MALWARE ANALYSIS WITH MACHINE LEARNING

TAN CARLVEN

SESSION 2022/2023

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERISITY
JANUARY 2023

MALWARE ANALYSIS WITH MACHINE LEARNING

BY

TAN CARLVEN

SESSION
2022/2023

THE PROJECT REPORT IS
PREPAIDED FOR

FACULTY OF COMPUTING AND INFORMATICS
MULTIMEDIA UNIVERISITY
IN PARTIAL FULFILLMENT
FOR

BACHELOR OF COMPUTER SCIENCE
B.CS (HONS) CYBERSECURITY

FACULTY OF COMPUTING AND INFORMATICS

MULTIMEDIA UNIVERSITY
JANUARY 2023

Copyright of this report belongs to Universiti Telekom Sdn. Bhd. as qualified by Regulation 7.2 (c) of the Multimedia University Intellectual Property and Commercialisation Policy. No part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Universiti Telekom Sdn. Bhd. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022 Universiti Telekom Sdn. Bhd. ALL RIGHTS RESERVED.

DECLARATION

I hereby declare that the work has been done by myself and no portion of the work contained in this thesis has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Name of Candidate: TAN CARLVEN
Student ID: 118120094
Faculty of Computing & Informatics
Multimedia University
Date: 30/01/2023

ACKNOWLEDGEMENT

I extend my heartfelt thanks to Dr. Ng Hu, my supervisor for the final year project, for their invaluable guidance, support, and motivation throughout the research endeavor. I am genuinely grateful for their patience and understanding in providing continuous guidance.

I would also like to express my gratitude to my dear friends and family, who have consistently offered their unwavering support and valuable advice during the research and writing process.

Lastly, I am profoundly thankful for the opportunity to undertake this research and successfully complete my final year project within the designated timeframe. I extend my thanks to everyone who has supported me along the way.

ABSTRACT

Malware analysis is an important task in the field of cybersecurity, as it can detect and remove the malicious software from the computer systems. However, traditional methods of malware analysis were wasting time and resources. In recent years, machine learnings have become new method to solve these problems for improving the efficiency and accuracy of malware analysis.

The purpose of this research project was to explore the machine learning techniques for the malware analysis. This project proposed a malware detection system that applies machine learnings algorithms to classify malwares based on their behavior and characteristics. The system (model) was trained using the dataset that consists of different malware families and was evaluated using the evaluation metrics.

The dataset utilized in this project was obtained from a publicly available repository on GitHub, graciously provided by Marco Ramili. Prior to conducting the classification task, several stages were undertaken, including EDA, data preprocessing, data splitting, and feature selection. Data preprocessing involved multiple steps such as integrating the data, cleaning it, transforming it, and balancing it. The machine learning algorithms employed in constructing the model were DT, RF, and LightGBM.

Four experiments were conducted to compare different techniques in malware analysis across different train-test split ration. Among all the models, the findings indicated that LightGBM had the best performance. Using SMOTE without Boruta, LightGBM was able to attain the maximum accuracy for the 80-20 and 60-40 splits of 90.59% and 89.55%, respectively. LightGBM used SMOTE and Boruta to obtain the greatest accuracy for 70-30 split, 90.33%. Malware detection using machine learning API will be deployed using LightGBM.

TABLE OF CONTENTS

DECLARATION ..... **III**

ACKNOWLEDGEMENT ..... **IV**

ABSTRACT ..... **V**

TABLE OF CONTENTS ..... **VI**

LIST OF TABLES ..... **IX**

LIST OF FIGURES ..... **X**

LIST OF ABBREVIATIONS/ SYMBOLS ..... **XII**

CHAPTER 1 INTRODUCTION **1**

1.1 Project Overview 1

1.2 Problem Statement 4

1.3 Project Objective 5

1.4 Project Scope 6

1.5 Project Planning 7

CHAPTER 2 LITERATURE REVIEW **8**

2.1 Introduction 8

2.2 Malware Analysis 8

2.2.1 Static Analysis 8

2.2.2 Dynamic Analysis 9

2.2.3 Summary of Malware Analysis 9

2.3 Machine Learning Techniques 10

2.3.1 Decision Tree 10

2.3.2 Random Forest 11

2.3.3 Gradient Boosting 12

2.4 Feature Selection 13

2.5 REST API 15

2.6 Related Works 15

2.6.1 Malware Training Sets 15

2.6.2 Different Datasets 18

2.7 Summary of Methods 23

CHAPTER 3 METHODOLOGY **24**

3.1 Introduction 24
3.2 Data Collection 25
3.3 Exploratory Data Analysis 25
3.3.1 Bar Chart Analysis 25
3.3.2 Box Plot Analysis 26
3.3.3 Density Plot Analysis 27
3.3.4 Scatter Plot Analysis 29
3.3.5 Correlation Matrix Analysis 31
3.3.6 Histogram Analysis 32
3.4 Data Preprocessing 34
3.4.1 Data Integration 34
3.4.2 Data Cleaning 36
3.4.3 Data Transformation 37
3.4.4 Data Balancing 38
3.5 Data Splitting 39
3.6 Feature Selection 40
3.7 Model Construction 42
3.7.1 Hyperparameter Tuning 42
3.7.2 Decision Tree 42
3.7.3 Random Forest 44
3.7.4 Light Gradient-Boosting Machine 46
3.8 Model Evaluation 47
3.8.1 Confusion Matrix 48
3.8.2 Accuracy 49
3.8.3 Precision 49
3.8.4 Recall 49
3.8.5 F1 Score 50
3.9 Application Deployment with Machine Learning API 50

CHAPTER 4 RESULT AND DISCUSSION 53

4.1 Introduction 53
4.2 Classification Result 53
4.2.1 Experiment 1: Without SMOTE & Boruta 53
4.2.2 Experiment 2: Using Boruta Without SMOTE 58

4.2.3 Experiment 3: Using SMOTE Without Boruta 62
4.2.4 Experiment 4: Using SMOTE & Boruta 66
4.3 Comparison Result 70
4.3.1 Models with Different Train-Test Split Ratio 70
4.3.2 Confusion Matrix of Models 72
4.4 Application Deployment Result 77

CHAPTER 5 CONCLUSION 79

5.1 Conclusion 79
5.2 Future Work 80

REFERENCES 83

APPENDIX A RESEARCH PAPER 84

APPENDIX B MEETING LOG 99

APPENDIX C SIMILARITY INDEX REPORT 130

LIST OF TABLES

Table 1.1: FYP 1 Timeline....7
Table 1.2: FYP 2 Timeline....7
Table 2.1: Summary of Malware Analysis Approaches....9
Table 2.2: Performance using NFE....16
Table 2.3: Evaluation Results....18
Table 2.4: Results for The CLAMP RAW-5184....18
Table 2.5: Results for CSDMC_API_Train....19
Table 2.6: Static Model Tests Results....21
Table 2.7: Hybrid Model Tests Results....22
Table 2.8: Result of Overall Accuracy for Different Classifier....22
Table 3.1: Malware Families Used....25
Table 3.2: Colour of Malware Families used in Scatter Plot....29
Table 3.3: Correlation Coefficient Represent the Relationship....31
Table 3.4: Best DT Parameters (Imbalanced Data)....43
Table 3.5: Best DT Parameters (Balanced Data)....43
Table 3.6: Best RF Parameters (Imbalanced Dataset)....44
Table 3.7: Best RF Parameters (Balanced Dataset)....45
Table 3.8: Best LightGBM Parameters (Imbalanced Dataset)....46
Table 3.9: Best LightGBM Parameters (Balanced Dataset)....47
Table 4.1: Performance of DT (Without SMOTE & Boruta)....53
Table 4.2: Performance of RF (Without SMOTE & Boruta)....54
Table 4.3: Performance of LightGBM (Without SMOTE & Boruta)....54
Table 4.4: Performance of DT (Using Boruta Without SMOTE)....58
Table 4.5: Performance of RF (Using Boruta Without SMOTE)....58
Table 4.6: Performance of LightGBM (Using Boruta Without SMOTE)....58
Table 4.7: Performance of DT (Using SMOTE Without Boruta)....62
Table 4.8: Performance of RF (Using SMOTE Without Boruta)....62
Table 4.9: Performance of LightGBM (Using SMOTE Without Boruta)....62
Table 4.10: Performance of DT (Using SMOTE & BORUTA)....66
Table 4.11: Performance of RF (Using SMOTE & BORUTA)....66
Table 4.12: Performance of LightGBM (Using SMOTE & BORUTA)....66
Table 4.13: Comparison Result of the Model (80-20 Split)....70
Table 4.14: Comparison Result of the Model (70-30 Split)....71
Table 4.15: Comparison Result of the Model (60-40 Split)....72

LIST OF FIGURES

Figure 1.1: Types of Malwares 1
Figure 1.2: Distribution of Web-Attack Sources by Country, Q2 2021 (AMR, et al, 2021)... 2
Figure 2.1: Example of Decision Tree (Charbuty, B., & Abdulazeez, A, 2021)..... 10
Figure 2.2: Bootstrap Aggregation (Bagging) (Tyralis H, Papacharalampous G, Langousis A, 2019) 12
Figure 2.3: Histogram-based decision tree algorithm (Liang, Weizhang, et al, 2020)..... 13
Figure 2.4: Overview of Classification Model Building Process (Masabo, E, 2019) 16
Figure 2.5: Class distribution: 'APT':0, 'Crypto':1, 'Locker':2, 'shadowbrokers':3, 'Zeus':4. (Masabo E, Kaawaase K S, Sansa-Otim J, 2018) 17
Figure 2.6: Results for the ClaMP Raw5184 Testing Set (Morales-Molina, Carlos Domenick, et al, 2018) 19
Figure 2.7: Results for the CSDMC_API_Train Testing Set (Morales-Molina, Carlos Domenick, et al, 2018)..... 20
Figure 2.8: Overview Research Methodology (Hadiprakoso, Kabetta, & Buana, 2020) 21
Figure 2.9: Architecture of the proposed malware detection system (Chowdhury, Mozammel, Azizur & Rafiqul, 2017) 22
Figure 3.1: Flow of Methodology Process..... 24
Figure 3.2: Distribution of Features in Five Families Malware 26
Figure 3.3: Box Plot Analysis (First 30 Features) 27
Figure 3.4: Density Plot Analysis (First 18 Features)..... 28
Figure 3.5: Scatter Plot Analysis 30
Figure 3.6: Plot of Correlation Matrix (First 15 Features)..... 32
Figure 3.7: Histogram Analysis of Balanced Data (First 12 Features)..... 33
Figure 3.8: A JSON File Read as CSV Format 34
Figure 3.9: Dataset Combined from All JSON Files 35
Figure 3.10: Algorithm of Generating Dataset as CSV Format..... 35
Figure 3.11: Dataset before Data Cleaning..... 36
Figure 3.12: Dataset after Data Cleaning..... 36
Figure 3.13: Algorithm of Label Encoding..... 37
Figure 3.14: Dataset before Label Encoding 37
Figure 3.15: Dataset after Label Encoding 38
Figure 3.16: Imbalanced Data..... 38
Figure 3.17: Balanced Data with SMOTE..... 39
Figure 3.18: Algorithm of Data Splitting..... 40
Figure 3.19: Boruta Algorithm 40
Figure 3.20: Features with Ranking (Balance Data)..... 41
Figure 3.21: Features with Ranking (Imbalance Data)..... 41
Figure 3.22: Hyperparameter Tuning 42
Figure 3.23: Decision Tree Algorithm..... 44
Figure 3.24: Random Forest Algorithm..... 45
Figure 3.25: LightGBM Algorithm..... 47
Figure 3.26: Confusion Matrix from Binary Class 48
Figure 3.27: Serialize and Save the LightGBM Model 50
Figure 3.28: API Endpoints 51
Figure 3.29: Function of Prediction..... 51
Figure 3.30: Running Application 52
Figure 4.1: Confusion Matrix of DT (Without SMOTE & Boruta) 55

Figure 4.2: Confusion Matrix of RF (Without SMOTE & Boruta) 56
Figure 4.3: Confusion Matrix of LightGBM (Without SMOTE & Boruta) 57
Figure 4.4: Confusion Matrix of DT (Using Boruta Without SMOTE) 59
Figure 4.5: Confusion Matrix of RF (Using Boruta Without SMOTE) 60
Figure 4.6: Confusion Matrix of LightGBM (Using Boruta Without SMOTE) 61
Figure 4.7: Confusion Matrix of DT (Using SMOTE Without Boruta) 63
Figure 4.8: Confusion Matrix of RF (Using SMOTE Without Boruta) 64
Figure 4.9: Confusion Matrix of LightGBM (Using SMOTE Without Boruta) 65
Figure 4.10: Confusion Matrix of DT (Using SMOTE & Boruta) 67
Figure 4.11: Confusion Matrix of RF (Using SMOTE & Boruta) 68
Figure 4.12: Confusion Matrix of LightGBM (Using SMOTE & Boruta) 69
Figure 4.13: Confusion Matrix Without SMOTE & Boruta 73
Figure 4.14: Confusion Matrix Using Boruta Without SMOTE 74
Figure 4.15: Confusion Matrix Using SMOTE Without Boruta 75
Figure 4.16: Confusion Matrix Using SMOTE and Boruta 76
Figure 4.17: Preview Malware Data 77
Figure 4.18: The Scanning Result is True 78
Figure 4.19: The Scanning Result is False 78

LIST OF ABBREVIATIONS/ SYMBOLS

EDA	Exploratory Data Analysis
AI	Artificial Intelligence
NFE	Novel Feature Engineering
SMOTE	Synthetic Minority Over-Sampling Technique
LDA	Linear Discriminant Analysis
RF	Random Forest
DT	Decision Tree
GT	Gradient Boosting
LightGBM	Light Gradient-Boosting Machine
KNN	K-Nearest Neighbors
SVM	Support Vector Machine
PCA	Principal Component Analysis
TF-IDF	Term Frequency-Inverse Document Frequency
MLP	Multilayer Perceptron
BAM	Binary Associative Memory
TF	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
Imblearn	Imbalanced-learn
Sklearn	Scikit-learn
REST API	Representational State Transfer Application Programming Interface
HTML	Hypertext Markup Language

CHAPTER 1 INTRODUCTION

1.1 Project Overview

Malware refers to any type of software or code that is intended to cause harm to a computer system or network. It can come in various forms to attack a computer system or network as shown in Figure 1.1, such as spyware, adware, ransomware, trojan, worms, rootkits and backdoors. It can be used to accomplish tasks such as stealing protected data, installing unwanted software, or deleting confidential documents without authorization. Due to the widespread use of the internet, malware can spread quickly and target both individuals and organizations, making it a significant security threat. The cyber criminals are allowed to take advantages to steal the data from the infected target over the internet (Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V, 2019). Therefore, detecting and preventing malware is a top priority for many organizations and individuals.

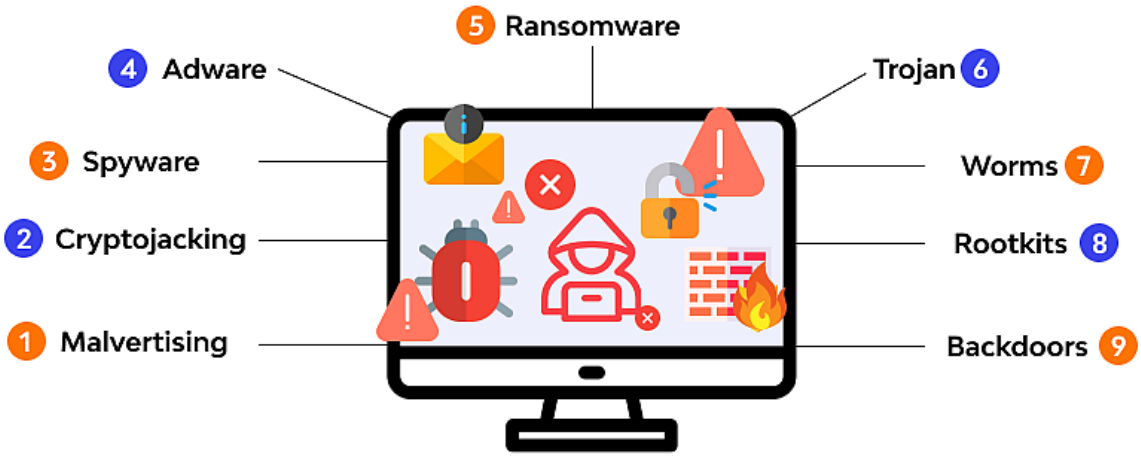

Figure 1.1: Types of Malwares

In recent times, traditional malware detection systems have faced challenges in effectively detecting modern malware as attackers are constantly evolving their techniques and spreading their malicious software in large number. The prevalent method of malware detection, signature-based detection, involves identifying the signature of known malware in a database and removing it. However, this approach is not effectively as it struggles to detect polymorphic malware and zero-days attacks, with detection rate currently estimated to be between 25% and 50% currently (Masabo E, Kaawaase K S, Sansa-Otim J, 2018). This is because attackers can

easily change the byte sequence within malware to avoid detection. Even after a new unknow threat is discovered, it is difficult to research and identify patterns in the malware as it is constantly changing. Additionally, malware cryptors can keep altering the malware until it becomes unrecognized to antivirus scans.

According to AMR, et al (2021), in the report of It Threat Evolution in Q2 2021, around one million new threats created were injected daily into the wild in 2021. In 2021, Kaspersky solutions prevented 1,686,025,551 attacks from online resource worldwide. They also reported web antivirus recognized 675,832,360 unique URL as malicious. Moreover, the attacker attempts to run malware for stealing money from online bank accounts were blocked on the computers of 119,252 unique users. Ransomware attacks were also detected and defeated on the computers of 97,451 unique users. Total 68,294,298 unique malicious and potentially unwanted objects were detected by Kaspersky file antivirus. Figure 1.2 shows the distribution by country of the sources of internet attacks blocked by Kaspersky products on user computer in 2021.

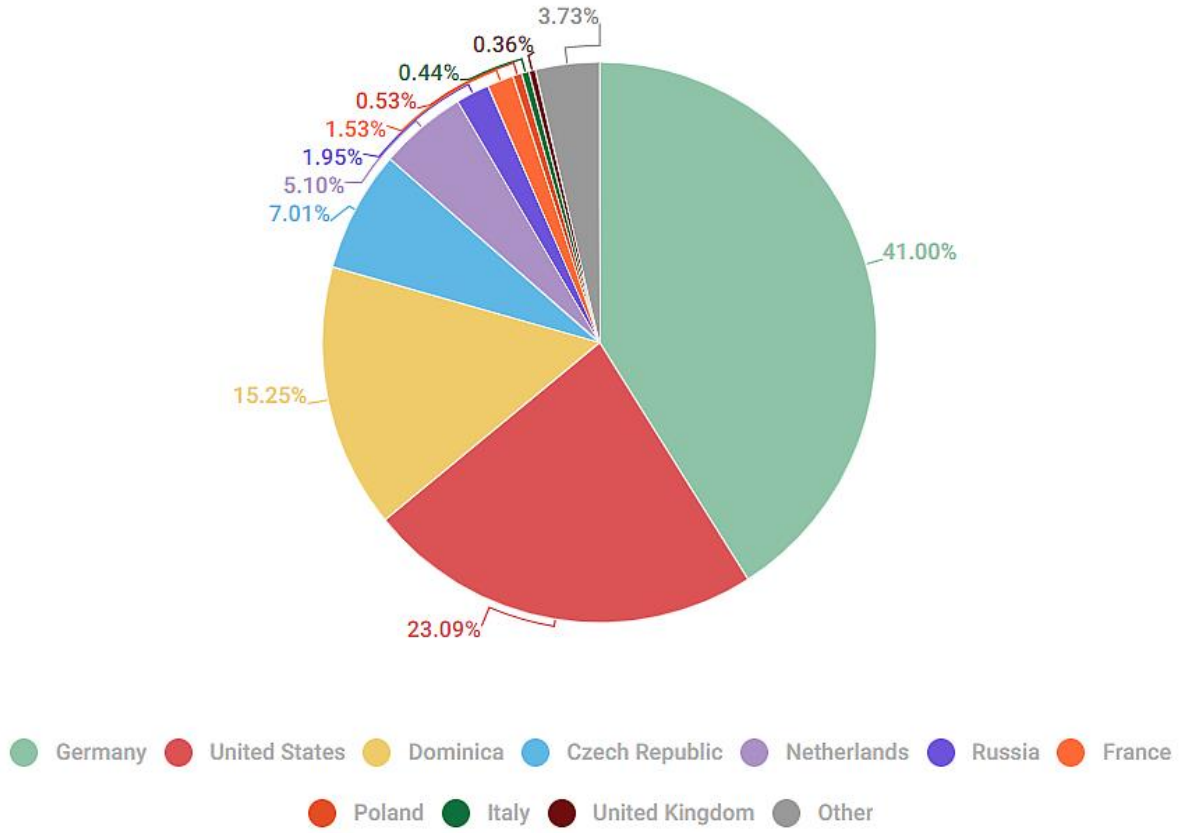

Figure 1.2: Distribution of Web-Attack Sources by Country, Q2 2021 (AMR, et al, 2021)

To address the issue of malware, researchers have developed new method for accurately identifying and detecting it. These methods typically involve two important steps. The first step is analyzing the malware to extract information about its specific characteristics. This analysis can help to detect and mitigate any potential threat by understanding the malware's function, purposes and potential impact. There are two types of malware analysis that are commonly used, which are static analysis and dynamic analysis (Ijaz M, Durad M H, Ismail M, 2019). Static analysis examines samples without executing it, while dynamic analysis examines the execution of samples while they are running to observe their behaviours. After the samples have been analyzed, the information about their pattern and behaviour is used to identify the malware. In the second step, classification and detection systems are developed based on the information provide by malware analysis. These systems store signature patterns in their database and update them regularly (Singh J, Singh J., 2018). However, this approach may not be effective against polymorphic malware, which can change its identity at any time.

The current limitations of malware detection systems, such as their inability to detect modern and advanced forms of malware, have led to the need for alternative methods. Malware detection can be addressed by leveraging machine learning models as a potential solution. Further research is needed to identify the key features that best characterize the malware, as the efficiency of the employed machine learning techniques heavily relies on the careful selection of appropriate features. Feature selection, recognized as a critical stage in the creation of an effective machine learning model, is a fundamental process. With various machine learning algorithms available, making it challenging to identify the one that would be most suitable for a particular dataset and enhance the model's performance. In many cases, real-world datasets are complex and trying multiple algorithms may be necessary to achieve the desired results.

1.2 Problem Statement

In order to effectively combat new forms of malwares, it is crucial to develop advanced machine learning models to fulfill the tasks of detection and classification. To achieve this goal, a clear and specific statement of the problems is provided to ensure that the project's resources are directly towards the most pressing issues. The following are the common problems affect the malware analysis with machine learnings.

A. Advanced malware

Advanced malware, also known as advanced persistent threats (APTs), is often complex and polymorphic, making it difficult to understand its behaviors to classify it correctly with machine learning models. This malware is designed to remain hidden for long periods of time, allowing it to avoid detection to steal sensitive information. It is a big problem to detect and analyze, because it may remain hidden for months or even years before it is discovered. Advanced malware is rare and less common than other types of malwares, which means it is hard to obtain the large dataset to train machine learning models.

B. Evasion techniques used in the malware

Evasion techniques are specifically designed to avoid detection by traditional security tools and methods, such as signature-based detection and sandbox analysis. It is difficult to use traditional signature-based detection to keep up because it always changes over time and it is dynamic. Machine learning models may not be able to provide interpretability on the decision they made and why they made them, making it difficult to explain why a particular evasion technique was not detected, and how to improve the decision in the future. The malware that uses evasion technique is very difficult to detect.

C. False positive and negative rates

False positive and negative rates can affect the overall performance of a malware detection system. High false positive rates can lead lost productivity and system destruction, while high false negative rates can lead to undetected malware. Balancing the trade-off between false positive and false negative is a difficult task because generally the detection rate for one is up will lead to the detection rate for other down. Malware with its evasion techniques is getting evolved, so it is difficult to maintain a low false positive rate and false negative rate over time.

1.3 Project Objective

Several objectives of this project are expected to be achieved for malware detection. The following are the project objectives:

A. To analyze the malware feature with exploratory data analysis (EDA)

EDA provides different methods to understand the characteristic of the malware dataset, such as the distribution of various features and the presence of outliers. It provides different methods to visualise the data, which to identity the relationship and pattern within the data that may not be apparent from the raw data alone. It can help to prepare the data by identifying important features and data quality issues that need to be handle for the machine learning models.

B. Developing a machine learning model to accurately classify and detect the malware

Machine learning models can adapt to the evolving malware landscape, making them more effective in detecting new and unknown malware variants. These models can automate the process of detecting and classifying the malware, which to improve the efficiency of malware detection. These models can handle the problem of evasion techniques used in the malware, which are specifically designed for evade detection.

C. To evaluate the performance of machine learning models

Evaluation can help to identify the best hyperparameter to optimize the performance of machine learning models, which to reduce error rates and improve the accuracy of malware detection. It helps to compare the performance of different machine learning models, which to identify the best model to use for a specific problem and datasets. It can identify the trade-off between false positive and false negatives and balance them based on the problem and dataset.

1.4 Project Scope

The scope of this project is to develop a malware detection system that utilizes advanced machine learning techniques to analyze and classify malware based on its behaviour and characteristics, in order to identity unknown types of malwares and improves defences against them.

A. Data Collection

Data collection is the process of gathering, recording, and storing data for a specific purpose. The malware dataset from Marco Ramili (Ramilli, Marco, 2016), is downloaded from a public repository github.

B. Exploratory Data Analysis (EDA)

EDA is a process of analyzing and understanding the datasets. It involves several techniques such as visualising the data using plots and charts, summarizing the data using statistical measures and identifying the patterns and relationships in the data.

C. Feature Selection

Feature selection is the process of selecting and extracting relevant features from the dataset. Boruta, a wrapper built around the random forest classification algorithm, is chosen as feature selection.

D. Machine Learning Models

Three machine learning algorithm that can handle the multi-class classification problem are chosen to train the models, such as DT, RF and LightGBM.

E. Evaluation Metrics

There are several common evaluation metrics used in machine learning, such as accuracy, precision, recall, f1 score and confusion matrix.

F. Application Deployment

Machine learning models was deployed as web service through HTTP requests with the method of REST API.

1.5 Project Planning

Table 1.1: FYP 1 Timeline

Description			Week														
Task	Start	End	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CHAPTER 2 LITERATURE REVIEW																	
2.1 Introduction	07/10/2022	09/10/2022															
2.2 Malware Analysis	09/10/2022	12/10/2022															
2.3 Machine Learning Technique	12/10/2022	17/10/2022															
2.4 Feature Selection	17/10/2022	20/10/2022															
2.5 Related Works	20/10/2022	25/10/2022															
2.6 Summary of Methods	25/10/2022	27/10/2022															
CHAPTER 3 METHODOLOGY																	
3.1 Introduction	27/10/2022	28/10/2022															
3.2 Data Collection	28/10/2022	31/10/2022															
3.3 Exploratory Data Analysis	31/10/2022	13/11/2022															
3.4 Data Preprocessing	05/12/2022	19/12/2022															
3.5 Data Splitting	19/12/2022	21/12/2022															
3.6 Feature Selection	21/12/2022	26/12/2022															
3.7 Model Construction	26/12/2022	02/01/2023															
3.8 Model Evaluation	02/01/2023	04/01/2023															
CHAPTER 4 RESULTS AND DISCUSSION																	
4.1 Introduction	04/01/2023	05/01/2023															
4.2 Classification Results	05/01/2023	09/01/2023															
4.3 Comparison Results	09/01/2023	16/01/2023															
CHAPTER 1 INTRODUCTION																	
1.1 Project Overview	16/01/2023	19/01/2023															
1.2 Problem Statement	19/01/2023	21/01/2023															
1.3 Poject Objective	21/01/2023	23/01/2023															
1.4 Project Scope	23/01/2023	25/01/2023															
1.5 Project Planning	25/01/2023	26/01/2023															
CHAPTER 5 CONCLUSION																	
5.1 Work Done In Phase 1	26/01/2023	28/01/2023															
5.2 Work Plan in Phase 2	28/01/2023	30/01/2023															

Table 1.2: FYP 2 Timeline

Description			Week															
Task	Start	End	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CHAPTER 2 LITERATURE REVIEW																		
2.3 Machine Learnig Techniques	21/03/2023	24/03/2023																
2.5 REST API	24/03/2023	27/03/2023																
CHAPTER 3 METHODOLOGY																		
3.9 Application Deployment with Machine Learning API	27/03/2023	08/05/2023																
3.5 Data Spliting	08/05/2023	11/05/2023																
3.7 Model Construction	11/05/2023	22/05/2023																
CHAPTER 4 RESULTS AND DISCUSSION																		
4.2 Classification Results	22/05/2023	29/05/2023																
4.3 Comparison Results	29/05/2023	01/06/2023																
4.4 Application Deployment Result	01/06/2023	05/06/2023																
CHAPTER 1 INTRODUCTION																		
1.4 Project Scope	05/06/2023	08/06/2023																
1.5 Project Planning	08/06/2023	12/06/2023																
CHAPTER 5 CONCLUSION																		
5.1 Conclusion	12/06/2023	15/06/2023																
5.2 Future Work	15/06/2023	19/06/2023																
SUBMISSION WORKS																		
Finalize Report	19/06/2023	23/06/2023																
Research Paper Writing	23/06/2023	03/07/2023																
Presentation Poster	03/07/2023	07/07/2023																
Video Demo	07/07/2023	14/07/2023																

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

This chapter examines the field of malware analysis and the current approaches being utilized. The application of machine learning techniques has demonstrated promise in effectively identifying and categorizing malware. The literature reviewed in this chapter highlights the availability of diverse machine learning algorithms, including DT, RF, and LightGBM. Additionally, feature selection methods like wrapper, filter, and embedded approaches are employed to optimize the dataset. Deploying machine learning models as web services through REST API is also explored for malware detection. Several significant research papers on malware analysis with machine learning are discussed in this chapter.

2.2 Malware Analysis

Malware analysis is a crucial aspect of computer security, as it aims to identify and classify new and unknown types of malwares to improve defences against them (Ucci D, Aniello L, Baldoni R, 2019). One of the main challenges in malware analysis is the ability to detect malicious software that has been created using obfuscation techniques such as polymorphism, metamorphism, and packing. These techniques make it difficult for traditional malware detection methods to identify and block the malware before it causes harm (Singh, J., & Singh, J. 2018). To address this issue, the research community has developed various approaches for malware analysis, including both static and dynamic analysis techniques.

2.2.1 Static Analysis

Static analysis examines the malware files without executing them in a controlled environment. This makes it more effective than dynamic analysis, but it also has limitations (Ijaz M, Durad M H, Ismail M, 2019). For instance, it does not allow for reverse engineering of malware, which limits the source code for analysis. Despite this, static analysis is still an important step in the malware analysis process as it provides valuable information that can be used to determine the next steps or conclude if a file is malicious (Chikapa, M., & Namanya, A. P, 2018). However, complex and sophisticated malware cannot be fully detected through static

analysis alone and require dynamic analysis to be fully identified (Chakkaravarthy, SS, Sangeetha, D, & Vaidehi, V, 2019).

2.2.2 Dynamic Analysis

Behavioural analysis, often referred to as dynamic analysis, involves running the malware file and observing its interaction with the computer system. To prevent harm to the host system, malware files are executed in a virtual environment created using tools such as VMware or Virtual Box, as well as emulators and hypervisors. By monitoring the malware's activites, such as file creation or deletion, the malware can be detected (Singh, J., & Singh, J, 2018). However, dynamic analysis can be time consuming and requires specialised tools. The more complex the malware, the more extensive the dynamic analysis required. This can be a tactic used by malware creators to waste the resources of malware analysts and increase the chances of the malware achieving its goals (Chikapa, M., & Namanya, A. P, 2018).

2.2.3 Summary of Malware Analysis

Table 2.1: Summary of Malware Analysis Approaches

Static Analysis	**Dynamic Analysis**
Examines malware files in a controlled environment without actually executing them.	Involves running the malware file in a controlled environment and observing its behavior.
It uses a methodology based on signatures for the analysis of malware.	It uses a methodology focused on behavior for the analysis of malware.
It cannot reverse engineer complex malware or observe its behavior during execution.	It identifies sophisticated malware that may exhibit different behaviors in different environments.
It is faster and more efficient.	It is more time-consuming and resource intensive.

2.3 Machine Learning Techniques

Machine learning, a branch of AI, utilizes data and algorithms to mimic human pattern recognition and enhance performance accuracy (Greener, Joe G., et al, 2022). In malware analysis, machine learning automates the process of analyzing and classifying malware, using techniques like clustering and classification to identify patterns and categorize unknown malware into specific families (Patil, Rajvardhan, and Wei Deng, 2020). DT, RF, and GT are commonly employed for malware classification.

2.3.1 Decision Tree

DT is widely acknowledged as highly effective and efficient methodologies in the realm of machine learning. They have been successfully employed to address real-world challenges within the field of artificial intelligence (Trabelsi, A., Elouedi, Z., & Lefevre, E, 2019). It exhibits similarities to the cognitive process of human decision-making, rendering it easily understand. It possesses the capability to handle both discrete and continuous data inputs, enabling it to provide solutions in various situations as shown in Figure 2.1. When the data does not provide any advantages in terms of splitting, the execution process is stopped directly (Charbuty, B., & Abdulazeez, A, 2021).

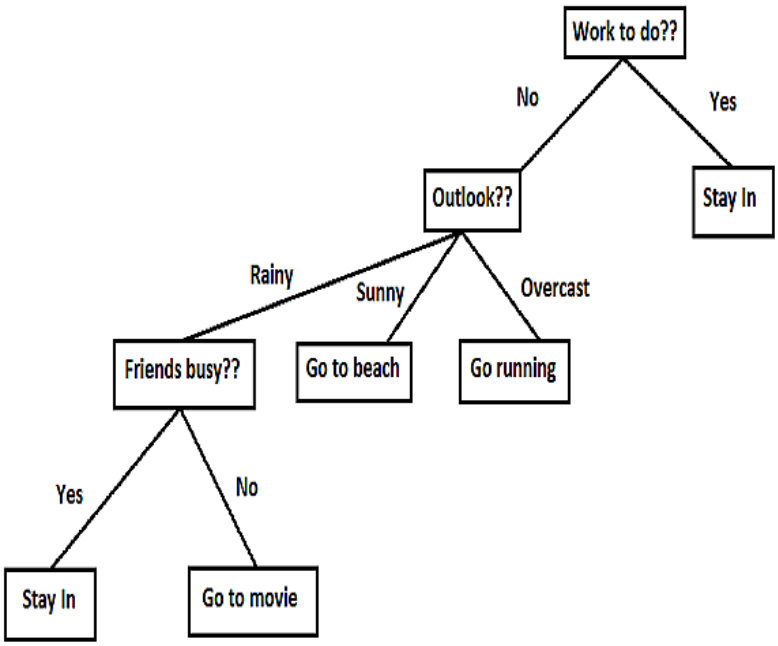

Figure 2.1: Example of Decision Tree (Charbuty, B., & Abdulazeez, A, 2021)

DT is also a commonly employed technique in data mining that involves the creation of classifiers. It offers the capability to derive assumptions about the names of categorical classes, categorize data using sets of training data and class labels, and classify recently acquired data. It comprises nodes and branches, with nodes representing feature categories for classification. Subsets determine the value of associated with each node. Due to their simplicity and accuracy across various data forms, decision trees have found widespread implementation (Charbuty, B., & Abdulazeez, A, 2021).

DT classifiers were always chosen to be used for four main reasons. Firstly, they can be learned and updated quickly from data compared to other methods. Secondly, they provide a visually intuitive and easy-to-interpret representation. Thirdly, decision tree classifiers enable data-driven root cause analysis, tracing the sequence of events from the end state to the initiating event. Lastly, they have demonstrated comparable or superior accuracy, especially when using ensemble decision tree classifiers, offering a balanced combination of error rate and speed (Abdallah, Imad, et al, 2018).

2.3.2 Random Forest

RF is a widely used ensemble learning technique that can be utilized for a diverse array of classification and regression tasks. This algorithm was developed by Breiman. It utilizes multiple decision trees to generate predictions, with the final outcome determined by a majority vote of these tree. Ensemble learning is a method of integrating multiple models to improve accuracy and stability, by combining the predictions from multiple classifiers. This technique is especially useful in cases where a single classifier may be unstable, as the integration of multiple classifiers reduces the variance and produces more reliable results (Sheykhmousa, Mohammadreza, et al, 2020).

RF is widely used because of bootstrap aggregation techniques, known as bagging (as shown in Figure 2.2). This technique generates several bootstrap samples derived from the initial dataset and trains an independent model on each of these samples. The predictions from each of these models are then combined, typically through averaging, to generate a final prediction. This process, known as bagging, enhances the stability and precision of the ultimate model by diminishing variance (Tyralis H, Papacharalampous G, Langousis A, 2019).

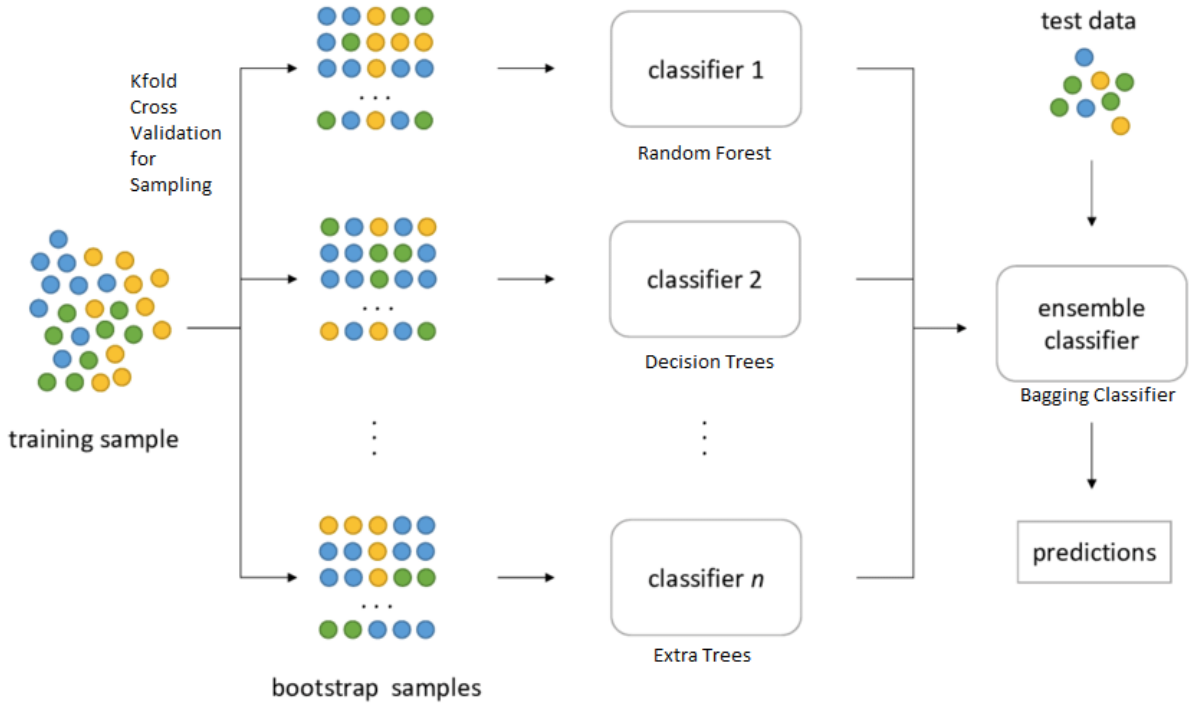

Figure 2.2: Bootstrap Aggregation (Bagging).

(Tyralis H, Papacharalampous G, Langousis A, 2019)

The RF classifier is also popular in a variety of applications, such as bioinformatics, malware analysis and more. RF possesses the capability to effectively handle datasets with missing attributes and extensive sample sizes, which is one of its key advantages. The algorithm works by dividing the dataset into training set and test set, and then using the training set to predict and label the classes. After the completion of the classification process, the model is initialized according to its level of accuracy. The RF classifier is particularly useful for handling complex and uncertain data, making it a powerful tool for a wide range of machine learning tasks (Huljanah, Mia, et al, 2019).

2.3.3 Gradient Boosting

A GB algorithm has been created with excellent predictive capabilities. However, its adoption is restricted because the algorithm mandates the sequential reduction of errors from previous trees in the model, one tree at a time. Consequently, the training process becomes laborious, even for the most basic models. In response to this concern, a new algorithm known as XGBoost has been uncovered (Weldegebriel, Halefom Tekle, et al, 2019).

XGBoost stands out as a remarkably effective realization of gradient boosted decision trees and has garnered acclaim as one of the leading algorithms of machine learning utilized in Kaggle competitions. It offers improved performance and reduced execution time in comparison to various algorithms used in machine learning, including models based on deep learning (Dhieb, Najmeddine, et al, 2019). XGBoost's success can be attributed to its unmatched scalability, surpassing existing solutions by running faster on a single machine. It seamlessly scales to handle a huge number of examples in distributed settings (Weldegebriel, Halefom Tekle, et al, 2019).

Besides XGBoost algorithm, LightGBM is a powerful gradient boosting framework known as the "Light Gradient Boosting Machine". The extraordinary capabilities of the LightGBM algorithms excel in managing massive data volumes and performs distributed data processing. This remarkable creation emerged from the depths of Microsoft Research Asia as an open-source marvel (Taha, A. A., & Malebary, S. J, 2020). LightGBM adopts an algorithm based on histograms and utilizes a growth strategy that prioritizes individual leaves while enforcing a maximum depth constraint. This combination accelerates the speed of training and minimizes memory consumption. The algorithm for decision trees based on histograms as observed in Figure 2.3 (Liang, Weizhang, et al, 2020).

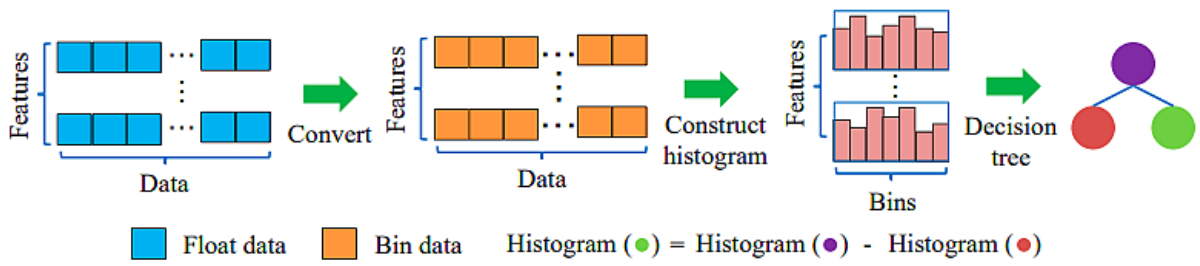

Figure 2.3: Histogram-based decision tree algorithm (Liang, Weizhang, et al, 2020)

2.4 Feature Selection

The process of selecting relevant attributes from a dataset, known as feature selection, is a vital step in constructing a model. This can greatly impact the model's prediction performance, runtime, generalization, and ability to avoid overfitting (Moran & Gordon, 2019). The available methods for feature selection encompass the wrapper approach, the filter approach, and the embedded approach. These approaches aim to improve the overall performance of the model

by selecting the most informative features from the dataset (Noorbehbahani, Rasouli & Saberi, 2019).

Wrapper methods in feature selection involve using a predictive model to assess the performance of various feature subsets. The model assesses the accuracy of different feature subsets by training a model with each subset and evaluating its performance. Algorithms such as Sequential, the Branch and Bound or Genetic Algorithm are commonly used to find the best subset. Although wrapper methods are computationally intensive, they provide the most optimal feature set for the training model, but they come with the risk of overfitting (Moran & Gordon, 2019).

Filter methods in feature selection involve evaluating individual features based on their intrinsic characteristics, rather than their relationship to the model or the target variable. These methods are less computationally compared to wrapper methods, and they do not take into account the specific model being used. Despite their simplicity, filter methods can provide a robust feature subset that can boost the effectiveness of a predictive model. (Moran & Gordon, 2019). Additionally, these methods are often referred to as Univariate Feature Selection, as they analyze each feature individually to evaluate its predictive power (Rodríguez-Rodríguez, Ignacio, et al, 2021).

Embedded methods are a type of feature selection technique that combines the benefits of both wrapper and filter methods. These methods involve performing feature selection and classification using a specific learning algorithm. This approach can improve performance by allowing the algorithm to interact with the feature selection process. However, embedded methods tend to be computationally intensive compared to other methods (Moran & Gordon, 2019). Additionally, by embedding feature selection into the learning process, it can avoid the need to repeatedly evaluate different subset, which can help to improve computational efficiency. Overall, embedded methods provide a trade-off solution between filter and wrapper methods (Li, Wange, et al, 2021).

2.5 REST API

A REST API, which is a form of web API, utilizes HTTP requests to retrieve and modify resources, commonly in JSON or XML format. Currently, it is employed via the HTTP protocol to handle resources, allowing clients to access and manipulate resources through stateless operations. These APIs offer a uniform interface for performing CRUD operations (read, update, create, and delete) on resources. The resources are identified by HTTP URLs and mapped to corresponding HTTP methods like GET, POST, PUT, and DELETE. (Kim, Myeongsoo, et al, 2022).

For example, there exists a REST API designed for handling a group of pets. To obtain the list of pets, one can utilize the HTTP method GET with the endpoint /pets. To include a new pet in the collection, one can employ the HTTP method POST with the endpoint /pets. Input parameters such as object IDs or structured objects can be specified in the resource URL and HTTP methods serve to furnish supplementary details for executing API operations. The URL may include the pet's ID (/pets/<petId>) if retrieving a specific pet (Viglianisi, E., Dallago, M., & Ceccato, M, 2020).

Despite the lack of attention in research, the researchers believe that utilizing a REST API to develop research prototypes can be beneficial in simplifying the overall system development process. REST API can be utilized with any programming language, such as Python, Java and C (Bach, K., Mathisen, B. M., & Jaiswal, A, 2019). Also, REST API implementation might incorporate external libraries or frameworks in compiled code format, posing difficulties for automated testing in scenarios the source code is not fully accessible (Viglianisi, E., Dallago, M., & Ceccato, M, 2020).

2.6 Related Works

2.6.1 Malware Training Sets

In this section, the dataset that most of the researcher use for the malware analysis, which is "Malware Training Sets: A machine learning dataset for everyone", provided by Marco Ramili. According to Masabo, E. (2019), there were not benign files in this dataset. This dataset only consists of 5 types of polymorphic malware and can be used in the classification model, which are APT1, Crypto, Locker, Zeus and ShadowBrokers.

In a study conducted by Masabo, E. (2019), a novel NFE approach was suggested that combines features related to the structure and behavior, proving to be an effective method for classifying and detecting polymorphic malware. In the dataset, most real-world datasets are imbalanced, including Malware Training Sets. To address the issue of imbalanced data, the original training set was balanced by generating synthetic data using the SMOTE technique. In the development of classification model, the researcher employed multiple machine learning algorithms for the classification such as GB, LDA and the KNN classifier. These algorithms were selected due to their ability to handle multi-class problems. The dataset was partitioned into distinct sets for training and testing purposes. The training set was utilized to train the proposed classifier, while the test set was employed to assess its performance on unseen data. Figure 2.4 describe the overview of classification model building process. As a result, the highest accuracy on the balanced dataset was 94% of Gradient Boosting model. The accuracy of performance is shown in Table 2.2.

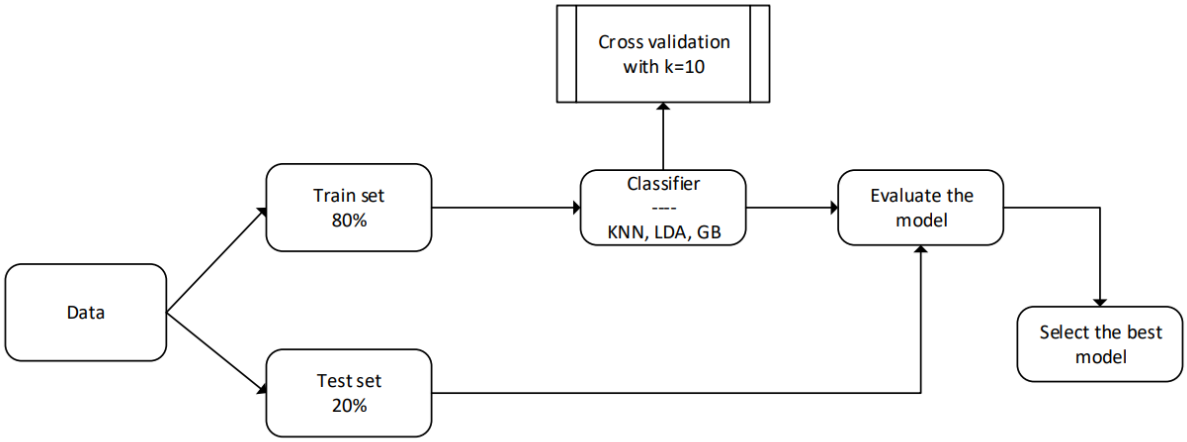

Figure 2.4: Overview of Classification Model Building Process (Masabo, E, 2019)

Table 2.2: Performance using NFE

Model	Accuracy	Recall	Precision	F1-Score
GB	0.94	0.94	0.94	0.94
KNN	0.79	0.66	0.66	0.66
LDA	0.68	0.7	0.81	0.74

In a study conducted by Masabo E, Kaawaase K S, Sansa-Otim J (2018), a method for detecting malware which leverages big data analytics and machine learning techniques. Their approach was assessed on an intricate dataset to evaluate its efficacy. In order to achieve optimal results, researchers perfomed data cleaning and ensured a balanced dataset. The primary focus of the last experiment revolved around the dominant malware families known as Crypto and Zeus, as shown in Figure 2.5. Therefore, dataset total has 2957, out of which Crypto were 1815 and Zeus were 1142. A standard scaler transformation and label encoding were used to convert Crypto into 0 and Zeus into 1. To assess the robustness of the deep learning method, Keras Deep Learning and SVM were used in model development. Using train-test split resampling, the dataset was divided into train and test sets, and the prediction accuracy was determined. Based on the results of the test and train sets, the top accuracy of the Keras Deep Learning model is 97%. Other result was reported in Table 2.3.

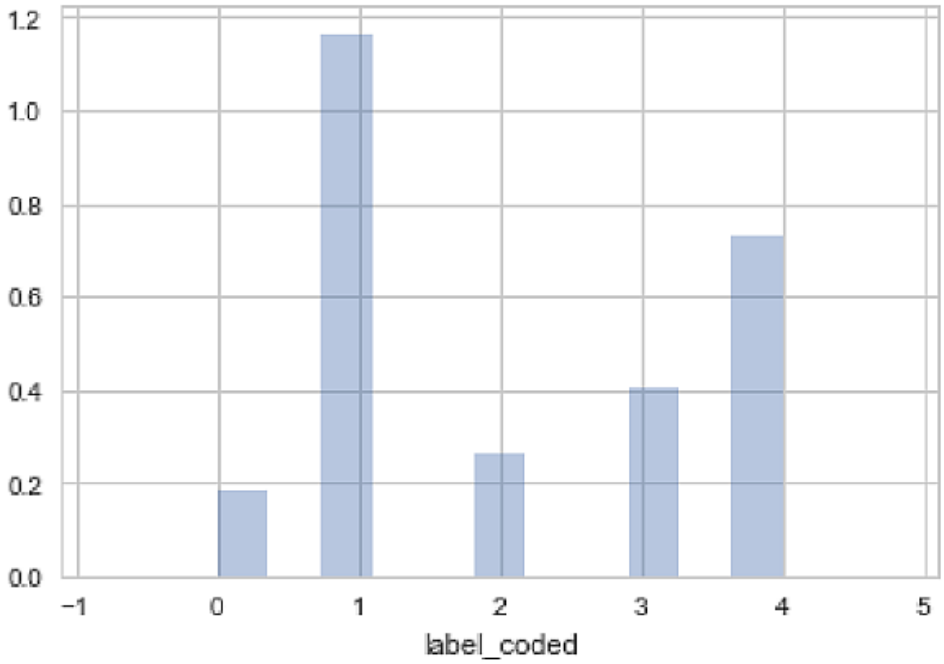

Figure 2.5: Class distribution: 'APT':0, 'Crypto':1, 'Locker':2, 'shadowbrokers':3, 'Zeus':4. (Masabo E, Kaawaase K S, Sansa-Otim J, 2018)

Table 2.3: Evaluation Results

Evaluation Metric	SVM	Keras Deep Learning
Precision	0.9489	0.9636
Recall	0.9139	0.9754
F1	0.9311	0.9695
Accuracy on Training Set	0.95	0.97
Accuracy on Test Set	0.94	0.97

2.6.2 Different Datasets

In order to enhance the performance of existing widely used algorithms, Morales-Molina, Carlos Domenick, et al. (2018) suggested an approach for classifying malware that makes use of the RF algorithm. To test their methodology, two datasets were used for classifying malware, which are ClaMP Raw-5184 and CSDMC_API_Train. There are 5184 samples in the ClaMP Raw-5184 dataset, 2683 of which are malware and 2501 of which are benign, comprising 55 characteristics. There are 388 samples in the CSDMC_API_Train dataset altogether, 320 of which are malicious programs and 68 benign ones. In feature selection and extraction, the ClaMP Raw-5184 dataset was processed using PCA algorithms to reduce dimensionality, while the CSDMC_API_Train dataset was processed to turn textual sequences into terms-weight using the TF-IDF algorithm. After processing, the data was divided into training and test sets. The researcher utilized the training sets as the input for a variety of machine learning algorithms, including LR, DT, RF and SVM. The performance of the trained models was then evaluated using the test sets. As can be observed in Table 2.4, Table 2.5, Figure 2.6, and Figure 2.7, RF method has the best accurate classification across various datasets.

Table 2.4: Results for The CLAMP RAW-5184

Classifier Algorithm	Precision	ReCall	F1 Score
LG	66.91	67.77	67.34
RF + PCA	89.83	89.50	89.67
DT	87.73	88.21	87.97
SVM	61.54	78.08	68.83

Table 2.5: Results for CSDMC_API_Train

Classifier Algorithm	Precision	ReCall	F1 Score
LG	85.12	93.27	96.33
RF + TF-IDF	96	98.72	96.54
DT	86	75	80
SVM	89.3	98.3	94.16

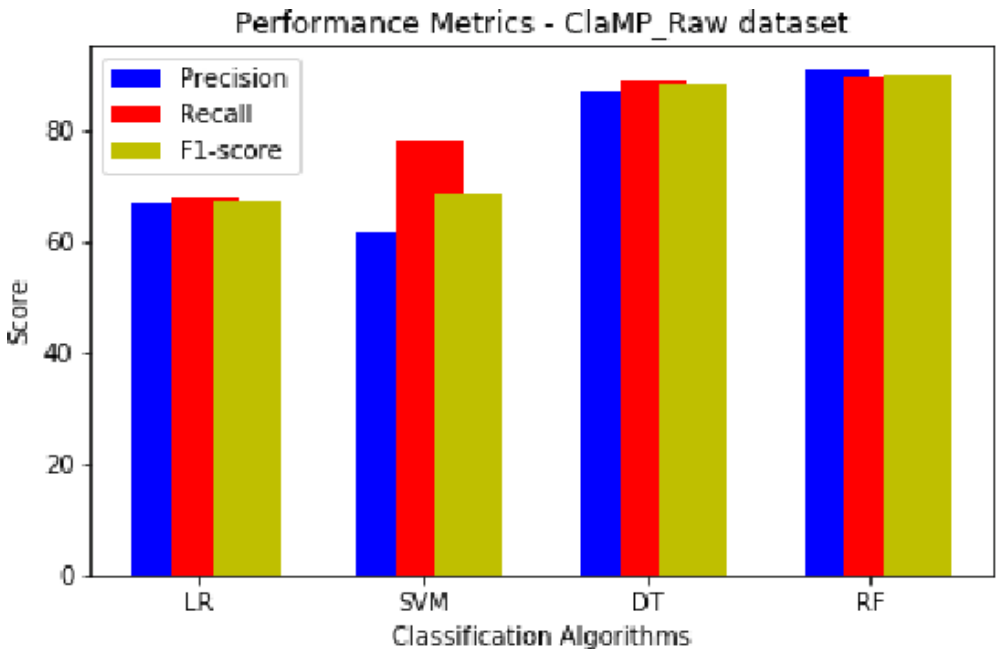

Figure 2.6: Results for the ClaMP Raw5184 Testing Set (Morales-Molina, Carlos Domenick, et al, 2018)

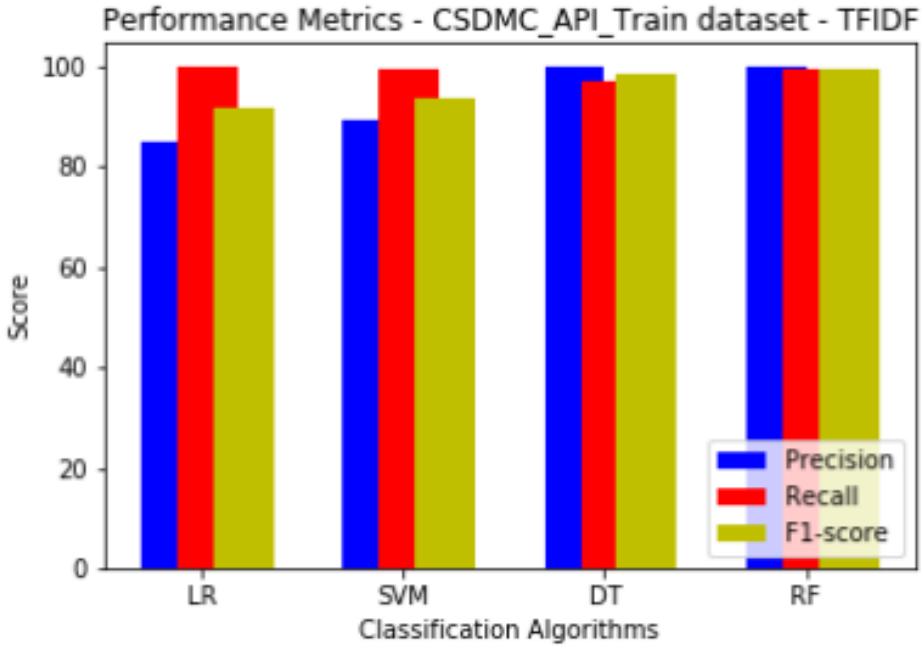

Figure 2.7: Results for the CSDMC_API_Train Testing Set (Morales-Molina, Carlos Domenick, et al, 2018)

Hadiprakoso, Kabetta, & Buana (2020) proposed a novel approach to malware detection, which not only focuses on achieving high accuracy but also prioritizes being resource efficient. This research went through several procedures as shown in Figure 2.8. There were two datasets they used to test the model with different malware analysis, which are Android Malgenome project and DREBIN (Static analysis) and CICMalDroid (Dynamic analysis). Given that the datasets were unbalanced, the SMOTE method and oversampling methodology were utilized to correct the imbalance. The 10-fold cross-validation approach was then used to divide the datasets into training and validation groups. SVM, Decision Tree, RF, Naive Bayes, K-NN, MLP, and GB are just a few of the machine learning and deep learning algorithms that were used in the model development. PCA was used for feature extraction to reduce the number of features while still maintaining the key information. A malware analysis model based on hybridization was developed by combining the findings from static and dynamic analysis. Through experiment results, GB performed the best overall in terms of precision and response time, with an accuracy of around 96% in static model tests as shown in Table 2.6 and around 99% in hybrid model tests as shown in table 2.7.

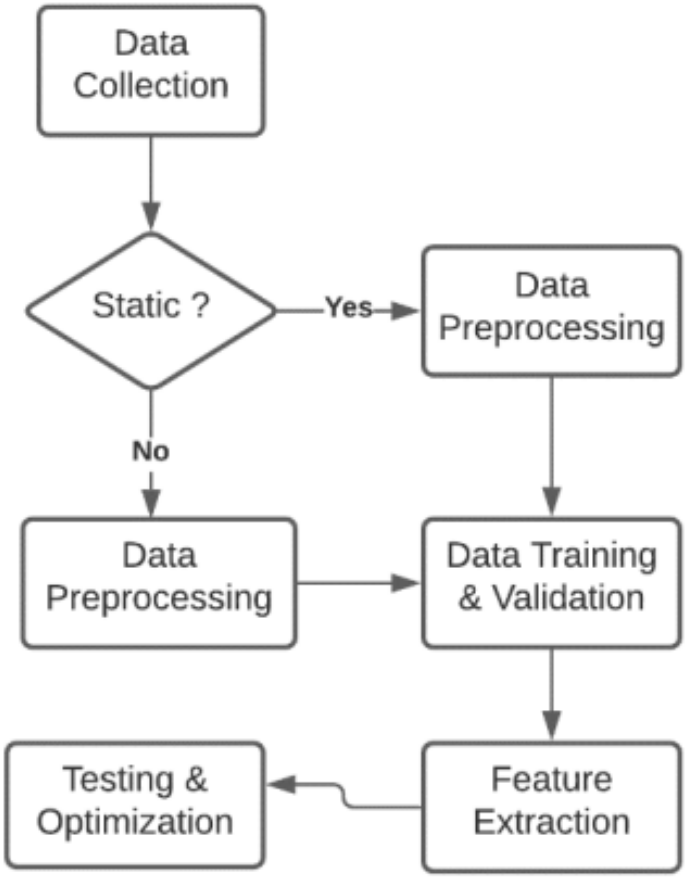

Figure 2.8: Overview Research Methodology (Hadiprakoso, Kabetta, & Buana, 2020)

Table 2.6: Static Model Tests Results

Algorithm	Average Score	Train & Val. Time (In Seconds)
SVM	0.943572	103.566
K-NN	0.808146	1.722
MLP	0.960026	205.45
RF	0.959159	23.33
DT	0.929516	4.092
Naïve Bayes	0.808146	2.769
GB	0.963543	139.897

Table 2.7: Hybrid Model Tests Results

Algorithms	Average Score	Prediction time (In Seconds)
GB	0.993572	0.667
RF	0.978146	1.222
MLP	0.950026	2.455

Chowdhury, Mozammel, Azizur, and Rafiqul (2017) proposed a solution for malware detection that uses a hybrid architecture that combines an MLP neural network with a BAM. Their approach is outlined in Figure 2.9. In this work, two different datasets were used, which are malware dataset and benign dataset. There are 10920 benign files from websites like Download.com and Softpedia.com and 41265 malicious files from VX Heaven. They merge these datasets to create a new common dataset. The suggested method uses behavior-based API sequences for malware analysis along with signature-based n-gram features. They employ SVM, Decision Tree (J48), Nave Bayes, and RF to evaluate the performance of their proposed technique with existing machine learning classifiers. Each classifier was validated using K-fold cross validation with K=10. Evidently, they discovered that the classifier performs better when K = 10. The result was reported that our proposed classifier gets a highest accuracy of 98.6% among machine learning classifier as seen in Table 2.8.

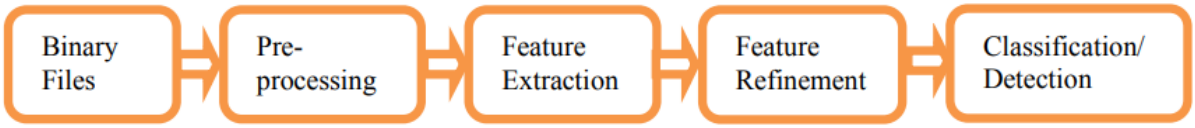

Figure 2.9: Architecture of the proposed malware detection system (Chowdhury, Mozammel, Azizur & Rafiqul, 2017)

Table 2.8: Result of Overall Accuracy for Different Classifier

Methods	Accuracy (%)	TPR (%)	FPR (%)
Naïve Bayes	88.5	89	12
J48	91.1	92	8
RF	94.8	95.5	6
SVM	95.7	97	3.6
Proposed Approach	98.6	99.5	2

2.7 Summary of Methods

No.	Authors	Classifier	Dataset Used	Result	Limitations
1.	Masabo, E. (2019)	A NFE approach based on structural and behavioural features	Malware Training Sets (APT1, Locker, Crypto, ShadowBrokers, Zeus)	94% of Gradient Boosting model.	Dataset was difficult to find filtered samples and their variants.
2.	Masabo E, Kaawaase K S, Sansa-Otim J (2018)	A malware detection approach based on big data analytics and machine learning	Malware Training Sets (Crypro & Zeus)	97% of Keras Deep Learning on both train and test sets	The training process of model is slow.
3.	Morales-Molina, Carlos Domenick, et al (2018)	Malware classification using a RF	ClaMP Raw-5184 & CSDMC_API_Train	89.83% of RF + PCA and 96% of RF + TF-IDF.	Not mentioned in paper.
4.	Hadiprakoso, Kabetta, & Buana (2020)	A hybrid-based malware analysis model	Android Malgenome project and DREBIN & CICMalDroid	Around 96% in static model tests and around 99% in hybrid model tests	Hybrid analysis is failed to ensure the efficiency of resource.
5.	Chowdhury, Mozammel, Azizur & Rafiqul (2017)	A malware approach with integrated features and employing hybrid network (BAM with MLP)	VX Heaven & online sources like Download.com and Softpedia.com	98.6% of proposed approach	Not mentioned in paper.

CHAPTER 3 METHODOLOGY

3.1 Introduction

In this project, several important stages were required to be performed, which are data collection, data pre-processing, data splitting, feature selection, model construction, model evaluation and application deployment with machine learning API. Figure 3.1 shows the flow of methodology process.

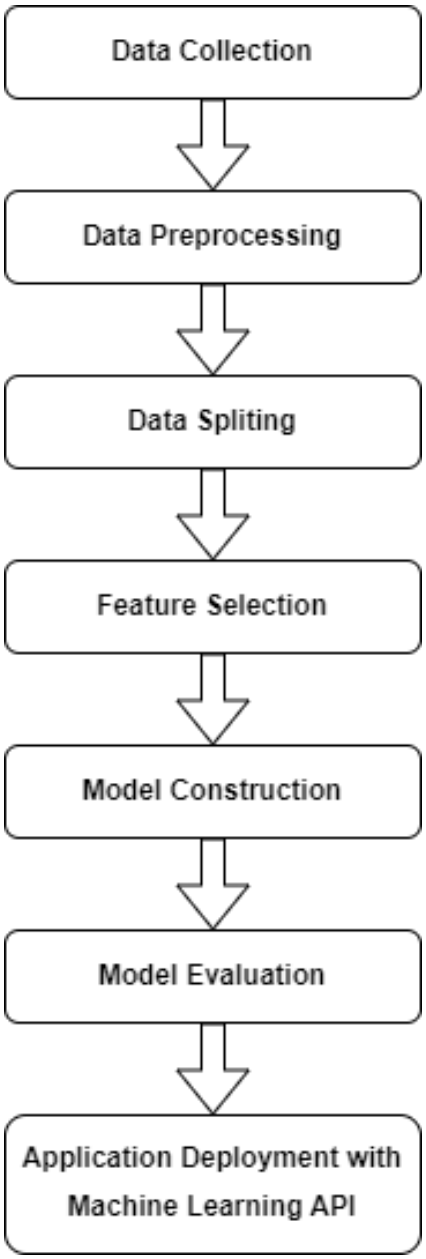

Figure 3.1: Flow of Methodology Process

3.2 Data Collection

In this project, the dataset used was compiled by gathering various features of executable malware from a public repository on Github, provided by Marco Ramili (Ramilli, Marco, 2016). This dataset consists of multiple folders, each containing numerous JSON files that store various characteristic and behaviors of different types of malwares. It is collected by combining multiple features extracted from various executable malware by using sandbox tools. There are all malware files in this dataset without any benign files. There are five families of malware used for classification, such as APT1, Crypto, Locker, Zeus and Shadowbrokers. Table 3.1 shows the five families of malware used in the dataset.

Table 3.1: Malware Families Used

Malware Family	Number of Samples
APT1	292
Crypto	2020
Locker	431
Zeus	2019
Shadowbrokers	1270
Total	6032

3.3 Exploratory Data Analysis

An Exploratory data analysis (EDA) was conducted that investigate on data used to discover their patterns and provide data visualization methods. Several data visualization methods were used in EDA, such as bar chart analysis, box plot analysis, density plot analysis, scatter plot analysis, confusion matrix analysis and histogram analysis. Due to the large number of features, only some of them were used in EDA.

3.3.1 Bar Chart Analysis

Bars indicate the counts of observations having five malware families in each feature as show n in Figure 3.2. The number of each is feature is different, and the highest number of features is 6029 of “str”, while the lowest number of features is 10 of “sig_creates_leargekey”. Crypto

family is described by almost all the available features, while shadowbrokers family is descri bed by very few features in the dataset.

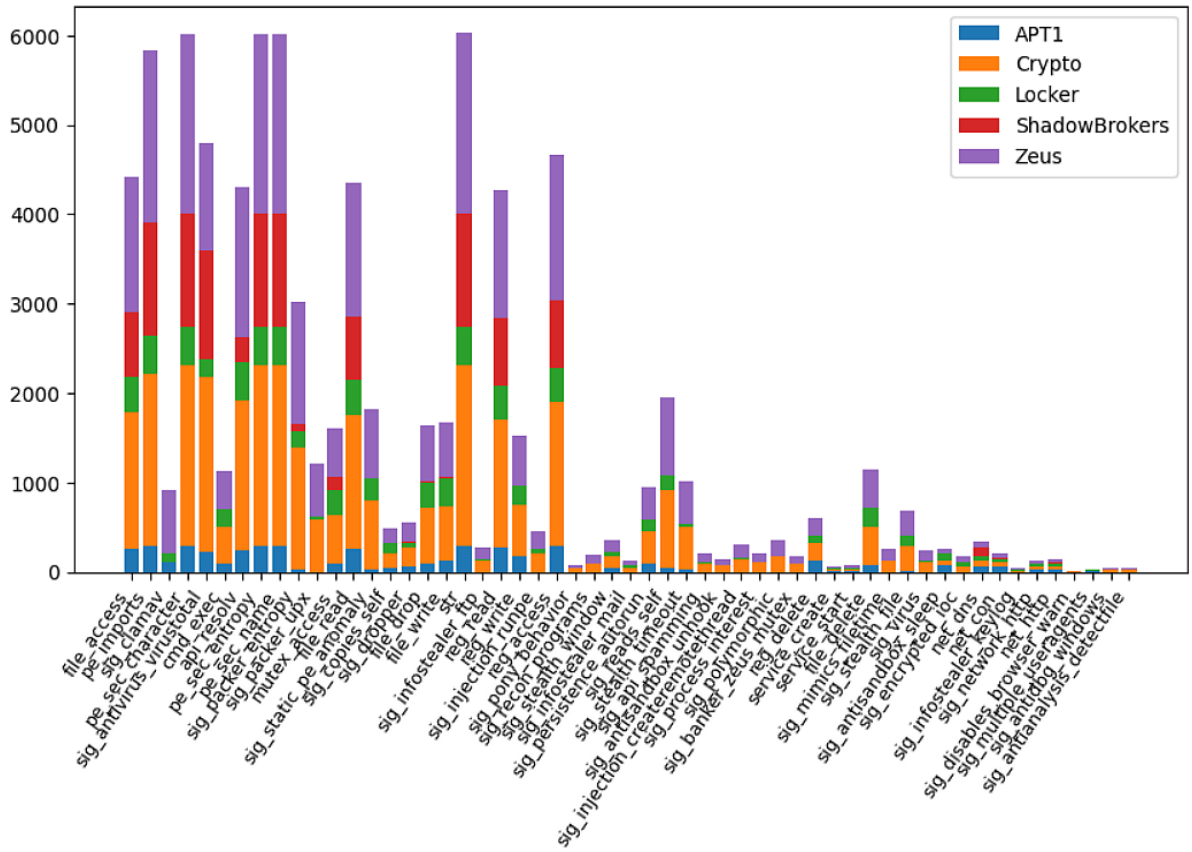

Figure 3.2: Distribution of Features in Five Families Malware

3.3.2 Box Plot Analysis

The boxplot shape shows some features are normally distributed or skewed as shown in Figure 3.3. The feature of “file access” is negative skew, “pe_imports” is normal distributions, and “api_resolv” is positive skew. The line at the center represents the median of all the data points. There was no outlier observed in this dataset. Outliers are the data points that falls outside of the interquartile range, which is calculated as (Q1-1.5(Q3-Q1) or Q3+1.5(Q3- Q1)). The longer the box, the more dispersed the data, such as “pe_sec_entropy”. The smaller the box, the less dispersed the data, such as “pe_sec_character”.

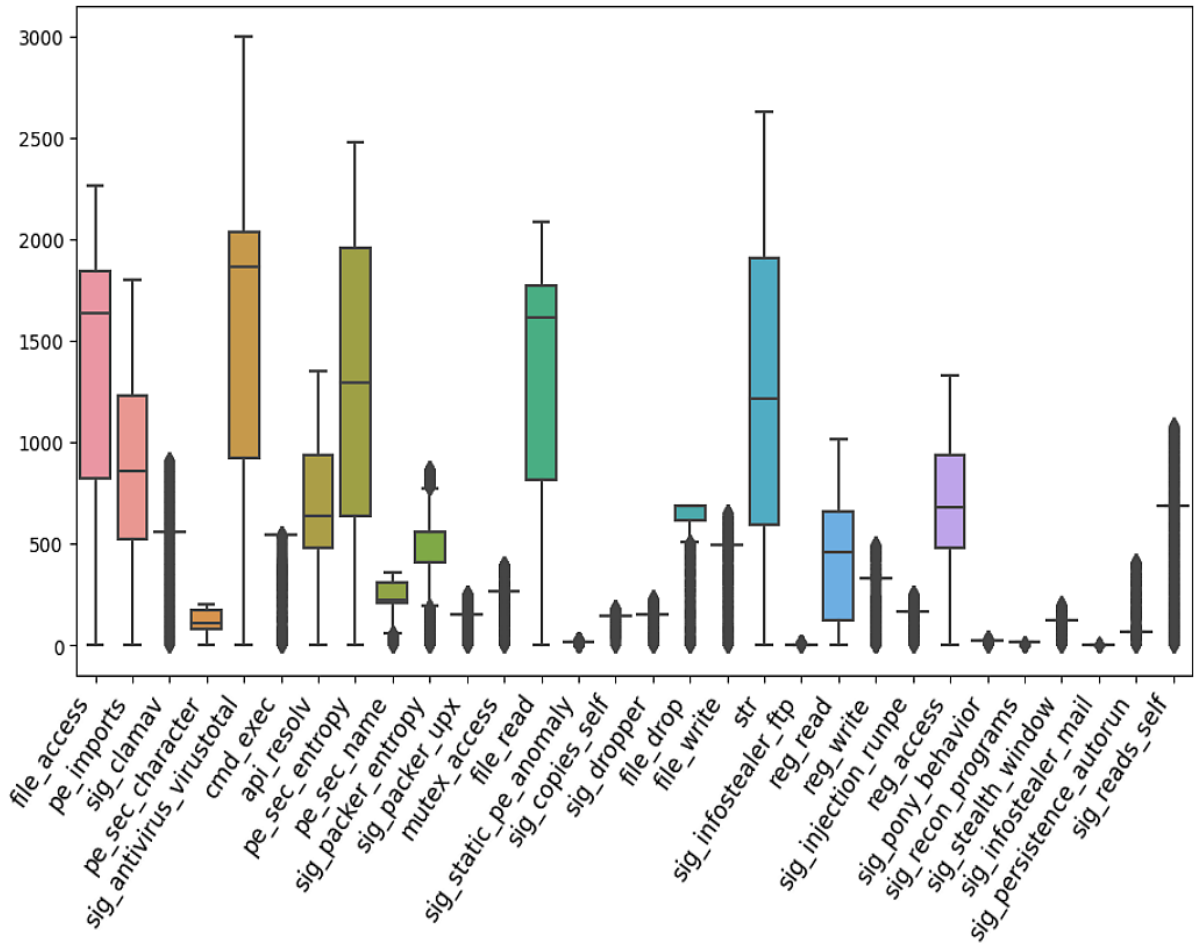

Figure 3.3: Box Plot Analysis (First 30 Features)

3.3.3 Density Plot Analysis

A density plot shows the density curves, which are shapes and sizes of some features as shown in figure 3.4. It is allowed to see some of the feature is left skewed, right skewed, or not skew. The feature of "file_access" is right skewed, which means the mean is greater than the median. The other one "pe_imports" is no skew, which means the mean is equal to the median. The features such as "pe_sec_character", "api_resolv" and "pe_sec_name" had multiple peaks, which is called bimodal. The feature such as "sig_clamav" and "sig_packer_upx" have only one peal, which is called unimodal. The features only have one peak is useful for the prediction.

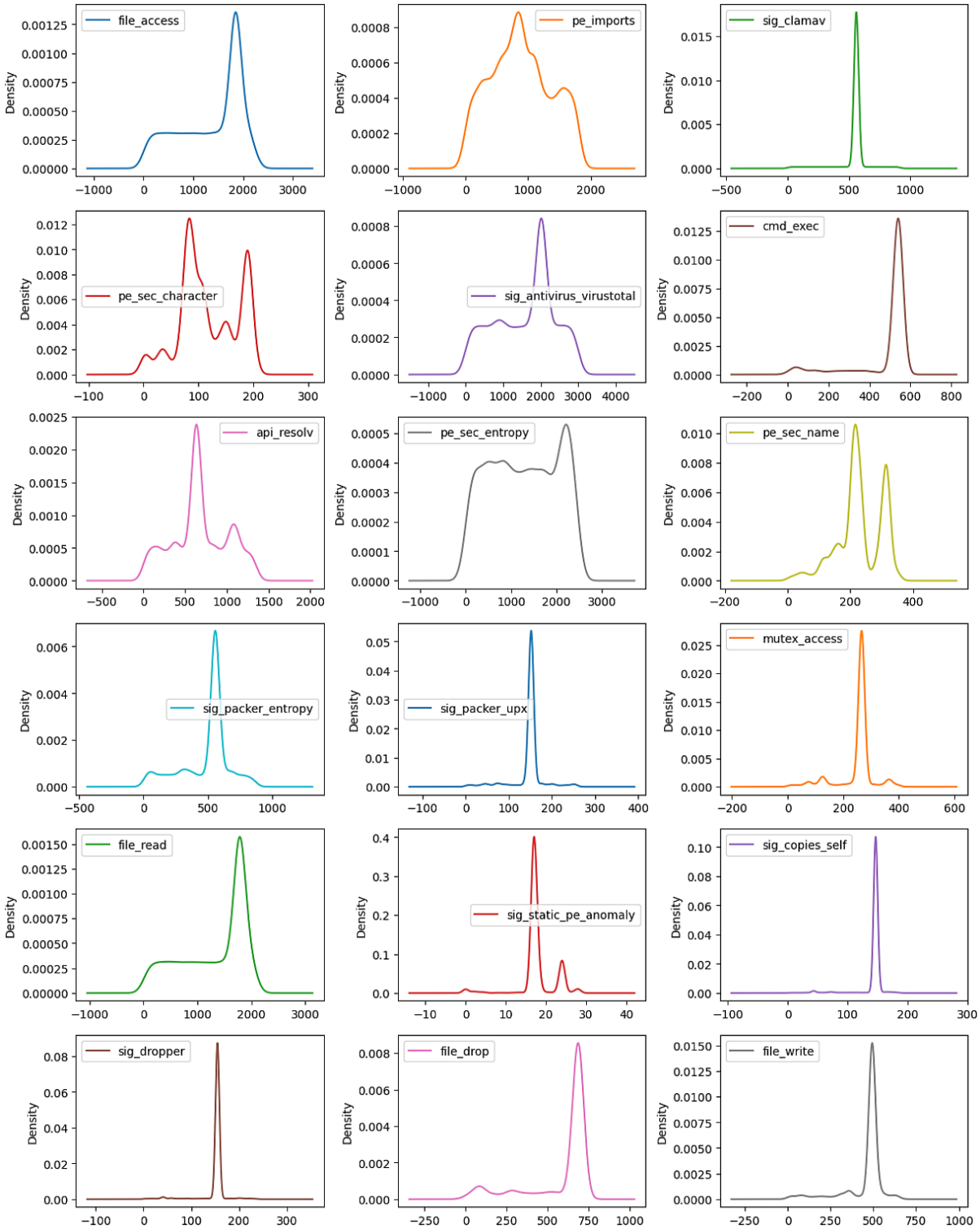

Figure 3.4: Density Plot Analysis (First 18 Features)

3.3.4 Scatter Plot Analysis

Scatter plot analysis is used to determine the relationship between two features in the dataset as shown in Figure 3.5. Table 3.2 shows each malware family represent their own colour in the scatter plot. The data point of five malware families can be seen in the relationship of different features. Overall, the relationship between different features has multiple clusters of points due to the five malware families, which can indicate the features is not useful for the prediction. The Shadowbrokers family displays a stronger relationship that produce a tighter clustering of data points. The relationship between “file_access” and “file_read” was strong, positive and linear. The other malware families show the relationship between features were moderate, negative, and linear.

Table 3.2: Colour of Malware Families used in Scatter Plot

Families	**Colour**
APT1	Blue
Crypto	Orange
Locker	Green
Zeus	Purple
Shadowbrokers	Red

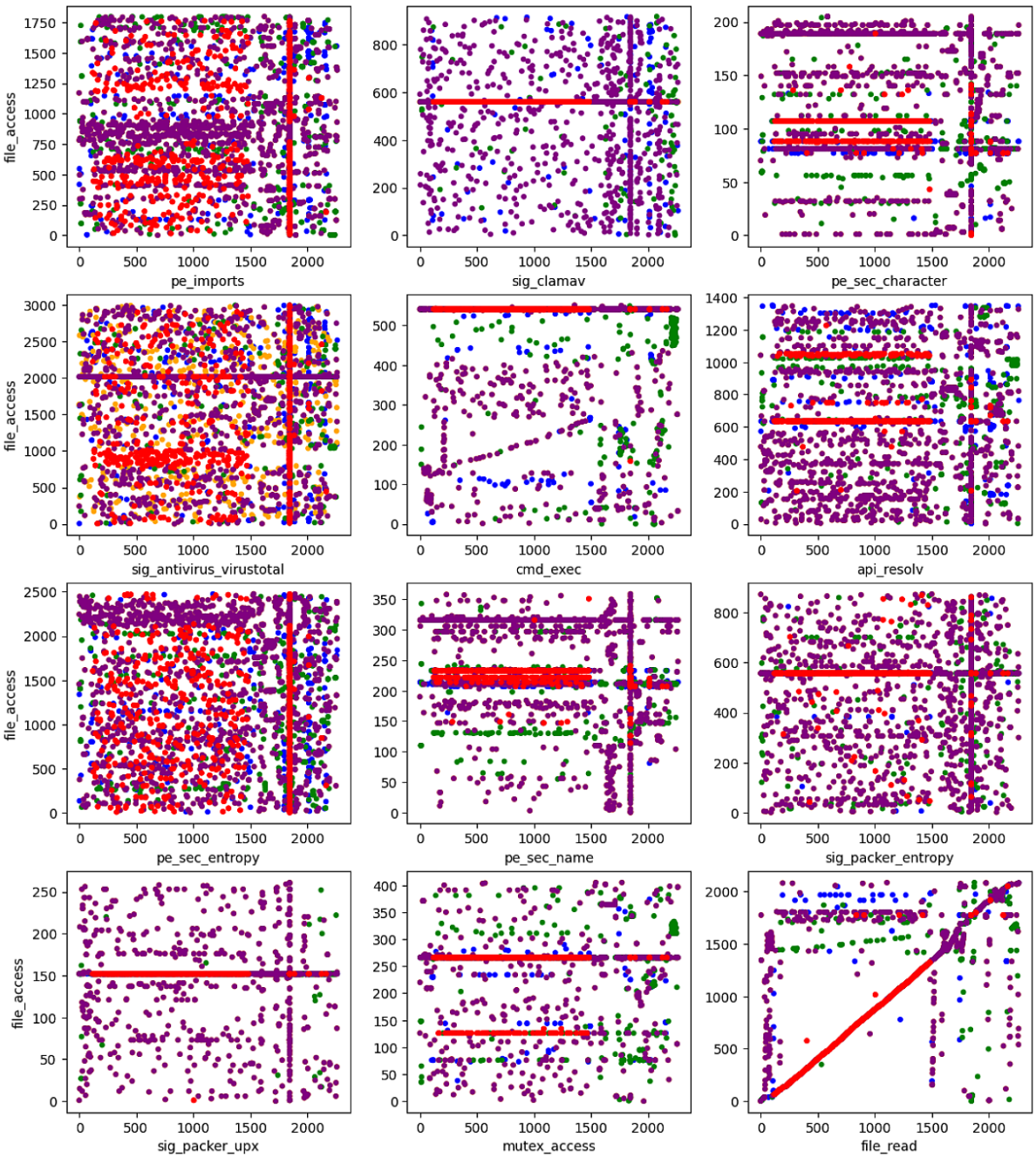

Figure 3.5: Scatter Plot Analysis

3.3.5 Correlation Matrix Analysis

A correlation matrix is a table to show the value of the correlation coefficient between multiple features. Table 3.3 shows the correlation coefficient is a value between -1 to +1 that indicate the strength of the relationship between two features. Figure 3.6 shows confusion matrix with different colour correlation coefficient. The relationship between the features of “files_read” and “files_access” have a 0.81 of high correlation coefficient, which means it is strong positive and useful for the prediction. The relationship between the features of “sig_clamav” and “pe_sec_character” have 0 of correlation coefficient, which means it is not relationship and less useful for the prediction. The relationship between the features of “file_access” and “pe_sec_name” is -0.19, which is weak negative and worse for the prediction.

Table 3.3: Correlation Coefficient Represent the Relationship

Colour	**Value**	**Linear Relationship**
Linen	-0.25	Weak Negative
Tea Green	0	No linear Relationship
Maya Blue	+0.25	Weak Positive
Dodger Blue	+0.50	Moderate Positive
Blue	+0.75	Strong Positive
Egyptian Blue	+1	Perfect Positive

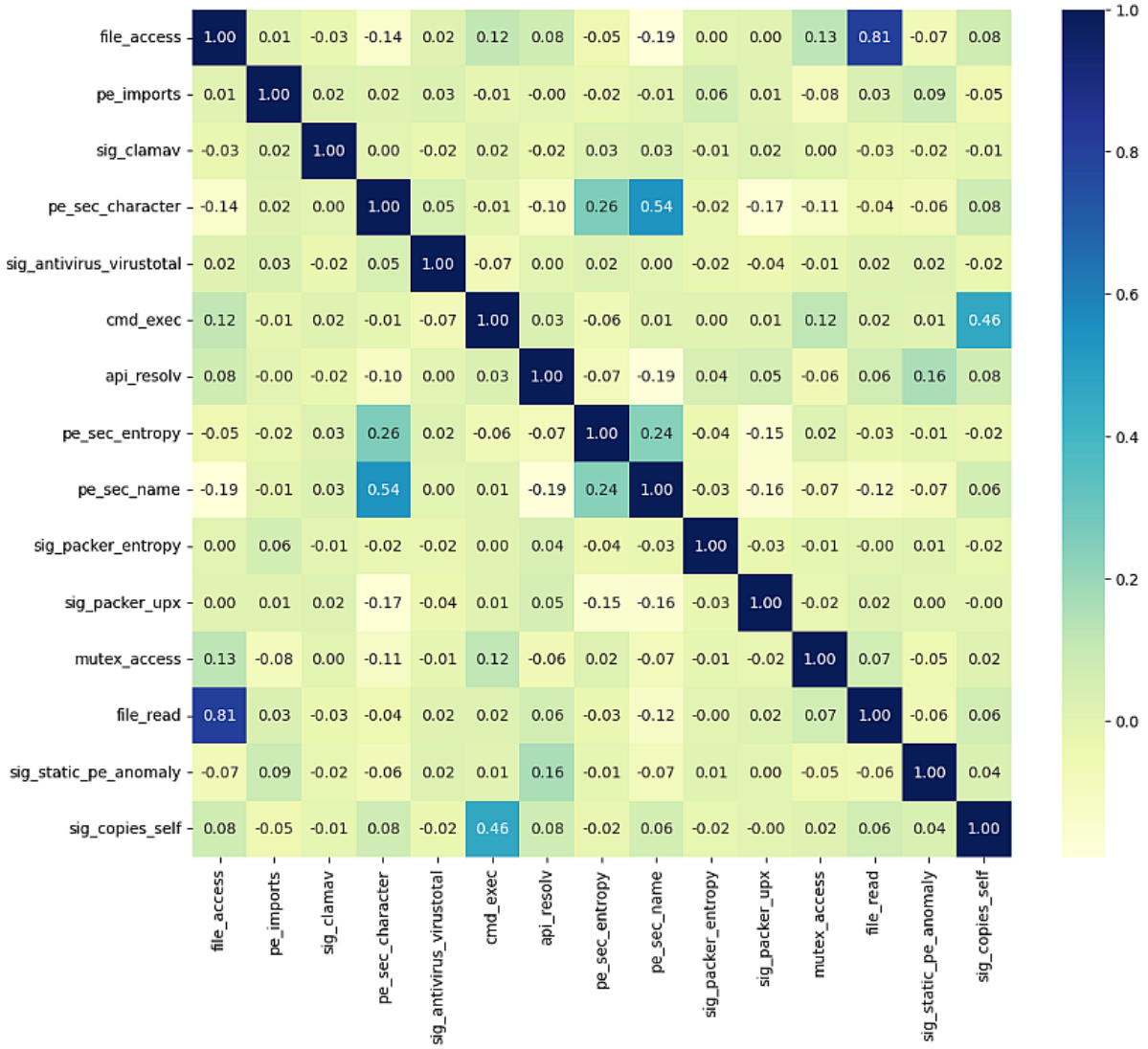

Figure 3.6: Plot of Correlation Matrix (First 15 Features)

3.3.6 Histogram Analysis

Histogram analysis is a method that visually presents the distribution of continuous data, as shown in Figure 3.7. This analysis employs a set of balanced data, setting it apart from the previous analyses conducted. The center of feature such as “seg_antivirus_virustotal” was 1500 and it was a right skewed distribution. The center of feature such as “pe_sec_entropy” was 1250 and it was left skewed distribution. These two features were skewed distribution, which was poor fit. The center of feature such as “api_resolv” was around 700 and it was symmetric distribution, which is the good fit. The features shown in the histogram did not have any outlier.

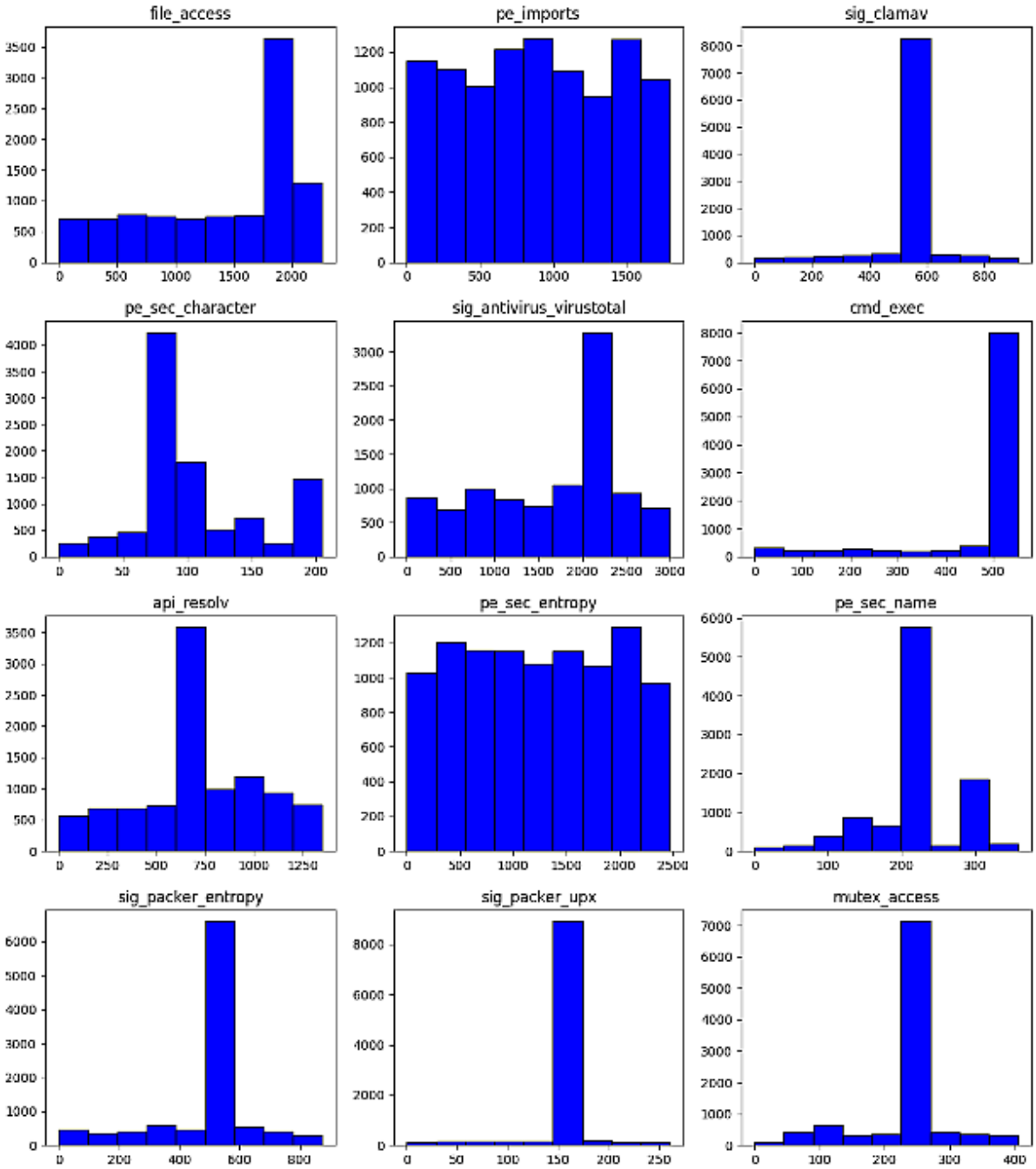

Figure 3.7: Histogram Analysis of Balanced Data (First 12 Features)

3.4 Data Preprocessing

Data preprocessing was performed to clean, format and organize data, making it suitable for a building and training machine learning models. This stage consisted of several steps, which are data integration, data cleaning, data transformation and data balancing.

3.4.1 Data Integration

Data integration was performed to combine data from different sources into a single dataset. The format of downloaded dataset are JSON files, so all the JSON files from five malware families must be combined and converted into a dataset of CSV format. Each JSON file had its own feature information, which it terms of dataset, refers to an individual row of data. Reading a single raw JSON file was the first step of data integration. A raw JSON file was read and converted into a dataset of CSV format as shown in Figure 3.8.

	entityId	entityType	event	eventTime	properties
api_resolv	2987	content	malware	2016-12-15T09:08:14.205+0000	8c70d8b3 5d067c04 4b03b39a a8b45284 a54e4c0e 8...
file_access	2987	content	malware	2016-12-15T09:08:14.205+0000	3d801aa5 47307f02 40d70447 3d801aa5 47307f02 9...
file_read	2987	content	malware	2016-12-15T09:08:14.205+0000	bda9f42e 249a6812 06416233 8b4939bc 080a35bc c...
label	2987	content	malware	2016-12-15T09:08:14.205+0000	Zeus
mutex_access	2987	content	malware	2016-12-15T09:08:14.205+0000	bb31764b 540a1cbf
pe_imports	2987	content	malware	2016-12-15T09:08:14.205+0000	839ca61c 82b1a48c a5c1f7de cec36c4f 805da537 9...
pe_sec_character	2987	content	malware	2016-12-15T09:08:14.205+0000	d78064fc eb95869a ad8225f3
pe_sec_entropy	2987	content	malware	2016-12-15T09:08:14.205+0000	f7ddd489 dba157bc 1a53cc44
pe_sec_name	2987	content	malware	2016-12-15T09:08:14.205+0000	c57b5e67 46040620 1f9f1073
reg_access	2987	content	malware	2016-12-15T09:08:14.205+0000	c0f87020 9ea6efd4 9ea6efd4 40d70447 c0f87020 7...
reg_read	2987	content	malware	2016-12-15T09:08:14.205+0000	9ea6efd4 40d70447 7e523f39 40d70447 c1c40440 0...
reg_write	2987	content	malware	2016-12-15T09:08:14.205+0000	f9fa10ba 5f706b1c

Figure 3.8: A JSON File Read as CSV Format

To combine and convert all the JSON files into a dataset, all the files from five malware families have to be combined into a folder. After loading the data, the data from all the files were appended into a data list to construct the dataset of CSV format. Reading data list with the data frame will generate each data into each row to form a completed dataset as shown in Figure 3.9. The raw dataset was ready once transforming the dataset into the prepared excel

files (CSV format). Figure 3.10 explained the algorithm how to generate dataset as CSV format from JSON files.

	entityId	entityType	event	eventTime	file_access	pe_imports	sig_clamav	sig_process_needed	pe_sec_character	sig_antivirus_virustotal	..
0	1000	content	malware	2016-12-15T09:05:57.778+0000	3d801aa509e592c498e833793d801aa5e38ac3949...	839ca61c82b1a48ca5c1f7decec36c4f805da5379...	08b3b9a7065508b8		d78064fceb95869aad8225f3	62e8160b b7373dfe2eb75834 9b49c8d0a184f465 6...	
1	1000	content	malware	2016-12-15T09:01:48.640+0000	3d801aa509e592c498e833793d801aa5e38ac3949...	839ca61c82b1a48ca5c1f7decec36c4f805da5379...	NaN		d78064fceb95869aad8225f3	62e8160b b7373dfe2eb75834 9b49c8d0a184f465 6...	
2	1001	content	malware	2016-12-15T09:01:48.684+0000	d41d8cd9913f9c4996da6d378b4939bc080a35bcc...	c9fab33ec50ba0ab1bdc1f0162c1e75e398706132...	NaN	NaN	0f24c321ad8225f3852ae548ad8225f3	7a7e816e 0b59c8fb90c0312f de63a19dbcee9ca5 4...	
3	1001	content	malware	2016-12-15T09:05:57.827+0000	d41d8cd9913f9c4996da6d378b4939bc080a35bcc...	c9fab33ec50ba0ab1bdc1f0162c1e75e398706132...	214b85fa	NaN	0f24c321ad8225f3852ae548ad8225f3	NaN	
4	1002	content	malware	2016-12-15T09:05:57.872+0000	NaN	a882be8ee8a591aefa3edd01bf1ade9af45919b2a...	NaN	NaN	9271b28c9271b28c2f0acaf7 ad8225f3 2f0acaf7	4ca137a9 9b64f8aa00a57f89 9ad5bbf8ab825f73 0...	

Figure 3.9: Dataset Combined from All JSON Files

```
data = glob('Datasets/*') # Read the folder
```

```
data_list = []

for d in data:
    with open (d) as doc: # Read all the files in the folder
        exp = json.load(doc) # Load the data
        data_list.append(exp) # Data is appended into a data list
```

```
df = pd.DataFrame(data_list) # Generate dataset with dataframe
```

```
row = pd.concat([df[['entityId','entityType','event','eventTime']],pd.json_normalize(df['properties'])],axis=1)
# Select the features in the properties
```

```
row.to_csv('Dataset2.csv', index=False) # Transform dataset into the excel file
```

Figure 3.10: Algorithm of Generating Dataset as CSV Format

3.4.2 Data Cleaning

Data cleaning was performed to identify and rectifying errors, inconsistencies, and missing data in the dataset. When merging the multiple data into a single dataset, there is potential for errors such as duplicate or inaccurate data to occur. Upon examination of the dataset, it is found that certain features contain missing or null values as shown in Figure 3.11. The method of handling the missing value or null values is to fill in a constant value like NA, instead of dropping them unless the feature is null values at all as shown in Figure 3.12.

I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
pe_sec_ch	sig_antivir	cmd_exec	api_resolv	pe_sec_en	pe_sec_na	sig_packer	sig_packer	label	mutex_ac	sig_injecti	file_read	sig_static_	sig_copies	sig_droppe	file_drop	file_write	str
d78064fc	62e8160b	8d6095a6	d998672a	f7ddd489	c57b5e67	ee655128	5ecff229	Zeus	aa8a9301	1a6d7805	3d801aa5	a9f2c818	3d801aa5	3d801aa5	00528c00	3d801aa5	1f9f1073
d78064fc	62e8160b	8d6095a6	d998672a	f7ddd489	c57b5e67	ee655128	5ecff229	Crypto	aa8a9301	1a6d7805	3d801aa5	a9f2c818	3d801aa5	3d801aa5	00528c00	3d801aa5	1f9f1073
0f24c321	7a7e816e	8d6095a6	ae5ad3a8	00d034f2	916e7571	3615174a		Crypto			d41d8cd9	a9f2c818			00000000	45b96339	916e7571
0f24c321	ad8225f3 8	8d6095a6	ae5ad3a8	00d034f2	916e7571	3615174a		Zeus			d41d8cd9	a9f2c818			00000000	45b96339	916e7571
9271b28c	4ca137a9	9b64f8aa 0	e6e88ee5	f7ddd489	df5bcfa6 9	26d42194		Zeus									df5bcfa6 2
9271b28c	4ca137a9	9b64f8aa 0	e6e88ee5	f7ddd489	df5bcfa6 9	26d42194		Crypto									df5bcfa6 2
ad8225f3	d24d7368	f6a66178 d	e6e88ee5	27579832	b7b66a05	6a234ff0		Crypto			bda9f42e	0475f535	a9f2c818				b7b66a05
ad8225f3	d24d7368	f6a66178 d	e6e88ee5	27579832	b7b66a05	6a234ff0		Zeus			bda9f42e	0475f535	a9f2c818				b7b66a05
9271b28c	5ad893ac	e90904e8 8	e6e88ee5	6c3aad79	916e7571	bdc4f61e		Zeus			bda9f42e	a9f2c818					916e7571
9271b28c	5ad893ac	e90904e8 8	e6e88ee5	6c3aad79	916e7571	bdc4f61e		Crypto			bda9f42e	a9f2c818					916e7571
9271b28c	d081f581	9b64f8aa 0	e6e88ee5	f7ddd489	df5bcfa6 9	18cac074		Crypto									df5bcfa6 2
9271b28c	d081f581	9b64f8aa 0	e6e88ee5	f7ddd489	df5bcfa6 9	18cac074		Zeus									df5bcfa6 2
9271b28c	199c2999	8d6095a6	aa9ee92a	609a9ece	916e7571	f3ff0e07	4782ab3c e2	Locker	11294b74	3f47bedd 9	0a245039	30ddd2e4	0f4137ed	97bc6c86 9	00000000	0a245039	916e7571
9271b28c	773ac449	fb8ca8e8 4	084c45c6	0fffb79e 2	c1336794	0f882a0c		Zeus				a28f50fa					abfc75e1
9271b28c	773ac449	fb8ca8e8 4	084c45c6	0fffb79e 2	c1336794	0f882a0c		Crypto				a28f50fa					abfc75e1
9271b28c	d2c1036c	8d6095a6	887e3e2b	2a175ad5	916e7571	b6236dc1		Crypto			3d801aa5	4e71ff71	98e83379				916e7571
9271b28c	d2c1036c	8d6095a6	887e3e2b	2a175ad5	916e7571	b6236dc1		Zeus			3d801aa5	4e71ff71	98e83379				916e7571
d78064fc	eb95869a	ad8225f3	b0cb98e9	f7ddd489	c57b5e67	c30542b1	37ac2c27	Zeus			3d801aa5	d81fbd71	98e83379				1f9f1073
d78064fc	a22067a8	05a1f277 3	b0cb98e9	f7ddd489	c57b5e67	c30542b1	37ac2c27	Crypto			3d801aa5	d81fbd71	98e83379				1f9f1073
9271b28c	2f0acaf7	ad8225f3 2f	2340ff16 3	63790b7d	916e7571	5617b9b7		Zeus			3d801aa5	de47fce5	98e83379				916e7571
9271b28c	31909eab	04a043ee 8	2340ff16 3	63790b7d	916e7571	5617b9b7		Crypto			3d801aa5	de47fce5	98e83379				916e7571
0f24c321	85b6735d	1fa01084	1b4faa03 4	66ba5bf7	916e7571	94fe2c55		Crypto									916e7571
0f24c321	f64ccb6c			66ba5bf7	916e7571	94fe2c55		Zeus									916e7571
9271b28c	b5523d5f	3df902ba d	fac82940	eff8a960 2	916e7571	a050642c		Zeus			3d801aa5	96a72a0a	98e83379				916e7571

Figure 3.11: Dataset before Data Cleaning

I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
pe_sec_ch	sig_antivir	cmd_exec	api_resolv	pe_sec_en	pe_sec_na	sig_packer	sig_packer	mutex_ac	file_read	sig_static_	sig_copies	sig_droppe	file_drop	file_write	str	sig_infoste	reg_read
9271b28c	b4adabef	8d6095a6	fa897852	81b16c30	916e7571	NA	NA	8907effb	d41d8cd9	NA	NA	NA	NA	d41d8cd9	916e7571	NA	c1c40440
9271b28c	NA	8d6095a6	fa897852	81b16c30	916e7571	NA	NA	8907effb 9	d41d8cd9	NA	NA	NA	NA	d41d8cd9	916e7571	NA	c1c40440
9271b28c	NA	NA	4294d6d6	111e4b9c	916e7571	NA	NA	NA	ea03fcb8	NA	NA	NA	NA	ea03fcb8	916e7571	NA	NA
9271b28c	ae3912eb	8d6095a6	ec2b83df	1c826789	916e7571	6a93484d	NA	acc19e13	3d801aa5	NA	NA	NA	00043800	3d801aa5	916e7571	NA	7cb31435
9271b28c	1426c3e4	8d6095a6	NA	0a8f373e	916e7571	NA	NA	NA	NA	NA	NA	NA	NA	NA	916e7571	NA	1fe39017
9271b28c	NA	NA	aa9ee92a	d92d2ee2	916e7571	NA	NA	NA	d41d8cd9	NA	NA	NA	NA	NA	916e7571	NA	c1c40440
9271b28c	NA	NA	NA	3859c0a4	916e7571	NA	NA	NA	NA	NA	NA	NA	NA	NA	916e7571	NA	NA
9271b28c	e8016325	8d6095a6	4d0fa58e	4a4212d1	916e7571	ee58d6ff	NA	540a1cbf	3d801aa5	NA	NA	3d801aa5	00001c00	3d801aa5	916e7571	NA	fb216d9e
41	14a949da	f8091e41	576a254b	f8091e41	353ffc5b	f8091e41	de2bedc1	ff389bb9	f5e0533b	9f7c35ba	85f342c9	fd7a1487	58ec0cff	09e592c4	c15ab7a8	09e592c4	7dd74765
1e4e0	fb216d9e	cbb84746	1471e4e0	c4ca4238	efc50fc9	35e4625d	35e4625d	06238034	35e4625d	a7eb6bbe	1553200d	a71b44c5	73d5c173	2579bf65	2579bf65	0aaa8742	2579bf65
ffa	2e5d8aa3	e4c116ba	e4c116ba	70a17ffa	2e5d8aa3	475b3877	475b3877	70a17ffa	2e5d8aa3	ba582312	ba582312	70a17ffa	2e5d8aa3	b9a3dab2	b9a3dab2	70a17ffa	2e5d8aa3
31c	950a0c49	f14d1698	950a0c49	67daf92c	950a0c49	7a946f10	950a0c49	54ed4630	950a0c49	145adfed	950a0c49	5157c798	950a0c49	baec6461	950a0c49	e91e6348	950a0c49
NA	NA	NA	NA	NA	NA	e17184bc	NA	NA	3d801aa5	NA	fed36e93	NA	NA	NA	8757a543	e165d4f2	NA
9271b28c	040bbf7a	8d6095a6	be1d1888	356e382c	916e7571	NA	NA	5d0fa1c9	d41d8cd9	NA	55451ac9	55451ac9	00004c00	55451ac9	916e7571	NA	41e18397
9271b28c	619af825	NA	aa9ee92a	9b511809	916e7571	NA	NA	NA	NA	NA	NA	NA	NA	NA	916e7571	NA	1fe39017
9271b28c	9131b496	NA	aa9ee92a	69dbcc21	916e7571	NA	NA	NA	3d801aa5	NA	NA	NA	NA	NA	916e7571	NA	2e5d8aa3
9271b28c	c30ddf56	NA	aa9ee92a	2a81859d	916e7571	NA	NA	NA	3d801aa5	a9f2c818	NA	NA	NA	NA	916e7571	NA	2e5d8aa3
9271b28c	de3e7de7	NA	e326ee78	53382718	916e7571	NA	NA	646767f9	45b96339	NA	NA	NA	00000000	45b96339	916e7571	NA	2e5d8aa3
ad8225f3	8f642bf0	NA	NA	d9e7f554	b7b66a05	NA	NA	NA	NA	0475f535	NA	NA	NA	NA	b7b66a05	NA	NA
9271b28c	05326120	NA	bc5e5cec	768148e9	916e7571	NA	NA	NA	d41d8cd9	NA	NA	NA	NA	NA	916e7571	NA	7214dce3
9271b28c	719a0e52	8d6095a6	bc5e5cec	df3ccd30	916e7571	NA	NA	NA	d41d8cd9	NA	NA	NA	00038c00	bda9f42e	916e7571	NA	cd32a766
9271b28c	c4593977	NA	c0b14a53	2a81859d	916e7571	NA	NA	NA	bda9f42e	NA	NA	NA	NA	NA	916e7571	NA	c1c40440
9271b28c	ebaf35bc	NA	fa897852	a18b4c04	916e7571	NA	NA	NA	d41d8cd9	NA	NA	NA	NA	d41d8cd9	916e7571	NA	c1c40440
9271b28c	1df6ba5d	NA	8e16a687	f7ddd489	916e7571	1e4fdf05	NA	NA	bda9f42e	NA	NA	NA	NA	NA	916e7571	NA	c1c40440

Figure 3.12: Dataset after Data Cleaning

3.4.3 Data Transformation

Data transformation was performed to change the format, structure, or values of data. Before performing data transformation, the dataset was categorical variables, such as string data types as shown in Figure 3.14. The categorical data was unable to use on some machine learning (ML) algorithms to train the model. The categorical data had to be converted into the numerical data for further analysis. A general algorithm used to convert the data into numeric data is Label Encoding technique as shown in Figure 3.13. Label encoding was chosen to use in this process because it is a popular encoding technique for handling categorical variables. Alphabetical ordering was used to assign a unique integer to each label as shown in Figure 3.15. Malware families are represented as: 'APT':0, 'Crypto':1, 'Locker':2, 'Zeus':3, 'shadowbrokers':4.

```
# Import required libraries
from sklearn.preprocessing import LabelEncoder

# Creating instance of labelencoder
cols = df.iloc[:, 4:92].columns

# Assigning numerical values and storing in another column
df[cols] = df[cols].apply(LabelEncoder().fit_transform)
df
```

Figure 3.13: Algorithm of Label Encoding

	entityId	entityType	event	eventTime	label	file_access	pe_imports	sig_clamav	pe_sec_character	sig_antivirus_virustotal	...	sig_antianalysis
0	780	content	malware	2016-12-15T09:01:42.068+0000	APT1	3d801aa576498be898e83379d41d8cd9913f9c499...	21a39751b7847ae6afb0f3daf6735a66c9fab33eb...	NA	9271b28c2f0acaf7 ad8225f3 2f0acaf7	b4adabef d8802580 d93fa703 a7b4b7e2 1465158f 7...	...	
1	824	content	malware	2016-12-15T09:01:43.818+0000	APT1	3d801aa5a12ac23c98e83379d41d8cd9913f9c499...	21a39751b7847ae6afb0f3daf6735a66c9fab33eb...	NA	9271b28c2f0acaf7 ad8225f3 2f0acaf7	NA	...	
2	825	content	malware	2016-12-15T09:01:43.857+0000	APT1	ea03fcb820826a3c6bf6834fea03fcb820826a3cc...	ba22de758dddb18d5b59ec1366016422 4737ac20a...	NA	9271b28c2f0acaf7 ad8225f3	NA	...	
3	826	content	malware	2016-12-15T09:01:44.085+0000	APT1	3d801aa59c8b24b8437175ba8b4939bc080a35bcc...	888b1f30c7ce5a43b7847ae62b78df95f666c32cd...	NA	9271b28c2f0acaf7 ad8225f3 2f0acaf7	ae3912eb 6ea594a7 4cc59460 9c17a8f4 ae3912eb 3...	...	
4	679	content	malware	2016-12-15T09:01:36.290+0000	APT1	8d6095a60f4137ed3d801aa540770804f5ddaf0c3...	b11e642ee9b9792abe7c3552395f4f47ce04b343a...	NA	9271b28c2f0acaf7 ad8225f3	1426c3e4 0004a5f9 0cc49ed3 5b9f997d ac130e01 3...	...	

Figure 3.14: Dataset before Label Encoding

	entityId	entityType	event	eventTime	label	file_access	pe_imports	sig_clamav	pe_sec_character	sig_antivirus_virustotal	...	sig_antianal
0	780	content	malware	2016-12-15T09:01:42.068+0000	0	764	129	560	81	2280	...	
1	824	content	malware	2016-12-15T09:01:43.818+0000	0	984	129	560	81	2018	...	
2	825	content	malware	2016-12-15T09:01:43.857+0000	0	2164	1394	560	77	2018	...	
3	826	content	malware	2016-12-15T09:01:44.085+0000	0	946	962	560	81	2214	...	
4	679	content	malware	2016-12-15T09:01:36.290+0000	0	1743	1345	560	77	216	...	
...	...	...	...	...	...	...	...	...	...	...	...	
6027	1577487	content	malware	2017-04-22T16:24:17.276+0000	4	561	1235	560	88	1447	...	
6028	1577776	content	malware	2017-04-22T16:24:30.030+0000	4	1341	978	560	88	422	...	

Figure 3.15: Dataset after Label Encoding

3.4.4 Data Balancing

Data balancing was performed to balance out the dataset so that the result can be equally justified. Five malware families used in the dataset were imbalanced as shown in Figure 3.16. The malware families such as Cryto and Zeus, had a very high number of observations and the other had a very low number of observations. Cryto and Zeus families were 33.49% and 33.47% in the dataset, which were called majority class. APT1, Locker and Shadowbrokers families were 4.84%, 7.15% and 21.05% in the dataset, which were called minority class.

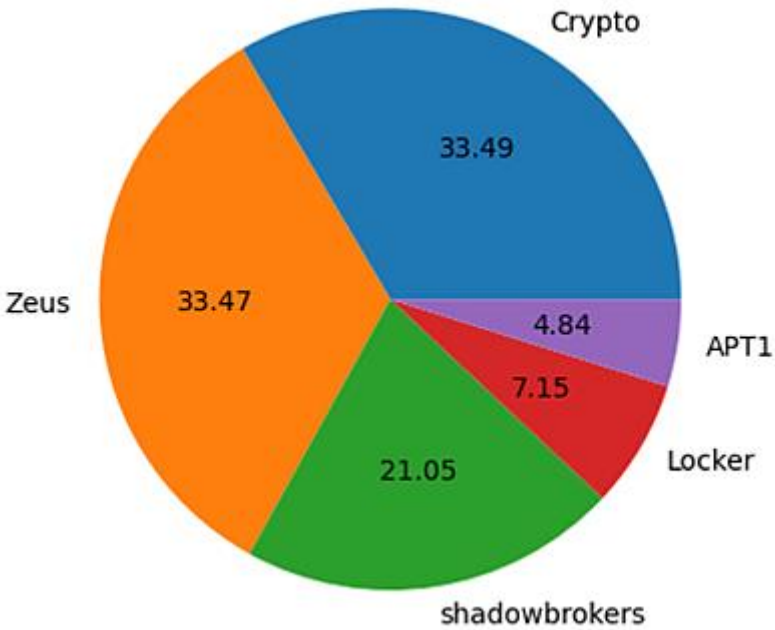

Figure 3.16: Imbalanced Data

To prevent the accuracy was no longer a proper measure, SMOTE (Synthetic Minority Over-sampling Technique) was performed to solve the imbalance problem of the dataset (Masabo, E, 2019). SMOTE was an oversampling technique, where it performed data augmentation by randomly increasing minority class data by replicating them. Figure 3.17 show the data size after using SMOTE technique. Each of the malware families had the same number of data, which were all 20% in the dataset.

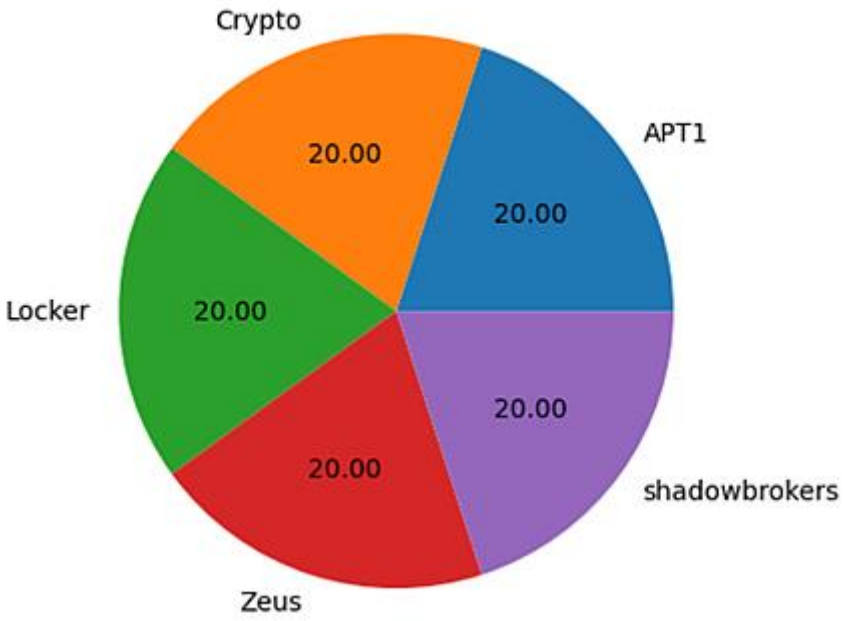

Figure 3.17: Balanced Data with SMOTE

3.5 Data Splitting

The dataset was divided into different sets for training and testing purposes. 80% of the data was used for training (8080 samples) and 20% for testing (2020 samples). The split was 70% for training (7070 samples) and 30% for testing (3030 samples). A 60% training set (6060 samples) and a 40% testing set (4040 samples) were created. The training set was used to develop machine learning models and feature sets, while the testing set was used to evaluate the models' performance. The algorithm was applied to perform data splitting as shown in Figure 3.18.

```
from sklearn.model_selection import train_test_split

# Splitting the data into a train and a dev set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state = 1)
```

Figure 3.18: Algorithm of Data Splitting

3.6 Feature Selection

Feature selection was performed to select the important features in the dataset before constructing machine learning models (Residentmario, 2018). Boruta was chosen as the feature selection algorithm in this project. It was a wrapper algorithm used to construct around Random Forest (RF) model. It used permutation of feature values to identify the importance of each feature. Figure 3.19 shows the algorithm of Boruta. After performing Boruta algorithm, the important features will be selected based on the feature ranking. Figure 3.20 shows the feature with ranking in balanced dataset. Figure 3.21 shows the feature with ranking in imbalance dataset. Only the feature with the feature ranking 1 were selected and utilized in training and testing the machine learning models.

```
from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

rfc = RandomForestClassifier(criterion='gini',max_depth=12, max_features='sqrt',
                             min_samples_split=10, n_estimators=200, random_state=0)

boruta_selector = BorutaPy(rfc, n_estimators='auto', verbose=2, random_state=0)

boruta_selector.fit(np.array(X_train), np.array(y_train))
```

Figure 3.19: Boruta Algorithm

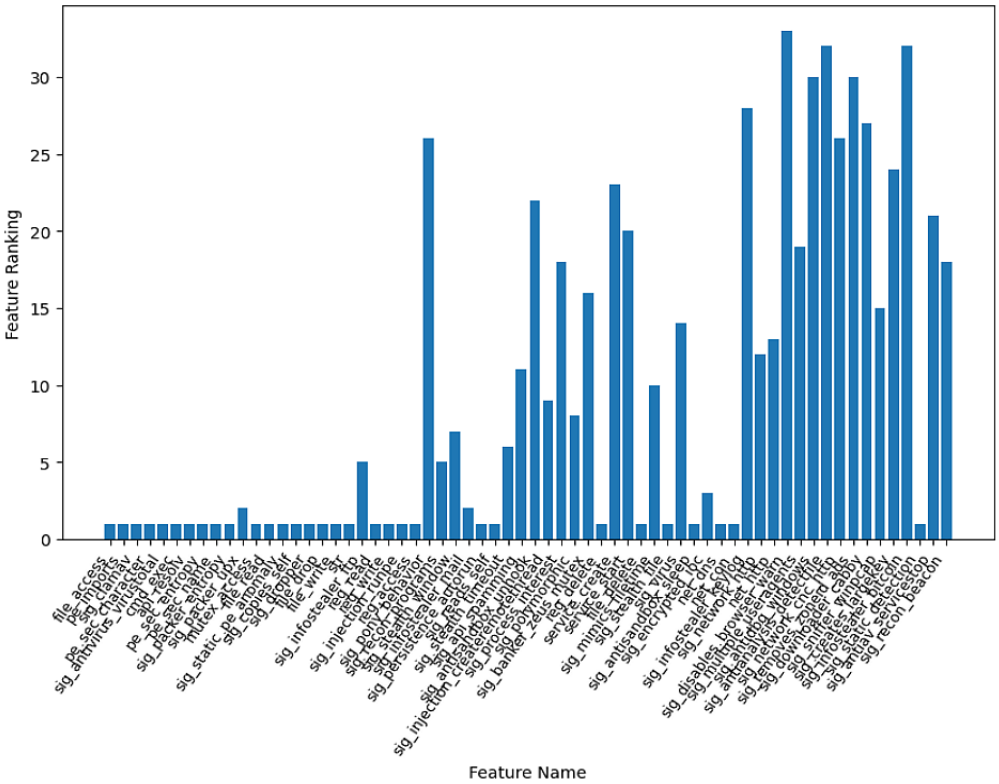

Figure 3.20: Features with Ranking (Balance Data)

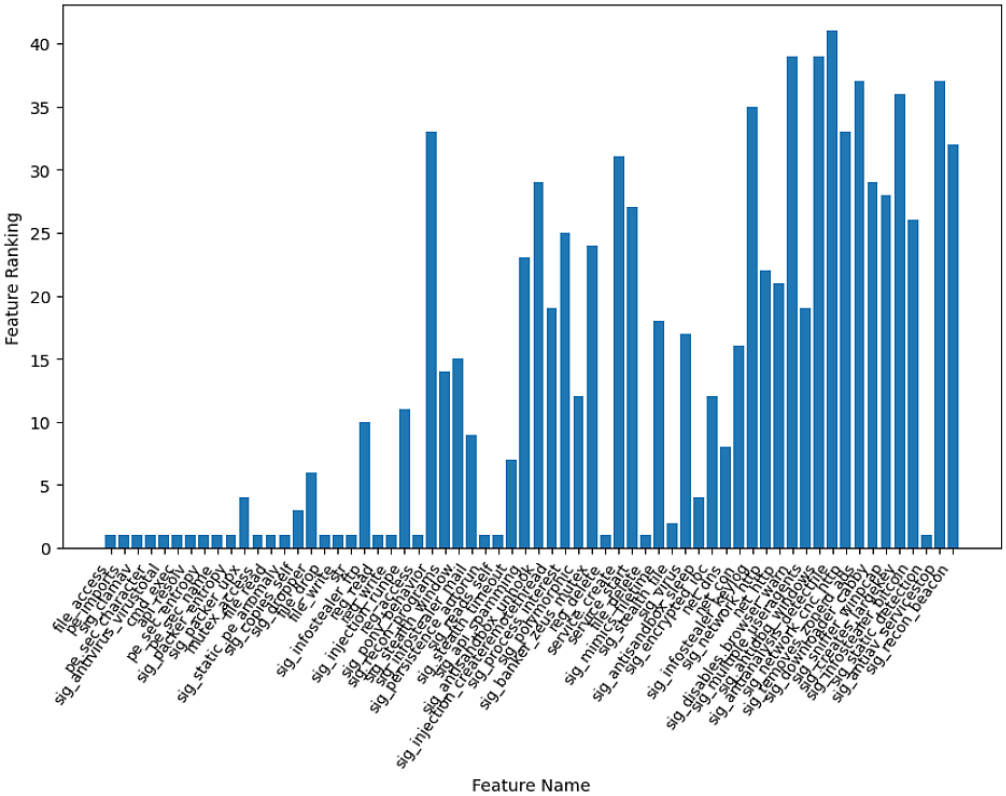

Figure 3.21: Features with Ranking (Imbalance Data)

3.7 Model Construction

In this project, the thrilling task was to classify data into five distinct malware families. Since the dataset consisted of more than two classes, a multi-class classification approach was adopted. To tackle this challenge, three machine learning algorithms, namely DT, RF and LightGBM were chosen. These algorithms were considered suitable for addressing the multi-class classification problem within the dataset.

3.7.1 Hyperparameter Tuning

Grid search was employed as the method for hyperparameter tuning in this project. It involved exhaustively exploring a range of possible hyperparameter values to determine the combination that yields the best results for the model as shown in Figure 3.22.

```
from sklearn.model_selection import GridSearchCV
import lightgbm as lgb

lgb_boruta = lgb.LGBMClassifier(random_state=0)

# Define search space
param_grid = {
    'boosting_type': ['gbdt', 'rf', 'dart'],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [200, 300, 400],
    'bagging_fraction': [0.5, 0.75, 0.9],
    'bagging_freq': [5, 10, 15]
}

# Create GridSearchCV object
grid_lgb = GridSearchCV(lgb_boruta, param_grid, refit=True, n_jobs=-1, cv=10, verbose=3, scoring='accuracy')

# Fit the data
grid_lgb.fit(X_important_train, y_train)

# Print best parameters and accuracy score
print("Best parameters found: ", grid_lgb.best_params_)
print("Best accuracy score found: {:.4f}".format(grid_lgb.best_score_))
```

Figure 3.22: Hyperparameter Tuning

3.7.2 Decision Tree

DT was applied for classification because they partition data based on features, facilitating interpretation and visualization. They handle nonlinear relationships, missing values, and outliers effectively.

Table 3.4 shows the best DT parameters for the imbalanced data. For all train-test splits, the “criterion” parameter, “max_depth” parameter and "max_features" parameter was set to "gini",

15 and "sqrt". The parameter "min_samples_split" varied, with values of 5, 2, and 10 for the 80-20, 70-30, and 60-40 splits.

Table 3.4: Best DT Parameters (Imbalanced Data)

	Train-Test Split		
Best Parameter	80-20	70-30	60-40
criterion	gini	gini	gini
max_depth	15	15	15
min_samples_split	5	2	10
max_features	sqrt	sqrt	sqrt

Table 3.5 presents the best parameters for the balanced data. For all train-test splits, the “criterion” parameter and “max_depth” parameter was once again set to "gini" and 16. The parameter "min_samples_split" varied, with values of 2, 5, and 2 for the 80-20, 70-30, and 60-40 splits. The "max_features" parameter changed to "log2" for the 80-20 split, while "sqrt" was used for the 70-30 and 60-40 splits.

Table 3.5: Best DT Parameters (Balanced Data)

	Train-Test Split		
Best Parameter	80-20	70-30	60-40
criterion	gini	gini	gini
max_depth	16	16	16
min_samples_split	2	5	2
max_features	log2	sqrt	sqrt

To construct a DT model, scikit-learn library was used to perform DT algorithm as shown in Figure 3.23. DT classifier was created with the best parameter for further processes. Training data was used to train the model using ‘fit’ method. Test data was used to predict the model using ‘predict’ method.

```python
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

# Create Decision Tree classifer object
dt1 = DecisionTreeClassifier(criterion='gini', max_depth=16, min_samples_split=2,
                             max_features='log2', random_state=0)

# Train the model on training data
dt1.fit(X_important_train, y_train)

# Use the forest's predict method on the test data
dt_predict = dt1.predict(X_important_test)
```

Figure 3.23: Decision Tree Algorithm

3.7.3 Random Forest

RF was applied for classification because they combine decision trees, benefiting from diversity and aggregation to boost performance. They mitigate overfitting by training each tree on random subsets of data and features.

Table 3.6 shows the best RF parameters for the imbalanced data. For all train-test splits, the “criterion” parameter, “min_samples_split” parameter and “max_features” parameter was set to "gini", 10 and “sqrt”. The "n_estimators " parameter changed to 400 for the 70-30 split, while 300 was set for the 80-20 and 60-40 splits. The “max_depth” parameter was changed to 8 for the 60-40 split, while 10 was set for the 80-20 and 70-30 splits.

Table 3.6: Best RF Parameters (Imbalanced Dataset)

	Train-Test Split		
Best Parameter	80-20	70-30	60-40
criterion	gini	gini	gini
n_estimators	300	400	300
max_depth	10	10	8
min_samples_split	10	10	10
max_features	sqrt	sqrt	sqrt

Table 3.7 presents the best parameters for the balanced data. For all train-test splits, the “criterion” parameter, “min_samples_split” parameter and “max_features” parameter was once again set to "gini", 10 and “sqrt”. The "n_estimators " parameter changed to 300 for the 80-20 split, while 400 was set for the 70-30 and 60-40 splits. The “max_depth” parameter was changed to 10 for the 70-30 split, while 12 was set for the 80-20 and 60-40 splits.

Table 3.7: Best RF Parameters (Balanced Dataset)

	Train-Test Split		
Best Parameter	80-20	70-30	60-40
criterion	gini	gini	gini
n_estimators	300	400	400
max_depth	12	10	12
min_samples_split	10	10	10
max_features	sqrt	sqrt	sqrt

To construct a RF model, scikit-learn library was used to perform RF algorithm as shown in Figure 3.24. RF classifier was created with the best parameter for further processes. Training data was used to train the model using ‘fit’ method. Test data was used to predict the model using ‘predict’ method.

```
from sklearn.ensemble import RandomForestClassifier

# Instantiate model
rfc1 = RandomForestClassifier(criterion='gini',max_depth=12, max_features='sqrt',
                              min_samples_split=10, n_estimators=300, random_state=0)

# Train the model on training data
rfc1.fit(X_important_train, y_train)

# Use the forest's predict method on the test data
rfc_predict = rfc1.predict(X_important_test)
```

Figure 3.24: Random Forest Algorithm

3.7.4 Light Gradient-Boosting Machine

LightGBM was chosen to apply for classification because it is a gradient boosting framework that offers high performance and efficiency. LightGBM uses a tree-based approach similar to decision trees, but with optimizations that make it faster and more scalable.

Table 3.8 shows the best LightGBM parameters for the imbalanced data. For all train-test splits, the “bagging_fraction” parameter, “boosting_type” parameter, “feature_fraction” parameter and “learning_rate” parameter was set to 0.5, “dart”, 0.5 and 0.1. The "max_depth" parameter changed to 5 for the 60-40 split, while 7 was set for the 80-20 and 70-30 splits. The “num_leaves” parameter was changed to 30 for the 60-40 split, while 20 was set for the 80-20 and 70-30 splits.

Table 3.8: Best LightGBM Parameters (Imbalanced Dataset)

	Train-Test Split		
Best Parameter	80-20	70-30	60-40
bagging_fraction	0.5	0.5	0.5
boosting_type	dart	dart	dart
feature_fraction	0.5	0.5	0.5
learning_rate	0.1	0.1	0.1
max_depth	7	7	5
num_leaves	20	20	30

Table 3.9 presents the best parameters for the balanced data. For all train-test splits, the “bagging_fraction” parameter, “feature_fraction” parameter and “max_depth” parameter was set to 0.5, 0.5 and 7. The "boosting_type" parameter changed to “gbdt” for the 70-30 split, while “dart” was used for the 80-20 and 60-40 splits. The " learning_rate" parameter changed to 0.05 for the 70-30 split, while 0.1 was set for the 80-20 and 60-40 splits. The “num_leaves” parameter was changed to 30 for the 80-20 split, while 20 was set for the 70-30 and 60-40 splits.

Table 3.9: Best LightGBM Parameters (Balanced Dataset)

	Train-Test Split		
Best Parameter	80-20	70-30	60-40
bagging_fraction	0.5	0.5	0.5
boosting_type	dart	gbdt	dart
feature_fraction	0.5	0.5	0.5
learning_rate	0.1	0.05	0.1
max_depth	7	7	7
num_leaves	30	20	20

To construct a LightGBM model, scikit-learn library was used to perform LightGBM algorithm as shown in Figure 3.25. LightGBM classifier was created with the best parameter for further processes. Training data was used to train the model using 'fit' method. Test data was used to predict the model using 'predict' method.

```
import lightgbm as lgb

# Instantiate model
lgb_model = lgb.LGBMClassifier(bagging_fraction=0.5, boosting_type='dart', feature_fraction=0.5,
                               learning_rate=0.1, max_depth=7, num_leaves=30, random_state=0)

# Train the model on training data
lgb_model.fit(X_important_train, y_train)

# Use the forest's predict method on the test data
lgb_predict = lgb_model.predict(X_important_test)
```

Figure 3.25: LightGBM Algorithm

3.8 Model Evaluation

Model evaluation was performed using different evaluation metrices to evaluate and observe the machine learning model's performance, as well as its strengths and weaknesses. Evaluation metrics were used to find the best model that works well and estimate the generalization accuracy of the model for future data. Popular evaluation metrices include accuracy, precision, recall and F1 score are commonly used for this purpose. A confusion matrix is also often employed to compare the true positive and negative outcomes with the false positive and negative outcomes of the model.

3.8.1 Confusion Matrix

Confusion Matrix was employed to analyze the performance of a multi-class classification model. It gave a clear picture of the accuracy of the model by comparing the actual and predicted values of the target variables. A confusion matrix was represented as an N x N matrix, where N was the number of classes in the target variable. The evaluation metrics such as accuracy, precision, recall and F1 Score, were computed using the values in the confusion matrix. A graphical representation of the confusion matrix is shown in Figure 3.26 for the binary classification case. It is a 2 x 2 matrix that contains four critical values, such as True Positive (TF), True Negative (TN), False Positive (FP) and False Negative (FN). Each of them is represented below:

- **True Positive (TF):** The actual and predicted values are both being positive.
- **True Negative (TN):** The actual and predicted values are both being negative.
- **False Positive (FP):** Actual value is negative, but the model predicted it as positive, which results in an incorrect prediction, also known as Type I error.
- **False Negative (FN):** Actual value is positive but the model predicted it as negative, which results in an incorrect prediction, also known as Type II error.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 3.26: Confusion Matrix from Binary Class

3.8.2 Accuracy

Accuracy is a common evaluation metric used to evaluate the performance of a model in a classification problem. It is the proportion of correct predictions made by the model over the total number of predictions made. An accuracy of 1 indicates that the model has made no errors and has correctly classified all instances, while an accuracy of 0 indicates that the model has made errors on all instances. The formula for accuracy was as follow:

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TN + TP}}{\boldsymbol{TP + TN + FP + FN}} \tag{3.1}$$

3.8.3 Precision

Precision is an evaluation metric used to correctly identify positive instances of a particular class in a classification problem. It is a measure of how many of the positive prediction made by a model are actually correct. A precision value of 1 indicates that the model has not made any false positive predictions, while a precision value of 0 indicates that all predictions made by the model are false positives. The formula for precision was as follows:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FP}} \tag{3.2}$$

3.8.4 Recall

Recall is an evaluation metric used to correctly identity all relevant instances of a particular class in a classification problem. It is a measure of how many of the actual positive instances are correctly predicted by the model. A recall value of 1 indicates that the model has not identified any positive instances, while a recall value of 0 indicates that the model has not identified any positive instance. The equation for recall was as follow:

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}} \tag{3.3}$$

3.8.5 F1 Score

F1 Score is an evaluation metric that combines precision and recall. It was performed in classification problems, particularly when the classes are imbalanced. F1 score is calculated as the harmonic mean of precision and recall with a range of 0 to 1. A perfect score is 1 and worst score is 0. A value of 0.5 is the minimum threshold for a model to be considered better than random guessing. A higher F1 score indicates that the model has a better balance of precision and recall. The equation of computing F1 Score is shown below:

$$\boldsymbol{F1\ Score = 2\left(\frac{Precision \times Recall}{Precision + Recall}\right)} \tag{3.4}$$

3.9 Application Deployment with Machine Learning API

This method was performed to deploy machine learning model as web service that can be accessed over the internet. The machine learning model used to deploy was LightGBM. There were some Python libraries are required to deploy the LightGBM model, which were FLASK (for creating REST API) and Joblib (for exporting the model). There were several deployment steps required to deploy the RF model as a REST API. The LightGBM model must be serialized and saved to a file using Joblib library as shown in Figure 3.27.

```
#Serialize the model and save
import joblib

joblib.dump(lgb_model, 'lgbModel.pkl')
print("LightBGM Model Saved")
```

Figure 3.27: Serialize and Save the LightGBM Model

The LightGBM model was used to make predictions based on the input data received through the Flask API. Flask was used to deploy the model as a web service that listens to incoming requests and provides predictions as responses. The API endpoints was used for rendering the home page and predicting outcomes using the LightGBM model as shown in Figure 3.28. The home endpoint simply renders an HTML page using the ‘render_template’ function of Flask. The prediction endpoint was more complex and define a function named ‘predict’ that is triggered when the endpoint receives a POST request.

```python
application = Flask(__name__, static_url_path='/static', static_folder='static')

@application.route('/')
def home():
    return render_template('index.html')

@application.route('/prediction', methods=['POST'])
```

Figure 3.28: API Endpoints

The prediction function was defined to handle the POST requests made to the '/prediction' route. It retrieved the input data from the form data sent with the request. The input data was parsed from JSON format to a Pandas DataFrame. The LightGBM model was then used to make predictions on the input DataFrame. The result of the predictions was converted to a string and stored in the predictions key of the result dictionary. Based on the predicted value, a message indicating the malware type is added to the result dictionary. The result dictionary was returned as a JSON response. If any errors occur during the prediction process, the function returns a traceback of the error as a JSON response. Figure 3.29 shows the function of prediction.

```python
def predict():
    if request.method == 'POST':
        try:
            data = request.form['data']
            input_dict = json.loads(data)
            input_df = pd.DataFrame.from_dict(input_dict)
            predictions = lgb_model.predict(input_df)

            result = {'predictions': str(predictions)}

            if predictions == 0:
                result['message'] = 'Malware Type: APT1'
            elif predictions == 1:
                result['message'] = 'Malware Type: Crypto'
            elif predictions == 2:
                result['message'] = 'Malware Type: Locker'
            elif predictions == 3:
                result['message'] = 'Malware Type: Zeus'
            elif predictions == 4:
                result['message'] = 'Malware Type: Shadowbrokers'
            else:
                result['message'] = 'Unknown malware type'

            return jsonify(result)

        except Exception as e:
            print(str(e))
            return jsonify({'error': str(e)})
    else:
        return jsonify({'error': 'Invalid request method'})
```

Figure 3.29: Function of Prediction

(if __name__ == '__main__':) was used to check if the application was being executed as the main entry point as shown in Figure 3.30. It attempted to read the port number from the command-line argument. If no valid port was provided, the default port 3000 was used. The LightGBM model was loaded using joblib. If the model failed to load, an error message was printed, and the application exited with a status code of 1. Finally, the Flask application is started by calling application.run() with the specified port and the debug parameter set to True.

```
if __name__ == '__main__':
    # Attempt to read the port from the command-line argument
    try:
        port = int(sys.argv[1])
    except (IndexError, ValueError):
        # If no valid port is provided, use default port 3000
        port = 3000

    # Attempt to load the model
    try:
        lgb_model = joblib.load("lgbModel.pkl")
        print('Model loaded')
    except Exception:
        # If the model fails to load, print an error message and exit the program
        print('Failed to load model')
        sys.exit(1)

    # Start the Flask application on the specified port
    application.run(port=port, debug=True)
```

Figure 3.30: Running Application

CHAPTER 4 RESULT AND DISCUSSION

4.1 Introduction

The following section focus on three main aspects: classification results, comparison results, and malware detection using REST API. The experiments were divided into four categories: Without SMOTE & Boruta, Using Boruta Without SMOTE, Using SMOTE Without Boruta, and Using SMOTE & Boruta. Three machine learning models, namely DT, RF, and LightGBM, were compared in the different experiments. The performance results were evaluated across different train-test splits, and the confusion matrix was utilized to assess the True Positive (TP) rates.

4.2 Classification Result

4.2.1 Experiment 1: Without SMOTE & Boruta

This experiment used a dataset with 5 malware families such as APT1, Crypto, Locker, Zeus and Shadowbrokers. SMOTE technique and Boruta were not applied. All the available features were used in the model. The dataset was divided into three different train-test splits: 80-20, 70-30, and 60-40. The machine learning algorithms such as DT, RF and LightGBM, were applied to train the model. The performance results of each model for the different train-test splits are summarized in Tables 4.1, 4.2, and 4.3.

Table 4.1: Performance of DT (Without SMOTE & Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.6015	0.6212	0.6015	0.6007
70-30	0.6392	0.6402	0.6392	0.6388
60-40	0.6680	0.6718	0.6680	0.6663

Table 4.2: Performance of RF (Without SMOTE & Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.7854	0.8079	0.7854	0.7826
70-30	0.7713	0.7870	0.7713	0.7693
60-40	0.7721	0.7892	0.7721	0.7703

Table 4.3: Performance of LightGBM (Without SMOTE & Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.8351	0.8618	0.8351	0.8319
70-30	0.8271	0.8496	0.8271	0.8234
60-40	0.8189	0.8409	0.8189	0.8158

From Table 4.1, the DT model achieved an accuracy of 0.6015 for the 80-20 split. The accuracy improved slightly for the 70-30 split, reaching 0.6392, and further increased to 0.6680 for the 60-40 split. From Tables 4.2, the RF model showed significant improvement compared to DT. For the 80-20 split, RF achieved an accuracy of 0.7854. For the 70-30 and 60-40 splits, the RF model maintained high accuracies of 0.7713 and 0.7721. From Table 4.3, the LightGBM model exhibited exceptional performance, surpassing both DT and RF consistently. As the train-test split ratio increased, LightGBM demonstrated higher performance. For the 80-20 split, LightGBM achieved an accuracy of 0.8351.

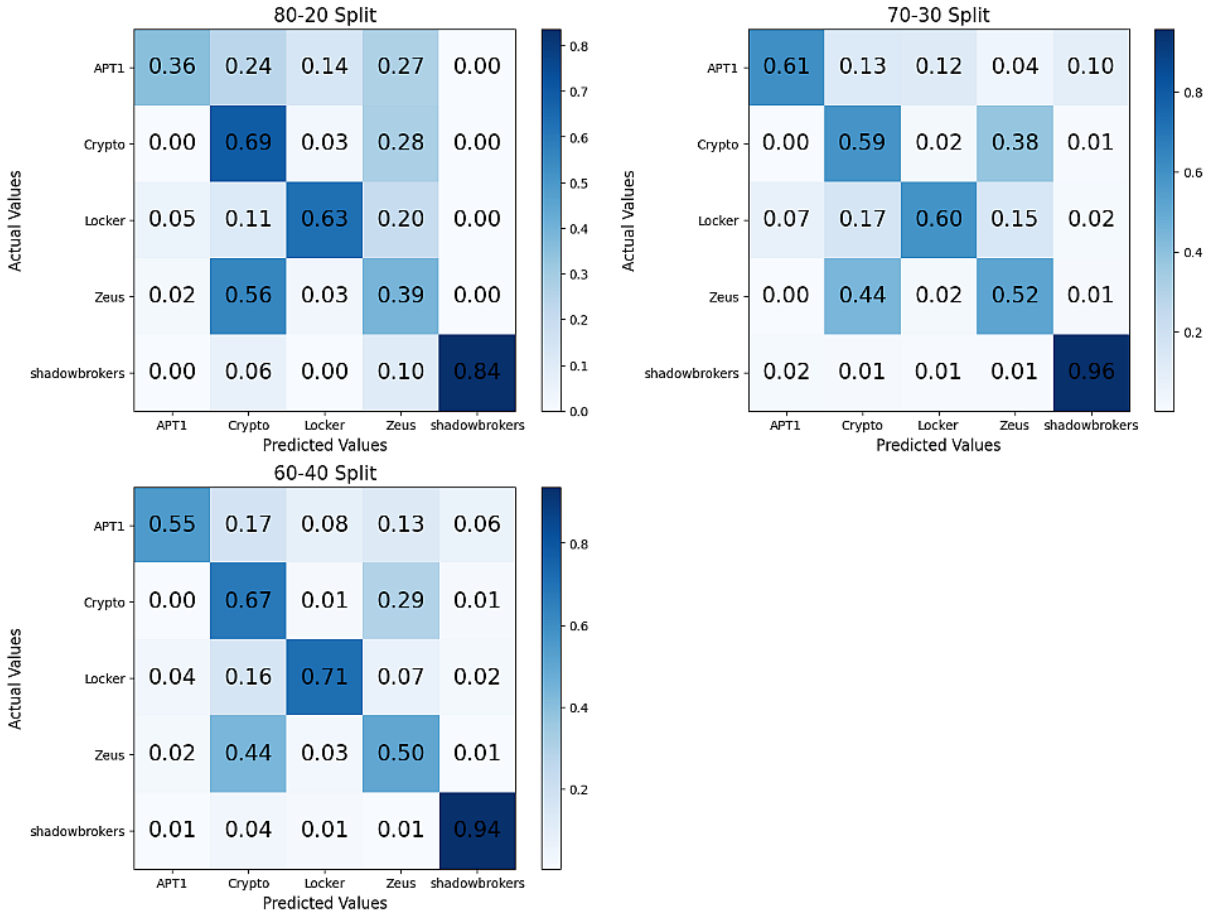

Figure 4.1: Confusion Matrix of DT (Without SMOTE & Boruta)

From Figure 4.1, the confusion matrix revealed that the DT model shares the same high TP rates for Shadowbrokers only, while exhibiting the low TP rates for APT1, Locker, Crypto and Zeus. Among the different train-test split ratios, the 80-20 split achieved the highest TP rate of 0.84 exclusively for Shadowbrokers. The 70-30 split achieved a TP rate of 0.96 for Shadowbrokers, while the 60-30 split achieved a TP rate of 0.94 for the same category.

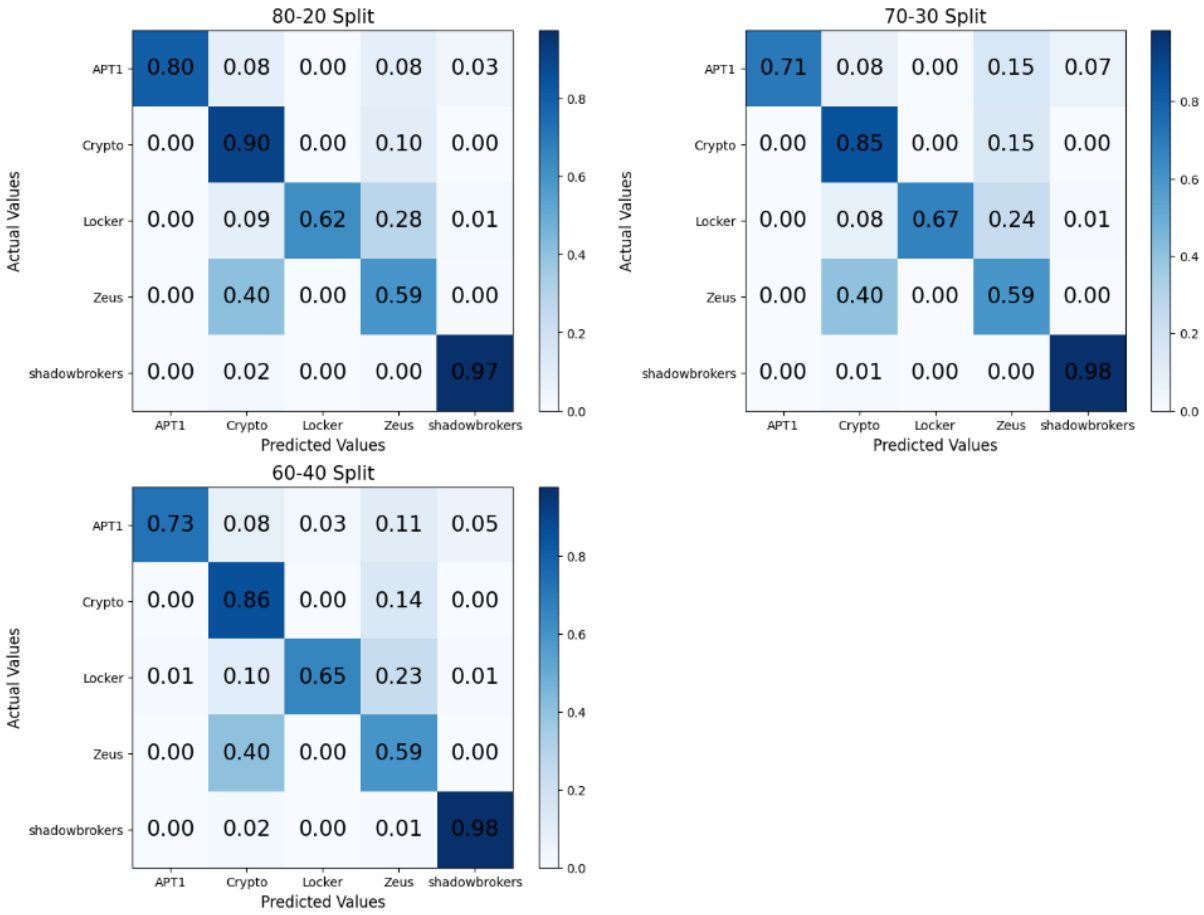

Figure 4.2: Confusion Matrix of RF (Without SMOTE & Boruta)

From Figure 4.2, the confusion matrix revealed that the RF model achieved the highest TP rates for APT1, Crypto, and ShadowBrokers in the 80-20 split, with values of 0.80, 0.90, and 0.97 respectively. For the 70-30 split, it achieved the highest TP rates for Crypto and ShadowBrokers, with values of 0.85 and 0.98. Lastly, in the 60-40 split, the RF model achieved the highest TP rates for Crypto and ShadowBrokers, with values of 0.86 and 0.98 respectively. Comparing the RF model to the DT model, the RF model generally achieves higher TP rates.

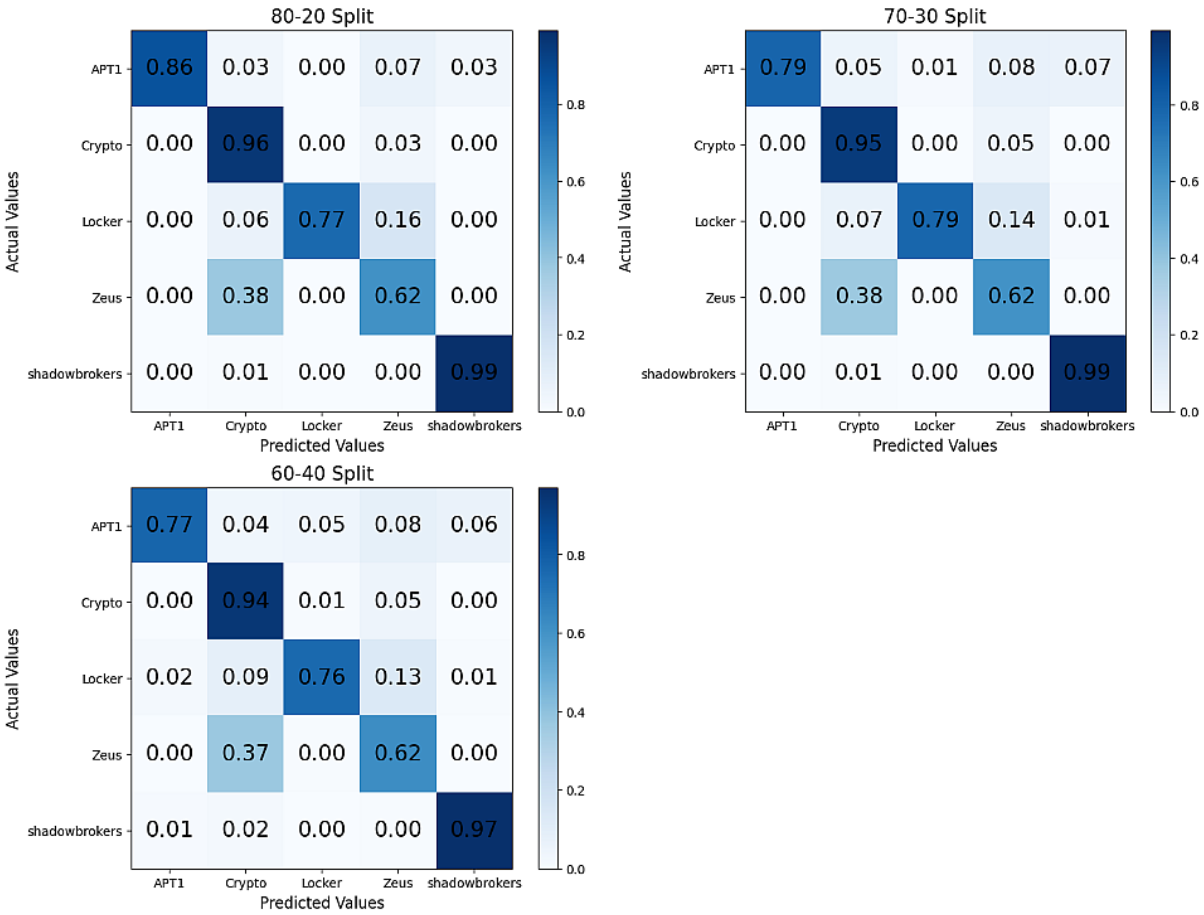

Figure 4.3: Confusion Matrix of LightGBM (Without SMOTE & Boruta)

From Figure 4.3, the confusion matrix revealed that the LightGBM model achieved the highest TP rates for APT1, Crypto, and ShadowBrokers in the 80-20 split, with values of 0.86, 0.96, and 0.99 respectively. For the 70-30 split, it achieved the highest TP rates for Crypto and ShadowBrokers, with values of 0.95 and 0.99. Lastly, in the 60-40 split, the LightGBM model achieved the highest TP rates for Crypto and ShadowBrokers, with values of 0.94 and 0.97 respectively. Overall, the LightGBM model consistently outperformed the DT and RF models, showcasing the highest TP rates for each malware family.

4.2.2 Experiment 2: Using Boruta Without SMOTE

This experiment used a dataset with 5 malware families such as APT1, Crypto, Locker, Zeus and Shadowbrokers. It applied Boruta for feature selection, without applying the SMOTE technique for data augmentation and oversampling. The important features were identified and used in the model. The dataset was divided into three different train-test splits: 80-20, 70-30, and 60-40. The machine learning algorithms such as DT, RF and LightGBM, were applied to train the model. The performance results of each model for the different train-test splits are summarized in Tables 4.4, 4.5, and 4.6.

Table 4.4: Performance of DT (Using Boruta Without SMOTE)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.6744	0.6715	0.6744	0.6714
70-30	0.6840	0.6894	0.6840	0.6785
60-40	0.6797	0.6808	0.6797	0.6774

Table 4.5: Performance of RF (Using Boruta Without SMOTE)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.7978	0.8230	0.7978	0.7949
70-30	0.7934	0.8102	0.7934	0.7906
60-40	0.7944	0.8176	0.7944	0.7912

Table 4.6: Performance of LightGBM (Using Boruta Without SMOTE)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.8376	0.8650	0.8376	0.8343
70-30	0.8315	0.8544	0.8315	0.8282
60-40	0.8247	0.8489	0.8247	0.8217

From Table 4.4, the DT model achieved an accuracy of 0.6744 for the 80-20 split. The accuracy slightly increased to 0.6840 for the 70-30 split and reached 0.6797 for the 60-40 split. From Tables 4.5, the RF model showed significant improvement compared to DT. For the 80-20 split, RF achieved an accuracy of 0.7978. The accuracy remained consistently high across various train-test splits, with values of 0.7934 and 0.7944 for the 70-30 and 60-40 splits. From Table 4.6, the LightGBM model exhibited exceptional performance, surpassing both DT and RF consistently. As the train-test split ratio increased, LightGBM demonstrated higher performance. For the 80-20 split, LightGBM achieved an accuracy of 0.8376.

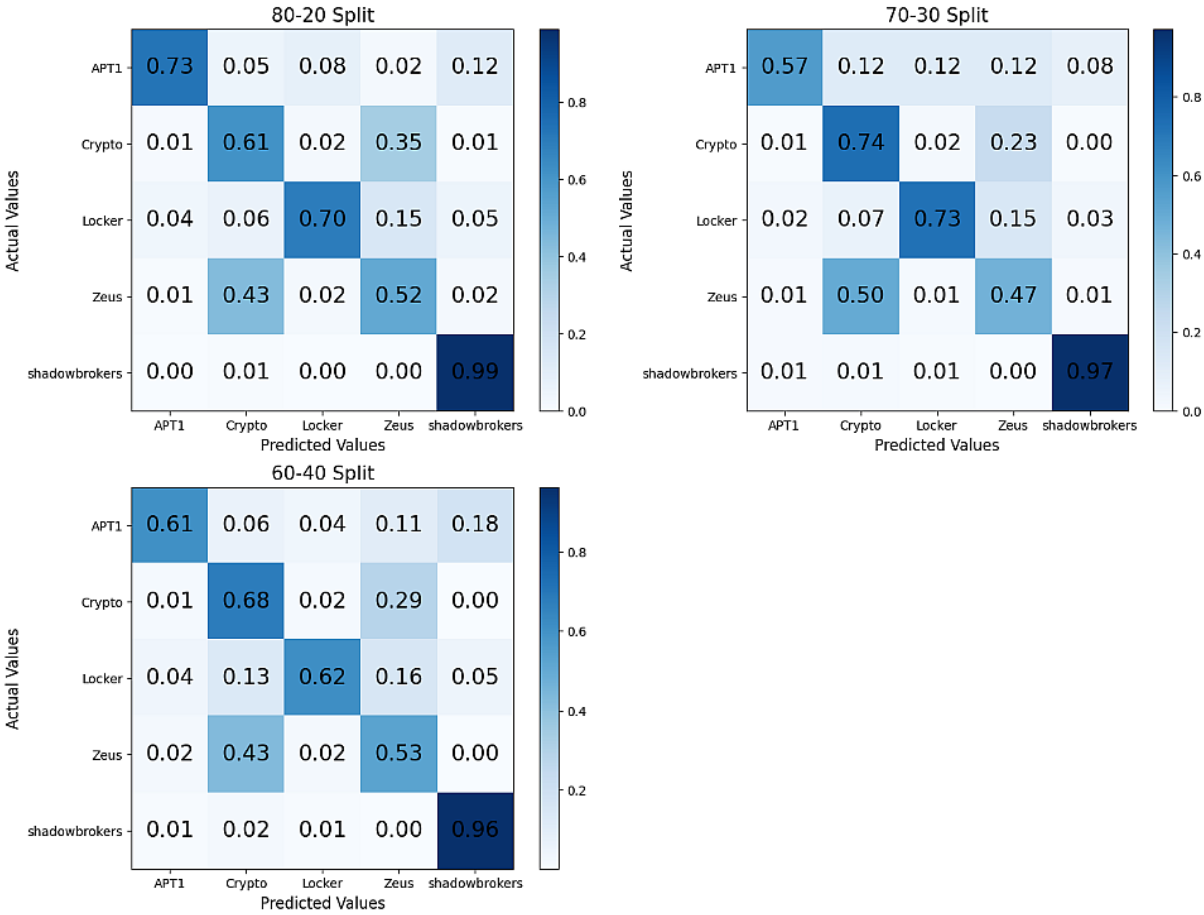

Figure 4.4: Confusion Matrix of DT (Using Boruta Without SMOTE)

From Figure 4.4, the confusion matrix revealed that the DT model shares the same high TP rates for Shadowbrokers only, while exhibiting the low TP rates for APT1, Locker, Crypto and Zeus. Among the various train-test split ratios, the 80-20 split achieved the highest TP rate of 0.99 for Shadowbrokers only. The 70-30 split achieved a TP rate of 0.97, while the 60-30 split achieves a TP rate of 0.96, both specifically for Shadowbrokers.

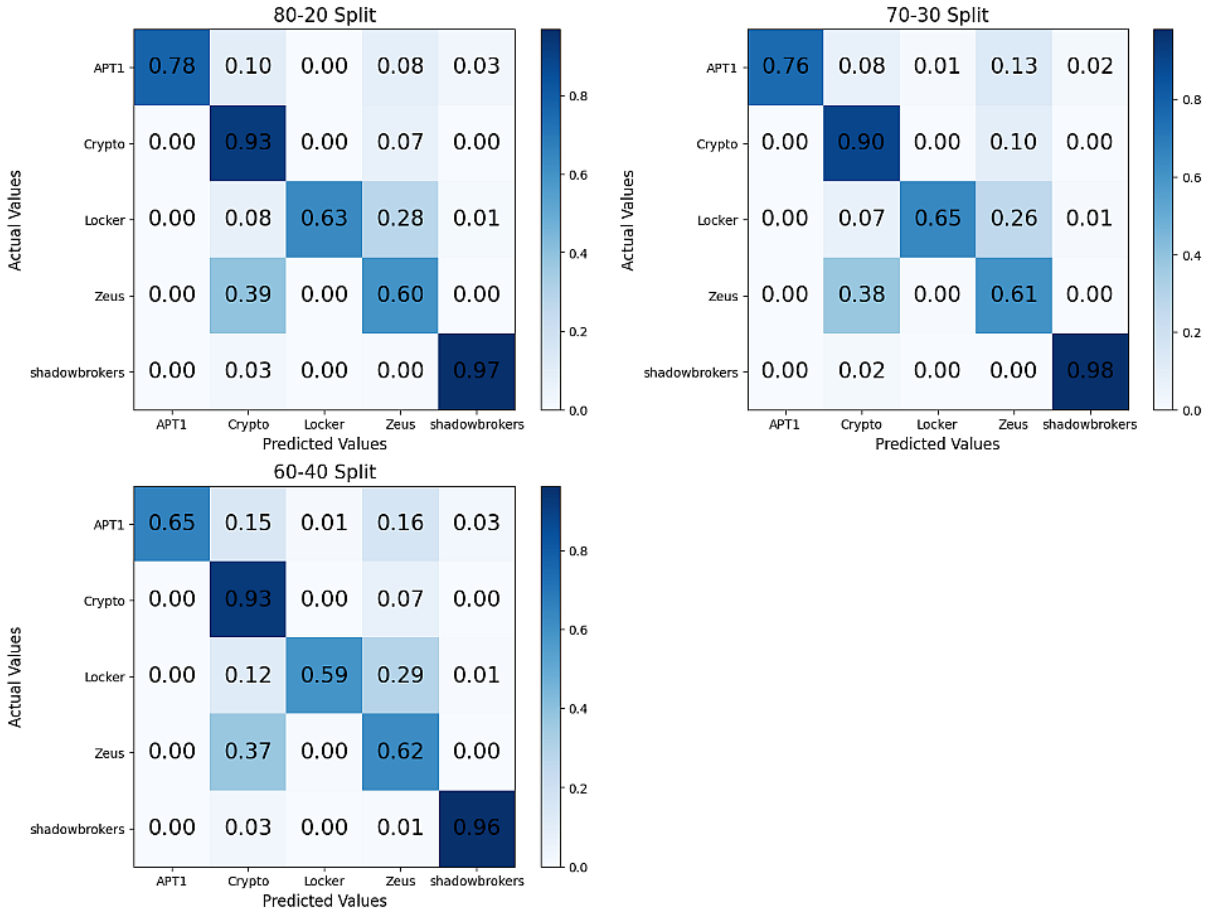

Figure 4.5: Confusion Matrix of RF (Using Boruta Without SMOTE)

From Figure 4.5, the confusion matrix revealed that the RF model shares the same high TP rates for Crypto and Shadowbrokers, while exhibiting the low TP rates for APT1, Locker and Zeus. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for Crypto and ShadowBrokers, with values of 0.93 and 0.97. The 70-30 split achieved the highest TP rates for Crypto and Shadowbrokers, with the value of 0.90 and 0.98. The 60-40 split achieved the highest TP rates for Crypto and Shadowbrokers, with value of 0.93 and 0.96. RF model has higher TP rates compared to the DT model.

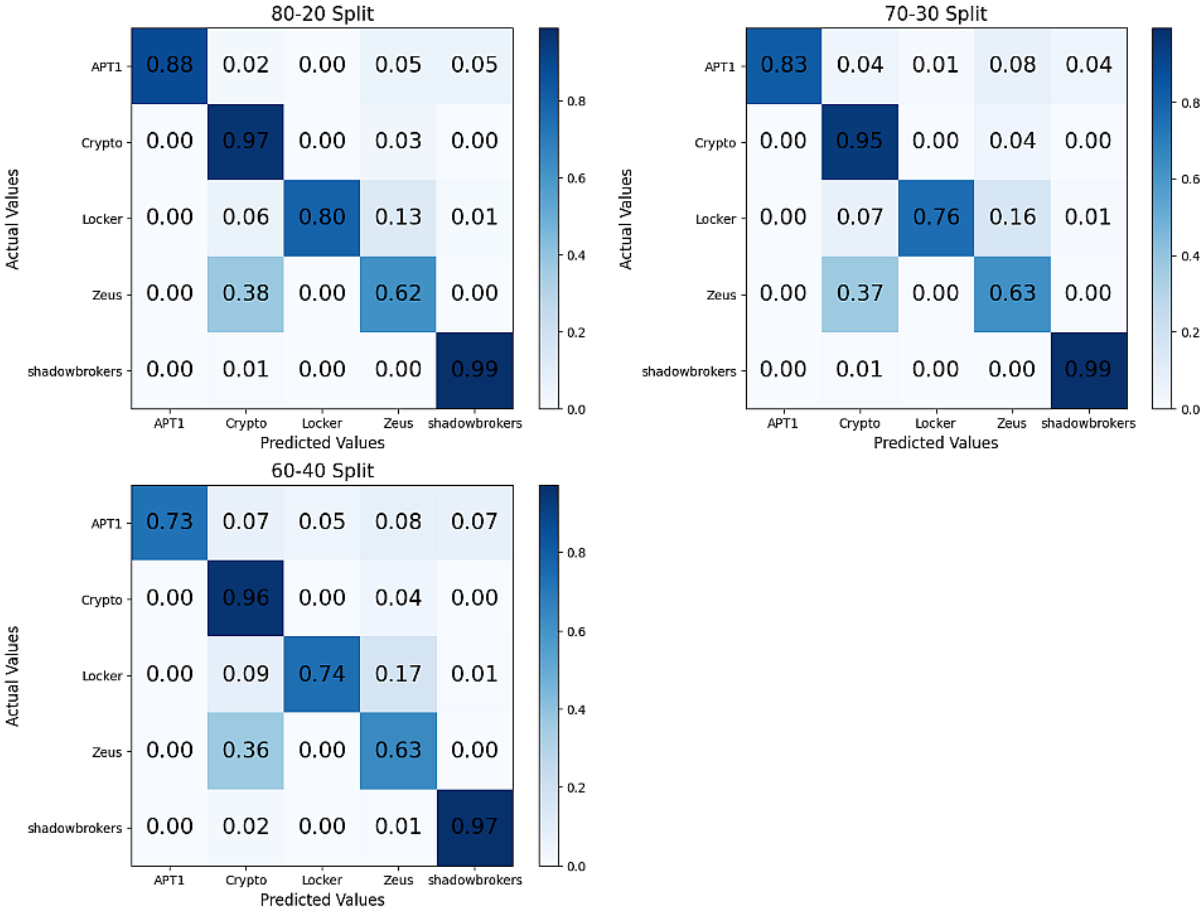

Figure 4.6: Confusion Matrix of LightGBM (Using Boruta Without SMOTE)

From Figure 4.6, the confusion matrix revealed that the LightGBM model achieved the highest TP rates for APT1, Crypto, Locker, and ShadowBrokers in the 80-20 split, with values of 0.88, 0.97, 0.80, and 0.99 respectively. For the 70-30 split, it achieved the highest TP rates for APT1, Crypto, and ShadowBrokers, with values of 0.83, 0.95, and 0.99. Lastly, in the 60-40 split, the LightGBM model achieved the highest TP rates for Crypto and ShadowBrokers, with values of 0.96 and 0.97 respectively. Overall, the LightGBM model consistently outperformed the DT and RF models, showcasing the highest TP rates for each malware family.

4.2.3 Experiment 3: Using SMOTE Without Boruta

This experiment used a dataset with 5 malware families such as APT1, Crypto, Locker, Zeus and Shadowbrokers. It applied the SMOTE technique for data augmentation and oversampling, without applying Boruta for feature selection. All the available features were used in the model. The dataset was divided into three different train-test splits: 80-20, 70-30, and 60-40. The machine learning algorithms such as DT, RF and LightGBM, were applied to train the model. The performance results of each algorithm for the different train-test splits are summarized in Tables 4.7, 4.8, and 4.9.

Table 4.7: Performance of DT (Using SMOTE Without Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.7594	0.7560	0.7533	0.7503
70-30	0.7578	0.7656	0.7562	0.7593
60-40	0.7401	0.7413	0.7397	0.7362

Table 4.8: Performance of RF (Using SMOTE Without Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.8426	0.8421	0.8378	0.8352
70-30	0.8360	0.8367	0.8342	0.8326
60-40	0.8277	0.8298	0.8272	0.8263

Table 4.9: Performance of LightGBM (Using SMOTE Without Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.9059	0.9158	0.9041	0.9013
70-30	0.9020	0.9121	0.9006	0.8989
60-40	0.8955	0.9065	0.8959	0.8936

From Table 4.7, the DT model achieved an accuracy of 0.7594 for the 80-20 split, and its performance remained relatively stable as the train-test split ratio increased. From Tables 4.8, the RF model showed significant improvement compared to DT, consistently outperforming it across all train-test splits. For the 80-20 split, RF achieved an accuracy of 0.8426. From Table 4.9, the LightGBM model exhibited exceptional performance, surpassing both DT and RF consistently. As the train-test split ratio increased, LightGBM demonstrated higher performance. For the 80-20 split, LightGBM achieved an accuracy of 0.9059.

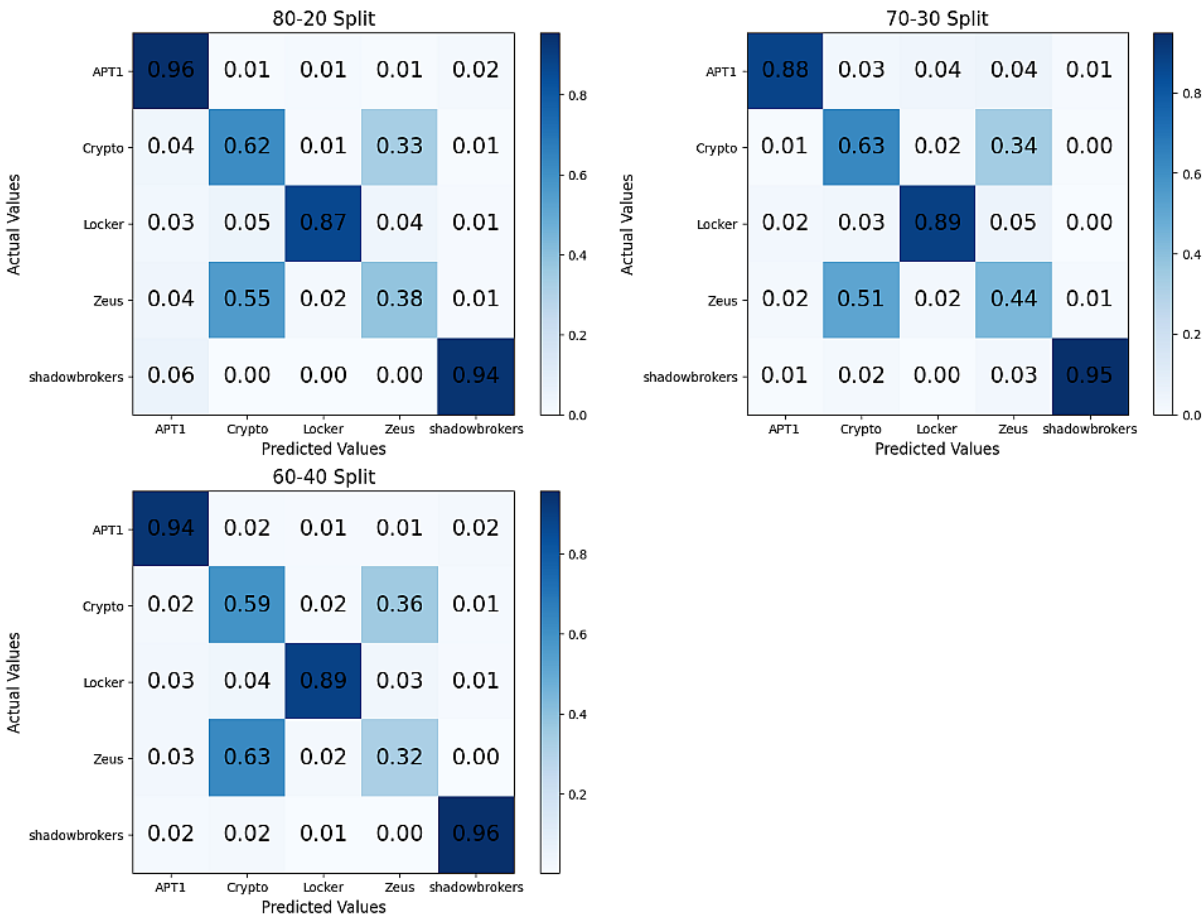

Figure 4.7: Confusion Matrix of DT (Using SMOTE Without Boruta)

From Figure 4.7, the confusion matrix revealed that the DT model shares the same high TP rates for APT1, Locker and Shadowbrokers, while exhibiting the low TP rates for Crypto and Zeus. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with the values of 0.96, 0.87 and 0.94. The 70-30 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with the values of 0.88,

0.89 and 0.95. The 60-40 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with the values of 0.94, 0.89 and 0.96.

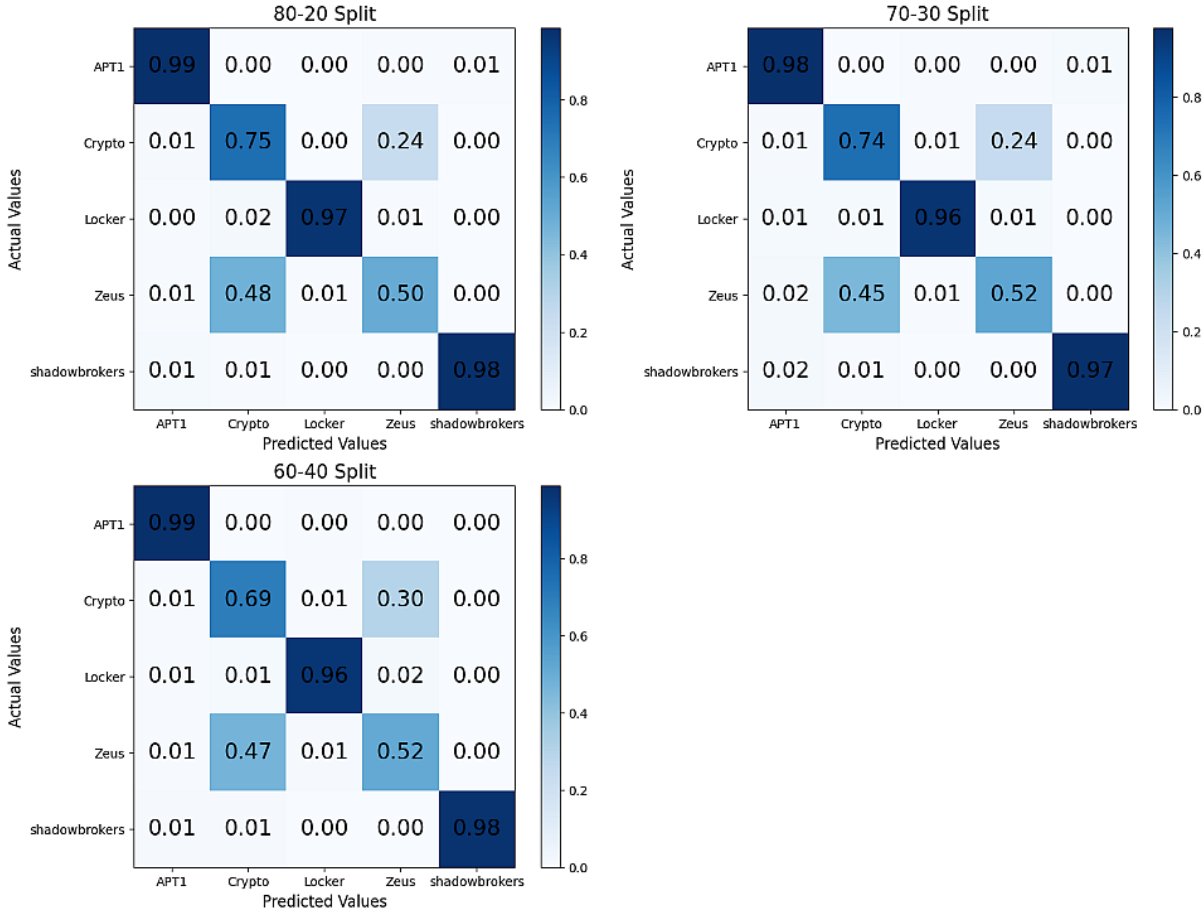

Figure 4.8: Confusion Matrix of RF (Using SMOTE Without Boruta)

From Figure 4.8, the confusion matrix revealed that the RF model shares the same high TP rates for APT1, Locker and Shadowbrokers, while exhibiting the low TP rates for Crypto and Zeus. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with values of 0.99, 0.97 and 0.98. The 70-30 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with values of 0.98, 0.96 and 0.97. The 60-40 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with values of 0.99, 0.96 and 0.98. RF model has higher TP rates compared to the DT model.

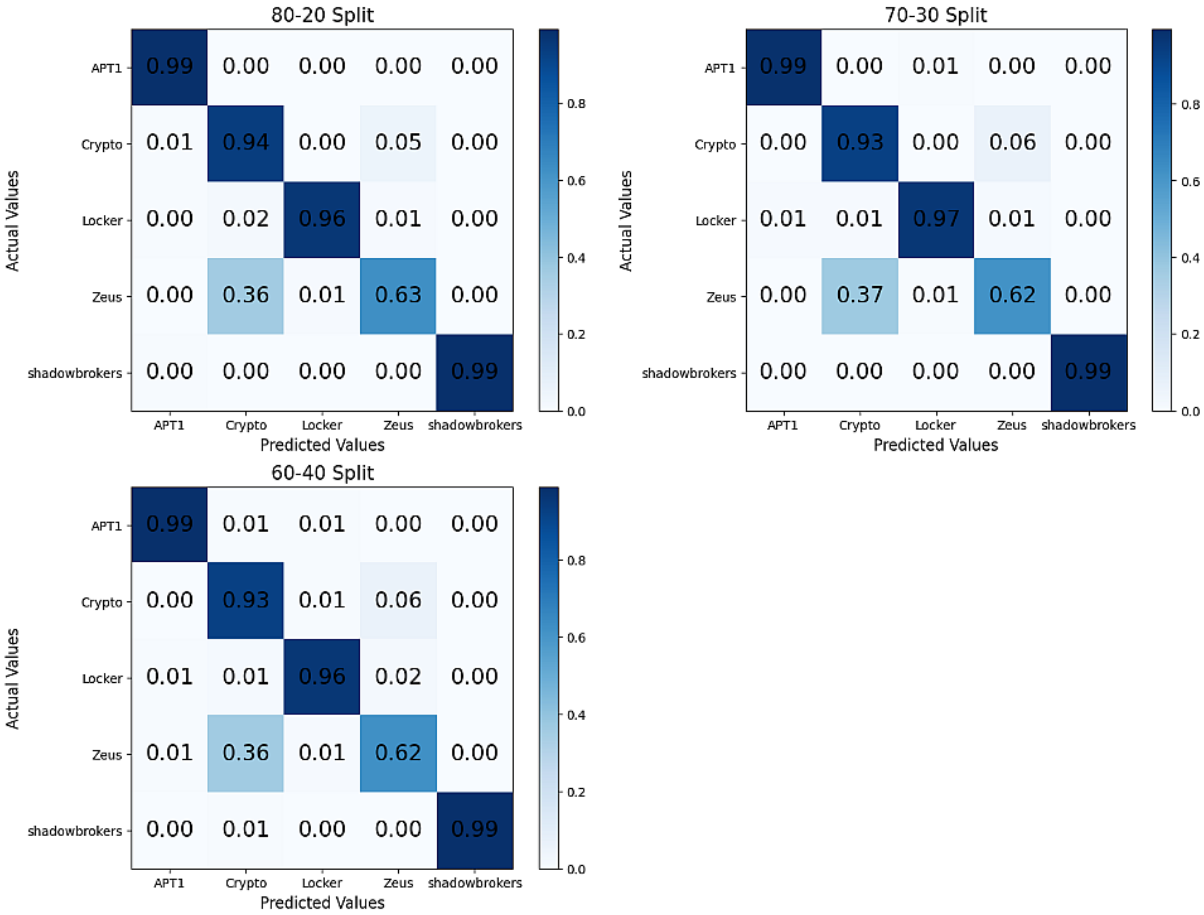

Figure 4.9: Confusion Matrix of LightGBM (Using SMOTE Without Boruta)

From Figure 4.9, the confusion matrix revealed that the LightGBM model shares the same high TP rates for APT1, Crypto, Locker and Shadowbrokers, while exhibiting the low TP rates for Zeus only. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for APT1, Crypto, Locker and Shadowbrokers, with values of 0.99, 0.94, 0.96, and 0.99. The 70-30 split achieved the highest TP rates for APT1, Crypto, Locker and Shadowbrokers, with values of 0.99, 0.93, 0.97, and 0.99. The 60-40 split achieved the highest TP rates for APT1, Crypto, Locker and Shadowbrokers, with values of 0.99, 0.93, 0.96, and 0.99. The LightGBM model consistently outperforms both DT and RF models, showcasing the highest TP rates for each malware family.

4.2.4 Experiment 4: Using SMOTE & Boruta

This experiment used a dataset with 5 malware families such as APT1, Crypto, Locker, Zeus and Shadowbrokers. It applied the SMOTE technique for data augmentation and oversampling and Boruta for feature selection. The dataset was divided into three different train-test splits: 80-20, 70-30, and 60-40. The machine learning algorithms such as DT, RF and LightGBM, were applied to train the model. The performance results of each algorithm for the different train-test splits are summarized in Tables 4.10, 4.11, and 4.12.

Table 4.10: Performance of DT (Using SMOTE & Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.7916	0.7946	0.7868	0.7853
70-30	0.7601	0.7606	0.7581	0.7569
60-40	0.7720	0.7777	0.7722	0.7675

Table 4.11: Performance of RF (Using SMOTE & Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.8540	0.8540	0.8496	0.8478
70-30	0.8389	0.8415	0.8370	0.8359
60-40	0.8470	0.8504	0.8468	0.8453

Table 4.12: Performance of LightGBM (Using SMOTE & Boruta)

	Evaluation Metrics			
Train-Test Split	Accuracy	Precision	Recall	F1 Score
80-20	0.9050	0.9128	0.9029	0.9002
70-30	0.9033	0.9148	0.9020	0.9003
60-40	0.8943	0.9055	0.8947	0.8920

From Table 4.10, the DT model obtained an accuracy of 0.7916 for the 80-20 split, while the accuracy decreased slightly to 0.7601 for the 70-30 split. From Table 4.11, the RF model showed significant improvement compared to DT. With the 80-20 split, RF achieved an accuracy of 0.8540, but this value decreased to 0.8389 for the 70-30 split. From Table 4.12, the LightGBM model consistently outperformed DT and RF, especially as the train-test split ratio increased. For the 80-20 split, LightGBM achieved an accuracy of 0.9059, showcasing its exceptional performance.

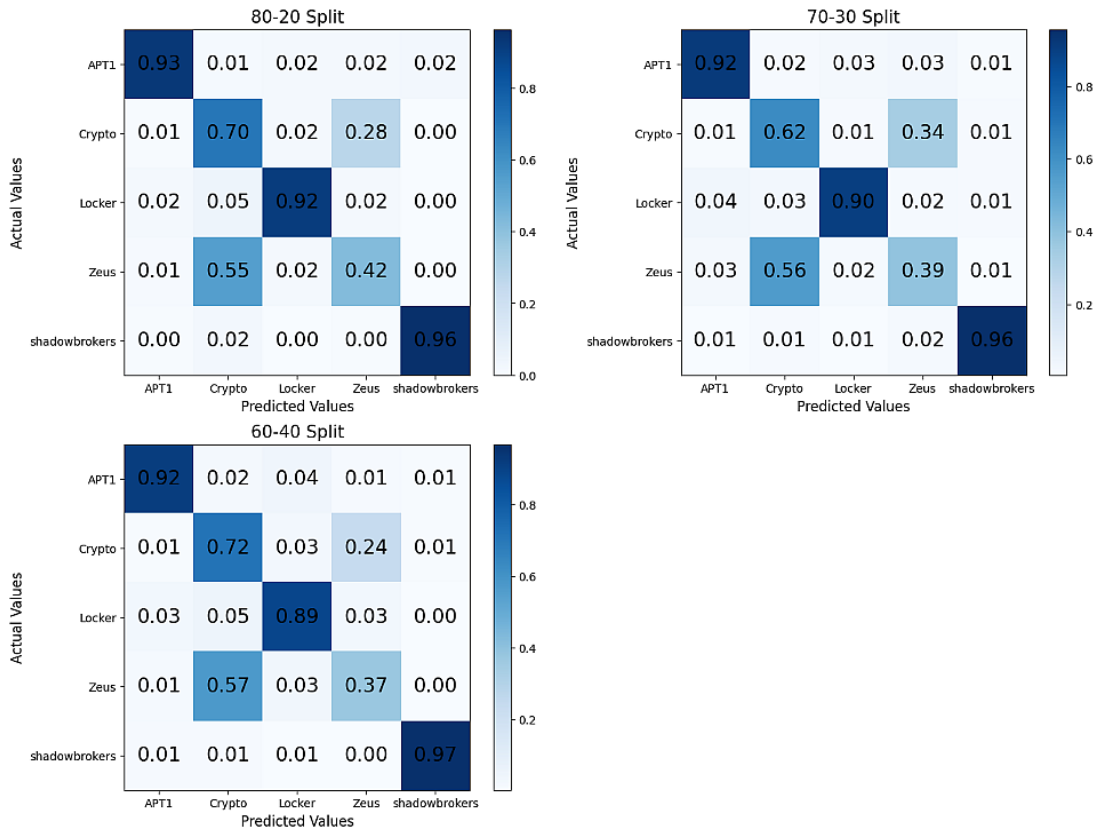

Figure 4.10: Confusion Matrix of DT (Using SMOTE & Boruta)

From Figure 4.10, the confusion matrix revealed that the DT model shares the same high TP rates for APT1, Locker and Shadowbrokers, while exhibiting the low TP rates for Crypto and Zeus. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with the values of 0.93, 0.92 and 0.96. The 70-30 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with the values of 0.92,

0.90 and 0.96. The 60-40 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with the values of 0.92, 0.89 and 0.97.

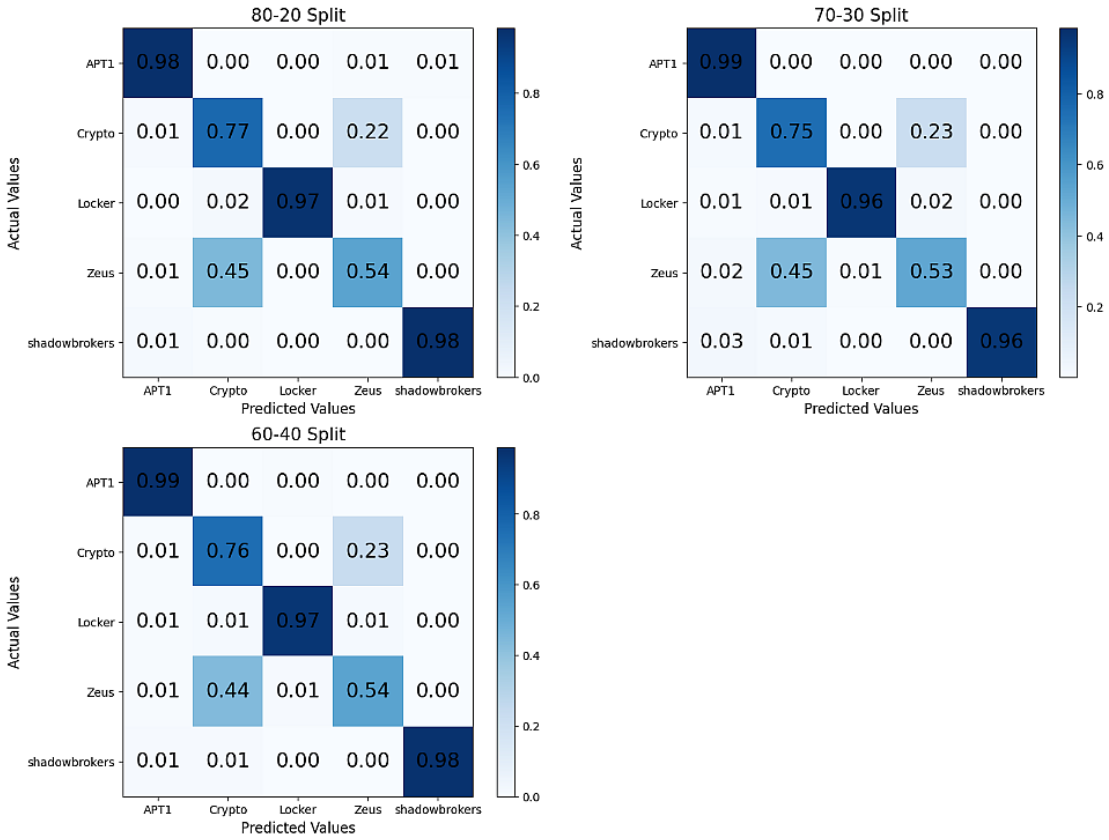

Figure 4.11: Confusion Matrix of RF (Using SMOTE & Boruta)

From Figure 4.11, the confusion matrix revealed that the RF model shares the same high TP rates for APT1, Locker and Shadowbrokers, while exhibiting the low TP rates for Crypto and Zeus. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with values of 0.98, 0.97 and 0.98. The 70-30 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with values of 0.99, 0.96 and 0.96. The 60-40 split achieved the highest TP rates for APT1, Locker and Shadowbrokers, with values of 0.99, 0.97 and 0.98. RF model has higher TP rates compared to the DT model.

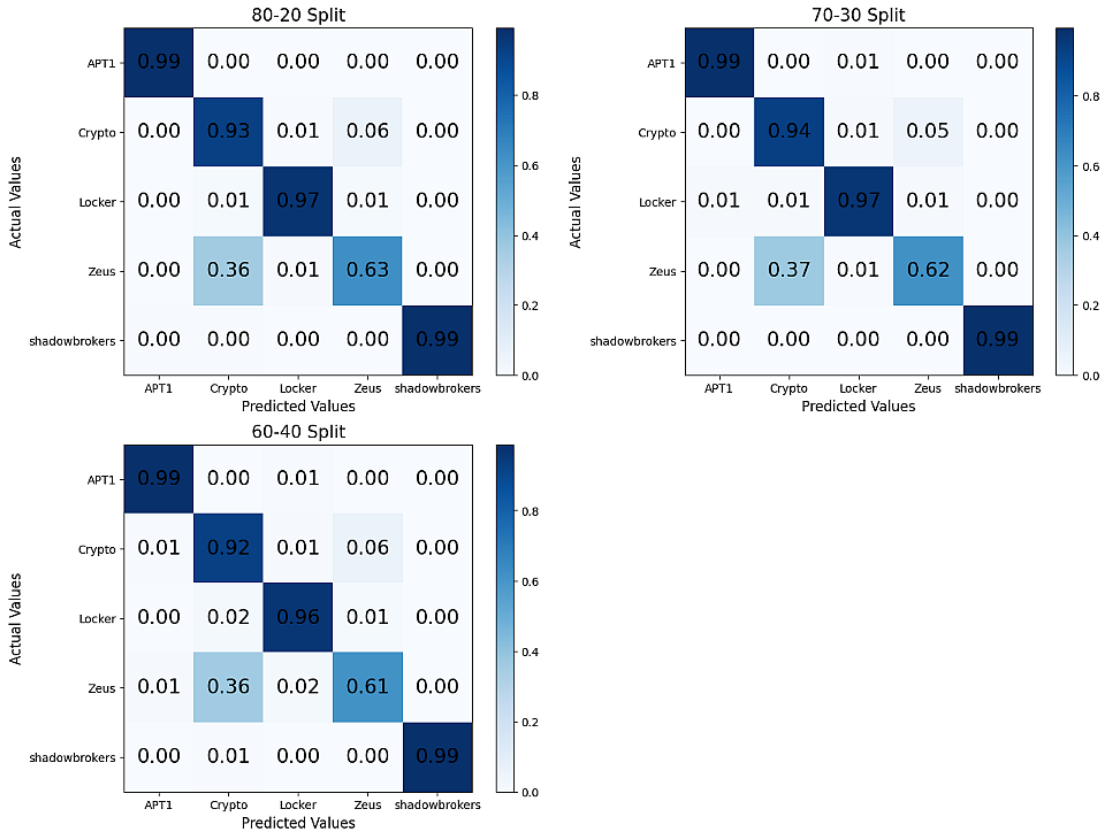

Figure 4.12: Confusion Matrix of LightGBM (Using SMOTE & Boruta)

From Figure 4.12, the confusion matrix revealed that the LightGBM model shares the same high TP rates for APT1, Crypto, Locker and Shadowbrokers, while exhibiting the low TP rates for Zeus only. Among the different train-test split ratios, the 80-20 split achieved the highest TP rates for APT1, Crypto, Locker and Shadowbrokers, with values of 0.99, 0.93, 0.97, and 0.99. The 70-30 split achieved the highest TP rates for APT1, Crypto, Locker and Shadowbrokers, with values of 0.99, 0.94, 0.97, and 0.99. The 60-40 split achieved the highest TP rates for APT1, Crypto, Locker and Shadowbrokers, with values of 0.99, 0.92, 0.96, and 0.99. The LightGBM model consistently outperforms both DT and RF models, showcasing the highest TP rates for each malware family.

4.3 Comparison Result

The classification results included four experiments that compared the performance of machine learning models, specifically DT, RF, and LightGBM, across various train-test split ratios. The confusion matrices of these models were compared in the four experiments, utilizing the best train-test split ratio of 80-20.

4.3.1 Models with Different Train-Test Split Ratio

As shown in Table 4.13, four experiments were conducted to evaluate the impact of feature selection (Boruta) and class imbalance handling (SMOTE) on the models' performance. In the first experiment, no feature selection or oversampling was applied, resulting in accuracy values of 0.6015 for DT, 0.7854 for RF, and 0.8351 for LightGBM. The second experiment used Boruta without SMOTE, showing increased accuracy values of 0.6744 for DT, 0.7978 for RF, and 0.8376 for LightGBM. This suggests that feature selection alone can enhance model accuracy. In the third experiment, SMOTE was used without Boruta, leading to further accuracy improvements of 0.7594 for DT, 0.8426 for RF, and 0.9059 for LightGBM, highlighting the effectiveness of SMOTE in addressing class imbalance. The fourth experiment utilized both SMOTE and Boruta, resulting in the highest accuracy values: 0.7916 for DT, 0.8540 for RF, and 0.9050 for LightGBM.

Table 4.13: Comparison Result of the Model (80-20 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6015	0.7854	0.8351
Using Boruta without SMOTE	0.6744	0.7978	0.8376
Using SMOTE without Boruta	0.7594	0.8426	0.9059
Using SMOTE & Boruta	0.7916	0.8540	0.9050

As shown in Table 4.14, the accuracy results of DT, RF, and LightGBM models were compared in four experiments using a 70-30 split of the data. In the first experiment, no feature selection or oversampling was applied, resulting in accuracies of 0.6392 for DT, 0.7713 for RF, and 0.8271 for LightGBM. The second experiment used Boruta without SMOTE, showing increased accuracies of 0.6840 for DT, 0.7934 for RF, and 0.8315 for LightGBM, indicating the effectiveness of feature selection alone. The third experiment used SMOTE without Boruta, achieving further accuracy improvements of 0.7578 for DT, 0.8360 for RF, and 0.9020 for LightGBM, highlighting the effectiveness of SMOTE in addressing class imbalance. The fourth experiment utilized both SMOTE and Boruta, resulting in slightly increased accuracies compared to the third experiment, with values of 0.7601 for DT, 0.8389 for RF, and 0.9033 for LightGBM.

Table 4.14: Comparison Result of the Model (70-30 Split)

	Accuracy of Models		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6392	0.7713	0.8271
Using Boruta without SMOTE	0.6840	0.7934	0.8315
Using SMOTE without Boruta	0.7578	0.8360	0.9020
Using SMOTE & Boruta	0.7601	0.8389	0.9033

As shown in Table 4.15, the accuracy results of DT, RF, and LightGBM models were compared in four experiments using a 60-40 split of the data. In the first experiment, no feature selection or oversampling was applied, resulting in accuracies of 0.6680 for DT, 0.7721 for RF, and 0.8189 for LightGBM. The second experiment used Boruta without SMOTE, showing slightly increased accuracies of 0.6797 for DT, 0.7944 for RF, and 0.8247 for LightGBM, indicating the positive impact of feature selection alone. The third experiment used SMOTE without Boruta, resulting in further accuracy improvements of 0.7401 for DT, 0.8277 for RF, and 0.8955 for LightGBM, highlighting the effectiveness of SMOTE in addressing class imbalance. The fourth experiment utilized both SMOTE and Boruta, resulting in slightly increased accuracies compared to the third experiment, with values of 0.7720 for DT, 0.8470 for RF, and 0.8943 for LightGBM.

Table 4.15: Comparison Result of the Model (60-40 Split)

	Accuracy of Models		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6680	0.7721	0.8189
Using Boruta without SMOTE	0.6797	0.7944	0.8247
Using SMOTE without Boruta	0.7401	0.8277	0.8955
Using SMOTE & Boruta	0.7720	0.8470	0.8943

For the 80-20 split, using both SMOTE and Boruta leads to the highest accuracy values for all models. A similar trend is observed with a 70-30 split, where Boruta and SMOTE, whether used individually or in combination, improve accuracies compared to the baseline models. LightGBM consistently outperforms the other models. For a 60-40 split, reinforces these findings, with Boruta and SMOTE positively affecting model accuracy and LightGBM showing superior performance. In summary, the results emphasize the effectiveness of feature selection and class imbalance handling in enhancing model accuracy, with LightGBM consistently outperforming DT and RF. These findings highlight the significance of preprocessing techniques in achieving better performance in predictive modelling.

4.3.2 Confusion Matrix of Models

As shown in Figure 4.13, when SMOTE and Boruta were not used in the experiment, the DT model achieved the highest TP rate of 0.84 for the Shadowbrokers malware family. The RF model achieved the highest TP rates for the APT1, Crypto, and Shadowbrokers families, with values of 0.80, 0.90, and 0.97. The LightGBM model had the highest TP rates for APT1, Crypto, and Shadowbrokers, with values of 0.86, 0.96, and 0.99. In comparison, the LightGBM model consistently outperformed both the DT and RF models, demonstrating the highest TP rates for each malware family.

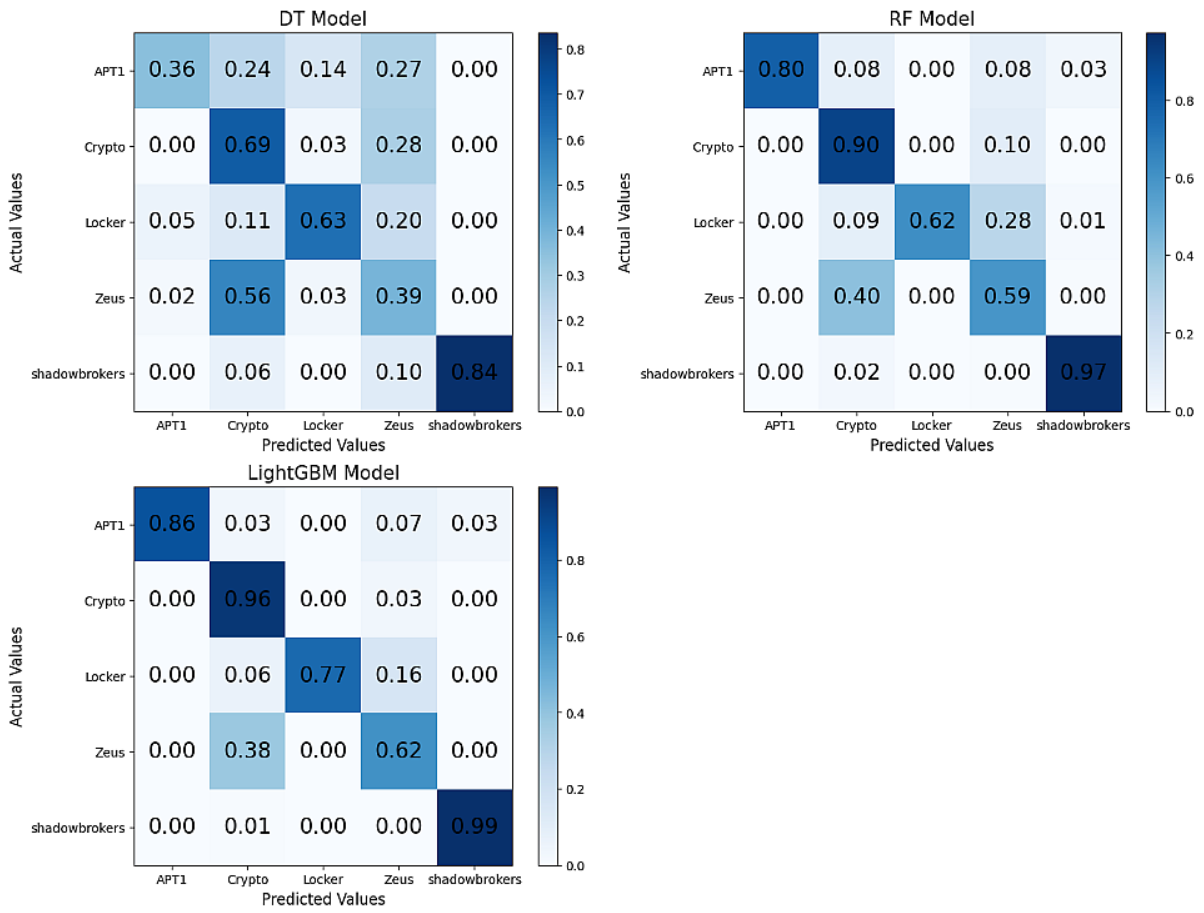

Figure 4.13: Confusion Matrix Without SMOTE & Boruta

As shown in Figure 4.14, when Boruta was used without SMOTE in the experiment, the DT model achieved the highest TP rate of 0.99 specifically for the Shadowbrokers malware family. The RF model had the highest TP rates for the Crypto and Shadowbrokers families, with values of 0.93 and 0.97. The LightGBM model demonstrated the highest TP rates for the APT1, Crypto, and Shadowbrokers families, with values of 0.88, 0.97, and 0.99. Overall, the LightGBM model consistently outperformed both the DT and RF models, exhibiting the highest TP rates for each malware family.

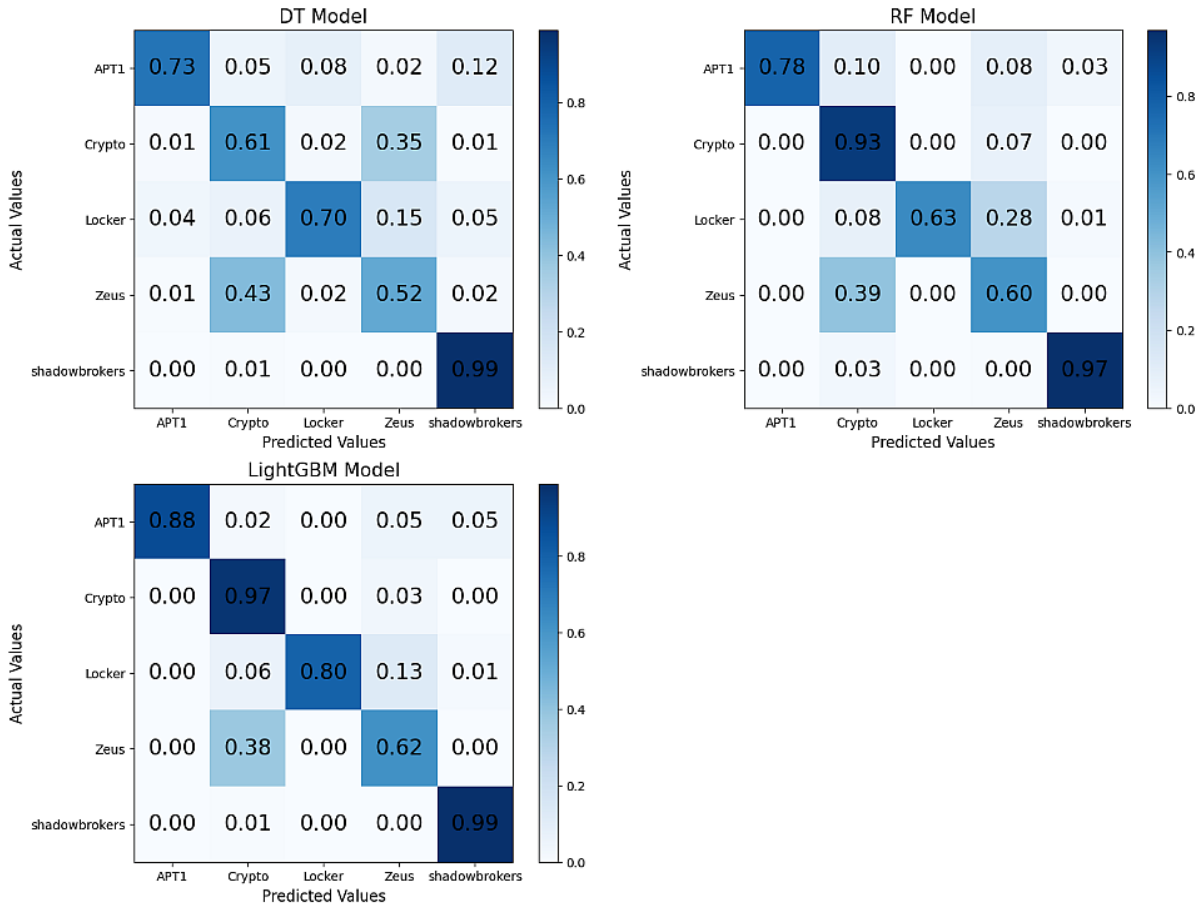

Figure 4.14: Confusion Matrix Using Boruta Without SMOTE

As shown in Figure 4.15, when SMOTE was used without Boruta in the experiment, the DT model achieved the highest TP rates for the APT1, Locker, and Shadowbrokers malware families, with values of 0.96, 0.87, and 0.94. The RF model had the highest TP rates for the APT1, Locker, and Shadowbrokers families, with values of 0.99, 0.97, and 0.98. The LightGBM model demonstrated the highest TP rates for the APT1, Crypto, Locker, and Shadowbrokers families, with values of 0.99, 0.94, 0.96, and 0.99. Throughout the experiment, the LightGBM model consistently outperformed both the DT and RF models, achieving the highest TP rates for each malware family.

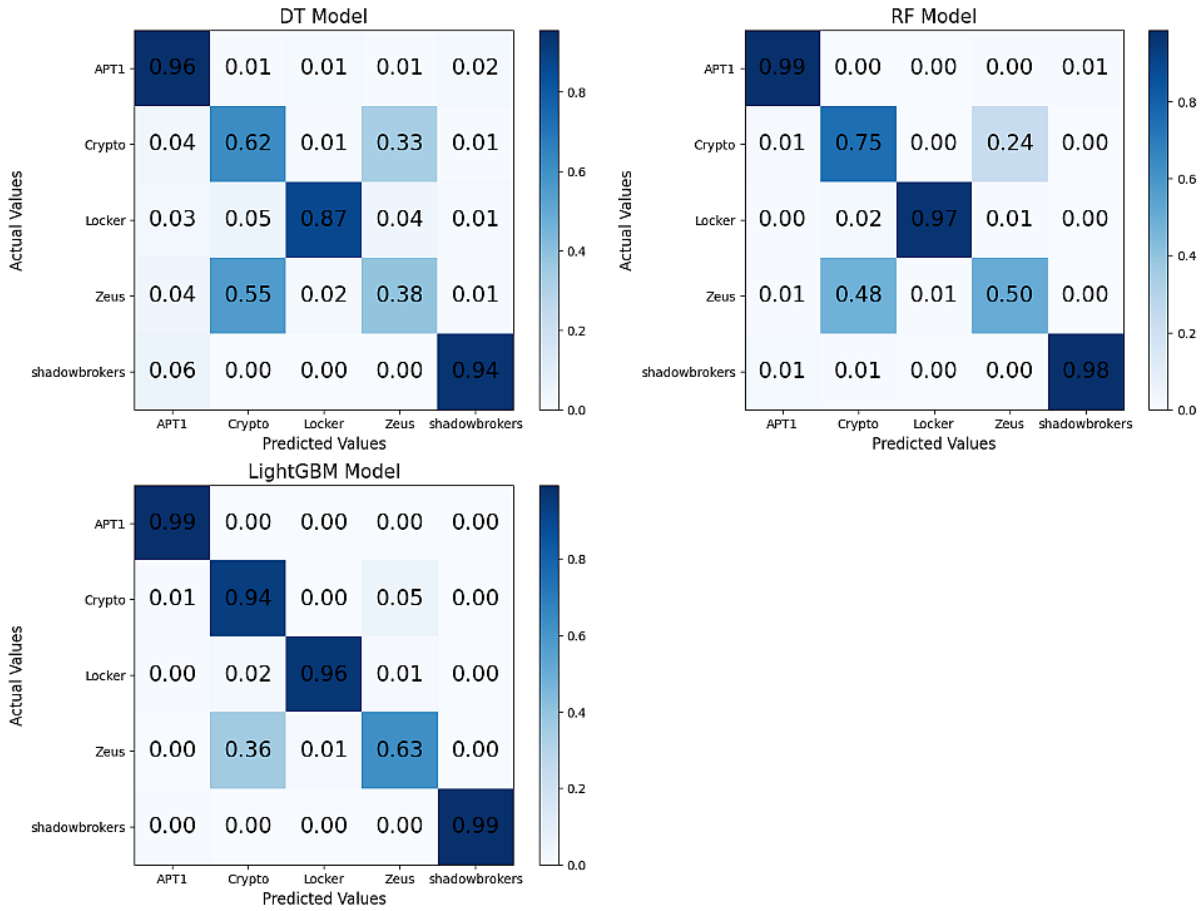

Figure 4.15: Confusion Matrix Using SMOTE Without Boruta

As shown in Figure 4.16, when both SMOTE and Boruta were used in the experiment, the DT model achieved the highest TP rates for the APT1, Locker, and Shadowbrokers malware families, with values of 0.93, 0.92, and 0.96. The RF model had the highest TP rates for the APT1, Locker, and Shadowbrokers families, with values of 0.98, 0.97, and 0.98. The LightGBM model demonstrated the highest TP rates for the APT1, Crypto, Locker, and Shadowbrokers families, with values of 0.99, 0.93, 0.97, and 0.99. Consistently, the LightGBM model outperformed both the DT and RF models by showcasing the highest TP rates for each malware family.

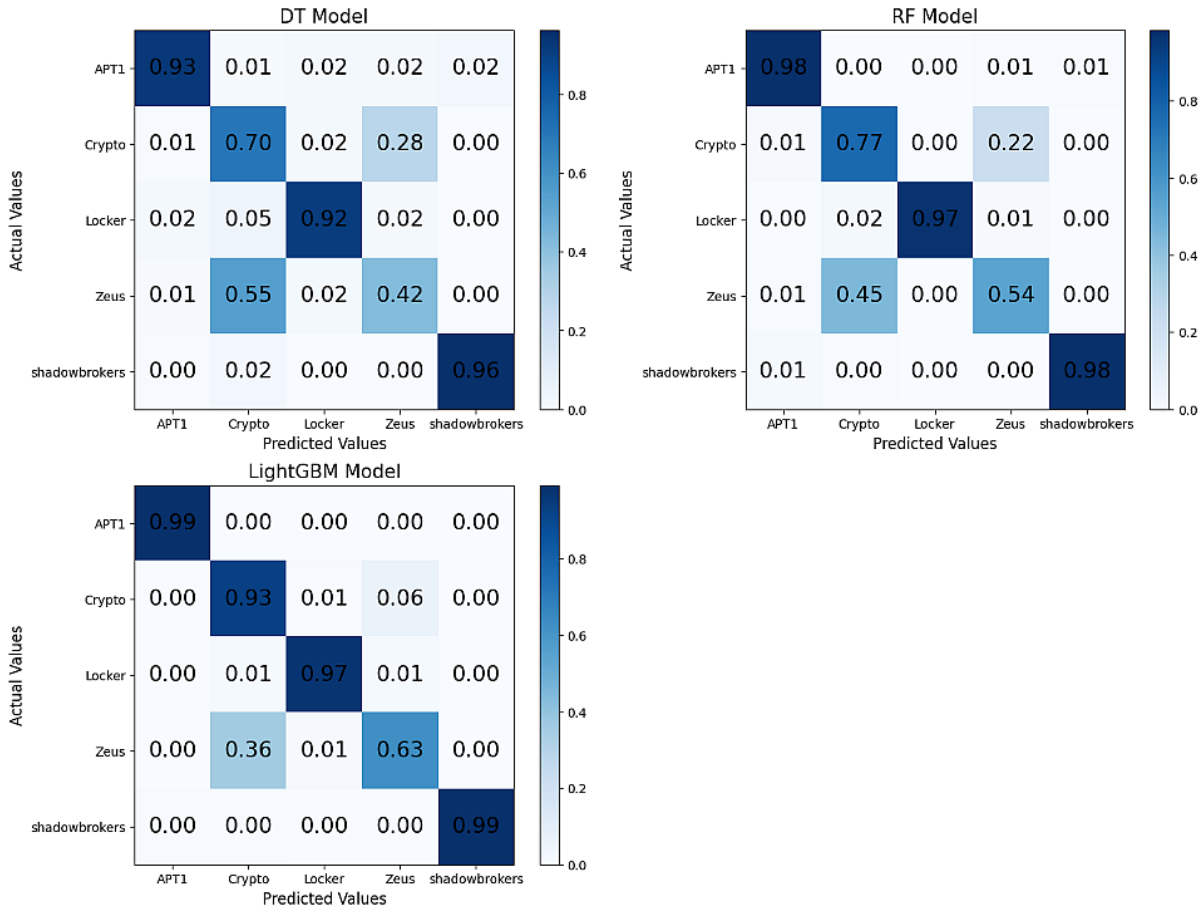

Figure 4.16: Confusion Matrix Using SMOTE and Boruta

To summarize, the results demonstrate that the DT model had the lowest TP rates for all malware families compared to the RF and LightGBM models. The RF model showed slightly better TP rates than the DT model. However, consistently, the LightGBM model outperformed both the DT and RF models in all experiments, achieving the highest TP rates for each malware family. These findings indicate that the LightGBM model is highly effective in accurately identifying and differentiating various types of malwares.

4.4 Application Deployment Result

The application of the model involves utilizing LightGBM, a machine learning model, for the purpose of detecting malware. This detection system is implemented through a REST API. When using the malware detection feature, the user is provided with a selection of malware types, such as APT1, Crypto, Locker, Zeus, Shadowbrokers, and Random. If no malware type is selected, the option to preview malware data is disabled. Upon selecting a malware type and clicking on "Preview Malware Data," the system displays the malware's label and provides a preview of the corresponding malware data in Figure 4.17.

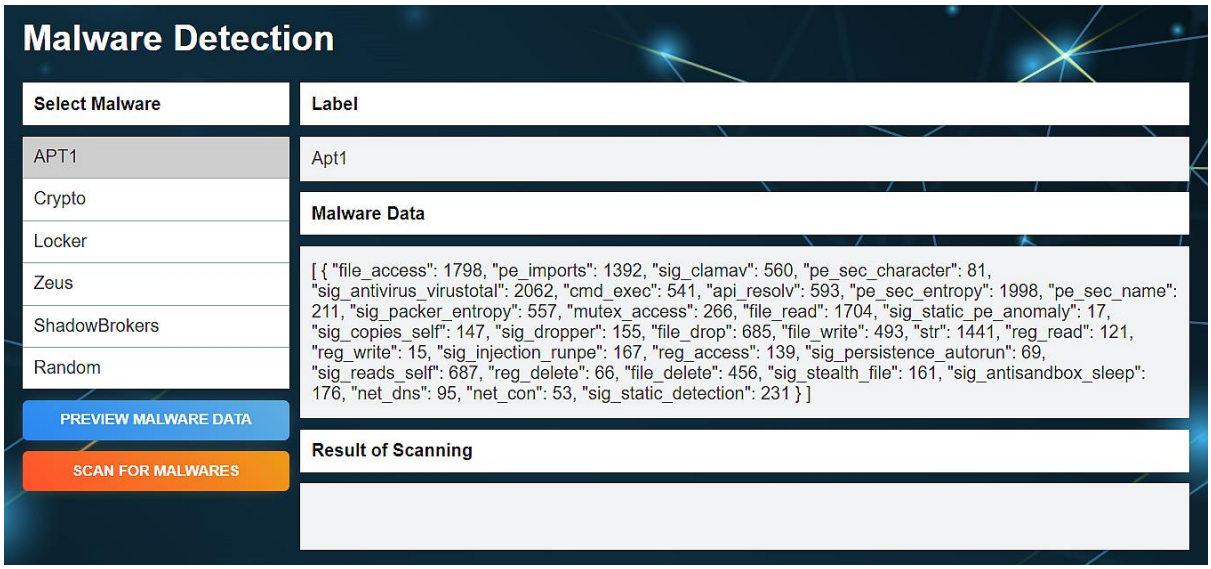

Figure 4.17: Preview Malware Data

Once the user has previewed the malware data, they can proceed by clicking the "Scan for Malwares" button to view the scanning results. However, if the "Preview Malware Data" button is not clicked, the option to click the "Scan for Malwares" button will be disabled. The scanning results, which indicate whether the malware is true or false, will be displayed in Figure 4.18 and Figure 4.19.

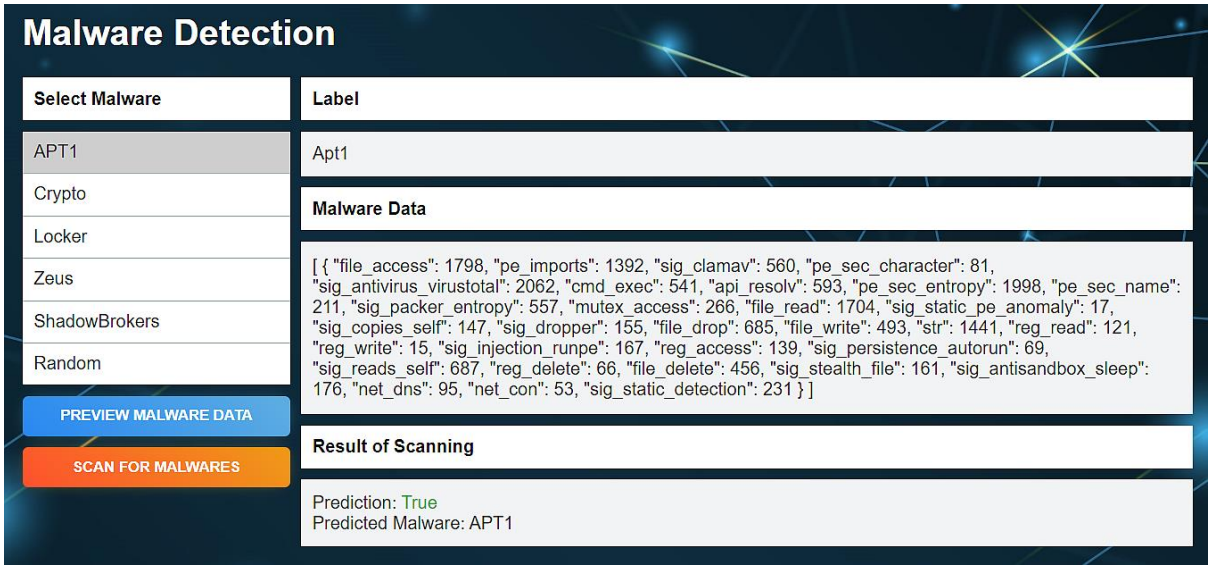

Figure 4.18: The Scanning Result is True

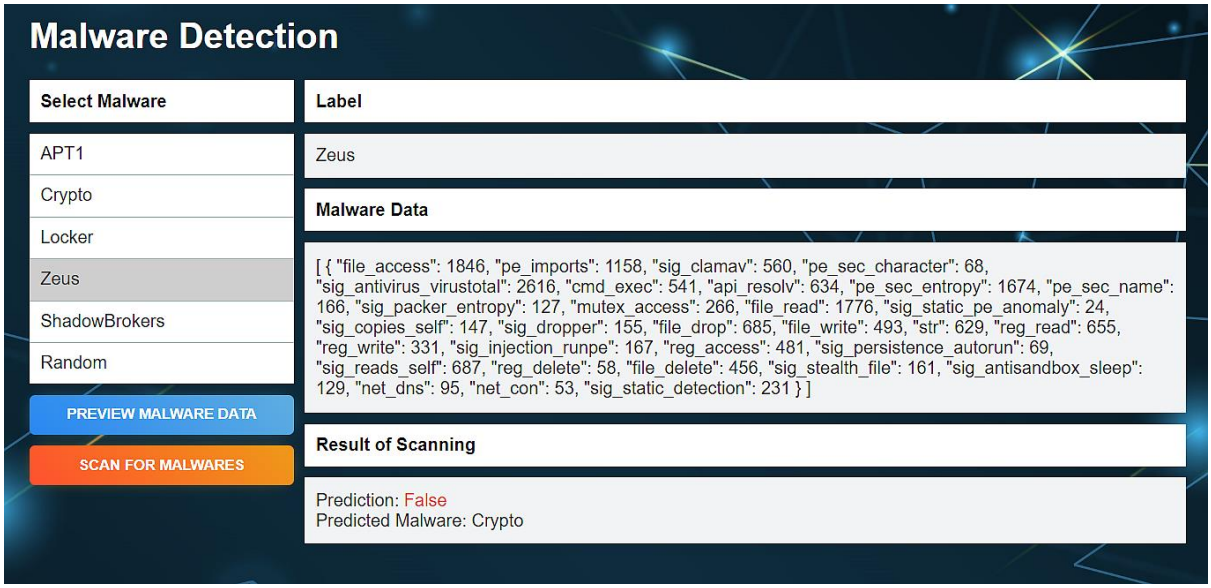

Figure 4.19: The Scanning Result is False

CHAPTER 5 CONCLUSION

5.1 Conclusion

In conclusion, malware analysis with machine learning techniques has demonstrated its significant worth in the detection and classification of malicious software. Leveraging the capabilities of machine learning algorithms, the identification and classification of malware can be conducted with great efficacy by examining intricate patterns, behaviors, and characteristics extracted from extensive datasets. This approach not only enables the swift identification of malware but also facilitates its effective categorization, thereby enhancing our ability to combat the ever-evolving landscape of malicious software threats. By harnessing the power of machine learning, we are empowered with an invaluable tool to safeguard against the pernicious activities orchestrated by malware.

In order to select the most suitable machine learning model for implementing a malware detection system, an evaluation was conducted comparing the performance of three models: DT, RF, and LightGBM. The objective was to determine which model would excel in detecting malware effectively. Through thorough analysis and experimentation, the results conclusively demonstrate that LightGBM outperforms both DT and RF by a significant margin. Among the three models, LightGBM exhibited exceptional performance, achieving the highest accuracy score of 0.9059 in the experiment where SMOTE was utilized without Boruta, specifically for the 80-20 split. These findings underscore the superiority of LightGBM as the preferred choice for accurate and efficient malware detection.

This research aims to accomplish several objectives. Firstly, EDA provides insights and a deeper understanding of the nature of malware. Secondly, the robust machine learning model is developed that can effectively identify different types of malwares and distinguish them from legitimate files or applications. Lastly, the performance of various machine learning models is evaluated to assess their effectiveness in detecting malware. These objectives have been successfully achieved through the research process. They also help in enhancing the field of malware detection and classification.

There are certain constraints that come with developing this project. One limitation involves the availability of adequate training data. Machine learning models require ample high-quality data to learn patterns and make precise predictions. However, obtaining a diverse and comprehensive dataset of malware samples can be challenging due to malware's dynamic

nature and the difficulties in obtaining labeled samples. Another limitation is the significant time and resources required for training complex machine learning models. It requires significant computational resources, including powerful hardware, and can be time-consuming, posing challenges for resource-constrained organizations or situations requiring swift analysis and decision-making.

5.2 Future Work

In the future, the field of analyzing malware using machine learning is expected to make significant advancements. Currently, analysts spend a lot of time manually examining malware samples and creating ways to detect them. However, in the future, there could be automated systems that use machine learning to analyze malware more efficiently. These systems would be able to extract features, detect malicious behavior, and classify malware into different types, reducing the workload of human analysts.

Furthermore, attribution in malware analysis refers to identifying the individuals, groups, or countries responsible for a specific malware attack. In the future, machine learning techniques could be used to analyze various characteristics of malware, such as code similarities, infrastructure patterns, or behavior, to attribute the malware to specific threat actors. This attribution can provide valuable information for threat intelligence and help in taking proactive measures against future attacks.

Lastly, traditional antivirus software relies on known patterns to detect and prevent malware. However, future work in malware analysis with machine learning would focus on developing dynamic systems that can detect and prevent malware in real-time. These systems would continuously analyze system behavior, network traffic, and other relevant data streams using machine learning algorithms, enabling them to detect and block malware in real-time, even if it has never been encountered before.

REFERENCES

[1] Ramilli, M. (2016). Malware Training Sets: a machine learning dataset for everyone. *Marco Ramilli Web Corner*.

[2] Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, *81*, 123-147.

[3] Singh, J., & Singh, J. (2018). Challenge of malware analysis: malware obfuscation techniques. *International Journal of Information Security Science*, *7*(3), 100-110.

[4] Chikapa, M., & Namanya, A. P. (2018, August). Towards a fast off-line static malware analysis framework. In *2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)* (pp. 182-187). IEEE.

[5] Ijaz, M., Durad, M. H., & Ismail, M. (2019, January). Static and dynamic malware analysis using machine learning. In *2019 16th International bhurban conference on applied sciences and technology (IBCAST)* (pp. 687-691). IEEE.

[6] Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V. (2019). A survey on malware analysis and mitigation techniques. *Computer Science Review*, *32*, 1-23.

[7] Patil, R., & Deng, W. (2020, March). Malware Analysis using Machine Learning and Deep Learning techniques. In *2020 SoutheastCon* (Vol. 2, pp. 1-7). IEEE.

[8] Greener, J. G., Kandathil, S. M., Moffat, L., & Jones, D. T. (2022). A guide to machine learning for biologists. *Nature Reviews Molecular Cell Biology*, *23*(1), 40-55.

[9] Sheykhmousa, M., Mahdianpari, M., Ghanbari, H., Mohammadimanesh, F., Ghamisi, P., & Homayouni, S. (2020). Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13, 6308-6325.

[10] Tyralis, H., Papacharalampous, G., & Langousis, A. (2019). A brief review of random forests for water scientists and practitioners and their recent history in water resources. *Water*, *11*(5), 910.

[11] Huljanah, M., Rustam, Z., Utama, S., & Siswantining, T. (2019, June). Feature selection using random forest classifier for predicting prostate cancer. In *IOP Conference Series: Materials Science and Engineering* (Vol. 546, No. 5, p. 052031). IOP Publishing.

[12] Moran, M., & Gordon, G. (2019). Curious feature selection. *Information Sciences*, *485*, 42-54.

[13] Noorbehbahani, F., Rasouli, F., & Saberi, M. (2019, August). Analysis of machine learning techniques for ransomware detection. In *2019 16th International ISC (Iranian*

Society of Cryptology) Conference on Information Security and Cryptology (ISCISC) (pp. 128-133). IEEE.

[14] Rodríguez-Rodríguez, I., Rodríguez, J. V., Woo, W. L., Wei, B., & Pardo-Quiles, D. J. (2021). A comparison of feature selection and forecasting machine learning algorithms for predicting glycaemia in type 1 diabetes mellitus. *Applied Sciences*, *11*(4), 1742.

[15] Li, W., Chen, L., Zhao, J., & Wang, W. (2021). Embedded feature selection based on relevance vector machines with an approximated marginal likelihood and its industrial application. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *52*(4), 2601-2614.

[16] Masabo, E. (2019). *A Feature Engineering Approach for Classification and Detection of Polymorphic Malware using Machine Learning* (Doctoral dissertation, Makerere University).

[17] Masabo, E., Kaawaase, K. S., & Sansa-Otim, J. (2018, May). Big data: deep learning for detecting malware. In *2018 IEEE/ACM Symposium on Software Engineering in Africa (SEiA)* (pp. 20-26). IEEE.

[18] Morales-Molina, C. D., Santamaria-Guerrero, D., Sanchez-Perez, G., Perez-Meana, H., & Hernandez-Suarez, A. (2018, November). Methodology for malware classification using a random forest classifier. In *2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)* (pp. 1-6). IEEE.

[19] Hadiprakoso, R. B., Kabetta, H., & Buana, I. K. S. (2020, November). Hybrid-based malware analysis for effective and efficiency android malware detection. In *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)* (pp. 8-12). IEEE.

[20] Chowdhury, M., Rahman, A., & Islam, R. (2017, June). Malware analysis and detection using data mining and machine learning classification. In *International conference on applications and techniques in cyber security and intelligence* (pp. 266-274). Edizioni della Normale, Cham.

[21] AMR, et al. "It Threat Evolution in Q2 2021. PC Statistics." Securelist English Global Securelistcom, 13 May 2021, securelist.com/it-threat-evolution-in-q2-2021-pc-statistics/103607/.

[22] Residentmario. (2018, April 20). Automated feature selection with Boruta. Retrieved January 26, 2023, from https://www.kaggle.com/code/residentmario/automated-feature-selection-with-boruta/notebook

[23] Horkoff, J., Hammouda, I., & Knauss, E. (2018). Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. *IEEE software*, *37*(2), 52-59.

[24] Viglianisi, E., Dallago, M., & Ceccato, M. (2020, October). Resttestgen: automated black-box testing of restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)* (pp. 142-152). IEEE.

[25] Kim, M., Xin, Q., Sinha, S., & Orso, A. (2022, July). Automated test generation for rest apis: No time to rest yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 289-301).

[26] Bach, K., Mathisen, B. M., & Jaiswal, A. (2019). Demonstrating the myCBR Rest API. In *ICCBR Workshops* (pp. 144-155).

[27] Weldegebriel, H. T., Liu, H., Haq, A. U., Bugingo, E., & Zhang, D. (2019). A new hybrid convolutional neural network and eXtreme gradient boosting classifier for recognizing handwritten Ethiopian characters. *IEEE Access*, *8*, 17804-17818.

[28] Dhieb, N., Ghazzai, H., Besbes, H., & Massoud, Y. (2019, September). Extreme gradient boosting machine learning algorithm for safe auto insurance operations. In *2019 IEEE international conference on vehicular electronics and safety (ICVES)* (pp. 1-5). IEEE.

[29] Taha, A. A., & Malebary, S. J. (2020). An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine. *IEEE Access*, *8*, 25579-25587.

[30] Liang, W., Luo, S., Zhao, G., & Wu, H. (2020). Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms. *Mathematics*, *8*(5), 765.

[31] Trabelsi, A., Elouedi, Z., & Lefevre, E. (2019). Decision tree classifiers for evidential attribute values and class labels. Fuzzy Sets and Systems, 366, 46-62.

[32] Charbuty, B., & Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. Journal of Applied Science and Technology Trends, 2(01), 20-28.

[33] Abdallah, I., Ntertimanis, V., Mylonas, C., Tatsis, K., Chatzi, E., Dervilis, N., ... & Eoghan, M. (2018). Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. Safety and Reliability–Safe Societies in a Changing World, 3053-3061.

APPENDIX A RESEARCH PAPER

Malware Analysis with Machine Learning

Carlven Tan [1], Hu Ng [1], Timothy Tzen Vun Yap [1]

[1] Faculty of Computing and Informatics, Multimedia University, 63100 Cyberjaya, Malaysia

Abstract. Malware analysis is crucial in cybersecurity for identifying and eliminating malicious software. Traditional methods have proven to be time-consuming and resource intensive. This research project aimed to leverage machine learning techniques to enhance the efficiency and accuracy of malware analysis. A malware detection system was developed, utilizing machine learning algorithms to classify malware based on their behavior and characteristics. The system was trained on a diverse dataset comprising various malware families and evaluated using appropriate metrics. The dataset was sourced from a publicly available repository on GitHub, provided by Marco Ramili. Prior to classification, several stages were completed, including exploratory data analysis, data preprocessing, data splitting, and feature selection. Notably, the machine learning algorithms employed were Decision Trees (DT), Random Forests (RF), and LightGBM. Four experiments were conducted, comparing different techniques across varying train-test split ratios. The results demonstrated that LightGBM outperformed other models consistently. For the 80-20 and 60-40 splits, LightGBM achieved accuracy rates of 90.59% and 89.55% respectively, using SMOTE without Boruta. Furthermore, LightGBM attained the highest accuracy of 90.33% for the 70-30 split, employing SMOTE and Boruta. The deployment of the malware detection system through a machine learning API will utilize LightGBM.

Keywords: Malware Detection, Machine Learning, Classification.

1. Introduction

Malware refers to any type of software or code that is intended to cause harm to a computer system or network. It can come in various forms to attack a computer system or network, such as spyware, adware, ransomware, trojan, worms, rootkits and backdoors. It can be used to accomplish tasks such as stealing protected data, installing unwanted software, or deleting confidential documents without authorization. Due to the widespread use of the internet, malware can spread quickly and target both individuals and organizations, making it a significant security threat. Cyber criminals are allowed to take advantage of stealing the data from the infected target over the internet (Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V, 2019). Therefore, detecting and preventing malware is a top priority for many organizations and individuals.

In recent times, traditional malware detection systems have faced challenges in effectively detecting modern malware as attackers are constantly evolving their techniques and spreading their malicious software in large numbers. The prevalent method of malware detection, signature-based detection, involves identifying the signature of known malware in a database and removing it. However, this approach is not effectively as it struggles to detect polymorphic malware and zero-days attacks, with detection rate currently estimated to be between 25% and 50% currently (Masabo E, Kaawaase K S, Sansa-Otim J, 2018). This is because attackers can easily change the byte sequence within malware to avoid detection. Even after a new unknow threat is discovered, it is difficult to research and identify patterns in the malware as it is constantly changing. Additionally, malware cryptors can keep altering the malware until it becomes unrecognized to antivirus scans.

According to AMR, et al (2021), in the report of It Threat Evolution in Q2 2021, around one

million new threats created were injected daily into the wild in 2021. In 2021, Kaspersky solutions prevented 1,686,025,551 attacks from online resources worldwide. They also reported web antivirus recognized 675,832,360 unique URL as malicious. Moreover, the attacker attempts to run malware for stealing money from online bank accounts were blocked on the computers of 119,252 unique users. Ransomware attacks were also detected and defeated on the computers of 97,451 unique users. A total of 68,294,298 unique malicious and potentially unwanted objects were detected by Kaspersky file antivirus.

To address the issue of malware, researchers have developed new method for accurately identifying and detecting it. These methods typically involve two important steps. The first step is analyzing the malware to extract information about its specific characteristics. This analysis can help to detect and mitigate any potential threat by understanding the malware's function, purposes and potential impact. There are two types of malware analysis that are commonly used, which are static analysis and dynamic analysis (Ijaz M, Durad M H, Ismail M, 2019). Static analysis examines samples without executing it, while dynamic analysis examines the execution of samples while they are running to observe their behaviours. After the samples have been analyzed, the information about their pattern and behaviour is used to identify the malware. In the second step, classification and detection systems are developed based on the information provided by malware analysis. These systems store signature patterns in their database and update them regularly (Singh J, Singh J., 2018). However, this approach may not be effective against polymorphic malware, which can change its identity at any time.

The current limitations of malware detection systems, such as their inability to detect modern and advanced forms of malware, have led to the need for alternative methods. Malware detection can be addressed by leveraging machine learning models as a potential solution. Further research is needed to identify the key features that best characterize the malware, as the efficiency of the employed machine learning techniques heavily relies on the careful selection of appropriate features. Feature selection, recognized as a critical stage in the creation of an effective machine learning model, is a fundamental process. With various machine learning algorithms available, making it challenging to identify the one that would be most suitable for a particular dataset and enhance the model's performance. In many cases, real-world datasets are complex and trying multiple algorithms may be necessary to achieve the desired results.

2. Literature Review

2.1. Malware Analysis

Malware analysis is a crucial aspect of computer security, as it aims to identify and classify new and unknown types of malwares to improve defences against them (Ucci D, Aniello L, Baldoni R, 2019). One of the main challenges in malware analysis is the ability to detect malicious software that has been created using obfuscation techniques such as polymorphism, metamorphism, and packing. These techniques make it difficult for traditional malware detection methods to identify and block malware before it causes harm (Singh, J., & Singh, J. 2018). To address this issue, the research community has developed various approaches for malware analysis, including both static and dynamic analysis techniques. Static analysis inspects the malware files in a safe environment without running them. Although this makes it more efficient than dynamic analysis, it still has drawbacks (Ijaz, Durad, and Ismail, 2019). For example, it prevents the reverse engineering of malware, which restricts the investigation of the source code. Dynamic analysis entails executing the malicious file and seeing how it interacts with the computer network. Malware files are run in a virtual environment made with the aid of emulators, hypervisors, and programs like VMware or Virtual Box in order to prevent damage to the host system. Malware can be identified by keeping track of its activities, such as the creation or deletion of files (Singh, J., & Singh, 2018).

2.2. Machine Learning Techniques

Machine learning, a branch of AI, utilizes data and algorithms to mimic human pattern recognition and enhance performance accuracy (Greener, Joe G., et al, 2022). In malware analysis, machine learning automates the process of analyzing and classifying malware, using techniques like clustering and classification to identify patterns and categorize unknown malware into specific families (Patil, Rajvardhan, and Wei Deng, 2020). DT, RF, and GT are commonly employed for malware classification.

Decision Trees (DT) are highly effective and efficient methodologies in machine learning, successfully addressing real-world challenges in artificial intelligence (Trabelsi et al., 2019). They resemble human decision-making processes, handling both discrete and continuous data inputs (Charbuty & Abdulazeez, 2021). DT is commonly used in data mining to create classifiers, categorize data with training sets, and derive assumptions about categorical class names (Charbuty & Abdulazeez, 2021). Decision trees are widely implemented due to their simplicity, accuracy, and applicability to diverse data forms. They offer quick learning, visually intuitive representation, support for data-driven root cause analysis, and comparable or superior accuracy, especially with ensemble decision tree classifiers (Abdallah et al., 2018).

Random Forest (RF) is a popular ensemble learning technique for classification and regression tasks. It combines multiple decision trees to make predictions through majority voting. Ensemble learning improves accuracy and stability by integrating predictions from multiple classifiers, reducing variance (Sheykhmousa et al., 2020). RF's effectiveness is attributed to its use of bagging, where models are trained on bootstrap samples and predictions are combined through averaging. This enhances model precision and stability (Tyralis et al., 2019). RF is widely used in various domains, handling missing attributes and large sample sizes. It excels in complex and uncertain data, making it a powerful tool for machine learning tasks (Huljanah et al., 2019).

XGBoost is a highly effective algorithm that improves upon the limitations of traditional gradient boosting methods, offering better performance and reduced execution time (Weldegebriel et al., 2019). It has gained recognition as one of the leading algorithms in machine learning, particularly in Kaggle competitions, surpassing deep learning models in terms of speed and scalability (Dhieb et al., 2019). Another powerful gradient boosting framework is LightGBM, known as the "Light Gradient Boosting Machine." LightGBM excels in managing massive data volumes and distributed data processing, utilizing a histogram-based algorithm and a growth strategy that prioritizes individual leaves to accelerate training and minimize memory consumption (Taha & Malebary, 2020; Liang et al., 2020).

2.3. Feature Selection

Feature selection, the process of selecting relevant attributes from a dataset, is crucial for constructing accurate models and impacting prediction performance, runtime, generalization, and overfitting avoidance (Moran & Gordon, 2019). Three main approaches exist: wrapper, filter, and embedded. Wrapper methods involve using a predictive model to assess the performance of various feature subsets, but they are computationally intensive and carry the risk of overfitting. Filter methods, on the other hand, evaluate individual features based on their intrinsic characteristics and are computationally efficient, providing a robust feature subset. Filter methods are often referred to as Univariate Feature Selection as they analyze each feature individually to evaluate its predictive power (Rodríguez-Rodríguez et al., 2021). Embedded methods combine the benefits of wrapper and filter methods, performing feature selection and classification simultaneously using a specific learning algorithm. While they can improve performance, embedded methods tend to be computationally intensive, but they avoid the need to repeatedly evaluate different subsets, enhancing computational efficiency (Li et al., 2021).

2.4. REST API

A REST API is a type of web API that uses HTTP requests to retrieve and modify resources, typically in JSON or XML format. It enables clients to access and manipulate resources through stateless operations, providing a uniform interface for CRUD operations. REST APIs are identified by HTTP URLs and mapped to corresponding methods like GET, POST, PUT, and DELETE (Kim et al., 2022). For instance, a REST API designed for managing pets can use GET /pets to retrieve a list of pets and POST /pets to add a new pet to the collection. REST APIs can be utilized with various programming languages and simplify system development (Bach et al., 2019). However, testing REST API implementations that incorporate external libraries or frameworks can be challenging when the source code is not fully accessible (Viglianisi et al., 2020).

2.5. Related Works

Masabo (2019) proposed a novel NFE approach that combines structure and behavior features for effective classification and detection of polymorphic malware. To address data imbalance, the training set was balanced using SMOTE. Multiple machine learning algorithms, including GB, LDA, and KNN, were employed for classification, selected for their ability to handle multi-class problems. The dataset was partitioned into training and test sets, with the Gradient Boosting model achieving the highest accuracy of 94% on the balanced dataset.

Masabo et al. (2018) conducted a study on malware detection using big data analytics and machine learning techniques. Their approach was evaluated on a dataset after performing data cleaning and balancing. The focus was primarily on the Crypto and Zeus malware families, with a total of 2957 samples, consisting of 1815 Crypto and 1142 Zeus instances. The Crypto and Zeus classes were encoded as 0 and 1 respectively. The robustness of the deep learning method was assessed using Keras Deep Learning and SVM models. The dataset was split into train and test sets, and the Keras Deep Learning model achieved a top accuracy of 97%.

Morales-Molina et al. (2018) proposed using the RF algorithm to enhance the performance of malware classification. They tested their approach on two datasets: ClaMP Raw-5184 and CSDMC_API_Train. ClaMP Raw-5184 had 5184 samples with 55 characteristics, while CSDMC_API_Train had 388 samples. Feature selection involved PCA for ClaMP Raw-5184 and TF-IDF for CSDMC_API_Train. The data was divided into training and test sets, and machine learning algorithms (LR, DT, RF, and SVM) were trained on the training sets. RF achieved the best precision of 89.83% for ClaMP Raw-5184 and 96% for CSDMC_API_Train.

Hadiprakoso et al. (2020) proposed a resource-efficient approach to malware detection using two datasets: Android Malgenome, DREBIN, and CICMalDroid. They addressed data imbalance through SMOTE and oversampling. Machine learning and deep learning algorithms, such as SVM, Decision Tree, RF, Naive Bayes, K-NN, MLP, and GB, were utilized. PCA was employed for feature extraction. A hybrid malware analysis model combining static and dynamic analysis achieved the best performance, with GB demonstrating around 96% accuracy in static model tests and approximately 99% accuracy in hybrid model tests.

Chowdhury et al. (2017) proposed a hybrid architecture combining an MLP neural network with a BAM for malware detection. They used two datasets, one for malware and one for benign files, with a total of 10920 benign files and 41265 malicious files. These datasets were merged to create a common dataset. Their method utilized behavior-based API sequences and signature-based n-gram features. SVM, Decision Tree (J48), Naive Bayes, and RF classifiers were employed and evaluated using K-fold cross-validation with K=10. The proposed classifier achieved the highest accuracy of 98.6% among the machine learning classifiers.

3. Methodology

Several important stages need to be performed to ensure the effectiveness and usability of the model. These stages include data collection, data pre-processing, data splitting, feature selection, model construction, model evaluation, and application deployment with a machine learning API.

3.1. Data Collection

The dataset used for the study was compiled from a public repository on GitHub, provided by Marco Ramili (Ramilli, Marco, 2016). It consists of executable malware features stored in JSON files organized in different folders. The dataset includes characteristics and behaviors of various malware types, collected by utilizing sandbox tools. The dataset exclusively comprises malware files without any benign files. It focuses on classifying five malware families, namely APT1, Crypto, Locker, Zeus, and ShadowBrokers as shown in Table 1.

Table 1: Malware Families Used

Malware Family	Number of Samples
APT1	292
Crypto	2020
Locker	431
Zeus	2019
shadowbrokers	1270
Total	6032

3.2. Data Preprocessing

Data preprocessing was performed to clean, format and organize data, making it suitable for building and training machine learning models. This stage consisted of several steps, which are data integration, data cleaning, data transformation and data balancing.

Data integration involved combining JSON files from different malware families into a single dataset in CSV format. Each JSON file represented a row of data in the dataset. The process began by reading and converting individual JSON files into CSV format. To create the final dataset, all the files from the malware families were combined into a folder, and their data was appended to a data list, which was then used to construct the CSV dataset. The raw dataset was prepared by transforming the data into Excel files in CSV format.

Data cleaning involved identifying and rectifying errors, inconsistencies, and missing data in the dataset. When merging multiple data sources into a single dataset, errors such as duplicates or inaccuracies can occur. Examination of the dataset revealed missing or null values in certain features. The approach taken to handle these missing values was to fill them with a constant value, such as NA, except when the feature had null values throughout.

Data transformation involved converting categorical variables into numerical format to enable their use in machine learning algorithms. The dataset initially consisted of categorical variables, such as string data types, which were incompatible with certain machine learning algorithms. To address this, a Label Encoding technique (see Algorithm 1) was employed, as it is commonly used for handling categorical variables. Each label was assigned a unique integer based on alphabetical ordering, resulting in the representation of malware families as: 'APT':0, 'Crypto':1, 'Locker':2, 'Zeus':3, 'shadowbrokers':4.

Algorithm 1: Label Encoding

Algorithm 1 Label Encoding

```
import LabelEncoder from sklearn.preprocessing
cols = df.columns[4:69]
label_encoder = LabelEncoder()
for column in cols:
    df[column] = label_encoder.fit_transform(df[column])
```

Data balancing was performed to address the class imbalance issue in the dataset. The majority class, consisting of malware families Cryto and Zeus, had a high number of observations, while the minority classes (APT1, Locker, and Shadowbrokers) had a significantly lower number of observations. To overcome this imbalance, the SMOTE (Synthetic Minority Over-sampling Technique) technique was applied (Masabo, E, 2019), which involved augmenting the minority class data by replication. After applying SMOTE, each malware family had an equal number of data points, representing 20% of the dataset. This ensured a fair and unbiased evaluation of model performance.

3.3. Feature Selection

Feature selection was performed using the Boruta algorithm, a wrapper algorithm based on Random Forest (RF) models (Residentmario, 2018). Boruta utilized feature value permutations to determine the importance of each feature (see Algorithm 2). The resulting feature rankings were used to select the important features. In the balanced dataset, features with a ranking of 1 were considered important, while in the imbalanced dataset, only features with a ranking of 1 were selected. These selected features were then used for training and testing the machine learning models.

Algorithm 2: Boruta

Algorithm 2 Boruta

```
import RandomForestClassifier from sklearn.ensemble
import BorutaPy from Boruta
rfc = RandomForestClassifier(criterion='gini', max_depth=12,
        max_features='sqrt', min_samples_split=10, n_estimators=400,
        random_state=0)
boruta_selector = BorutaPy(rfc, n_estimators='auto', verbose=2,
random_state=0)
boruta_selector.fit(np.array(X_train), np.array(y_train))
```

3.4. Model Construction

The task was to classify data into five distinct malware families using a multi-class classification approach. To achieve this, three machine learning algorithms (DT, RF, and LightGBM) were chosen as suitable solutions for addressing the multi-class classification challenge in the dataset. The dataset was split into training and testing sets with various proportions, such as 80% training and 20% testing, 70% training and 30% testing, and 60% training and 40% testing. The training set was utilized to develop machine learning models and feature sets, while the testing set was used to evaluate model performance. Hyperparameter tuning was conducted using grid search, involving an exhaustive search for optimal parameter combinations to achieve the best model results. This process was described in Algorithm 3.

Algorithm 3: GridSearchCv

Algorithm 3 GridSearchCv

```
import GridSearchCV from sklearn.model_selection
classifier = YourClassifier()
param_grid = {
    'param_name_1': [param_value_1a, param_value_1b, ...],
    'param_name_2': [param_value_2a, param_value_2b, ...],
     ...
    }
grid_search = GridSearchCV(classifier, param_grid, n_jobs=-1, cv=10,
                    verbose=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
```

3.5. Model Evaluation

Different evaluation metrics were utilized to assess the performance of the machine learning model and identify its strengths and weaknesses. These metrics, including accuracy, precision, recall, and F1 score, helped determine the model's ability to generalize well to future data. The confusion matrix was employed to compare the model's true positive and negative outcomes with its false positive and negative outcomes.

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3}$$

$$\text{F1 Score} = 2\left(\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\right) \tag{4}$$

3.6. Application Deployment with Machine Learning API

To deploy the machine learning model as a web service, the LightGBM model was used and Flask and Joblib libraries were required. The deployment involved serializing and saving the model using Joblib. The Flask API was used to receive input data and provide predictions as responses. The home endpoint rendered an HTML page, while the prediction endpoint handled POST requests and triggered the 'predict' function.

The prediction function handled POST requests to the '/prediction' route (see Algorithm 4). It parsed the input data from JSON format to a Pandas DataFrame and used the LightGBM model to make predictions. The predictions were stored in the result dictionary, along with a message indicating the malware type. The function returned the result dictionary as a JSON response. The

name == 'main' check read the port number from the command-line argument, using the default port 3000 if no valid port was provided. The LightGBM model was loaded, and if it failed, an error message was displayed. Finally, the Flask application was started with the specified port and debug mode set to True.

Algorithm 4: Prediction

Algorithm 4 Prediction

```
function predict():
    if request.method == 'POST' then
        try:
            data = request.form['data']
            input_dict = json.loads(data)
            input_df = pd.DataFrame.from_dict(input_dict)
            predictions = lgb_model.predict(input_df)
            result = {'predictions': str(predictions)}
            if predictions == 0 then
                result['message'] = 'Predicted Malware: APT1'
            ….
            else
                result['message'] = 'Unknown Malware'
            end if
            return jsonify(result)
        except Exception as e:
            print(str(e))
            return jsonify({'error': str(e)})
        end
    else
        return jsonify({'error': 'Invalid request method'})
    end if
end function
```

4. Result and Discussion

The experiments compared the classification result of DT, RF and LightGBM using different techniques explained in classification result. Across different train-test split ratio, Tables 2, 3, and 4 showed the results of comparing the models with various experiments. The confusion matrices of different models for 80-20 split were compared as shown in Figures 1, 2, 3, and 4.

4.1. Classification Results

Applying both SMOTE and Boruta resulted in the highest accuracy scores for the DT, RF, and LightGBM models in the 80-20 split. The scores were 0.7916, 0.8540, and 0.9050 (see Table 2). The same approach yielded the highest accuracy scores in the 70-30 split (see Table 3), with scores of 0.7601, 0.8389, and 0.9033 for the DT, RF, and LightGBM models. Applying both SMOTE and Boruta led to the highest accuracy scores in the 60-40 split (see Table 4), with scores of 0.7720, 0.8470, and 0.8943 for the DT, RF, and LightGBM models.

Table 2: Comparison Result of the Model (80-20 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6015	0.7854	0.8351
Using Boruta Without SMOTE	0.6744	0.7978	0.8376
Using SMOTE Without Boruta	0.7594	0.8426	0.9059
Using SMOTE & Boruta	0.7916	0.8540	0.9050

Table 3: Comparison Result of the Model (70-30 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6392	0.7713	0.8271
Using Boruta Without SMOTE	0.6840	0.7934	0.8315
Using SMOTE Without Boruta	0.7578	0.8360	0.9020
Using SMOTE & Boruta	0.7601	0.8389	0.9033

Table 4: Comparison Result of the Model (60-40 Split)

	Accuracy		
Experiments	DT	RF	LightGBM
Without SMOTE & Boruta	0.6680	0.7721	0.8189
Using Boruta Without SMOTE	0.6797	0.7944	0.8247
Using SMOTE Without Boruta	0.7401	0.8277	0.8955
Using SMOTE & Boruta	0.7720	0.8470	0.8943

With an 80/20 split, the LightGBM model confusion matrix had the highest TP rate for each malware family among all models and all experiments. In experiments without SMOTE and Boruta (see Fig. 1), the LightGBM model achieved the highest TP rates with APT1, Crypto and Shadowbrokers values of 0.86, 0.96 and 0.99. The LightGBM model achieved the highest TP rates (values 0.88, 0.97, and 0.99) for APT1, Crypto, and Shadowbrokers in experiments with Boruta without SMOTE (see Fig. 2). The LightGBM model achieved the highest TP rates (values 0.99, 0.94, 0.96, and 0.99) for APT1, Crypto, Locker, and Shadowbrokers in experiments with SMOTE without Boruta (see Fig. 3). The LightGBM model achieved the highest TP rates (values 0.99, 0.93, 0.97, and 0.99) for APT1, Crypto, Locker, and Shadowbrokers in experiments with SMOTE and Boruta (see Fig. 4).

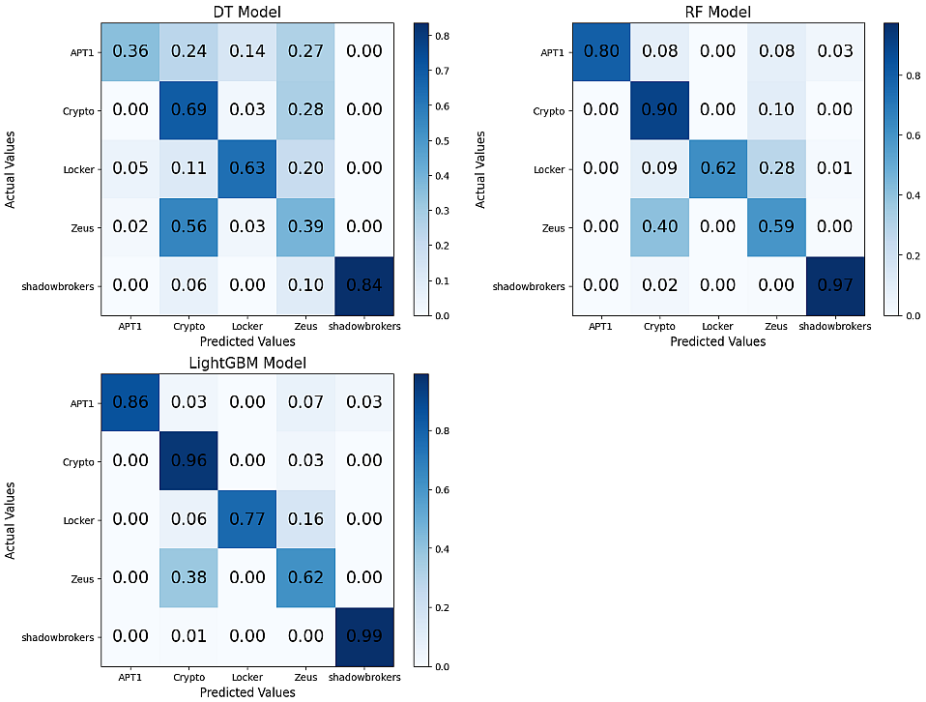

Fig. 1: Confusion Matrix for 80-20 Split (Without SMOTE & Boruta)

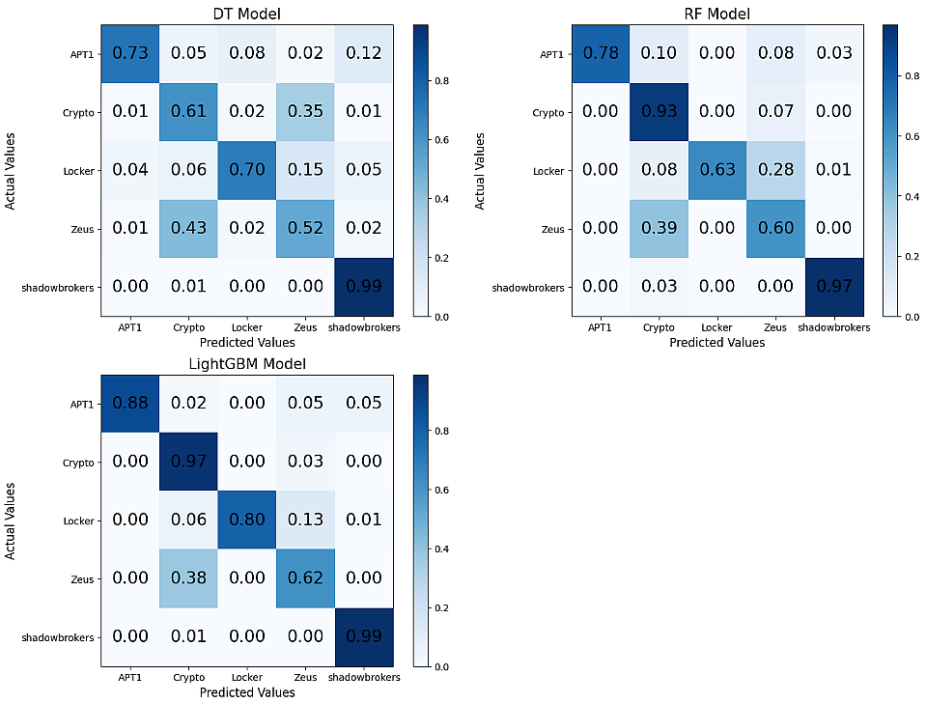

Fig. 2: Confusion Matrix for 80-20 Split (Using Boruta Without SMOTE)

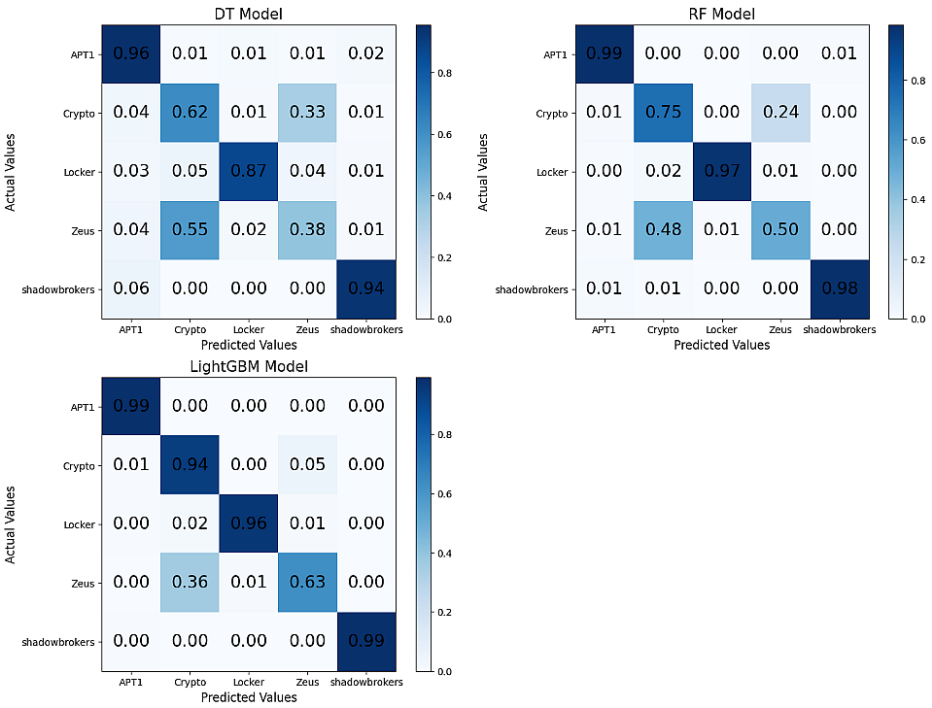

Fig. 3: Confusion Matrix for 80-20 Split (Using SMOTE Without Boruta)

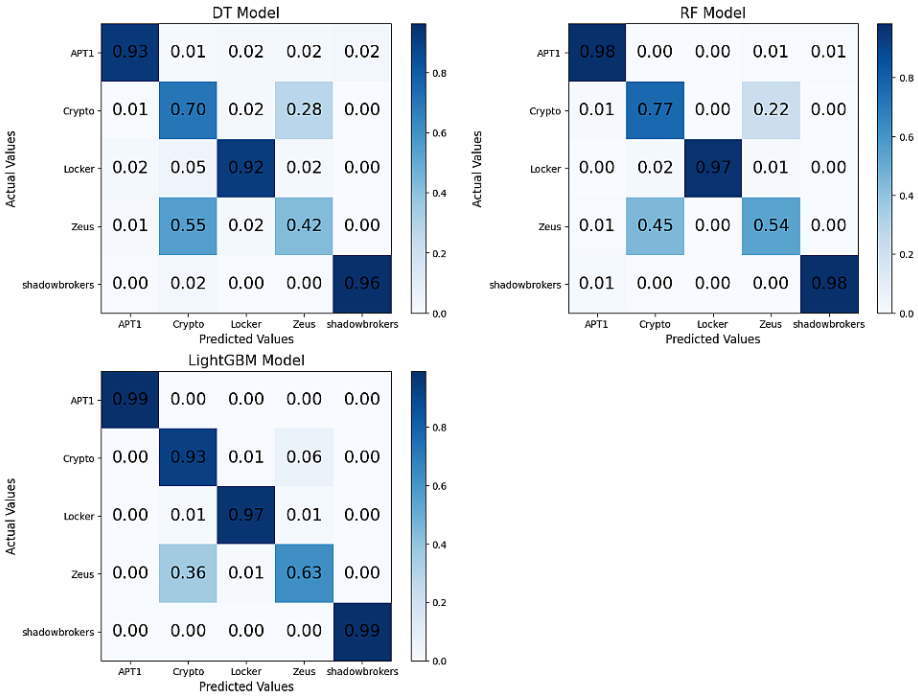

Fig. 4: Confusion Matrix for 80-20 Split (Using SMOTE & Boruta)

In summary, LightGBM consistently outperformed DT and RF models in terms of accuracy across all train-test split ratios. SMOTE without Boruta achieved the highest accuracy for both 80-20 and 60-40 splits, while combining SMOTE and Boruta yielded the best results for the 70-30 split. The LightGBM model consistently exhibited the highest true positive (TP) rates for each malware family according to the confusion matrix for 80-20 split, further confirming its superior performance compared to DT and RF models.

4.2. Application Deployment Results

The model utilizes LightGBM for malware detection through a REST API. Users can select from malware types like APT1, Crypto, Locker, Zeus, Shadowbrokers, and Random. If no type is chosen, the option to preview malware data is disabled. After selecting a type and clicking "Preview Malware Data," the system displays the label and corresponding data. Clicking "Scan for Malwares" shows the scanning results, indicating true or false for each malware (see Fig. 4). The scanning option is disabled if "Preview Malware Data" is not clicked.

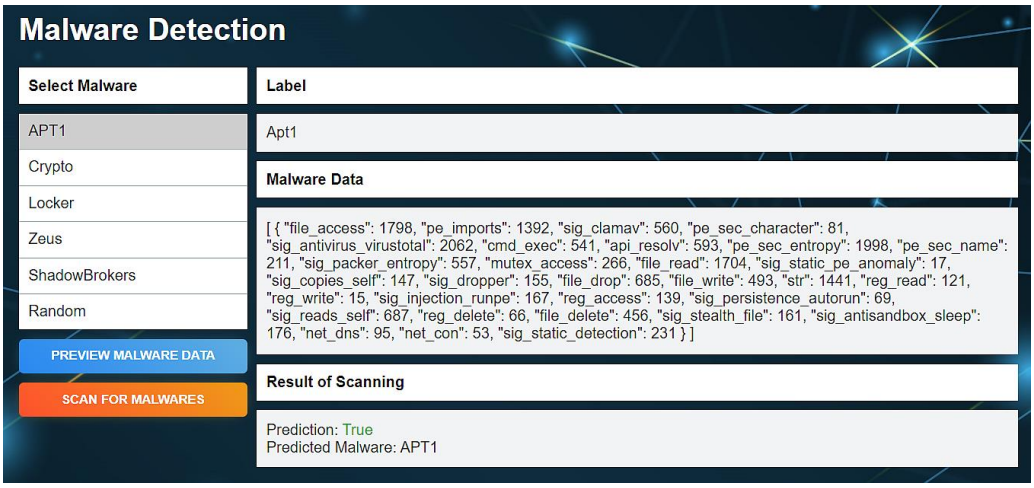

Fig. 4: Scanning Result of Malware Detection

5. Conclusion

Machine learning techniques have proven to be highly valuable in the detection and classification of malware. By leveraging these techniques, we can effectively identify and categorize malicious software by analyzing patterns, behaviors, and characteristics from large datasets. This approach enables swift detection and enhances our ability to combat evolving malware threats. An evaluation of three models, DT, RF, and LightGBM, was conducted to select the most suitable model for malware detection. The results clearly indicate that LightGBM outperforms DT and RF, achieving the highest accuracy score of 0.9059 in the experiment without Boruta but with SMOTE, specifically for the 80-20 split. These findings establish LightGBM as the preferred choice for accurate and efficient malware detection.

This research aims to achieve several objectives: conducting EDA to gain insights into malware, developing a robust machine learning model for malware identification, and evaluating the performance of different models. These objectives have been successfully accomplished, contributing to the field of malware detection and classification. However, certain constraints exist. Obtaining sufficient training data is challenging due to the dynamic nature of malware and the difficulty in obtaining labeled samples. Training complex machine learning models also requires significant time and resources, which can be a limitation for resource-constrained organizations or time-sensitive scenarios.

References

[1] Ramilli, M. (2016). Malware Training Sets: a machine learning dataset for everyone. Marco Ramilli Web Corner.

[2] Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. Computers & Security, 81, 123-147.

[3] Singh, J., & Singh, J. (2018). Challenge of malware analysis: malware obfuscation techniques. International Journal of Information Security Science, 7(3), 100-110.

[4] Chikapa, M., & Namanya, A. P. (2018, August). Towards a fast off-line static malware analysis framework. In 2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW) (pp. 182-187). IEEE.

[5] Ijaz, M., Durad, M. H., & Ismail, M. (2019, January). Static and dynamic malware analysis using machine learning. In 2019 16th International bhurban conference on applied sciences and technology (IBCAST) (pp. 687-691). IEEE.

[6] Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V. (2019). A survey on malware analysis and mitigation techniques. Computer Science Review, 32, 1-23.

[7] Patil, R., & Deng, W. (2020, March). Malware Analysis using Machine Learning and Deep Learning techniques. In 2020 SoutheastCon (Vol. 2, pp. 1-7). IEEE.

[8] Greener, J. G., Kandathil, S. M., Moffat, L., & Jones, D. T. (2022). A guide to machine learning for biologists. Nature Reviews Molecular Cell Biology, 23(1), 40-55.

[9] Sheykhmousa, M., Mahdianpari, M., Ghanbari, H., Mohammadimanesh, F., Ghamisi, P., & Homayouni, S. (2020). Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13, 6308-6325.

[10] Tyralis, H., Papacharalampous, G., & Langousis, A. (2019). A brief review of random forests for water scientists and practitioners and their recent history in water resources. Water, 11(5), 910.

[11] Huljanah, M., Rustam, Z., Utama, S., & Siswantining, T. (2019, June). Feature selection using random forest classifier for predicting prostate cancer. In IOP Conference Series: Materials Science and Engineering (Vol. 546, No. 5, p. 052031). IOP Publishing.

[12] Moran, M., & Gordon, G. (2019). Curious feature selection. Information Sciences, 485, 42-54.

[13] Noorbehbahani, F., Rasouli, F., & Saberi, M. (2019, August). Analysis of machine learning techniques for ransomware detection. In 2019 16th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC) (pp. 128-133). IEEE.

[14] Rodríguez-Rodríguez, I., Rodríguez, J. V., Woo, W. L., Wei, B., & Pardo-Quiles, D. J. (2021). A comparison of feature selection and forecasting machine learning algorithms for predicting glycaemia in type 1 diabetes mellitus. Applied Sciences, 11(4), 1742.

[15] Li, W., Chen, L., Zhao, J., & Wang, W. (2021). Embedded feature selection based on relevance vector machines with an approximated marginal likelihood and its industrial application. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 52(4), 2601-2614.

[16] Masabo, E. (2019). A Feature Engineering Approach for Classification and Detection of Polymorphic Malware using Machine Learning (Doctoral dissertation, Makerere University).

[17] Masabo, E., Kaawaase, K. S., & Sansa-Otim, J. (2018, May). Big data: deep learning for detecting malware. In 2018 IEEE/ACM Symposium on Software Engineering in Africa (SEiA) (pp. 20-26). IEEE.

[18] Morales-Molina, C. D., Santamaria-Guerrero, D., Sanchez-Perez, G., Perez-Meana, H., & Hernandez-Suarez, A. (2018, November). Methodology for malware classification using a random forest classifier. In 2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC) (pp. 1-6). IEEE.

[19] Hadiprakoso, R. B., Kabetta, H., & Buana, I. K. S. (2020, November). Hybrid-based malware analysis for effective and efficiency android malware detection. In 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS) (pp. 8-12). IEEE.

[20] Chowdhury, M., Rahman, A., & Islam, R. (2017, June). Malware analysis and detection using data mining and machine learning classification. In International conference on applications and techniques in cyber security and intelligence (pp. 266-274). Edizioni della Normale, Cham.

[21] AMR, et al. "It Threat Evolution in Q2 2021. PC Statistics." Securelist English Global Securelistcom, 13 May 2021, securelist.com/it-threat-evolution-in-q2-2021-pc-statistics/103607/.

[22] Residentmario. (2018, April 20). Automated feature selection with Boruta. Retrieved January 26, 2023, from https://www.kaggle.com/code/residentmario/automated-feature-selection-with-boruta/notebook

[23] Horkoff, J., Hammouda, I., & Knauss, E. (2018). Emerging perspectives of application programming interface strategy: A framework to respond to business concerns. IEEE software, 37(2), 52-59.

[24] Viglianisi, E., Dallago, M., & Ceccato, M. (2020, October). Resttestgen: automated black-box testing of restful apis. In 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (pp. 142-152). IEEE.

[25] Kim, M., Xin, Q., Sinha, S., & Orso, A. (2022, July). Automated test generation for rest apis: No time to rest yet. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 289-301).

[26] Bach, K., Mathisen, B. M., & Jaiswal, A. (2019). Demonstrating the myCBR Rest API. In ICCBR Workshops (pp. 144-155).

[27] Weldegebriel, H. T., Liu, H., Haq, A. U., Bugingo, E., & Zhang, D. (2019). A new hybrid convolutional neural network and eXtreme gradient boosting classifier for recognizing handwritten Ethiopian characters. IEEE Access, 8, 17804-17818.

[28] Dhieb, N., Ghazzai, H., Besbes, H., & Massoud, Y. (2019, September). Extreme gradient boosting machine learning algorithm for safe auto insurance operations. In 2019 IEEE international conference on vehicular electronics and safety (ICVES) (pp. 1-5). IEEE.

[29] Taha, A. A., & Malebary, S. J. (2020). An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine. IEEE Access, 8, 25579-25587.

[30] Liang, W., Luo, S., Zhao, G., & Wu, H. (2020). Predicting hard rock pillar stability using GBDT, XGBoost, and LightGBM algorithms. Mathematics, 8(5), 765.

[31] Trabelsi, A., Elouedi, Z., & Lefevre, E. (2019). Decision tree classifiers for evidential attribute values and class labels. Fuzzy Sets and Systems, 366, 46-62.

[32] Charbuty, B., & Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. Journal of Applied Science and Technology Trends, 2(01), 20-28.

[33] Abdallah, I., Ntertimanis, V., Mylonas, C., Tatsis, K., Chatzi, E., Dervilis, N., ... & Eoghan, M. (2018). Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. Safety and Reliability–Safe Societies in a Changing World, 3053-3061.

APPENDIX B MEETING LOG

FACULTY OF COMPUTING & INFORMATICS

TPT3101 Final Year Project (FYP2) Meeting Log
Trimester 2, 2022/23 (Trimester ID:2220)

Meeting Date: **8/4/2023**	**Meeting No.:** **1**
Meeting Mode: **Online meeting with google meet**	
Project ID: **2334**	**Project Type:** **Research-based**
Project Title : **Malware analysis with machine learning**	
Student ID : **1181200964**	**Student Name:** **Tan Carlven**
Student Programme and Specialisation: **Bachelor of Computer Science (Hons.) Cybersecurity**	
Supervisor Name: **Dr. Ng Hu**	**Co-Supervisor Name:** **(if applicable)**
Collaborating Company: **(if applicable)**	**Company Supervisor Name:** **(if applicable)**

1. WORK DONE

[Please write the details of the work done, after the last meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

-

2. WORK TO BE DONE

[Please write the details of the work to be done, before the next meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Deploy the machine learning model as REST API
- Write the Literature Review about REST API

3. PROBLEMS ENCOUNTERED AND SOLUTIONS

[Please write the details of the problems encountered, after the last meeting and provide the solutions / plan for the solutions]

4. COMMENTS (Supervisor / Co-Supervisor / Company Supervisor)

NG HU

…… ……...
Supervisor's Signature Student's Signature

…….. …….
Co-Supervisor's Signature Company Supervisor's Signature
(if applicable) **(if applicable)**

IMPORTANT NOTES TO STUDENTS:

1. Items 1 – 3 are to be completed by the students prior to the meeting. Item 4 is to be completed by the supervisor / co-supervisor / company supervisor.

2. Student must upload the soft copies of the meeting logs in Google Classroom and attach them along with final (FYP2) report.
   Minimum requirement is SIX Meeting Logs (Period: Week 4 to Week 14). Students can have fortnightly meetings with the supervisor.

3. Log sheets provide the basis for evaluating the General Effort (Project Management, Attitude, and Technical Competency) of the student, by the supervisor and for checking the attendance requirement of the student, by the FYP Committee.

   This also provides the student with feedback from the supervisor / co-supervisor / company supervisor on the tasks done and provides the plan for the upcoming tasks. This can provide the motivation for the student to give consistent and efficient effort throughout the period of FYP.

4. Student who fails to meet the minimum requirement (six nos.) of log sheets will not be allowed to submit FYP report.

FACULTY OF COMPUTING & INFORMATICS

TPT3101 Final Year Project (FYP2) Meeting Log
Trimester 2, 2022/23 (Trimester ID:2220)

Meeting Date: **26/4/2023**	**Meeting No.:** **2**
Meeting Mode: **Online meeting with google meet**	
Project ID: **2334**	**Project Type:** **Research-based**
Project Title : **Malware analysis with machine learning**	
Student ID : **1181200964**	**Student Name:** **Tan Carlven**
Student Programme and Specialisation: **Bachelor of Computer Science (Hons.) Cybersecurity**	
Supervisor Name: **Dr. Ng Hu**	**Co-Supervisor Name:** **(if applicable)**
Collaborating Company: **(if applicable)**	**Company Supervisor Name:** **(if applicable)**

1. WORK DONE
[Please write the details of the work done, after the last meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Export the machine learning model
- Done Literature review about REST API
- Run Flask web application

2. WORK TO BE DONE
[Please write the details of the work to be done, before the next meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Create webpage to run malware detection
- Make sure the malware detection can randomly select the data input
- Send the data input to the model API
- Return result and show in the webpage

3. PROBLEMS ENCOUNTERED AND SOLUTIONS
[Please write the details of the problems encountered, after the last meeting and provide the solutions / plan for the solutions]

4. COMMENTS (Supervisor / Co-Supervisor / Company Supervisor)

NG HU

…… ……...

Supervisor's Signature Student's Signature

…….. …….

Co-Supervisor's Signature Company Supervisor's Signature

(if applicable) **(if applicable)**

IMPORTANT NOTES TO STUDENTS:

1. Items 1 – 3 are to be completed by the students prior to the meeting. Item 4 is to be completed by the supervisor / co-supervisor / company supervisor.

2. Student must upload the soft copies of the meeting logs in Google Classroom and attach them along with final (FYP2) report.
   Minimum requirement is SIX Meeting Logs (Period: Week 4 to Week 14). Students can have fortnightly meetings with the supervisor.

3. Log sheets provide the basis for evaluating the General Effort (Project Management, Attitude, and Technical Competency) of the student, by the supervisor and for checking the attendance requirement of the student, by the FYP Committee.

   This also provides the student with feedback from the supervisor / co-supervisor / company supervisor on the tasks done and provides the plan for the upcoming tasks. This can provide the motivation for the student to give consistent and efficient effort throughout the period of FYP.

4. Student who fails to meet the minimum requirement (six nos.) of log sheets will not be allowed to submit FYP report.

FACULTY OF COMPUTING & INFORMATICS

TPT3101 Final Year Project (FYP2) Meeting Log
Trimester 2, 2022/23 (Trimester ID:2220)

Meeting Date: **26/4/2023**	**Meeting No.:** **3**
Meeting Mode: **Online meeting with google meet**	
Project ID: **2334**	**Project Type:** **Research-based**
Project Title : **Malware analysis with machine learning**	
Student ID : **1181200964**	**Student Name:** **Tan Carlven**
Student Programme and Specialisation: **Bachelor of Computer Science (Hons.) Cybersecurity**	
Supervisor Name: **Dr. Ng Hu**	**Co-Supervisor Name:** **(if applicable)**
Collaborating Company: **(if applicable)**	**Company Supervisor Name:** **(if applicable)**

1. WORK DONE
[Please write the details of the work done, after the last meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Web page is created to run malware detection.
- Add the feature where the user can randomly select the data input.
- The data input can be sent to the model API and return and display in the webpage.

2. WORK TO BE DONE
[Please write the details of the work to be done, before the next meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Show what kind of malware is when displaying the result.
- Add the feature selection to choose the malware type.
- Add a button to click for previewing the data.
- Add a button to click for displaying the result prediction.

3. PROBLEMS ENCOUNTERED AND SOLUTIONS
[Please write the details of the problems encountered, after the last meeting and provide the solutions / plan for the solutions]

4. COMMENTS (Supervisor / Co-Supervisor / Company Supervisor)

NG HU

…… ……...

Supervisor's Signature Student's Signature

…….. …….

Co-Supervisor's Signature Company Supervisor's Signature

(if applicable) **(if applicable)**

IMPORTANT NOTES TO STUDENTS:

1. Items 1 – 3 are to be completed by the students prior to the meeting. Item 4 is to be completed by the supervisor / co-supervisor / company supervisor.

2. Student must upload the soft copies of the meeting logs in Google Classroom and attach them along with final (FYP2) report.
   Minimum requirement is SIX Meeting Logs (Period: Week 4 to Week 14). Students can have fortnightly meetings with the supervisor.

3. Log sheets provide the basis for evaluating the General Effort (Project Management, Attitude, and Technical Competency) of the student, by the supervisor and for checking the attendance requirement of the student, by the FYP Committee.

   This also provides the student with feedback from the supervisor / co-supervisor / company supervisor on the tasks done and provides the plan for the upcoming tasks. This can provide the motivation for the student to give consistent and efficient effort throughout the period of FYP.

4. Student who fails to meet the minimum requirement (six nos.) of log sheets will not be allowed to submit FYP report.

FACULTY OF COMPUTING & INFORMATICS

TPT3101 Final Year Project (FYP2) Meeting Log
Trimester 2, 2022/23 (Trimester ID:2220)

Meeting Date: **17/5/2023**	**Meeting No.:** **4**
Meeting Mode: **Online meeting with google meet**	
Project ID: **2334**	**Project Type:** **Research-based**
Project Title : **Malware analysis with machine learning**	
Student ID : **1181200964**	**Student Name:** **Tan Carlven**
Student Programme and Specialisation: **Bachelor of Computer Science (Hons.) Cybersecurity**	
Supervisor Name: **Dr. Ng Hu**	**Co-Supervisor Name:** **(if applicable)**
Collaborating Company: **(if applicable)**	**Company Supervisor Name:** **(if applicable)**

1. WORK DONE
[Please write the details of the work done, after the last meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Show what kind of malware is when displaying the result.
- Add the feature selection to choose the malware type
- Add a button to click for previewing the data
- Add a button to click fir displaying the result prediction

2. WORK TO BE DONE
[Please write the details of the work to be done, before the next meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- For the malware detection, the prediction of result scan update to True
- Try to add one more selection is clean, which is not malware (Optional)
- Add one more random selection to show all the malware.

3. PROBLEMS ENCOUNTERED AND SOLUTIONS
[Please write the details of the problems encountered, after the last meeting and provide the solutions / plan for the solutions]

4. COMMENTS (Supervisor / Co-Supervisor / Company Supervisor)

NG HU

…… ……...

Supervisor's Signature Student's Signature

…….. …….

Co-Supervisor's Signature Company Supervisor's Signature

(if applicable) **(if applicable)**

IMPORTANT NOTES TO STUDENTS:

1. Items 1 – 3 are to be completed by the students prior to the meeting. Item 4 is to be completed by the supervisor / co-supervisor / company supervisor.

2. Student must upload the soft copies of the meeting logs in Google Classroom and attach them along with final (FYP2) report.
   Minimum requirement is SIX Meeting Logs (Period: Week 4 to Week 14). Students can have fortnightly meetings with the supervisor.

3. Log sheets provide the basis for evaluating the General Effort (Project Management, Attitude, and Technical Competency) of the student, by the supervisor and for checking the attendance requirement of the student, by the FYP Committee.

   This also provides the student with feedback from the supervisor / co-supervisor / company supervisor on the tasks done and provides the plan for the upcoming tasks. This can provide the motivation for the student to give consistent and efficient effort throughout the period of FYP.

4. Student who fails to meet the minimum requirement (six nos.) of log sheets will not be allowed to submit FYP report.

FACULTY OF COMPUTING & INFORMATICS

TPT3101 Final Year Project (FYP2) Meeting Log
Trimester 2, 2022/23 (Trimester ID:2220)

Meeting Date: **24/5/2023**	**Meeting No.:** **5**
Meeting Mode: **Online meeting with google meet**	
Project ID: **2334**	**Project Type:** **Research-based**
Project Title : **Malware analysis with machine learning**	
Student ID : **1181200964**	**Student Name:** **Tan Carlven**
Student Programme and Specialisation: **Bachelor of Computer Science (Hons.) Cybersecurity**	
Supervisor Name: **Dr. Ng Hu**	**Co-Supervisor Name:** **(if applicable)**
Collaborating Company: **(if applicable)**	**Company Supervisor Name:** **(if applicable)**

1. WORK DONE
[Please write the details of the work done, after the last meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Prediction of result scan update to true
- The random selection to show all the malware was added

2. WORK TO BE DONE
[Please write the details of the work to be done, before the next meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Add the malware label in the preview there.
- In the result, update the result display to “Malware Type: …”
- Color the word “True” and “False” s

3. PROBLEMS ENCOUNTERED AND SOLUTIONS
[Please write the details of the problems encountered, after the last meeting and provide the solutions / plan for the solutions]

4. COMMENTS (Supervisor / Co-Supervisor / Company Supervisor)

NG HU

…… ……...

Supervisor's Signature Student's Signature

…….. …….

Co-Supervisor's Signature Company Supervisor's Signature

(if applicable) **(if applicable)**

IMPORTANT NOTES TO STUDENTS:

1. Items 1 – 3 are to be completed by the students prior to the meeting. Item 4 is to be completed by the supervisor / co-supervisor / company supervisor.

2. Student must upload the soft copies of the meeting logs in Google Classroom and attach them along with final (FYP2) report.
   Minimum requirement is SIX Meeting Logs (Period: Week 4 to Week 14). Students can have fortnightly meetings with the supervisor.

3. Log sheets provide the basis for evaluating the General Effort (Project Management, Attitude, and Technical Competency) of the student, by the supervisor and for checking the attendance requirement of the student, by the FYP Committee.

   This also provides the student with feedback from the supervisor / co-supervisor / company supervisor on the tasks done and provides the plan for the upcoming tasks. This can provide the motivation for the student to give consistent and efficient effort throughout the period of FYP.

4. Student who fails to meet the minimum requirement (six nos.) of log sheets will not be allowed to submit FYP report.

FACULTY OF COMPUTING & INFORMATICS

TPT3101 Final Year Project (FYP2) Meeting Log
Trimester 2, 2022/23 (Trimester ID:2220)

Meeting Date: **28/6/2023**	**Meeting No.:** **6**
Meeting Mode: **Online meeting with google meet**	
Project ID: **2334**	**Project Type:** **Research-based**
Project Title : **Malware analysis with machine learning**	
Student ID : **1181200964**	**Student Name:** **Tan Carlven**
Student Programme and Specialisation: **Bachelor of Computer Science (Hons.) Cybersecurity**	
Supervisor Name: **Dr. Ng Hu**	**Co-Supervisor Name:** **(if applicable)**
Collaborating Company: **(if applicable)**	**Company Supervisor Name:** **(if applicable)**

1. WORK DONE
[Please write the details of the work done, after the last meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- The label title is updated in preview section
- Malware type is updated in the result display
- The color of prediction “True” and “False” have been added

2. WORK TO BE DONE
[Please write the details of the work to be done, before the next meeting]

Tasks: Implementation / Testing (Application-based projects) or Evaluation of Findings and Research Contribution (Research-based projects) / Commercialisation Proposal (Application-based projects) or Research Paper (Research-based Projects) / Draft Final Report Completion

(Please strike out the tasks, which are not applicable)

Details (in point form):

- Complete the final report
- Write the conference paper
- Prepare the poster.

3. PROBLEMS ENCOUNTERED AND SOLUTIONS
[Please write the details of the problems encountered, after the last meeting and provide the solutions / plan for the solutions]

4. COMMENTS (Supervisor / Co-Supervisor / Company Supervisor)

NG HU

…… ……...

Supervisor's Signature Student's Signature

…….. …….

Co-Supervisor's Signature Company Supervisor's Signature

(if applicable) **(if applicable)**

IMPORTANT NOTES TO STUDENTS:

5. Items 1 – 3 are to be completed by the students prior to the meeting. Item 4 is to be completed by the supervisor / co-supervisor / company supervisor.

6. Student must upload the soft copies of the meeting logs in Google Classroom and attach them along with final (FYP2) report.
   Minimum requirement is SIX Meeting Logs (Period: Week 4 to Week 14). Students can have fortnightly meetings with the supervisor.

7. Log sheets provide the basis for evaluating the General Effort (Project Management, Attitude, and Technical Competency) of the student, by the supervisor and for checking the attendance requirement of the student, by the FYP Committee.

   This also provides the student with feedback from the supervisor / co-supervisor / company supervisor on the tasks done and provides the plan for the upcoming tasks. This can provide the motivation for the student to give consistent and efficient effort throughout the period of FYP.

8. Student who fails to meet the minimum requirement (six nos.) of log sheets will not be allowed to submit FYP report.

APPENDIX C SIMILARITY INDEX REPORT

Malware Analysis with Machine Learning

ORIGINALITY REPORT

17%	14%	11%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	repository.ruforum.org Internet Source	1%
2	addictionresearchchair.com Internet Source	1%
3	inass.org Internet Source	1%
4	vulners.com Internet Source	1%
5	www.researchgate.net Internet Source	1%
6	pergamos.lib.uoa.gr Internet Source	<1%

9	eprints.utem.edu.my Internet Source	<1 %
10	www.mdpi.com Internet Source	<1 %
11	"Hybrid Intelligent Systems", Springer Science and Business Media LLC, 2023 Publication	<1 %
12	Carlos Domenick Morales-Molina, Diego Santamaria-Guerrero, Gabriel Sanchez-Perez, Hector Perez-Meana, Aldo Hernandez-Suarez. "Methodology for Malware Classification using a Random Forest Classifier", 2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), 2018 Publication	<1 %
13	dokumen.pub Internet Source	<1 %
14	hdl.handle.net Internet Source	<1 %

17	pure.ulster.ac.uk Internet Source	<1 %
18	ebin.pub Internet Source	<1 %
19	www.ijraset.com Internet Source	<1 %
20	Asima Sarwar, Abdullah M. Alnajim, Safdar Nawaz Khan Marwat, Salman Ahmed, Saleh Alyahya, Waseem Ullah Khan. "Enhanced Anomaly Detection System for IoT Based on Improved Dynamic SBPSO", Sensors, 2022 Publication	<1 %
21	michelepasqua.github.io Internet Source	<1 %
22	polgan.ac.id Internet Source	<1 %
23	www.igi-global.com Internet Source	<1 %

26	core-cms.prod.aop.cambridge.org Internet Source	<1 %
27	www.springerprofessional.de Internet Source	<1 %
28	"Advanced Intelligent Systems for Sustainable Development (AI2SD'2020)", Springer Science and Business Media LLC, 2022 Publication	<1 %
29	umpir.ump.edu.my Internet Source	<1 %
30	dehesa.unex.es:8443 Internet Source	<1 %
31	eprints.usm.my Internet Source	<1 %
32	oiji.utm.my Internet Source	<1 %
33	www.biorxiv.org Internet Source	<1 %

36	engrxiv.org Internet Source	<1 %
37	etd.lib.metu.edu.tr Internet Source	<1 %
38	ijarcce.com Internet Source	<1 %
39	moam.info Internet Source	<1 %
40	Jefferson G. Martins, Luiz E. S. Oliveira, Daniel Weingaertner, Andersson Barison, Gerlon A. R. Oliveira, Luciano M. Lião. "A database for automatic classification of gender in Araucaria angustifolia plants", Soft Computing, 2021 Publication	<1 %
41	shura.shu.ac.uk Internet Source	<1 %
42	southampton.ac.uk Internet Source	<1 %

45	Thomas Rincy N, Roopam Gupta. "Design and Development of an Efficient Network Intrusion Detection System Using Machine Learning Techniques", Wireless Communications and Mobile Computing, 2021 Publication	<1 %
46	d197for5662m48.cloudfront.net Internet Source	<1 %
47	Ignacio Rodríguez-Rodríguez, María Campo-Valera, José-Víctor Rodríguez, Alberto Frisa-Rubio. "Constrained IoT-Based Machine Learning for Accurate Glycemia Forecasting in Type 1 Diabetes Patients", Sensors, 2023 Publication	<1 %
48	aircconline.com Internet Source	<1 %
49	eprints.unsri.ac.id Internet Source	<1 %

52	dergipark.org.tr Internet Source	<1 %
53	hal.inria.fr Internet Source	<1 %
54	mdpi-res.com Internet Source	<1 %
55	vsip.info Internet Source	<1 %
56	www.hindawi.com Internet Source	<1 %
57	Emanuele Viglianisi, Michael Dallago, Mariano Ceccato. "RESTTESTGEN: Automated Black-Box Testing of RESTful APIs", 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020 Publication	<1 %
58	isif.asia Internet Source	<1 %

61	www.techscience.com Internet Source	<1 %
62	"Advances in Computing and Data Sciences", Springer Science and Business Media LLC, 2021 Publication	<1 %
63	"Proceedings of ICRIC 2019", Springer Science and Business Media LLC, 2020 Publication	<1 %
64	Bemah Ibrahim, Isaac Ahenkorah, Anthony Ewusi, Majeed Fareed. "A novel XRF-based lithological classification in the Tarkwaian paleo placer formation using SMOTE-XGBoost", Journal of Geochemical Exploration, 2022 Publication	<1 %
65	Berny Carrera, Victor Luid Piñol, Judit Bazin Mata, Kwanho Kim. "A machine learning based classification models for plastic recycling using different wavelength range	<1 %

67	Daniel Gibert, Jordi Planes, Carles Mateu, Quan Le. "Fusing feature engineering and deep learning: A case study for malware classification", Expert Systems with Applications, 2022 Publication	<1 %
68	Guizhou Hu. "Accuracy of Prediction Models in the Context of Disease Management", Disease Management, 02/2005 Publication	<1 %
69	Spessard, Gary O.. "Organometallic Chemistry", Oxford University Press Publication	<1 %
70	Wei - Meng Lee. "Python® Machine Learning", Wiley, 2019 Publication	<1 %
71	erepository.uonbi.ac.ke Internet Source	<1 %

74	Daniel Althoff, Santos Henrique Brant Dias, Roberto Filgueiras, Lineu Neiva Rodrigues. "ETo - Brazil: A daily gridded reference evapotranspiration dataset for Brazil (2000 – 2018)", Water Resources Research, 2020 Publication	<1 %
75	repository.ju.edu.et Internet Source	<1 %
76	repository.tudelft.nl Internet Source	<1 %
77	"International Conference on Applications and Techniques in Cyber Security and Intelligence", Springer Science and Business Media LLC, 2018 Publication	<1 %
78	Ji Zhou. "Laboratory Model Test on Foundation Reinforcement by Locating Pile at Both Side of Strip Footing", Lecture Notes in Electrical Engineering, 2012 Publication	<1 %

82	Muhammad Haziq Bin Roslan, Chwen Jen Chen. "Predicting students' performance in English and Mathematics using data mining techniques", Education and Information Technologies, 2022 Publication	<1 %
83	Ooi Nian Chze, Azian Azamimi Abdullah. "COVID-19 mRNA Vaccine Degradation Prediction By Using Deep Learning Algorithms", 2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT), 2022 Publication	<1 %
84	ceur-ws.org Internet Source	<1 %
85	library.nexteinstein.org Internet Source	<1 %
86	pp9c5.gch2020.eu Internet Source	<1 %

91	"Data Wrangling", Wiley, 2023 Publication	<1 %
92	Samaneh Mahdavifar, Ali A. Ghorbani. "Application of deep learning to cybersecurity: A survey", Neurocomputing, 2019 Publication	<1 %
93	arrow.tudublin.ie Internet Source	<1 %
94	deepai.org Internet Source	<1 %
95	www.frontiersin.org Internet Source	<1 %
96	Deepika Verma, Kerstin Bach, Paul Jarle Mork. "External Validation of Prediction Models for Patient-Reported Outcome Measurements collected using the selfBACK Mobile App", International Journal of Medical Informatics, 2022 Publication	<1 %

98	Nikulin, Vladimir. "On the method for data streams aggregation to predict shoppers loyalty", 2015 International Joint Conference on Neural Networks (IJCNN), 2015. Publication	<1 %
99	Zakaria Khoudi, Mourad Nachaoui, Soufiane Lyaqini. "Identifying the contextual factors related to the reading performance of Moroccan fourth-grade students from a Machine Learning-based Approach.", Education and Information Technologies, 2023 Publication	<1 %
100	epe.lac-bac.gc.ca Internet Source	<1 %
101	ir.unimas.my Internet Source	<1 %
102	jurnal.unprimdn.ac.id Internet Source	<1 %

105	publications.polymtl.ca Internet Source	<1 %
106	www.ijisae.org Internet Source	<1 %
107	"Intelligent Technologies and Applications", Springer Science and Business Media LLC, 2020 Publication	<1 %
108	"Proceedings of Third International Conference on Computing, Communications, and Cyber-Security", Springer Science and Business Media LLC, 2023 Publication	<1 %
109	"Security, Trust and Privacy Models, and Architectures in IoT Environments", Springer Science and Business Media LLC, 2023 Publication	<1 %
110	123dok.com Internet Source	<1 %

112	Kamran Shaukat, Suhuai Luo, Vijay Varadharajan, Ibrahim A. Hameed, Min Xu. "A Survey on Machine Learning Techniques for Cyber Security in the Last Decade", IEEE Access, 2020 Publication	<1 %
113	Moh Abdul Latief, Titin Siswantining, Alhadi Bustamam, Devvi Sarwinda. "A Comparative Performance Evaluation of Random Forest Feature Selection on Classification of Hepatocellular Carcinoma Gene Expression Data", 2019 3rd International Conference on Informatics and Computational Sciences (ICICoS), 2019 Publication	<1 %
114	Noah S. Diffenbaugh. "Vegetation sensitivity to global anthropogenic carbon dioxide emissions in a topographically complex region", Global Biogeochemical Cycles, 2003 Publication	<1 %

117	Umar Islam, Hathal Salamah Alwageed, Malik Muhammad Umer Farooq, Inayat Khan, Fuad A. Awwad, Ijaz Ali, Mohamed R. Abonazel. "Investigating the Effectiveness of Novel Support Vector Neural Network for Anomaly Detection in Digital Forensics Data", Sensors, 2023 Publication	<1 %
118	atrium.lib.uoguelph.ca Internet Source	<1 %
119	digitalcommons.library.tmc.edu Internet Source	<1 %
120	formative.jmir.org Internet Source	<1 %
121	lup.lub.lu.se Internet Source	<1 %
122	nmbu.brage.unit.no Internet Source	<1 %

125	repository.universitasbumigora.ac.id Internet Source	<1 %
126	techscience.com Internet Source	<1 %
127	www.isteams.net Internet Source	<1 %
128	"Computational Science and Its Applications – ICCSA 2020", Springer Science and Business Media LLC, 2020 Publication	<1 %
129	"Smart Trends in Information Technology and Computer Communications", Springer Science and Business Media LLC, 2016 Publication	<1 %
130	123dok.net Internet Source	<1 %

133	Gourinath Banda, Chaitanya Krishna Bommakanti, Harsh Mohan. "One IoT: an IoT protocol and framework for OEMs to make IoT-enabled devices forward compatible", Journal of Reliable Intelligent Environments, 2016 Publication	<1 %
134	Kolawole Ogunsina, Ilias Bilionis, Daniel DeLaurentis. "Exploratory data analysis for airline disruption management", Machine Learning with Applications, 2021 Publication	<1 %
135	Ruiqi Liu, Zhiting Xu, Jinyan Teng, Xiangchun Pan et al. " Evaluation of six machine learning classification algorithms in pig breed identification using array data ", Animal Genetics, 2022 Publication	<1 %
136	S Geetha, Yusuf Mohammed Khan, Rohan Sujay, Sai Pavan Yoganand, Rohan B.	<1 %

137	Suraj Kumar Bhagat, Tran Minh Tung, Zaher Mundher Yaseen. "Heavy Metal Contamination Prediction using Ensemble Model: Case Study of Bay Sedimentation, Australia", Journal of Hazardous Materials, 2020 Publication	<1 %
138	bioresources.cnr.ncsu.edu Internet Source	<1 %
139	cdn.manaraa.com Internet Source	<1 %
140	cps-vo.org Internet Source	<1 %
141	dspace.unimap.edu.my Internet Source	<1 %
142	ecommons.cornell.edu Internet Source	<1 %
143	habr.com Internet Source	<1 %

149	tojqi.net Internet Source	<1 %
150	uis.brage.unit.no Internet Source	<1 %
151	www.internationaljournalssrg.org Internet Source	<1 %
152	Chang, C.P.. "Defect prevention in software processes: An action-based approach", The Journal of Systems & Software, 200704 Publication	<1 %
153	HongFang Zhou, JiaWei Zhang, YueQing Zhou, XiaoJie Guo, YiMing Ma. "A feature selection algorithm of decision tree based on feature weight", Expert Systems with Applications, 2020 Publication	<1 %
154	Ying-xu. Lai, Zeng-hui. Liu. "Chapter 26 Unknown Malicious Identification", Springer Science and Business Media LLC, 2009	<1 %