S. et al. (2014). Understanding Machine Learning: From Theory to Algorithms. [Online]. Available at: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf [Accessed: 07-05-20202].

[40] Stratosphere Laboratory. *A labeled dataset with malicious and benign IoT network traffic*. January 22th. Agustin Parmisano, Sebastian Garcia, Maria Jose Erquiaga.

[41] W. An-na, Z. Yue, H. Yun-tao and L. I. Yun-lu, "A novel construction of SVM compound kernel function," *2010 International Conference on Logistics Systems and Intelligent Management (ICLSIM)*, Harbin, 2010, pp. 1462-1465, doi: 10.1109/ICLSIM.2010.5461210.

[42] Waskom, M., 2020. *Seaborn*. [online] PyPI. Available at: https://pypi.org/project/seaborn/ [Accessed 1 May 2020].

[43] Westra, E. (2016). Modular Programming with Python Introducing modu- lar techniques for building sophisticated programs using Python. [Online]. Available at: http://file.allitebooks.com/20161026/Modular%20Programming%20with%20Python.pdf

[Accessed: 28-04-2020].

[44] Witten, I. et al. (2016). Data Mining: Practical Machine Learning Tools and Techniques. Elsevier.

[45] Wylie, B., 2017. *Brothon*. [online] PyPI. Available at: https://pypi.org/project/brothon/ [Accessed 1 May 2020].