Optimizing the Performance of Machine Learning Algorithms in Detecting Malicious Files using Hybrid Models

Master's Thesis

A S M Farhan Al Haque

Aalborg University
Electronics and IT

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Optimizing the Performance of Machine Learning Algorithms in Detecting Malicious Files using Hybrid Models.

Theme:
Malicious Files Detection

Project Period:
Summer Semester 2023

Participant(s):
A S M Farhan Al Haque

Supervisor(s):
Jens Myrup Pedersen
Ashutosh Dhar Dwivedi

Copies: 1

Page Numbers: 71

Date of Completion:
August 3, 2023

Abstract:

The exfiltration of digital systems using malcrafted files has been an evolving issue for the last two decades. Malicious actors deploy diverse payloads through files that posses potentiality of evading possible detection mechanism and cause alarming harm. Leveraging the universal file format, support of advanced features like JavaScript, and inclusion of additional files make Portable Document File (PDF) and Portable Executable (PE) an apparent choice for to be weaponized by the hackers. This project explores the performance of different branches of machine learning approaches in malware detection. Two dataset each for PDF and PE files are selected after an extensive review of the existing research. At first, Gaussian Naive Bayes (GNB) and Logistic Regression (LR) algorithms are applied from the classical branch. Random Forest (RF) from bagging and Adaptive Boosting (AdaBoost) from boosting are selected from the ensemble classification. Next, three variants of Artificial Neural Network (ANN) are deployed to improve the detection. Finally, a novel hybrid approach integrating ANN and ensemble techniques is proposed for both PDF and PE files and discovered that the hybrid model outperforms all the previous models. The hybrid model combining ANN with AdaBoost achieve an accuracy of 99.51% and F1-score of 99.53% for malware detection in PDF. Similarly, 98.45% of accuracy and 98.95% of F1-score for PE files.

Contents

List of Tables viii

List of Figures ix

Preface xi

1 Introduction 1
- 1.1 How malicious actors target files? . . . 2
- 1.2 Major Attack Vectors using Files . . . 3
- 1.3 Machine Learning in Malware Detection . . . 3
- 1.4 Contribution to the domain . . . 5
- 1.5 Deliminations . . . 6
- 1.6 Structure of the Thesis . . . 7

2 Background 8
- 2.1 Malware Detection vs. Analysis . . . 8
- 2.2 Portable Document File (PDF) . . . 9
- 2.3 Implement PDF Exploits . . . 10
  - 2.3.1 JavaScript Code based Attacks . . . 10
  - 2.3.2 File Embedding Attacks . . . 11
  - 2.3.3 Form submission and URI attacks . . . 11
  - 2.3.4 ActionScript attacks . . . 11
- 2.4 Portable Executable (PE) File . . . 11
- 2.5 Exploitation of PE File . . . 13
- 2.6 Machine Learning (ML) Algorithms . . . 14
  - 2.6.1 Classical Approaches . . . 14
  - 2.6.2 Ensemble Techniques: bagging and boosting . . . 16
  - 2.6.3 Artificial Neural Network (ANN) . . . 17
- 2.7 Performance Evaluation . . . 19
  - 2.7.1 Accuracy . . . 20
  - 2.7.2 Precision . . . 20
  - 2.7.3 Recall . . . 21

2.7.4 F1-score 21
2.7.5 Receiver Operating Characteristic (ROC) Curve 21
2.7.6 Area Under the ROC Curve (AUC) 22

3 Literature Review 23
3.1 Search Engines and Keywords 23
3.2 Detection Techniques of Malicious PDF 23
3.3 Detection Techniques of Malicious PE 26
3.4 Summary 28

4 Design and Implementation 29
4.1 Methodology 29
4.2 Phase-1 (Project Initialization) 31
4.2.1 Data Collection 31
4.2.2 System Configurations 32
4.2.3 Experimental Setup 32
4.3 Phase-2 (Application of Machine Learning) 33
4.3.1 Data Preprocessing 34
4.3.2 Feature Selection & Dimensionality Reduction 37
4.3.3 Data Splitting 37
4.3.4 Implementation of Classical Machine Learning 38
4.3.5 Implementation of Ensemble Techniques (Bagging and Boosting) 40
4.3.6 Implementation of ANN 42
4.4 Phase-3 (Model Evaluation) 45
4.5 The Proposed Hybrid Solution 46
4.5.1 Architecture 46
4.5.2 Implementation 46

5 Result and Analysis 49
5.1 Result Analysis of Classical Machine Learning 49
5.2 Result Analysis of Ensemble Techniques 52
5.3 Result Analysis of ANN 52
5.4 Result Analysis of the Hybrid Solution 55
5.5 Result Analysis of PE Dataset 56
5.6 Summary 60

6 Conclusion 61

Bibliography 63

A PDF Dataset 66
A.1 Correlation Matrix of the Features 67

B PE Dataset **69**

List of Tables

1 Table with acronyms and abbreviations used throughout the report . . . xii

4.1 Libraries used for the project . . . 32
4.2 System configuration . . . 32
4.3 Number of samples in the datasets . . . 38

5.1 Evaluation metrics of classical ML approaches . . . 50
5.2 Evaluation metrics of ensemble techniques . . . 50
5.3 Evaluation metrics of the three ANN models . . . 50
5.4 Evaluation metrics of the ANN-ensemble hybrid models . . . 50
5.5 Results for evaluation metrics on PE dataset . . . 56
5.6 Comparison table with other related works . . . 59

A.1 Name and Type of the features in the dataset . . . 66
A.1 Name and Type of the features in the dataset . . . 67

List of Figures

1.1 Malicious Actors compromise systems with files 2
1.2 Overview of Intrusion Detection System or Prevention System (IDS/IPS) 4

2.1 Structure of PDF [22] 9
2.2 DOS Header in PE file 12
2.3 DOS stub header in PE file 12
2.4 File header in PE file 13
2.5 Sigmoid function for LR [21] 15
2.6 Overview of RF [35] 16
2.7 Overview of the AdaBoost model [24] 17
2.8 Overview of the ANN model [34] 18
2.9 ROC curve 22

4.1 Phase-1 of methodology of the solution 29
4.2 Phase-2 of methodology of the solution 30
4.3 Phase-3 of methodology of the solution 30
4.4 Overview of the solution 33
4.5 Datasets used for experiments in this project 34
4.6 Architectures of the ANN models 43
4.7 Architecture of the proposed solution 47
4.8 Architecture of the feature extractor using ANN 48

5.1 Confusion Matrix and ROC curves of classical ML approaches . . . 51
5.2 Confusion Matrix and ROC curve of Ensemble Techniques 51
5.3 Confusion Matrix of the three ANN models 51
5.4 Training and validation accuracy of the three ANN models 53
5.5 Training and validation loss of the three ANN models 53
5.6 ROC curve for the three ANN models 53
5.7 Confusion matrix and ROC curves of the two hybrid models 55
5.8 Feature extraction using ANN for hybrid models 56
5.9 ANN model architecture for implemented PE dataset 57

5.10 Confusion Matrix of different apporaches on PE dataset 58
5.11 ROC curves for PE dataset 58

A.1 Correlation of features 68

B.1 PE header fields in the dataset [36] 70
B.2 PE sections and fields in the dataset [36] 71

Preface

Being a student of Computer Science, I have always had interest in the field of Machine Learning. I have always desired to apply Machine Learning to a challenging problem like malware detection.

This thesis explores the performance of different branches of Machine Learning algorithms. Finally, hybrid solutions integrating Artificial Neural Network and ensemble techniques like bagging and boosting are proposed that has outperformed all the previous individual traditional Machine Learning techniques.

It is a matter of joy for me that, the results generated by the proposed models are outstanding. It has been an exciting journey to indulge into a challenging project like this and finish with what I planned in the beginning.

I would take the opportunity to thank my supervisors for supporting and inspiring me through out this project.

Aalborg University, August 3, 2023

Farhan

A S M Farhan Al Haque
<ahaque21@student.aau.dk>

Table 1: Table with acronyms and abbreviations used throughout the report

Acronym or abbreviation	**Definition**
DPI	Deep Packet Inspection
IDS/IPS	Intrusion Detection System/ Prevention System
PDF	Portable Document Format
GNB	Gaussian Naive Bayes
LR	Logistic Regression
PE	Portable Executable
ML	Machine Learning
RF	Random Forest
AdaBoost	Adaptive Boosting
ANN	Artificial Neural Networks
CVE	Common Vulnerabilities and Exposures
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
ROC	Receiver Operating Characteristic
AUC	Area Under the ROC curve
IEEE	Institute of Electrical and Electronics Engineers
TPR	True Positive Rate
FPR	False Positive Rate
AUB	Aalborg University Database
MLP	Multi-layer Perceptron Neural Network
SVM	Support Vector Machine
FLF	Function Length Frequency
DT	Decision Tree
KNN	K-nearest Neighbour
DBN	Deep Belief Network
CIC	Canadian Institute for Cybersecurity
GridSearchCV	Grid Search Cross-validation
DLL	Dynamic Link Library
API	Application Programming Interface

Chapter 1

Introduction

The ever-increasing amount of cyber attacks and the threats relating to it has been one of the most worrisome concerns over the last two decades [16]. With the immensely growing numbers, the level of sophistication and the magnitude of the threat landscape have been soaring as well. Owing to the technological evolution, mankind has embraced digitization ubiquitously. This emergence of computer technology in every aspect of modern civilization has expanded the potential of threats caused by the malicious actors. There are contrasting target and motives for cybercriminals for disrupting different systems by exploiting vulnerabilities. From financial gain to sensitive data theft or espionage to social engineering, it is difficult to identify the prominent [18]. There has been many different types of cyber attacks recorded: malware, man-in-the-middle, phishing, denial-of-service, ransomware [20]. A severe type of cyber attack is carried out by injecting malicious files into a system which is capable of stealing or manipulating sensitive data, corrupting other files, even taking control over the entire system the consequence of which is immense. A study in 2022 brings out that the two most common malware file types are Portable Executable (PE) and Portable Document Format (PDF).

PDF files being a ubiquitous format that is used worldwide for preparing and sharing documents. It is a trustworthy format in every sector from academic to industry. Another key feature is it's platform independence. PDF is accessible through diverse applications from any operating system. The support for the rich media contents, hyperlinks, and JavaScript is also one of the prime reason to make PDF a viable choice for the hackers spreading malware. For JavaScript, PDF files can contain forms accommodating check boxes, user inputs or buttons that enhance it's features than a normal document. Similarly, PE files also have certain features to draw the attentions of the attackers. It is a file format for executables, DLLs, object codes for Windows operating system. PE files have file headers allocating important information like size or control sequences which if abused, flow of the

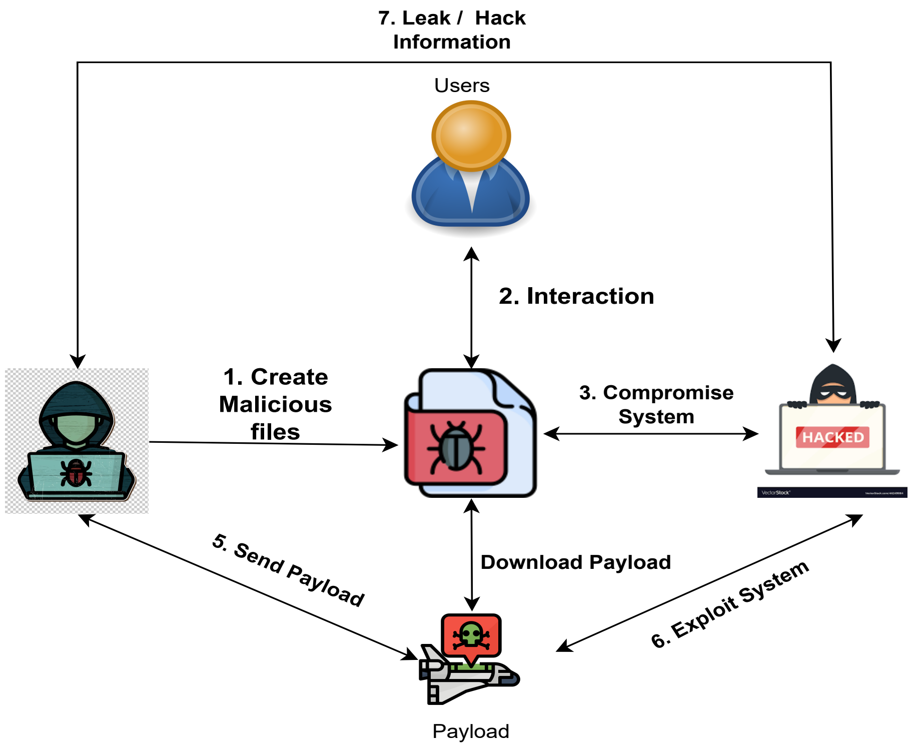


Figure 1.1: Malicious Actors compromise systems with files

program can be altered. PE files contain binary codes as well as metadata of the operating system. Attackers can also exploit exception handling of PE files. PE files can load external DLLs at runtime that be exploited by the attacker to alter legitimate DLL file with the malicious one.

1.1 How malicious actors target files?

Figure 1.1 represents a general flow and sequence how malicious actors can utilize files to compromise systems. Files of different categories are deployed as attack vectors leveraging potential of high risk. Files are exploited as convenient carriers to proliferate viruses or ransomware in the target systems. Files containing malicious contents appear as legitimate documents or executables. Unaware of the incoming threats users open those documents or run the file leading to compromise the system. Hackers can exploit a wide option of vulnerabilities in any software or even in operating system by weaponizing files. The attack vector using files can be categorized into four major type explained in the next section. Out of numerous numbers few vulnerabilities worth mentioning that had previously exploited by maliciously crafted PDFs are: CVE-2021-27045, CVE-2021-27036, CVE-2021-27037, CVE-2018-4993, CVE-2018-16011. Similarly, examples of recent vulnerabilities exploited by PE files: CVE-2023-26245, CVE-2023-26246, CVE-2022-28884, CVE-2022-0026 [3].

1.2 Major Attack Vectors using Files

Files can be used to deploy the following attacks:

- `Exploit:` Threat actors can craft malicious files by embedding JavaScript code in the file. When those files are run by the victim the malicious code will start running to exploit any vulnerability in the software to eventually compromise the system. For examples, exploiting vulnerabilities of the reader of the PDF files or the software processing the malformed PE file.
- `Phishing:` In these attacks, the files itself does not perform any malicious activities rather it is used as a bait to manipulate the victim. If the victim clicks on a specific button or a link in the PDF, the hacker can make certain actions like downloading and running external files in the system.
- `Privilege Escalation:` This attack is a consequence of both the previous two attacks. The hacker can get elevated access to perform unauthorized actions on sensitive data.
- `Trojan and Backdoor attacks:` Files specifically PE files can be disguised as legitimate ones in the form of crack version of any software or even pirated version. If executed on the system, the malware is installed and hackers can immediately take remote access on the infected device.

1.3 Machine Learning in Malware Detection

With the help of modern technologies, the advancements are remarkable in the cyber security landscape including digital forensics, intrusion detection and prevention system (IDS / IPS). Traditional detection techniques often fall short detecting the rapidly evolving threats and their continuously changing executions. Machine learning (ML) is a great inclusion in the field of IDS being adapt to detect new and obscured threats by analyzing behaviour of malwares and triggering the monitoring system in case of any abnormalities or deviation from the baseline. Other techniques like signature-based detection is not effective for detecting new exploits. Sandbox analysis of malwares is popular detection technique. In recent times, the malwares are capable of identifying whether they are in a real environment or in sandbox. Malwares are adaptive and tend to change their usual execution to evade the analysis process. On the contrary, ML offers dynamic and adaptive approach

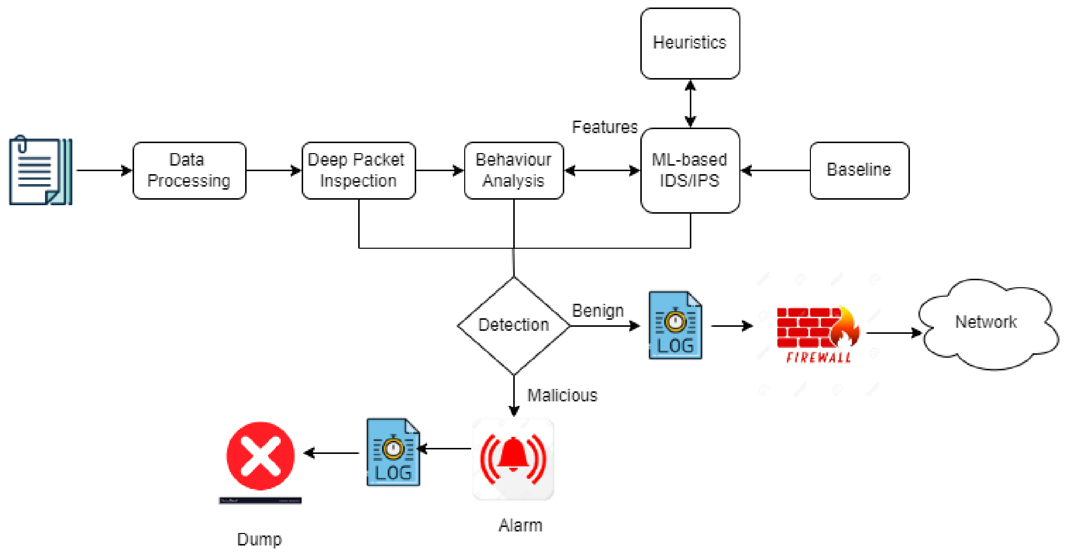


Figure 1.2: Overview of Intrusion Detection System or Prevention System (IDS/IPS)

that learns from it's past experiences and execute the knowledge acquired from training in detecting newer threats. It has enabled the security professionals to always be a step ahead. However, there will always be room for further improvements. Optimizing ML algorithms to improve computation overhead, reducing false positives are the areas requiring continuous research and refinement.

Figure 1.2 represents a potential IDS/IPS framework which inspects any incoming file from the external network. In modern times, a single mechanism is not able to provide the necessary security from potential harm. With trained and expert human interventions, diverse detection techniques must be applied together as a complete unit to keep the system safe. Specially in case of zero-day attacks, due to their unknown characteristics, it is a formidable task for a single detection mechanism like machine learning based detection or deep packet inspection. To prevent newer and newer threats, collective deployment of appropriate measures are recommended. In the above framework, any incoming file is passed through fundamental processing which is then fed to deep packet inspection (DPI). DPI looks into the contents of the packets, analyze the included protocols, identify the services used by the traffic. The next unit is behavior analysis, where any anomalies are identified through monitoring.

Although DPI and behavior analysis are both very good tools, they do have certain operational limitations. One such case worth mentioning here is encrypted malicious files. DPI cannot retrieve encrypted contents and does not turn out to be a useful option. Behavior analysis can process some metadata such as file name, file size, timestamp, source and destination of the encrypted file, sudden spike in the frequency of encrypted files etc. These metadata can also be valuable input feature for machine learning based detectors. Machine learning based detection techniques perform by detecting comparing the network traffic to the dynamic and adaptive baseline. Both DPI and behaviour analyzer can extract decent features for the machine learning detectors to work on. There is a significant impact due to inclusion of heuristics in detecting malwares for providing supplementary domain knowledge. Heuristics can contribute to better feature engineering, model optimization or selection, appropriate labeling of data, updating the baseline and even threshold adjustments. If any malicious indication in any file is detected, the IDS/IPS should detect the malware, trigger the alarm, log the behaviour to enhance the knowledge-base for future use and then dump the file. And in cases of benign files, still logging is required for future usage and then allow access through the firewall to the network.

1.4 Contribution to the domain

This paper aims at performing a malware detection in files using the best machine learning techniques. We are considering both PDF and PE files for carrying out experiments in this research. For this, we have used two datasets, one for each type of file which is discussed thoroughly in 4.1. There are plenty of effective machine learning algorithms. In this project, two classical machine learning approaches, bagging, and boosting techniques have been selected and compared them with more complicated Artificial Neural Network (ANN) techniques. Furthermore, we deployed hybrid models combining ANN with ensemble techniques to find out if further improvement of the results is possible or not. The goal is to understand and examine which algorithms are capable of learning the features of malicious and benign files the most accurately and precisely. As the effectiveness of the domain is critical even the robustness of the different models have been examined carefully with different possible scales. Summarizing, we focus to achieve the following problem statement:

Apply machine learning to analyze static features to detect malwares in PDF and PE files.

In addition to that, there are following sub-questions:

i. Evaluate the performance of Artificial Neural Network and Ensemble technique like bagging and boosting over the classical approaches.

ii. Examining the robustness and performance of each model using precise performance metrics

iii. Propose and formulate a novel hybrid approach comprised of ANN and ensemble techniques.

iv. Examine whether the previous results can be further optimized and improved by this hybrid models.

1.5 Deliminations

As stated earlier, the scope of the project is to investigate how well different ML techniques perform in detecting malicious files. In this project, only PDF and PE, two types of files are examined. The ML algorithms will be trained and tested against these two types of benign and malicious samples. This project does not concern the various other file types.

The Ml techniques deployed in this project will not classify malwares into their families. This project is restrained in detecting the existence of malicious files on the basis of useful features. The algorithms used in this project does not analyze any malicious codes in files.

Referring to the features used in this project are static features only. This project does not generate and analyze dynamic features during runtime. The project deploys fixed set of features to understand the impact of ML techniques in generalizing the data to classify the malwares effectively.

Furthermore, scrutinizing the malware detector with evasion attacks is regarded out of scope for this project. Finally, the goal is not to develop any end product i.e. antivirus software for detecting malwares. Rather, the focus is to explore different ML techniques like classical approaches, neural network, deep learning technique like CNN, or even transfer learning with pre-trained network as feasible solutions for the purpose of malware detection.

1.6 Structure of the Thesis

In this chapter, the motivation and importance of this project is highlighted in brief. The remaining part of the paper is organized to give a top-down approach to the problem. The following chapaters are as follows:

- **Chapter 2** provides a background study on the topic and the formats of PDF and PE files with a basic overview of the effectiveness machine learning based detection in this domain.
- **Chapter 3** serves a review of the related work in this field and also the search keywords to find those academic papers from different sources.
- **Chapter 4** illustrates the experiment thoroughly. It describes the tools used, the experimental setup, stages of data pre-processing, parameter selection of different machine learning algorithms and finally deploying the models in details.
- **Chapter 5** deliberates the results and evaluates performance of different models.
- Finally **chapter 6** concludes the paper with discussion and any further possible improvement.

Chapter 2

Background

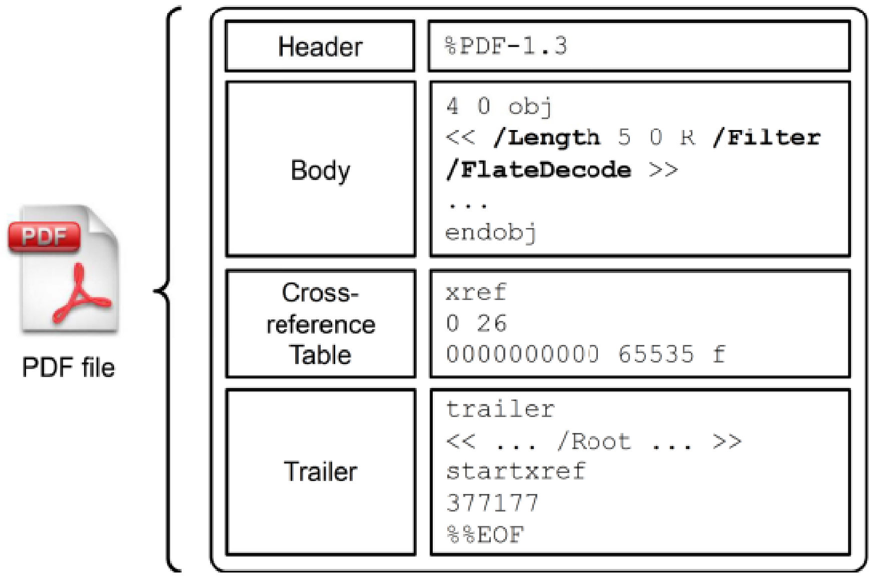
This chapter will reflect a detailed background that serves as a foundation for this project. Starting with the explanation of malware analysis and detection describing how the two terms vary. Next, the structure of both the file formats PDF and PE is discussed in details. It is vital to understand the concept how an attacker can take advantage of these files by maneuvering it's components which is stated in this chapter. Following, the machine learning (ML) algorithms deployed in this project are introduced and explained. Finally concluding by looking at how a ML-based detector might aid in the detection of malicious files in a system.

2.1 Malware Detection vs. Analysis

There should be no doubt regarding the necessity of malware detection and prevention. The paramount incurring loss caused by cyber attacks over time has been an issue of concerns for security professionals. Detection and analysis of malwares is a crucial research domain. Though detection and analysis, the two phrases sound similar but they refer to distinct approaches and outcomes.

Malware detection is the process of identifying and removing any malicious content from the system [12]. There are a handful of approaches for carrying out this detection. Two of the most popular approaches are signature-based and machine learning approach. `Signature-based` detection compares the files with a database of malicious signatures to detect the existence of any malware in the file. `Machine learning` algorithms are trained to learn from datasets having features of both malicious and benign files.

Once the malware is detected, it is vital to study the malware to gain a comprehensive picture of it's behaviours and patterns [8]. Analysis can be both static and dynamic. The fundamental difference between the two process is static analysis



```
%PDF-1.3
```

```
4 0 obj
<< /Length 5 0 R /Filter
/FlateDecode >>
...
endobj
```

```
xref
0 26
0000000000 65535 f
```

```
trailer
<< ... /Root ... >>
startxref
377177
%%EOF
```

Figure 2.1: Structure of PDF [22]

does not execute the file whereas the dynamic malware analysis involves proper execution of the malwares in a controlled environment (sandbox). Both detection and analysis are critical providing insights to identify the new malware strains and develop security defense mechanism.

2.2 Portable Document File (PDF)

The structure of a PDF file consists of the following four components illustrated in Figure 2.1:

- **Header:** This is usually a line at the beginning of the file that identifies the file as a PDF. It is a sequence of default characters providing the version number in the form "%PDF-1.N", where N can be a digit from 0 to 7 [15].
- **Body:** This section encapsulates different objects for defining pages, inclusion of any images or annotations, fonts, colors, bookmarks objects and so on.
- **Cross-reference (Xref) Table:** It is a table which is designed to store each object's locations and information with random access to the objects. It uses a byte offset format to map the object. This table makes it efficient for the PDF reader to retrieve an object without searching the entire file [2].

- **Trailer:** The final component is the trailer accommodating key information like the address of the Xref table, the total number of objects used in the file, and also the root object. Using these information the PDF reader can read from the start till the end of the file efficiently. The section initiate with the keyword *trailer* and terminates with *%%EOF*

2.3 Implement PDF Exploits

Selvaraj et el. [27] mentioned that the most suitable three channels to spread malicious PDFs are through targeted attacks, bulk e-mailing, and drive-by downloads. **Targeted attacks** is carried out when a PDF is specially crafted to attack an individual or an organization. This is executed by thorough research on the specific target which can also be referred to as social engineering. This attack is sophisticated and the distributed PDF will have the potential of high trust of the target to be opened. **Bulk e-mailing** has become popular as we are habituated to receiving PDF files via e-mails. In this way, malicious PDFs are designed to send to a huge number of users. The content covers interesting topics or recent incidents to entice the users to open the document. With the embedded feature the recent browsers are capable of opening PDF document which can be a reason for **drive-by downloads**. The attack is executed under complete stealth with zero knowledge of the users. The PDF contains codes that downloads crafted executables by the attacker from the internet.

PDF documents can be exploited in many ways for malicious intent. The most prominent attacks using PDFs are as follows:

- JavasScript code based attack
- File embedding attacks
- Form submission and URL attacks
- ActionScript attacks

2.3.1 JavaScript Code based Attacks

The highest volume of PDF attacks are carried out by exploiting JavaScript code [32]. PDF files support JavaScript code to serve purposes like form validation, any required action like changing the content as requested, even restricting user controls, displaying multimedia content. These code can also reside in different places in the file which facilitates the malicious purpose. The primary indicator of the presence of JavaScript code in a PDF file is the /JS keyword in the dictionaries. These dictionaries can be also be placed in filtered stream. So the code is not anyhow visible in plain text which evades any detection mechanism that solely depend on this keyword searching. Attackers can leverage the full potential of this scripting language to infect the users' system by exploiting vulnerabilities. The normal flow of execution can be halted by attacks like heap spraying or buffer overflow attacks. The code can be also be abused to download malicious files or executables from the internet, opening a malicious website without the knowledge of the user. Windows system is recorded to be compromised with multiple escalation of privilege attacks including CVE-2018-8166, CVE-2018-8164, CVE-2018-8124, CVE-2018-8120 [3].

2.3.2 File Embedding Attacks

In this type of attack, a malicious file is embedded inside a legitimate PDF file. So, the attacker can make use of the possible vulnerabilities. The files are embedded in such a way so that it tries to evade any sort of detection. These kind of embedded file can also be used as an obfuscation tool for other attacks. Again with the help of JavaScript code or PDF command features like `OpenAction` or `Launch` from the `ActionClass`, the corrupted embedded file can be opened immediately when the legitimate PDF file is opened. A recent vulnerability identified in 2022 is CVE-2022-40181, which was capable of executing arbitrary code, reading and altering other files, or even a carrying out a denial of service attack.

2.3.3 Form submission and URI attacks

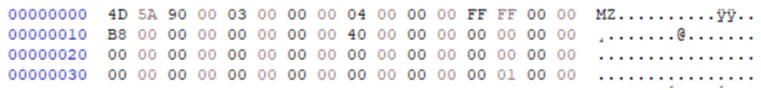
In 2013, Valentin et el. [11] illustrated this technique which attackers can use. Adobe Reader can submit PDF form to any server with the feature `SubmitForm` command. If the URL is from a remote web server the responses are stored in `AppData` directory which is pops up automatically in the default web browser. An attacker can exploit the user's web browser by compromising this pop up.

2.3.4 ActionScript attacks

ActionScript attacks are not as frequent as JavaScript attacks. These attacks exploit the Flash software that Adobe Reader used. The goal was to perform an arbitrary code execution. But in December 2020 Adobe's Flash had been deprecated and these attacks became less popular.

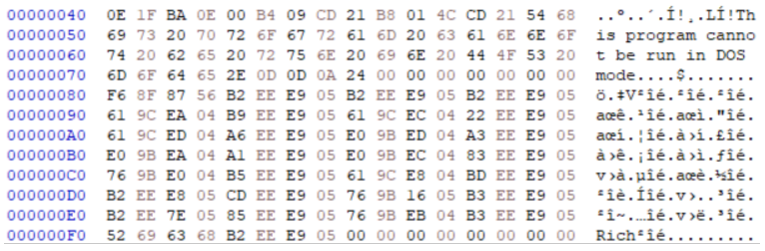
2.4 Portable Executable (PE) File

Portable Executable is a universal file format for Windows Operating System (OS). It helps to manage programs and resources in the Windows platform. It holds information necessary for the OS which is used to load any program into memory



```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ................
```

Figure 2.2: DOS Header in PE file



```
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  F6 8F 87 56 B2 EE E9 05 B2 EE E9 05 B2 EE E9 05  ö.‡V²îé.²îé.²îé.
00000090  61 9C EA 04 B9 EE E9 05 61 9C EC 04 22 EE E9 05  aœê.¹îé.aœì."îé.
000000A0  61 9C ED 04 A6 EE E9 05 E0 9B ED 04 A3 EE E9 05  aœí.¦îé.à›í.£îé.
000000B0  E0 9B EA 04 A1 EE E9 05 E0 9B EC 04 83 EE E9 05  à›ê.¡îé.à›ì.ƒîé.
000000C0  76 9B E0 04 B5 EE E9 05 61 9C E8 04 BD EE E9 05  v›à.µîé.aœè.½îé.
000000D0  B2 EE E8 05 CD EE E9 05 76 9B 16 05 B3 EE E9 05  ²îè.Íîé.v›..³îé.
000000E0  B2 EE 7E 05 85 EE E9 05 76 9B EB 04 B3 EE E9 05  ²î~.…îé.v›ë.³îé.
000000F0  52 69 63 68 B2 EE E9 05 00 00 00 00 00 00 00 00  Rich²îé.........
```

Figure 2.3: DOS stub header in PE file

and execute it. Not only code, Dynamic Link Libraries (DLL), any sort of drivers or kernel modules are stored in this format also. Upon execution of any program this PE file format is used to get all the data along with metadata for efficient resource allocations. PE files are used in tasks related to security by verifying and validating any alteration of data ensuing the integrity of the system. The structure of PE file is as follows:

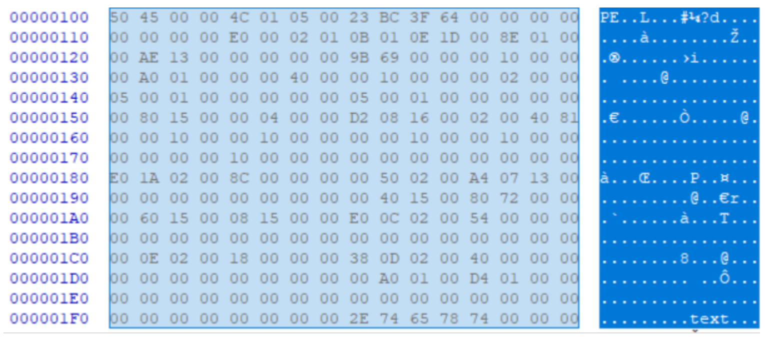
- **DOS Header:** The first 64 bytes or 4 rows of the file. Starting with `4d 5A` represents `MZ`, one of the developers of MS-DOS. The last 4 bytes represents the location of the PE header section which is `00 01 00 00`. Figure 2.2 shows the DOS header in a PE file. The file is generated by a Hex Editor using the `.exe` file of a browser.

- **DOS Stub:** The stub is a code for a message that is shown in the display `"This program cannot be run in DOS mode"` in case of non-compatibility with the OS. If any program executed in a WIN-32 environment which was not originally built for that environment, then this message will be printed out. Figure 2.3 shows the DOS stub message in the generated PE file.

- **PE Signature:** In the location `00 01 00 00`, the PE signature starts with a 4 byte signature `50 45 00 00`. Figure 2.4 shows the file header section where the first 4 bytes are PE signature.

- **File Header:** This section contains important parsing information regarding the file such as number of section, size of the optional header, time date stamp, pointer to the symbol table, characteristics etc.

- **Section Header Table:** The next component is the section headers which represent information of each section such as name of the section, size, address.



```
00000100  50 45 00 00 4C 01 05 00 23 BC 3F 64 00 00 00 00  PE..L...#¼?d....
00000110  00 00 00 00 E0 00 02 01 0B 01 0E 1D 00 8E 01 00  ....à.........Ž..
00000120  00 AE 13 00 00 00 00 00 9B 69 00 00 00 10 00 00  .®......›i......
00000130  00 A0 01 00 00 00 40 00 00 10 00 00 00 02 00 00  . ....@.........
00000140  05 00 01 00 00 00 00 00 05 00 01 00 00 00 00 00  ................
00000150  00 80 15 00 00 04 00 00 D2 08 16 00 02 00 40 81  .€......Ò.....@.
00000160  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
00000170  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00000180  E0 1A 02 00 8C 00 00 00 00 50 02 00 A4 07 13 00  à...Œ....P..¤...
00000190  00 00 00 00 00 00 00 00 00 40 15 00 80 72 00 00  .........@..€r..
000001A0  00 60 15 00 08 15 00 00 E0 0C 02 00 54 00 00 00  .`......à...T...
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 0E 02 00 18 00 00 00 38 0D 02 00 40 00 00 00  ........8...@...
000001D0  00 00 00 00 00 00 00 00 00 A0 01 00 D4 01 00 00  ......... ..Ô...
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001F0  00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00  .........text...
```

Figure 2.4: File header in PE file

There can be different sections in the file i.e. text, data, relocation table, resource container, debug information.

- **Sections:** This component holds the different section that is mapped in the section header table. There is not any particular way of organizing these sections. `.text` section usually contains the code and `.data` section stores all the variables used in the program. The address and size information import table is stored in `.idata` and for export table there is `.edata`. Import tables are used to track the APIs required to import by the current executable. And, export table is for the other executables to use this file.

2.5 Exploitation of PE File

The discussion above makes it obvious that there are clear scope where PE files can be altered on purpose to make it malicious. The popularity of Windows OS and the universal acceptance of the PE file make it an obvious choice. Another aspect of being a target of the hackers is the self-sufficient nature of the file type. Attacks can be launched solely by the PE files without the support of any additional data with it.

There are a plenty of mechanisms to exploit PE file format. Attackers are able to inject or append malicious files in a PE file to execute unauthorized commands to fulfill evil purposes. Components of OS like device drivers or libraries can be manipulated by applying rootkit techniques. Altering the metadata for example changing the file extension or timestamps can make it look benign. DLL hijacking is recorded to be a demonstrated technique where a malicious DLL replaces a legitimate DLL for arbitrary code execution. Alteration of resources like image or any supporting file is also possible.

2.6 Machine Learning (ML) Algorithms

This project entails a brief discussion of different ML algorithms that will be implemented and deployed in the next chapters. This section discusses classical ML approaches, ensemble techniques like bagging and boosting, and deep learning technique.

2.6.1 Classical Approaches

There is a long list of classical ML approaches, from where Gaussian Naive Bayes (GNB) and Logistic Regression (LR) are selected. The description of these two algorithms are as follows:

Gaussian Naive Bayes (GNB)

GNB is a simple and fast classification algorithm based on the probabilistic approach using Bayes' theorem and gaussian distribution. The algorithm best applied to independent features given the target class but still operates considerably on the dependant features. It is a good choice for high dimensional datasets. The steps associated with GNB are discussed below [26]:

Step 1: At first, calculate the probability of each class using the equation 2.1,

$$P(C_j = \frac{frequency(C_j)}{TotalNumber} \tag{2.1}$$

Step 2: The algorithm calculates the mean and standard deviation using each of the features for each class using the equation 2.2 and 2.3, where n is the total number of samples in the training set and x_i represents each sample in the dataset.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.2}$$

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2.3}$$

Step 3: To find the class of a new data in the test set, the probability distribution for each class is determined using the equation 2.4

$$P(X|C) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.4}$$

Step 4: Calculate the conditional probability of X at class C using the equation 2.5:

$$P(X|C) = \sum_{i=1}^{n} P(X_i|C) \tag{2.5}$$

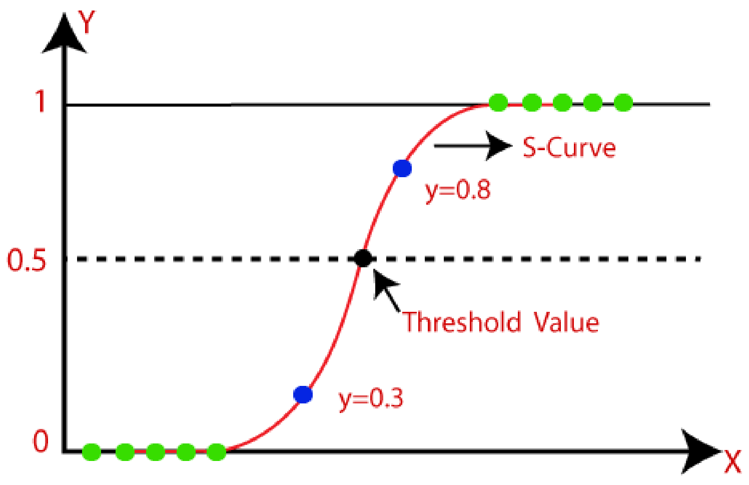


Figure 2.5: Sigmoid function for LR [21]

Step 5: Finally Calculate the posterior probability of X using the equation 2.6:

$$P(C|X) = P(X|C) \cdot P(C) \tag{2.6}$$

Step 6: Assign class labels by selecting the highest maximum probability value for all the classes.

Logistic Regression (LR)

LR is a popular ML technique that is widely used for classification tasks. The goal is to calculate the probability of a sample to be in one of classes. The algorithm works great where the features of the dataset are categorical and dependant. LR employs an 'S' shaped logistic function, also known as sigmoid function that predicts the inputs into two classes of values 0 and 1. The sigmoid function is represented in figure 2.5 and The mathematical equation of the sigmoid function is given in 2.7. The equation of LR for classification is given in 2.8.

$$P(y = 1|X) = \frac{1}{1 + e^{-(X \cdot \beta)}} \tag{2.7}$$

$$S(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

In the equation 2.8,

- *P(y=1 | X)* represents the probability of input X to be in class '1'.
- β denotes the parameters learned through training and also represent the decision boundary or threshold value above which the input falls into class '1' otherwise class '0'.

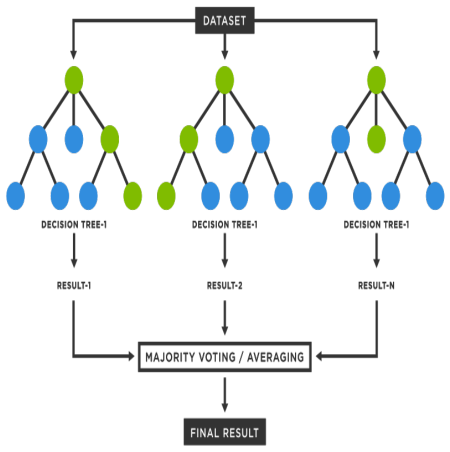


Figure 2.6: Overview of RF [35]

- The sigmoid function shown in 2.7 maps the linear combination $X \cdot \beta$ to the probability value between 0 and 1.

2.6.2 Ensemble Techniques: bagging and boosting

Ensemble techniques are a special ML branch of supervised learning that employs the collective strengths of different models to improve the performance. The limitations of different models can be mitigated by integrating them together. Ensemble techniques achieve better results than those individual models separately. Ensemble techniques have various mechanisms, of them bagging and boosting are two very popular ones. Bagging come from 'bootstrap aggregating', meaning making a subset of the training data by replacing. In bagging, a number of instances of the same base model is trained using the subset of the training data. Then the average of all the models' prediction is calculated for the final output. Boosting, builds weak models repeatedly where each model optimizes the flaws of the previous ones. The prominent difference between these two techniques is, in bagging the models are trained parallel, whereas boosting train the weak models sequentially. In this section, Random Forest (RF) from the bagging and Adaptive Boosting (AdaBoost) from boosting will be discussed.

Random Forest (RF)

The major phases of RF is demonstrated in figure 2.6. RF employs several decision trees on various subsets of training data. The fact that RF also chooses a random subset of characteristics for the various decision trees is crucial to ensure

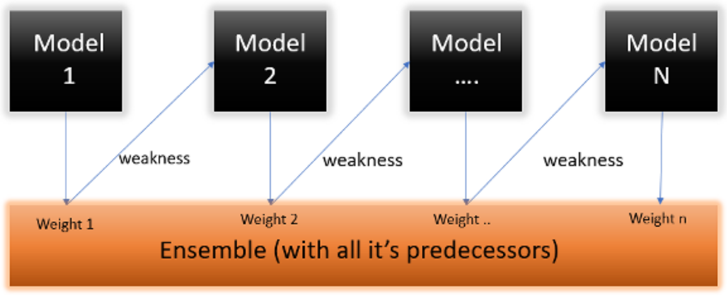


Figure 2.7: Overview of the AdaBoost model [24]

randomness to generalize diverse patterns. For each subset of the training set, a decision tree is deployed for the selected features. Next, the final model can make prediction on the unseen data and the performance is calculated. For classification problems, RF uses majority voting to determine the classes. For RF, multiple hyperparameters like the number of trees, depth of each trees, number of features etc. play critical roles in optimizing the results.

Adaptive Boosting(Adaboost)

Figure 2.7 visualizes the above description. AdaBoost also employs decision trees as the base learner but not independently like RF. The trees are built in parallel, each optimizing the result of the previous block. AdaBoost initializes weights on training data, initially equal weights for all the data. Then, the weak learner processes the data and the weighted error is calculated considering all the misclassified instances. Next, the weights are adjusted and the misclassified data are updated with higher weights and the correctly classified data with lower weights. The reason is that the misclassified data will be prioritized in the next iterations. Each classifier tree gains a weight based on its' classification result and integrated to the ensemble. The greater the tree's weight, the greater the contribution it provides. After training, the ensemble of trees are tested with unseen data. The weighted majority voting determines the class of each sample.

2.6.3 Artificial Neural Network (ANN)

ANN is the core of deep learning which is a popular branch of ML. As the name suggests, ANN is inspired by the nerve cells function in human brain. It is constructed by interconnected nodes or neuron in layers. The first layer is called input layer where the training data is fed for training the neurons to learn the patterns of the data. The intermediate layer is called the hidden layers which can have different combinations of layers in it. Each node has individual weight and threshold. The

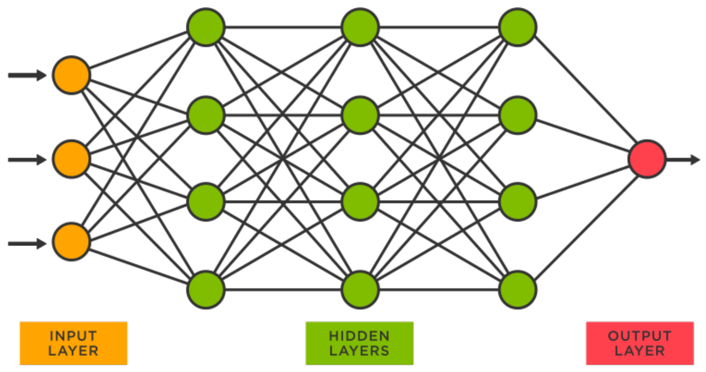


Figure 2.8: Overview of the ANN model [34]

nodes of all the layers are interconnected by weighted edges as well. The hidden layers process the result and send it to the final output layer. The structure is depicted in figure 2.8 Each node has an activation function that determines whether or not to activate it as an output. This activation function brings in non-linearity that makes the model capable of learning pattern and classifying instances. Few examples of the activation function are ReLU, sigmoid function, softmax, tanh, exponential etc. The choice of the activation function depends on the task.

The flow of the processed data is of two types: forward propagation and backward propagation. The flow of processed data from the input layer to the output layer is called forward propagation. If activated, each neuron perform calculation and generates outputs. And, the output of the final layer is the actual prediction made by the ANN model. In backward propagation, the model adjusts and fine-tunes the weights of neurons to minimize the error. This technique propagate the error of the output layer back to the input layer for adjustments.

Hyperparameters

It is important to mention about the hyperparameters for ANN which determines the efficiency of the model to a great detail. Few of the key hyperparameters are listed below:

- **Number of hidden layers:** It decides how deep the network will be constructed. Very deep network does not essentially ensure better performance. The number of layers is dependant on the expected task to perform, the complexity, and the size of the input data, For the model to produce the optimum results, the architecture of the ANN must be optimized.
- **Number of nodes in each layer:** Just like the number of layers, number of nodes play the same role for construction of the ANN. Each layer must

have optimal number of nodes. Hyperparameter tuning can be applied that employs grid search technique to discover the best architecture for the ANN model.

- **Activation function for the nodes:** As stated above, Each node has an activation function that controls whether or not the node will be active.
- **Output activation function:** Depending on the type of classification the ANN model performs. For regression problems linear activation function should be selected. Sigmoid function should be selected for binary classification problem. and softmax activation function is the choice for multi-class classification problem.
- **Learning rate:** It decides how quickly the ANN model is learning the training data. The selection of learning rate is critical, as larger learning rate can make the model learn fast but end up in a sub-optimal solution. Learning rate directly effects the training time as well. The default learning rate in Keras is 0.001 and in TensorFlow is 0.01.
- **Batch size:** It is the number of instances from the training set taken at a time for each epoch. It affects the convergence, training time, and memory usage as well. Less training and convergence time is required for larger batches. Conversely, large batch size require more memory to store the intermediate results.
- **Number of Epochs:** It decides the number ANN model processes the training data. It directly influences the training time, higher the number longer the training time is. However, large number of epochs enables the ANN model to learn the training data too well that might cause overfitting. It is a case when the model learns the training data specially well but perform poorly on the unseen test data.
- **Regularization:** Overfitting is an issue to handle while constructing the architecture of ANN model. To the rescue, regularization can be applied to make the model robust to overfitting. Few of such techniques are L1 or L2 regularization, dropout, early stopping, data augmentation etc.

2.7 Performance Evaluation

The different branches ML is explored in the previous section. Once implemented it is vital to examine the performance of individual algorithms. There exist particular metrics for evaluating algorithms to realize the strength and weaknesses of the model. For better understanding, there are 4 key terms that need to be defined:

- **True Positive (TP):** Model predicting a sample in Class A, where it actually belongs to Class A.
- **False Positive (FP):** Model predicting a sample in Class B, where it actually belongs to Class A.
- **True Negative (TN):** Model prdicting a sample in Class B, where it actually belongs to Class B.
- **False Negative (FN):** Model prdicting a sample in Class A, where it actually belongs to Class B.

2.7.1 Accuracy

Accuracy is the most frequently used metric for performance evaluation. It reflects the correctness of classification of any model by calculating the number of correct predictions over the total number of prediction as shown in equation 2.9. Another representation of accuracy can be devised applying TP, TN, FP, and FN as shown in equation 2.10. However, accuracy can be deceiving sometime as it does not always portrays the actual scenario. Specially in cases of the imbalanced dataset the number of samples of one class is way more than the other. In real world scenarios, it is difficult to find balanced datasets. For evaluating the robustness of the model, metrics like precision, recall, F1-score are taken into consideration.

$$\text{Accuracy} = \frac{\text{No. of Correct Predictions}}{\text{Total No. of Predictions}} \tag{2.9}$$

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+FP+FN+TN}} \tag{2.10}$$

2.7.2 Precision

Precision is a useful metric when the dataset is imbalanced. It measures ratio of the number of correct positive predictions to the total positive predictions as shown in equation 2.11. A machine learning model is expected to generate high precision value that points out the model does not predict the negative samples as positive. Precision is crucial in scenarios where FP might impact heavily.

$$\text{Precision} = \frac{\text{TP}}{\text{FP + TP}} \tag{2.11}$$

2.7.3 Recall

Recall is calculated as the ratio of hte number of correct positive predictions to the total number of actual positive samples as shown in equation 2.12. Recall is calculated in scenarios where FN can impact the results.

$$\text{Recall} = \frac{\text{TP}}{\text{TP + FN}} \tag{2.12}$$

2.7.4 F1-score

F1-score is the measure that takes both precision and recall into consideration and provides balance between the two metrics putting equal weights. F1-score solely can depict a clear overview of the model performance. It is calculated as the harmonic mean of precision and recall as shown in equation **??**.

$$\begin{aligned}\text{F1-score} &= \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \\ &= \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision + Recall}}\end{aligned} \tag{2.13}$$

2.7.5 Receiver Operating Characteristic (ROC) Curve

The curve is generated by tracing out sensitivity against (1-specificity) for each value of cut-off as shown in Figure 2.9. True positive rate (TPR) and false positive rate (FPR) can be calculated from the equations 2.14 and 2.16. ROC curves can be extensively use to select the threshold value for a classifier.

$$\text{TPR} = \text{Sensitivity} = \frac{\text{TP}}{\text{TP + FN}} \tag{2.14}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN + FP}} \tag{2.15}$$

$$\text{FPR} = 1 - \text{Specificity} = \frac{\text{FP}}{\text{FP + TN}} \tag{2.16}$$

When PFR = TPR, the model randomly classifies the samples and the curve becomes a diagonal straight line at 45-degree. That means the model does not learn to generalize the data and classify to their actual classes. The closer the ROC curve is to this diagonal line, the less accurate the model is. On the other hand, the more this curve shifts to the top left corner of the graph suggests better results.

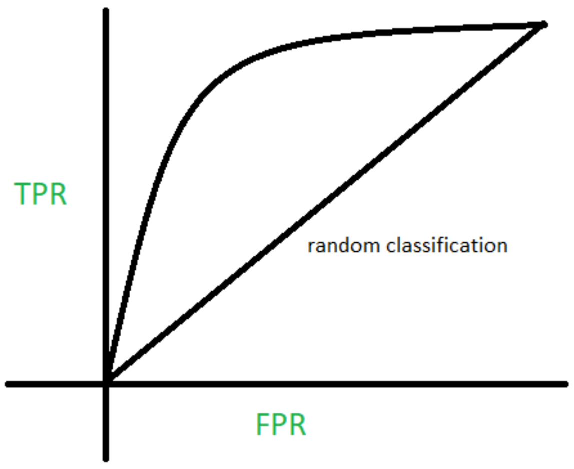


Figure 2.9: ROC curve

2.7.6 Area Under the ROC Curve (AUC)

This is another metric to understand the performance of any classifier. It represents the classification quality of the model. This is calculated by the total two-dimensional area under the ROC curve. Therefore, the better the model performs, the higher the AUC value leaps.

Chapter 3

Literature Review

This chapter lays a foundation of the project reflecting the existing academic works proposed. Beginning with the search terms and different sources used to find the articles in later sections. A brief review on the methodologies implemented by different authors on detection of malware on PDF and PE file is reflected sequentially.

3.1 Search Engines and Keywords

To have a good understanding of the mentioned domain "malware detection", it was necessary to find relevant academic articles. To serve the purpose, few of the search engines that was found very helpful are as follows: Google Scholar database, Institute of Electrical and Electronics Engineers (IEEE) Xplore, ResearchGate, Aalborg University Database (AUB), Scopus etc. There were many variety of contrasting keywords was used as search terms to find the relevant papers. Few of them are as follows:

- "Malware detection using Machine Leaning"
- "Malware detection on PDF"
- "Detection of malicious PE files"
- "Deep Learning for malware detection"
- "Malware family classification"

3.2 Detection Techniques of Malicious PDF

Jason Zhang et el. [37] proposed a novel approach named MLP_{df} to detect malwares in PDF documents. MLP indicates multi-layer perceptron neural network that uses stochastic gradient descent search to update. The author deployed two datasets generating 48 features from both benign and malicious PDF documents. The MLP detector yields a true positive rate (TPR) of 95.12% and is compared with the result of 8 prominent anti virus (AV) scanners. The MLP-based detector could outperform the other AV scanners substantially. The author however mentions the significance of the manual feature engineering to improve the learning of the model.

Laskov and Nedim et el. [19] devised a model named `PJScan` to detect the presence of malicious JavaScript code in PDF. The features of the documents are analyzed with static lexical process to identify benign or malicious that the authors claim to incur less run-time overhead compared to dynamic analysis. Support Vector Machine (SVM) is used for classification that has generated a decent detection rate of 85%. The authors could identify a shortcoming of the model which is high false positive rate (FPR) of 16-17%. A major issue with the model is that it only concerns JavaScript-based exploitation. The model is not adapt to perform well against different exploitation mechanisms.

Baptista and Shiaeles et el. [1] proposed a neural network based unsupervised malware detection technique for different categories of files such as `.exe, .doc, .txt, .pdf, .htm`. The model converts the binary data of the file into 2-dimensional binary visualizing images using an online tool named `binvis.io` [5]. The binary representation for optimal clustering of data is ensured by Hilbert space-filling curves. The authors used a balanced dataset with 4000 samples containing 2000 benign and 2000 malicious files obtained from `VirusShare` website. The model attained the average detection rate for all the file types is 74% approx. with 12% false positives. The highest accuracy achieved was 94% for PDF files. On the other hand, the least accuracy generated was 60.9% for .htm file. Even so a limitation of this experiment was the sample size of the dataset for different classes. The model is expected to perform better with more samples. On the bright side, the neural network was lightweight enough to be trained for 10k iterations within 15 seconds.

Optimal feature extraction process is key to improved classification results. In [13], the authors have combined function length frequency (FLF) [28] and printable string information (PSI) [29] techniques to extract features. This process generated better results than previous individual results. Classical ML techniques and ensemble techniques were deployed on the extracted feature set. Almost 1400 malware and 161 clean samples were used to train the classifiers. The result of the experiment was carried out by the WEKA tool. Weighted average results were calculated using *k-fold* cross validation. Decision Tree (DT) with AdaboostM1 classifier yields the overall maximum accuracy of over 98%.

Gavrilut and Cimpoesu et el. [10] built a framework that explores the capability of different versions of perceptrons and SVM. The authors deployed two datasets to train and test the classifiers. The training dataset had 2,73,133 benign and 27,475 malware files. The test dataset consisted of 6,522 clean and 11,605 malicious files. The best version of the perceptron produced an accuracy of over 88% for the test dataset. With SVM the best accuracy obtained was over 94% on the test set. However, the time consumption of the models is high and the authors mentioned to speed up the training time optimizing the models in future.

Maiorca and Giacinto et el. [23] presented a malicious PDF detecting tool named `PDF Malware Slayer (PDFMS)`. The tool was trained and tested with benign and malicious files both. Benign files were collected by downloading random PDF files from Yahoo search engine based on a dictionary of a benign source. The dataset provided by the Contagio team [4] was also used for collecting both types of files. For training, a balanced 6,000 benign and 6000 malicious PDF files, in total 12,000 files were used. The test set had 9,146 files containing 3,989 benign and 5157 malicious files. Experiments were carried out with various classical ML and ensemble algorithms However, Random Forest (RF) with 18 trees generated the best detection accuracy of 99% on the training set. The authors compared the result with other antiviruses of that time-frame. Although the authors mentioned few weaknesses of the tool. The architecture of the tool was based on keyword frequency only. There was no analysis of any code which makes the tool vulnerable to certain attacks like injection of a certain malicious keyword with high frequency. The author considered the necessity of the parser vital.

Torres and Santos et el. [30] presented a framework specifically for cloud computing services that analyze PDF documents online. The authors collected the benign PDF files from various trusted sources. The training was developed with care following certain criteria like not having any suspicious call to APIs or any obfuscation. The benign files were checked against antivirus engines to detect any malwares in it. The malicious samples were downloaded from sources email honeypots, public or private malware repositories like `malwr.com, Contagio`. The framework was trained with 995 training samples, 217 validation samples, and 500 test samples having 28 features. Three algorithms deployed in the framework were SVM, RF and Multilayer Perceptron (MLP). Out of the three, MLP performs the best with detection accuracy of 96%. However, the parsing process of the framework was quite complex that spawned high processing time.

Corum and Jenkins et el. [6] designed a PDF detector using image processing which converted PDFs to gray-scale images to extract features. The dataset used by the mentioned authors is Contagio PDF malware dataset. The dataset was evenly balanced with a equal 6000 benign and malicious samples The authors applied the optimized feature extraction techniques like texture features and keypoint descriptors. For classification, three algorithms were deployed RF, DT, and k-nearest neighbour (KNN). Out of all, RF turned out to be the best performing algorithm with F1-score of over 99%. The authors compared the result of this detector with some popular antivirus scanners where it holds significant values. Although the authors pointed out few possible further improvements by extracting more image feature and applying feature selection to make the model more optimized.

Maryam and Princy et el. [14] presented an computationally expensive stacking learning technique for the detection of malware in PDF files. The authors have accumulated the previously used extensively used contagio dataset and virustotal. And they extracted 28 features to build a new dataset. The authors applied four different models including RF, SVM, MLP, and AdaBoost as the base learner. And selected any of the LR, K-NN, DT as the meta learner. This heavily structured stacking algorithm could perform well on the dataset and achieved 99% of accuracy. The authors compared their results with the previous work done and issued better result. One of the prime weakness of this paper is the computation required for exetectuing 7 algorithms sequentially. The authors did not calculate processing time in the paper.

Fettaya and Mansour et el. [9] devised an ensemble of CNN algorithm that process on the byte level eliminating the necessity of extracting features. The size of the dataset used for this experiment was 18296 malicious PDF files and 70551 benign files. the malicious files were downloaded from VirusTotal [33]. The authors deployed three different models each having unique architecture. The best performing algorithm yielded a substantial detection rate of 94%. The authors compared the results of the model with 52 available antivirus and it held 7^{th} position.

3.3 Detection Techniques of Malicious PE

. So far different ML based approaches to detect malicious PDF file is discussed. This section will highlight some previous research carried out in detecting malicious executables specially PE.

At the outset a classical ML based approach is presented by Radwan et el. [25] to detect the malware in PE file. The approach used static analysis of both raw and derived features. The author employed a dataset of 5184 samples in which 2683 malware and 2501 benign. The dataset included 55 features out of which 28 features are fed to the classifiers. He derived 46 new features to improve the accuracy of classification. 10-fold cross validation was introduced with a train-test split 70 : 30 ratio. The author deployed seven ML algorithms within which Random Forest (RF) performs the best with 99% accuracy.

Uppal and Saha et el. [31] developed an approach to detect malicious application programming interface (API call sequence. Classical ML techniques were used in this experiment to analyze the behavior of the malwares. A novel feature selection technique is used applying N grams and odd ratio selection. The dataset contained 120 malicious PE files and 150 benign files. SVM produced the best classification results with over 98%accuracy whereas DT turns out to be least effective among all the algorithms.

Another ML based solution for classification of malwares in PE files is devised by Kumar and Kuppusamy et el. [17]. The authors used 10-fold cross validation to improve the accuracy. To minimize the computation overhead, top *N* features was investigated successfully. The authors built a dataset with 68 features out of which 28 are raw and 28 boolean features are derived. The dataset had 2722 malwares and 2488 benign samples in it. Random Forest again turns out to be the winner outperforming others with a 98.4% accuracy. Although the model fell short comparing with the previous work mentioned in the paper. For optimizing the feature set the authors investigated with top 5,10,15,20 and 25 features. RF shows the most promising result for top20 features than the classifier applied.

DeepSign elicited from a deep learning method for automatic signature generation and classification of the malwares utilizing those signatures. This method was designed by David and Netanyahu et el. [7]. They named the signature generator as deep belief network (DBN) which performed dynamic analysis of the malware behavior. The dataset used for this experiment contains six popular categories of malwares named `Zeus`, `Carberp`, `SpyEye`, `Cidox`, `Andromeda`, `DarkComet` and 300 samples for each one these categories that makes it a total of 1800 samples. Out of them, 1200 samples were used training and the rest 600 for testing the classification performance. The malware executables were run in the Cuckoo sandbox which accumulated necessary data and stored in a text file as JSON format. This file was converted to binary format and fed to the DBN which generated the signatures. For classification, SVM, KNN, and ANN were applied where SVM yielding a satisfactory 96.4% of accuracy. But ANN optimized the result outperforming the others with an accuracy of 98.6

3.4 Summary

The relevant academic research papers in the domain of malware detection in PDF and PE files was investigated and discussed through. The approaches used in these papers has turned out to be extremely helpful to get an overview of the recent works and their methodology in this field previously. The superiority of an ensemble technique RF and deep learning techniques are clearly visible the papers stated above. This overview constructs the foundation of the structure of this thesis explained in the next chapter in section 4.1. Collection of data is a vital step in machine learning projects like this. This chapter has helped to gather sources and details of the different available datasets. Furthermore, a deep understanding of different performance evaluating methods and application of those methods were developed that has been benefiting in the implementation of the experiment.

Chapter 4

Design and Implementation

This chapter begins with the illustration of the methodology to discuss each phase of the project. Next, a comprehensive technical specification of the experimental design is described. Following, the architecture of ML algorithms with different parameters are specified. Then the implementation of each phases of the project is demonstrated with necessary code snippets and explanations.

4.1 Methodology

The methodology of this project is broken down into three phases and depicted in figures 4.1, 4.2, and 4.3. In **phase-1**, the project was initiated by establishing an overview of the domain which was served by performing a thorough literature study in chapter 3. Putting in the overview, potential sources of data are listed. Two datasets are selected one each containing features of malicious and benign PDF and PE files. The raw data is then pre-processed to feed to the ML based malware detectors for training and testing purposes. Then, the system configurations and the environment of the experiment is setup. The design of the whole project is finalized by making sure the data is usable for the experimentation, the system is up and running with all the necessary tools and frameworks.

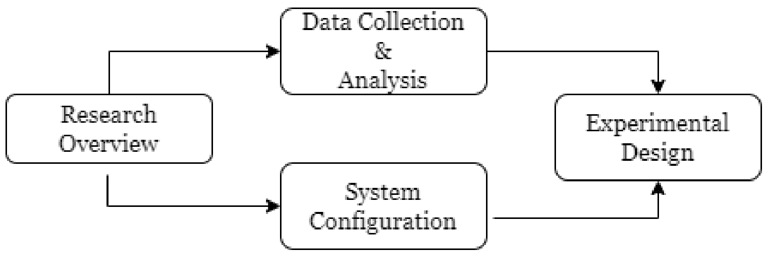


Figure 4.1: Phase-1 of methodology of the solution

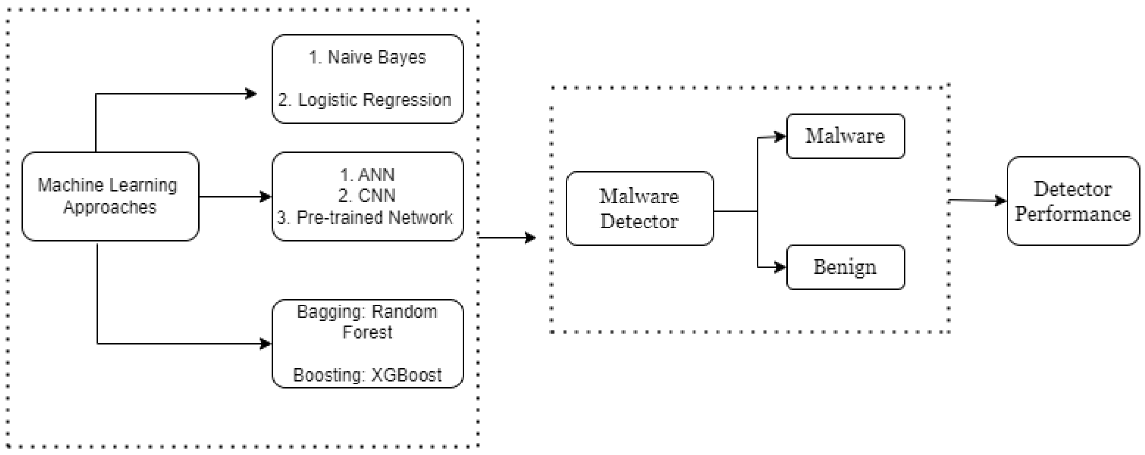


Figure 4.2: Phase-2 of methodology of the solution

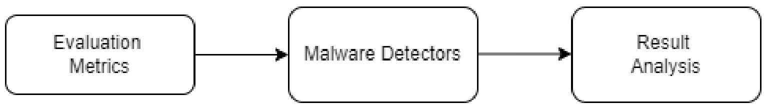


Figure 4.3: Phase-3 of methodology of the solution

Phase-2 selects the ML algorithms for the malware detector to classify the samples into malicious or benign. This project showcases the performance of different approaches of ML in the domain of malware detection. For example classical approaches: Logistic Regression, Gaussian Naive Bayes, deep learning techniques: ANN, ensemble techniques: boosting (AdaBoost) and bagging (Random Forest) are deployed. The models need to be configured well with different parameters to get the best classification results.

Next, a vital part of the experiment is to evaluate the classification performance of the detectors which is appointed in **phase-3**. Specific evaluation metrics are applied that are best-suited for this experiment details of which are explained in 2.7. And finally, the result is presented in a suitable approach and discussed in chapter 5. Next, the three phases are discussed in depth with the descriptions, specifications, implementations with the code snippets of each step.

4.2 Phase-1 (Project Initialization)

4.2.1 Data Collection

For data collection, benign and malicious samples of PDF and PE files are used to train, validate, and test the malware detectors. Upon pursuing the literature review, the sources of the datasets have been discovered. To be mentioned, one of the most widely recognized dataset is **Contagio** (2013) [1]. Another, sought-after source for malwares is **VirusTotal**[2]. Users can upload files in their service, then these files scanned against a number of antivirus. Then both the files and the scanned results are then available in the database. Another similar database containing old and new malicious samples is **VirusShare** [3].

The most recent published dataset concerning features of malicious PDF had been published in 2022 by the Canadian Institute for Cybersecurity (CIC) operating at University of New Brunswick in Fredericton [4]. The team has used the above mentioned sources to develop this dataset. They have accumulated 11,173 from Contagio and 20,000 from VirusTotal for malicious samples exhibiting evasive characteristics. For benign samples they have collected 9,109 samples from Contagio. Upon processing and removing the duplicate samples after merging the two datasets, the released version has a total 10.025 samples, 5,557 malicious and 4,468 benign. The dataset contains 32 features within which 12 are general features and the rest are structural. The list of the features are mentioned in table A.1 in appendix A.

The dataset utilized for detecting malicious PE files contains malwares from five families including: Spyware, Ransomware , Downloader, Backdoor, and the rest are Generic [36]. The dataset contains static features extracted with Python's *pefile library*. To prepare the dataset, a total of 79 features are generated analyzing imported Dynamic Link Libraries (DLLs), API calls, PE headers, and PE sections. The name of different PE header features and PE section features are given in figure B.1 andB.2 in appendix B. The dataset contains a total of 19,611 samples comprising of 14,599 malwares and 5,012 benign files. Both the datasets, and all the codes implemented in this project can be found in this link [5].

[1]https://https://contagiodump.blogspot.com/
[2]https://www.virustotal.com/gui/home/upload
[3]https://virusshare.com/
[4]https://www.unb.ca/cic/
[5]https://github.com/ASMFH/Thesis_Project

4.2.2 System Configurations

This section highlights system configurations for the project environment including tools, frameworks and libraries are mentioned. Jupyter Notebook has been used as environment for this project. It enables programmers to integrate code and documentation at the same place and has great result visualization interface. It supports the programming language Python for writing the codes of machine learning. Python 3.10.6 version is used in this project. Table 4.1 enlists the name and the purpose of the libraries used.

Table 4.1: Libraries used for the project

Python Library	**Purpose**
pandas	Data manipulation, use data structures like DataFrames
Scikit-Learn	Implement machine learning algorithms
Keras	Implement deep learning techniqeus
Matplotlib	Data visualization by creating graphs
seaborn	Creating heatmap to visualize correlation of the features

4.2.3 Experimental Setup

Table 4.2 displays the configuration of the system. The computation time to train heavily depends on the specifications of the hardware. In this case, the hardware used in this project is very moderate with no fancy processors and GPU used. As a consequence, the model is expected to produce better computation time when executed in a better hardware.

Table 4.2: System configuration

Environment	Oracle VirtualBox 7.0
Host System	Windows 11
Virtual OS	Ubuntu 23.04 (Lunar Lobster)
Kernel	6.2 Linux kernel
Processor Model	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
No. of Cores	4
Memory	8 GB
Graphics Card	NVIDIA GeForce MX330
Cores	384 CUDA Cores
Memory	2 GB GDDR5

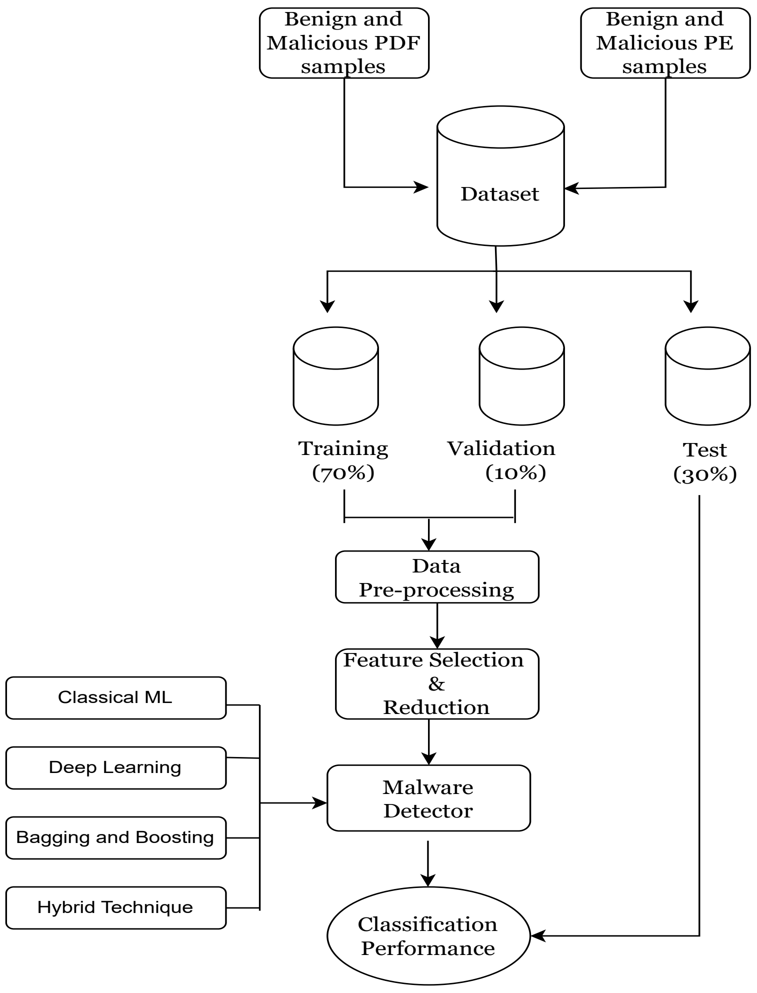
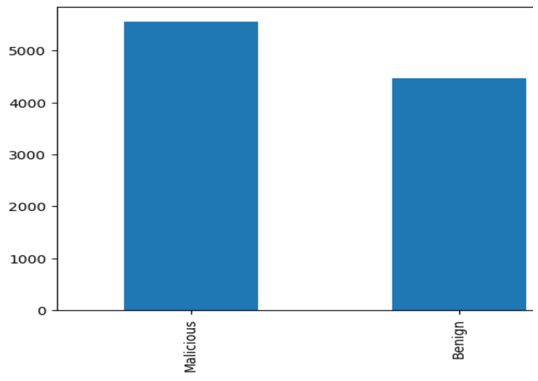


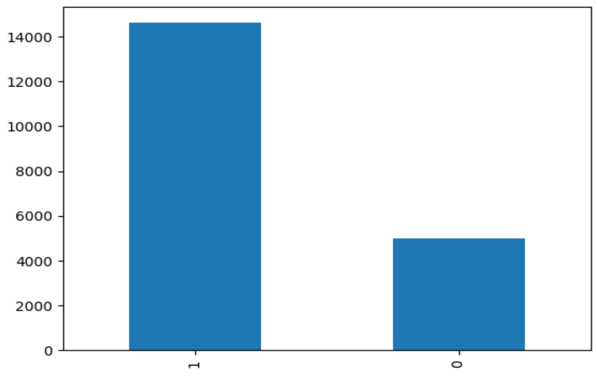
Figure 4.4: Overview of the solution

4.3 Phase-2 (Application of Machine Learning)

The overview of the entire architecture of the solution is depicted in figure 4.4. At first, the benign and malicious samples of PDF and PE files are used to make the datasets. The datasets are divided into three parts: training set (70%), validation set (10%), and test set (20%). Next, the data is pre-processed for the next phase of the solution. Then, feature selection is applied to the pre-processed data. And feature reduction is carried out to improve the performance of the model and optimize the computation overhead. The output of this phase is trained by the malware detector which is developed by different ML algorithms. After completion of the training, the classification performance is evaluated using the test samples. The implementation details are described in the following sections.



(a) PDF dataset



(b) PE dataset

Figure 4.5: Datasets used for experiments in this project

4.3.1 Data Preprocessing

Data preprocessing is required to make the raw data suitable and prepared for applying machine learning analysis. The actual goal is to refine and enhance the data so that the ML algorithms are capable of generalizing the pattern to classify correctly. Real-world data are generally noisy that is difficult to process. The first stage of data preprocessing is to develop a grasp over the data structure, formats, and the ranges of the values to understand any outlier. Cleaning the data removing any null values is the next stage. Afterwards, handling categorical data is crucial as ML algorithms can not process those categorical data. For that, data transformation into suitable formats is required. Lastly, the data is split into training set, validation set, and test set for operation of ML. Each of the data preprocessing stages are explained next.

Description of Data

The datasets contain malicious and benign samples of PDF and PE files. The project is a binary classification system where ML models classifies the files into either one of the two classes. The PDF dataset after deleting all the missing and *null* value contains 5,555 malicious samples and 4,468 benign samples. For PE dataset, the number of instances in the malicious category is 14599 and in benign category is 5012. The bar plot of the dataset is shown in figure 4.5 shows the number of samples of the two classes.

Data Cleaning

The task of the ML models becomes more challenging with the null values. To overcome that the null values for different samples are identified and excluded from the dataset. Listing 4.1 shows the source code for loading the dataset and finding for any missing value. Any row containing missing value are removed from the dataset and then the number of samples before and after deleting the missing value is printed. The output of this code is shown in listing 4.2. The *null* values of the PE dataset is also excluded in the same manner. In the following preprocessing steps only the relevant code snippets and explanations are provided for the PDF dataset. The same steps has also been carried out for the PE dataset as well.

```
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/PDFMalware2022.csv")
df.isnull().sum()
new_df=df.dropna(axis=0,how='any')
print("Old Dataframe length: ", len(df))
print("New Dataframe lenght: ", len(new_df))
print("Number of rows with at least 1 Null Values: ",(len(df)-len(
    new_df)))
```

Listing 4.1: Code for data loading and finding missing value

```
Old Dataframe length:  10026
New Dataframe lenght:  10023
Number of rows with at least 1 Null Values:  3
```

Listing 4.2: Output after deleting the rows containg missing values

Data Transformation

The models must be fed with the suitable data in order to provide excellent insights. Data transformation is that crucial preprocessing technique that involves modifying the data into the most compatible format with ML models. Usually, structured numerical data provides best output for the statistical analysis that ML algorithm performs. For data transformation, the categorical data is converted into A feature named `'text'` represents whether the sample contains text in it. In the dataset, there are five categories of data found which are `'yes'`, `'no'`, `'unclear'`, `'-1'`, and `'0'`. The number of instances found with the value `'-1'` and `'0'` is 315 which are removed from the experiment due to lack of having meaningful attribute. Listing 4.3 shows the code snippet of the function to transform the categorical data of `'text'` and `'class'` features into numerical data.

```python
def convert_categorical():
    print(new_df['text'].value_counts())

    new_df = new_df[new_df != '-1']
    new_df = new_df[new_df != '0']
    new_df['text'].replace(['No','Yes','unclear', '-1', '0'
    ],[0,1,2,3,4],inplace=True)
    print(new_df['text'].value_counts())

    print("Clas names converted to 1 and 0:")
    new_df['Class'].replace(['Malicious','Benign'],[1,0],inplace=True)
    print(new_df['Class'].value_counts())
```

Listing 4.3: Transmorning categorical data into numerical

In the datasets used in this project, the data types are `'object'` and `'float64'` whereas the data inside the datasets are discrete numerical values. For that reason both the typecasting is performed into `'int64'`. The reason to convert `float` data to `int` is for optimizing memory usage that has a slight possibility of enhancing the overall performance. The code snippet is shown in listing 4.4.

```python

def typecast_to_int(df):
    for column in df.columns:
        # Try converting the column to int type
        try:
            dataframe[column] = df[column].astype(int)
        except ValueError:
            # If the conversion fails, skip the column (e.g., if it
    contains non-numeric data)
            pass

    return dataframe
```

Listing 4.4: Data type transformation code

Data Scaling and Standardizing

Another data transformation technique is *StandardScaler()* to standardize the numerical features of the dataset. The output converts all the data in the dataset having a mean of 0 with a standard deviation of 1. ML is a statistical technique that generates the output processing huge mass of numbers. In real scenarios, systems have to process data that are different in scales and ranges. Consequently, the outliers having large numbers sometimes become dominant and impose significant impact on the classification performance of the ML models. So converting and scaling the features in the same standard helps ML models to handle different features uniformly leading to a more efficient model training.

Although, bagging and boosting technique might not require the StandardScaler as much as deep learning. In this project, it is only applied for the deep learning techniques. The code snippet to perform standardization is shown in listing 4.5. The training data is both fit and transformed using *fit_transform()* function. The validation and test data is supposed to be only transformed for being new and unseen.

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val=scaler.transform(X_val)
X_test=scaler.transform(X_test)
```

Listing 4.5: Code snippet for standard scaler

4.3.2 Feature Selection & Dimensionality Reduction

As stated before, the dataset containing PDF samples has 31 features excluding the target variable. To improve the computation speed and efficiency for the ML models most relevant features are selected in this process. In this project, RFC is applied to produce the sorted list of features based on their importance values. Listing 4.6 shows the code snippet for implementation of the feature selection process. Once the importance values are generated it becomes an simple task to decide which features can be dropped. Out of the 31 features, 13 found to be ineffective and thus dropped. The dimension of the dataset is then reduced to 18 features from 31.

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
y = fit_df['Class']
X = fit_df.drop('Class', axis=1)  # Replace 'target_column_name' with
    the actual target column name
clf = RandomForestClassifier(random_state=42)
clf.fit(X, y)
feature_importances = clf.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X.columns, '
    Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='
    Importance', ascending=False)
print(feature_importance_df)
```

Listing 4.6: Implementation of RFC for feature selection

4.3.3 Data Splitting

To build a reliable ML model it is required to split the dataset. In this project the dataset is split into three parts. 70% of the samples are used for training the model, 10% for validation, and the rest 20% is to test the system to evaluate the performance of the model on new and unseen data. The generalization capabilities of a model are significantly influenced by the validation set. It enables researchers to comprehend how effectively the model generalizes to new data that was not included in its training set. Validation data is also helpful to determine if the model has overfitted the training set. The function to split dataset is shown in listing 4.7. Table 4.3 represents the number of instances for training, validation, and test set.

```
from sklearn.model_selection import train_test_split

def data_split(data, data_target):
  X,X_test,Y,y_test= train_test_split(data, data_target,test_size=0.2,
     shuffle = True, random_state = 0)
  X_train,X_val,y_train,y_val = train_test_split(X, Y,test_size=0.2,
    shuffle = True, random_state = 0)
  print("X_train shape: {}".format(X_train.shape), "y_train shape: {}"
    .format(y_train.shape))
  print("X_val shape: {}".format(X_val.shape), "y_val shape: {}".
    format(y_val.shape))
  print("X_test shape: {}".format(X_test.shape), "y_test shape: {}".
    format(y_test.shape))
  return X_train,X_val,X_test,y_train,y_val,y_test

target_name = 'Class'
data_target = new_df['Class']
data = new_df.drop(['Class'], axis=1)
X_train,X_val,X_test,y_train,y_val,y_test = data_split(data,
    data_target)
```

Listing 4.7: Splitting the dataset into training, validaiton, and test set

Table 4.3: Number of samples in the datasets

Dataset	Training Set Samples (70%)	Validation Set Samples (10%)	Test Set Samples (20%)
PDF Dataset	6594	733	1832
PE Dataset	14119	1569	3923

4.3.4 Implementation of Classical Machine Learning

There exist a long list of algorithms falling into the category of classical approaches. Out of all, two approaches are chosen having simple computation and not so complex. Naive Bayes operating on the probabilistic prediction. And logistic regression assuming linear relationship between input features and output classes, specially designed for binary classification. A brief description of how these algorithms operate is given in section 2.6. In this section, the implementation of both the algorithms are shown in code snippets in listing 4.8 and 4.9. The models are run with their default parameters without any adjustments to see how well the simpler models perform when compared to more advanced deep learning or ensemble algorithms.

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
gaussian_naive = GaussianNB()
def implement_gaussian_naive_bayes(X_train,X_val,X_test,y_train,y_val,
    y_test):
    gaussian_naive.fit(X_train, y_train)
    acc_gaussian_training = round(gaussian_naive.score(X_train,
    y_train) * 100, 2)
    y_pred_validation = gaussian_naive.predict(X_val)
    accuracy_validation = accuracy_score(y_val, y_pred_validation)
    y_pred_test = gaussian_naive.predict(X_test)
    accuracy_test = accuracy_score(y_test, y_pred_test)
    display_result(acc_gaussian_training,accuracy_validation,
    accuracy_test)

implement_gaussian_naive_bayes(X_train,X_val,X_test,y_train,y_val,
    y_test)
```

Listing 4.8: Gaussian Naive Bayes model implementation

Firstly, the model is trained with the *fit()* function. The training features and the target class of those features are passed as arguments. After the training is completed, the model is applied to the validation set to make predictions. Finally, the model then makes predictions using the test set. Three accuracies are calculated for training set, validation set, and test set using the function $accuracy_score()$. The function takes two arguments: true labels of the instances and the predicted labels. The code to display the accuracy values for training, validation, and test set is shown in listing 4.10.

```python
from sklearn.linear_model import LogisticRegression
def implement_logistic_regression(X_train,X_val,X_test,y_train,y_val,
    y_test):
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)
    acc_logreg = round(logreg.score(X_train, y_train) * 100, 2)
    y_pred_validation = logreg.predict(X_val)
    accuracy_validation = accuracy_score(y_val, y_pred_validation)
    y_pred_test = logreg.predict(X_test)
    accuracy_test = accuracy_score(y_test, y_pred_test)
    display_result(acc_logreg, accuracy_validation, accuracy_test)

implement_logistic_regression(X_train,X_val,X_test,y_train,y_val,
    y_test)
```

Listing 4.9: Logistic regression model implementation

```
def display_result(acc_training, accuracy_validation, accuracy_test):
  print("Accuracy on the Training set:{:.2f}\%".format(acc_training))
  print("Accuracy on the Validation set: {:.2f}\%".format(
    accuracy_validation * 100))
  print("Accuracy on the Test set: {:.2f}\%".format(accuracy_test *
    100))
```

Listing 4.10: Code to display the training accuracy validation and test accuracy

4.3.5 Implementation of Ensemble Techniques (Bagging and Boosting)

Two very popular ensemble techniques: bagging and boosting are applied in this project. Random forest (RF) is chosen from the branch of bagging algorithms and adaptive boosting (AdaBoost) from the group of boosting algorithms. The description of the algorithms is found in section 2.6.2. In this section, the implementation of these two techniques will be discussed along with the suitable code snippets.

Random Forest (RF)

For the implementation of RF, ensemble of base learner is created and the results are aggregated to construct the final predictions. The base learner for RF is decision tree that are built independently. There are a number of hyperparameters required to be tuned properly to get the best outcome from the model. For tuning, Grid Search Cross-Validation (GridSearchCV) is implemented to find the best combination of parameters. GridSearchCV exhaustively looks over all conceivable combinations of hyperparameters. Also, a cross validation is applied for each combinations to bring out the best output. Although, while designing this sort of GridSreachCV, the processing time for training must a considering factor as this process might require high computational resources. The code snippet to implement RF and GridSearchCV is shown in listing 4.11. The values of the hyperparameters to search for are:

- *'n_estimators'* defines the number of decision trees built independently. Here, the optimal value GridSearchCV will search from the given set *[100, 200, 300]*.
- *'max_depth'* defines the depth of decision trees. Having a large number for max_depth can raise the complexity of the model. The assigned set of values to search for are: *[None, 10, 20, 30]*.
- *cv* defines the cross validation. The value assigned for cross validation is '5'. Hence, the training set will be divided into five random subsets. The model is trained on the four subsets and tested on the fifth one. This step is iterated for 5 times and the final result is obtained by computing the average of all the results.

```
from sklearn.ensemble import RandomForestClassifier,
    GradientBoostingClassifier, ExtraTreesClassifier
from sklearn.model_selection import train_test_split, cross_val_score,
     GridSearchCV
def implement_random_forest(X_train,X_val,X_test,y_train,y_val,y_test)
    :
    random_forest = GridSearchCV(estimator=RandomForestClassifier(),
                    param_grid={'n_estimators': [100, 200, 300],
                    'max_depth': [None, 10, 20, 30],},
                    cv=5).
                    fit(X_train, y_train)
    random_forest.fit(X_train, y_train)
    acc_random_forest = round(random_forest.score(X_train, y_train) *
    100, 2)
    print(acc_random_forest,random_forest.best_params_)
    y_pred_val_randomforest = random_forest.predict(X_val)
    accuracy_val_randomforest = accuracy_score(y_val,
    y_pred_val_randomforest)
    y_pred_test_randomforest = random_forest.predict(X_test)
    accuracy_test_randomforest = accuracy_score(y_test,
    y_pred_test_randomforest)    display_result_random_forest(
    acc_random_forest,accuracy_val_randomforest,
    accuracy_test_randomforest)
implement_random_forest(X_train,X_val,X_test,y_train,y_val,y_test)
```

Listing 4.11: Implementation of random forest with hyperparameter optimization

Adaptive Boosting (AdaBoost)

The implementation of AdaBoost is similar to RF and the code snippet is shown in listing 4.12. In this case, *DecisionTreeClassifier()* is used to build the ensemble of base learners. The function has an argument *'max_depth'* which specifies the depth of each decision tree. The optimal hyperparameters are investigated with *GridSearchCV* again. And cross validation is also applied to build a robust model. The values are:

- *'n_estimators'* gets the set of values: *[50, 100, 150]*.
- *'learning_rate'* defines the contribution of each base learner to the final ensemble model. After each iteration the weights of the instances of the training set and the base learner is updated. This *'learning_rate'* is multiplied with the updated weights of the base learner. It receives the value *[0.1, 0.01, 0.001]*.
- *cv* receives the value '5'.

```python
from sklearn.tree import DecisionTreeClassifier
base_estimator = DecisionTreeClassifier(max_depth=3)
adaboost_classifier = AdaBoostClassifier(base_estimator=base_estimator
    , random_state=42)
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.1, 0.01, 0.001],
}
grid_search = GridSearchCV(adaboost_classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print(best_params)
best_adaboost_classifier = AdaBoostClassifier(
    base_estimator=base_estimator,
    n_estimators=best_params['n_estimators'],
    learning_rate=best_params['learning_rate'],
    random_state=42
)
best_adaboost_classifier.fit(X_train, y_train)
```

Listing 4.12: Implementation of AdaBoost with hyperparameter optimization

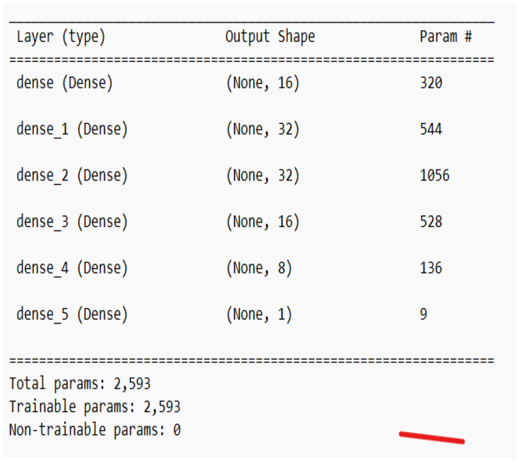
4.3.6 Implementation of ANN

The implementation of ANN is different than the previous ones. In this project, *keras* library is leveraged for the implementation purpose. *Keras* has a dedicated built in function named *Sequential()* to construct the layers and nodes in each layer of the ANN. In this project, three ANN models having different architectures are deployed to investigate what sort of ANN achieves the goal most accurately. First, the architecture and then the implementation of the three models are discussed next.

Architecture

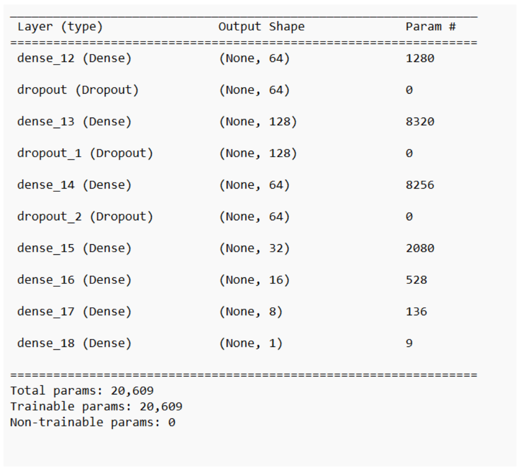
Figure 4.6 depicts the architecture of the three ANN models deployed. Model M1 and M2 having the same architecture and M3 have a deeper architecture. There is a difference of data pre-processing among the three models. M1 is deployed to test a neural network's performance using a dataset that has not been scaled to a certain range, as recommended by most researchers. So, M1 is not fed with standardized training set with *StandardScaler()* whereas both M2 and M3 is trained on the scaled data. There is no other difference in the feature extraction and selection phase among the models.

As seen in the figure 4.6a the network consists of six layers, one for input 4 for the hidden layers and 1 for the output layer. The input layer has 16 nodes, The 4 hidden layers have nodes 32, 32, 16, and 8. The models are pretty shallow



```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
dense (Dense)                (None, 16)                320

dense_1 (Dense)              (None, 32)                544

dense_2 (Dense)              (None, 32)                1056

dense_3 (Dense)              (None, 16)                528

dense_4 (Dense)              (None, 8)                 136

dense_5 (Dense)              (None, 1)                 9

================================================================
Total params: 2,593
Trainable params: 2,593
Non-trainable params: 0
```

(a) M1 and M2 model



```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
dense_12 (Dense)             (None, 64)                1280

dropout (Dropout)            (None, 64)                0

dense_13 (Dense)             (None, 128)               8320

dropout_1 (Dropout)          (None, 128)               0

dense_14 (Dense)             (None, 64)                8256

dropout_2 (Dropout)          (None, 64)                0

dense_15 (Dense)             (None, 32)                2080

dense_16 (Dense)             (None, 16)                528

dense_17 (Dense)             (None, 8)                 136

dense_18 (Dense)             (None, 1)                 9

================================================================
Total params: 20,609
Trainable params: 20,609
Non-trainable params: 0
```

(b) M3 model

Figure 4.6: Architectures of the ANN models

and simple. The third model M3 has a deeper architecture and immune to overfitting introducing regularization technique *Dropout()*. The argument in the function defines the number of neurons to be ignored during training. The input layer of M3 consists of a considerably larger 64 nodes. The 5 hidden layers are substantially wide with 128, 64, 32, 16, and 8 nodes sequentially. The input layer and the first 2 hidden layer has *Dropout* between them. Looking at the number of trainable parameters, the dimension of the model M3 can be imagined. The number of parameters are approximately 10 times more than M1 and M2 model.

Hyperparameters of Neural Network

In ANN, there are plethora of hyperparameters having substantial impact on the classification performance. It is a difficult task to choose the optimum hyperparameters. Few of them are listed below:

- **Activation function:** Different variants of the activation function have been tried and finalized with the *'relu function'* for input and hidden layers. And, *'sigmoid function'* is applied to the output layer so that the values are only '0' and '1'.
- ***Dropout()*:** The input layer has been assigned to drop 20% of the node for different iterations. The next two layers in the hidden layer has 30% nodes dropped.
- ***Optimizer*:** *'adam'* is used as the optimizer.
- ***Loss function*:** The target being classifying files into benign and malicious, $'binary_{c}rossentropy'$ is used as the loss function.

- ***Number of epochs***: The models have been trained with various number of epochs (like 50, 100, 200, 300) to discover the suitable number. And finally found that the models performs best to their ability when the *'epoch'* is '300'.
- ***Batch size***: Various batch sizes (8, 16, 32, 64) are tried, and 32 are chosen in the end. Putting a smaller batch size increases the training time substantially and adds more noise. Larger batch size requires more memory and counters issues in generalizing data. 32 is selected considering all these factors.
- ***Learning rate***: All three models have 0.001 as the learning rate.

Implementation

The code snippet to implement the M3 model is shown in listing 4.13. The function *create_model*() is designed to construct the desired neural network as represented in figure 4.6b. The model is compiled with the designated optimizer, loss and metrics. The model is trained for 300 epochs with *batch_size* 32. Once the training is finished, the test set is applied on the data to inspect the performance of the model. Other evaluation metrics like building confusion matrix and calculating precision, recall, F1-score is performed next. Finally, the ROC curve and AUC from the ROC curved is computed to acknowledge the results of the models.

```python
import tensorflow as tf
import keras
from keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Dropout, Conv1D,
    MaxPooling1D, Flatten
from sklearn.model_selection import cross_val_score, KFold
n_splits = 5  # No of folds
kf = KFold(n_splits=n_splits, shuffle=True, random_state=42)
def create_model():
    model = Sequential([
        Dense(64, input_dim =19, activation ="relu"),
        Dense(128, activation = "relu"),
        Dropout(0.2),
        Dense(64, activation = "relu"),
        Dropout(0.3),
        Dense(32, activation = "relu"),
        Dropout(0.3),
        Dense(16, activation = "relu"),
        Dense(8, activation = "relu"),
        Dense(1, activation ="sigmoid")
    ])
    model.summary()
    model.compile(optimizer='adam', loss='binary_crossentropy',
    metrics=['accuracy'])
    return model
```

```
tf.keras.wrappers.scikit_learn.KerasClassifier(build_fn=create_model,
    epochs=300, batch_size=32)
results = cross_val_score(model, data, data_target, cv=kf, scoring='
    accuracy')
for i, score in enumerate(results):
    print(f"Fold {i+1}: Accuracy = {score:.4f}")
mean_accuracy = np.mean(results)
print(f"Mean Accuracy: {mean_accuracy:.4f}")
```

Listing 4.13: Implementation of the ANN with cross validation = 5

4.4 Phase-3 (Model Evaluation)

Once the models are trained as stated in the previous sections, test set is applied to examine the classification performance of the models. This project deploys confusion matrix to calculate precision, recall, and f1-score besides accuracy. Furthermore, the ROC curve is used for visual representation of the performance and AUC value is generated from the curve. The explanations of these metrics can be seen in section 2.7, whereas this section shows the implementation of the evaluation techniques. Listing 4.14 shows the code snippet to generate and plot the confusion matrix from which the values of precision, recall, f1-score can be calculated for any model.

```
def build_confusion_matrix():
  y_pred_gauss = gaussian_naive.predict(X_test)
  cf_matrix = confusion_matrix(y_test, y_pred_gauss)
  ax = sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, fmt='.2%',
     cmap='Blues')
  ax.set_title('Confusion Matrix for Naive Bayes\n');
  ax.set_xlabel('\nPredicted Value')
  ax.set_ylabel('Actual Value ');
  plt.show()

build_confusion_matrix()
print(classification_report(y_test, y_pred_adaboost))
```

Listing 4.14: Implementation of confusion matrix

Next, ROC curve is plotted by putting FPR along x-axis and TPR along y-axis. The code written to plot the curve and generate the auc value from the values of TPR and FPR is displayed in listing 4.15.

```python

from sklearn.metrics import roc_curve, auc
gaussian_fpr, gaussian_tpr, threshold = roc_curve(y_test, y_pred_gauss
    )
auc_gaussian = auc(gaussian_fpr, gaussian_tpr)
plt.figure(figsize=(5, 5), dpi=100)
plt.plot(gaussian_fpr, gaussian_tpr, marker='.', label='Naive Bayes (
    auc = %0.2f)' % auc_gaussian)
plt.title('ROC curve for Malicious PDF Prediction')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 4.15: Code snippet to plot ROC curve and calculate AUC value

4.5 The Proposed Hybrid Solution

4.5.1 Architecture

As of now application of classical approach, ensemble technique, and neural network in malware detection is carried out and the result is inspected meticulously. Now, the strengths of ensemble technique and neural network are combined together for further potential improvement. In this project, a novel approach is proposed where neural network (model M3) is specially fabricated to perform feature extraction and then feed the output extracted feature to the bagging and boosting techniques for classification of the malwares. All the previous steps of data collection and data preprocessing is kept uniform. The architecture of the proposed hybrid solution is represented in figure 4.7 where the hybrid portion is shown in green. Both bagging and boosting techniques are applied to inspect which one performs the best.

4.5.2 Implementation

The implementation of the hybrid solution can be sliced into multiple blocks.

- The first one is to build the neural network model and train the model with the training set.
- Next, a key step where the output layer of the model M3 is sliced out as the objective of the neural network is not classification, but extracting the features. Listing 4.16 shows the code snippet to slice out the output layer of the model. The structure of the model is shown in figure 4.8. Now the model is ready for the feature extraction from the data.

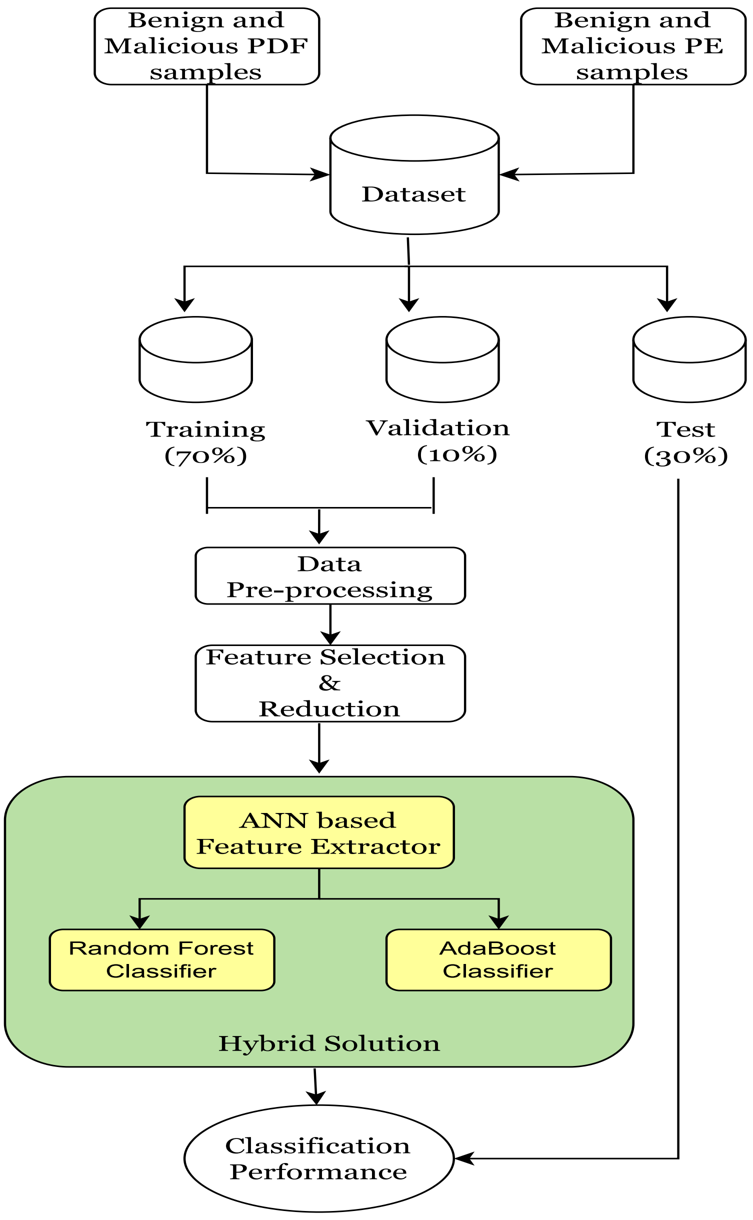
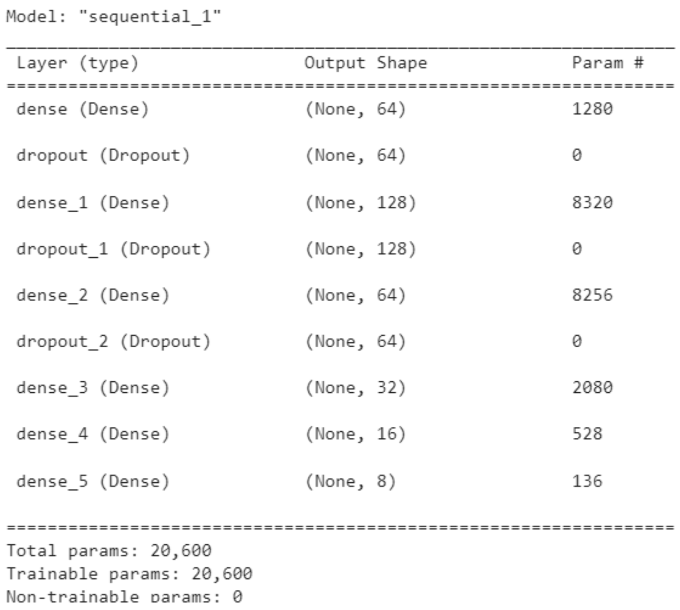

Benign and Malicious PDF samples
Benign and Malicious PE samples
Dataset
Training (70%)
Validation (10%)
Test (30%)
Data Pre-processing
Feature Selection & Reduction
ANN based Feature Extractor
Random Forest Classifier
AdaBoost Classifier
Hybrid Solution
Classification Performance

Figure 4.7: Architecture of the proposed solution



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                1280

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 128)               8320

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dropout_2 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 dense_4 (Dense)             (None, 16)                528

 dense_5 (Dense)             (None, 8)                 136

=================================================================
Total params: 20,600
Trainable params: 20,600
Non-trainable params: 0
```

Figure 4.8: Architecture of the feature extractor using ANN

- Next, the resulting model performs the feature extraction being fed with training set, validation set, and test set
- Then, the bagging and boosting techniques are trained on the extracted features of the training and validation set obtained from the neural network.
- Finally, both bagging and boosting techniques are applied for predicting the unseen data of the test set.
- Lastly, the model is examined with appropriate evaluation metrics to produce the results of each of the hybrid approach.

```python
feature_extraction_model = Sequential()
for layer in model.layers[:-1]:
    feature_extraction_model.add(layer)
feature_extraction_model.summary()
extracted_features_nn_train = feature_extraction_model.predict(X_train
    )
extracted_features_nn_val = feature_extraction_model.predict(X_val)
extracted_features_nn_test = feature_extraction_model.predict(X_test)
```

Listing 4.16: Implementation of the ANN based feature extractor for the proposed solution

Chapter 5

Result and Analysis

This chapter focuses on the outcomes produced by several ML approaches in classifying malwares. The models are subjected to appropriate metrics discussed in section 2.7 in order to evaluate their performances. To make it easy to comprehend, the findings of different methodologies are provided separately with the required tables, values, and graphs. It is experimentally shown that the proposed hybrid solution in this thesis achieves better results than the other approaches.

5.1 Result Analysis of Classical Machine Learning

The results of the two classical approaches LR and GNB are shown in table 5.1. On the provided dataset, both techniques deliver decent results. The confusion matrix and the ROC curve of both the techniques are shown in figure 5.1. Comparing between the two classical techniques, LR performs slightly better with the accuracy of 90.37% on the test set. More rigorous metrics like precision, recall, F1-score are calculated from the confusion matrix and AUC value is generated from the ROC curve. The model yields 91.25% for F1-score and the area under the ROC curve is 90.09%. From figure 5.1c the ROC curve of LR clearly superior to the curve of GNB. The possible grounds for LR performing better than GNB are as follows:

- LR employs a logistic function (sigmoid) to assign into the target class after constructing a complex linear relationship among the features. The quality of how the model performs vastly depends on this linear relationship. In this case, the features heavily relying on each other making LR a better performing algorithm than GNB given that GNB perceives features as continuous fitting the Gaussian or normal distribution independently given the target class.
- Another evident reason can be GNB performs well with continuous features but not with the categorical ones. GNB needs them to be converted to contin-

uous representations which is not carried out in the experiment. However, LR is effective with both categorical and continuous features making it perform moderately in this experiment.

Table 5.1: Evaluation metrics of classical ML approaches

	Training Acc.	**Val. Acc.**	**Test Acc.**	**Precision**	**Recall**	**F1-score**	**AUC**
GNB	89.76%	88.93%	88.62%	85.66%	94.37%	89.81%	88.23%
LR	91.24%	90.35%	90.37%	88.16%	94.57%	91.25%	90.09%

Table 5.2: Evaluation metrics of ensemble techniques

	Training Accuracy	**Val. Acc.**	**Test Acc.**	**Prec.**	**Recall**	**F1-score**	**AUC**
RF	99.81%	97.81%	98.35%	98.07%	98.83%	98.45%	98.32%
AdaBoost	99.30%	98.33%	98.46%	98.45%	98.64%	98.54%	98.44%

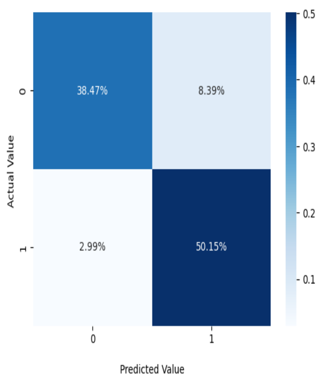
Table 5.3: Evaluation metrics of the three ANN models

	Training Acc.	**Validation Acc.**	**Test Acc.**	**Precision**	**Recall**	**F1-score**	**AUC**
M1	99.76%	97.55%	97.99%	97.78%	98.44%	98.11%	97.96%
M2	100%	97.68%	98.04%	97.69%	98.64%	98.16%	98.00%
M3	99.97%	98.33%	98.92%	99.46%	99.51%	98.98%	98.87%

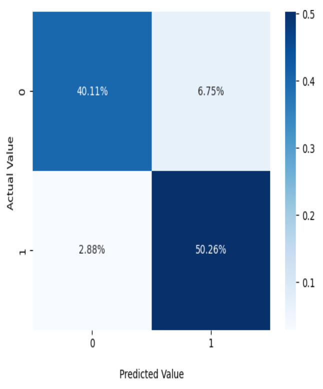
Table 5.4: Evaluation metrics of the ANN-ensemble hybrid models

	Train. Acc.	**Val. Acc.**	**Test Acc.**	**Prec.**	**Rec.**	**F1**	**AUC**
Hybrid-1	99.92%	98.91%	98.53%	98.28%	98.96%	98.61%	98.50%
Hybrid-2	100%	99.73%	99.51%	99.48%	99.58%	99.53%	99.51%

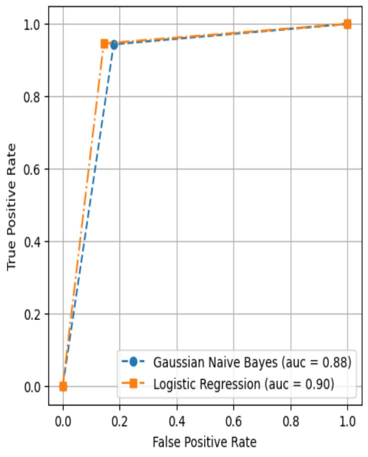
The two classical methods work mediocrely, and contemporary systems need to be more accurate in sensitive cases like malware analysis. Triggered false positives are critical disrupting usual workflow. Even LR generates 6.75% of false positives and 2.88% of false negatives. In pursuit of better performance, ensemble techniques are employed next which is discussed in following section.



(a) Gaussian Naive Bayes

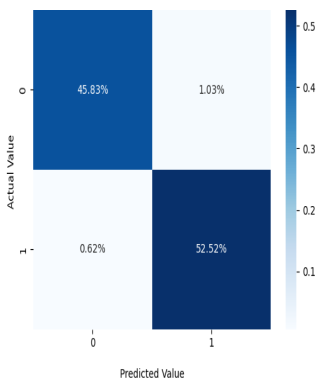


(b) Logistic Regression

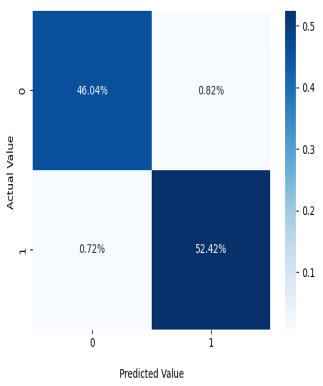


(c) ROC curves

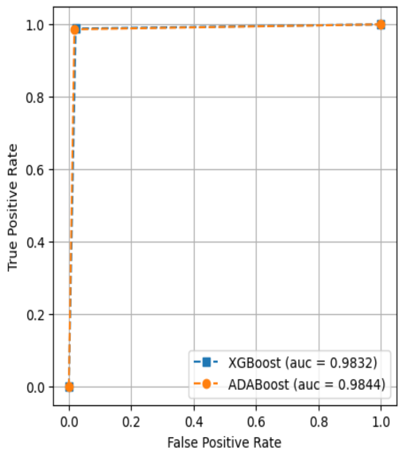
Figure 5.1: Confusion Matrix and ROC curves of classical ML approaches



(a) Random forest

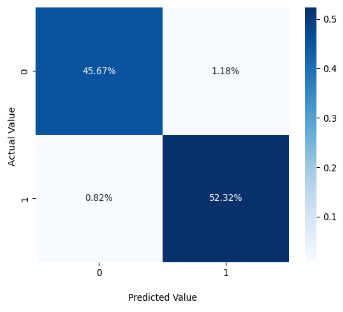


(b) AdaBoost

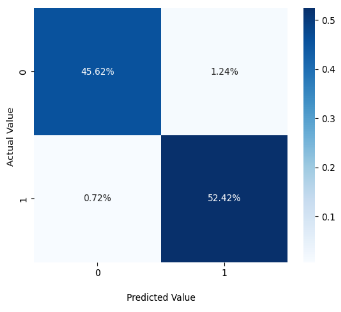


(c) ROC curve

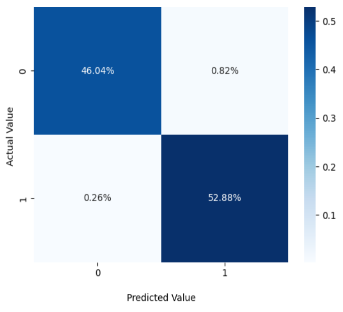
Figure 5.2: Confusion Matrix and ROC curve of Ensemble Techniques



(a) M1 model



(b) M2 model



(c) M3model

Figure 5.3: Confusion Matrix of the three ANN models

5.2 Result Analysis of Ensemble Techniques

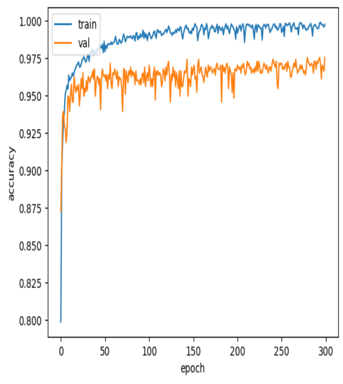
Table 5.2 depicts the results of the two ensemble techniques: bagging (RF) and boosting (AdaBoost). The confusion matrix and ROC curves for the mentioned techniques is illustrated in figure 5.2. Discussing about the results, there are massive improvements seen for ensemble techniques. In comparison to the earlier classical techniques, both the bagging and boosting algorithms perform significantly better. The accuracies by RF and AdaBoost for the test set are 98.35% and 98.46% respectively. The F1-score for RF is 98.45% whereas AdaBoost has a thin improvement of 98.54%. The ROC curve shown in 5.2c also affirms the upper hand of AdaBoost over RF producing an AUC value of 98.44%.

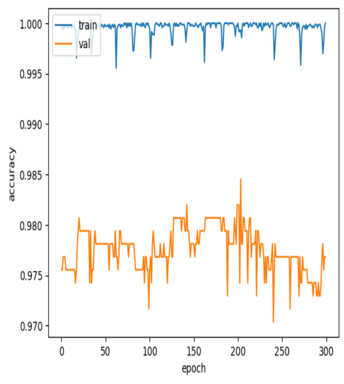
If the reason of AdaBoost performing better than previously discussed LR is examined few intriguing aspects emerge. Ensemble techniques like AdaBoost builds non-linear relationships among the feature to generate stronger and more complex classifier. This allows the decision boundary of AdaBoost to operate better by correctly classifying samples that can not be separable by the linear classifier of LR. Also, AdaBoost can handle outliers better by assigning higher weights for them during training. However, LR receive impacts from outliers that contribute to higher errors. At first glance, it might seem there is only one ROC curve. However, both the curves having same pattern and AdaBoost having more dominance over RF, the AdaBoost curve has eclipsed the curve of RF. Despite the excellent performance of AdaBoost, we will look at potential improvements by deploying ANN in the next section.

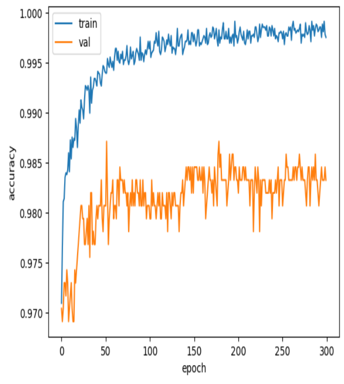
5.3 Result Analysis of ANN

It must be stated that the outcome achieved by AdaBoost is already outstanding. It will be exceedingly challenging to continue advance from here, and the advancements will be quite minute. Three models have been implemented in this study to evaluate precisely how ANNs accomplish in malware detection. The implementation and design specifications of various hyperparameters like `batch size, layers, dropout, no of epochs to train, etc.` are discussed previously in section 4.3.6 and figure 4.6. Here, the results of each of the models are discussed in this section and illustrated in table 5.3.

Both model M1 and M2 does not perform better than AdaBoost. M1 and M2 generate a modest accuracies 97.99% and 98.04% for the test set, which is lower than what AdaBoost achieved previously. The confusion matrix shown in 5.3 for model M1 and M2 also assert the misclassications with false positives of 1.18%

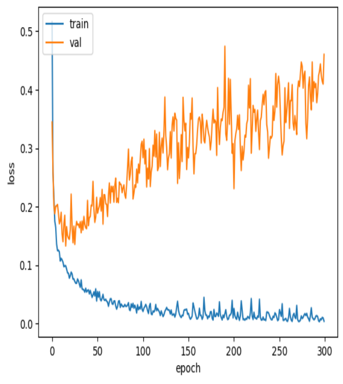


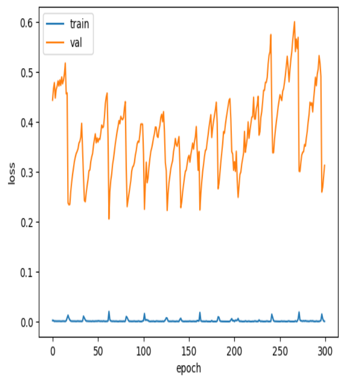


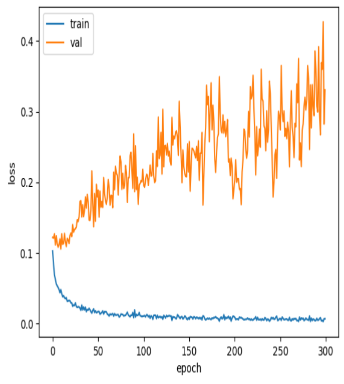


(a) M1 model **(b)** M2 model **(c)** M3model

Figure 5.4: Training and validation accuracy of the three ANN models







(a) M1 model **(b)** M2 model **(c)** M3model

Figure 5.5: Training and validation loss of the three ANN models

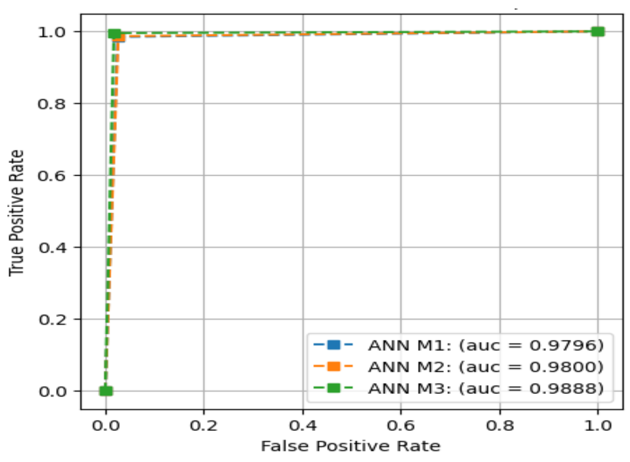
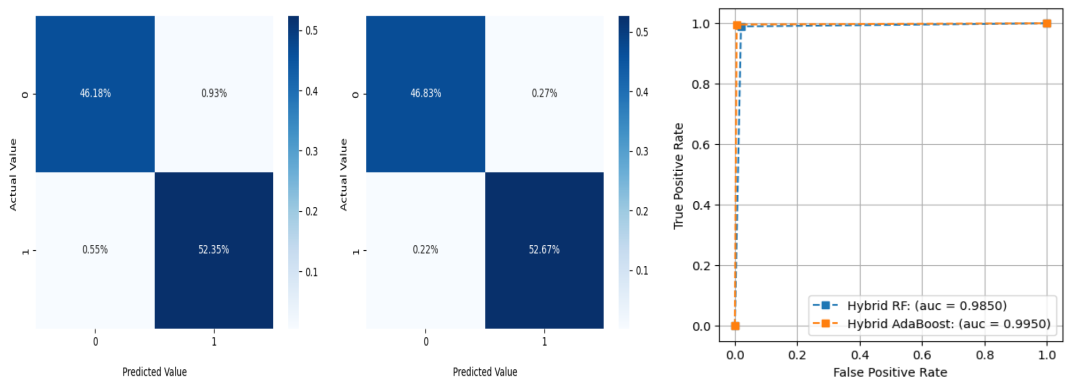


Figure 5.6: ROC curve for the three ANN models

for M1 and 1.24% for M2. Besides the confusion matrix, another visualization is present for the deep learning models illustrating the accuracy and loss curves during training shown in figure 5.4 and 5.5. Looking at the accuracy curves of M1 in 5.4a the clear difference between the training and validation curves is observed though the curves appear orthodox. But the curves for M2 in figure 5.4b exhibits an unusual pattern being distinctly apart from each other confirming that the model has overfit the training data with 100% of accuracy whereas the validation and test accuracy is nowhere close. Overfitting is a critical issue in machine learning and require to be addressed. Therefore, the third ANN model M3 is implemented and deployed with a new specification and methodology that enables the model to be resilient to overfitting.

The results and accuracy curve in 5.4c of the model M3 validate the immunity against overfitting. The accuracy curve of M3 gets back it's typical shape and yields an accuracy of 99.97% for the training set. Finally, M3 could outperform AdaBoost considerable margin with a test accuracy of 98.92% and emerges as the most effective strategy out of all those used up to this point.The improvement for F1-score for M3 is 98.98% superior than the previous best AdaBoost having F1-score 98.54%. The ROC curve for the three models are depicted in figure 5.6. M3 turns out to be the best model with the highest AUC of 98.88% concealing the ROC curve of the other two models.

ANN appears to be the foremost algorithm up to this moment bringing out extraordinary results. Further advancements are attempted to discover any favourable outcome. Now the proposed novel solution in this thesis will be discussed in the next section.



(a) ANN-RF-hybrid model **(b)** ANN-AdaBoost-hybrid model **(c)** ANN-AdaBoost-hybrid model

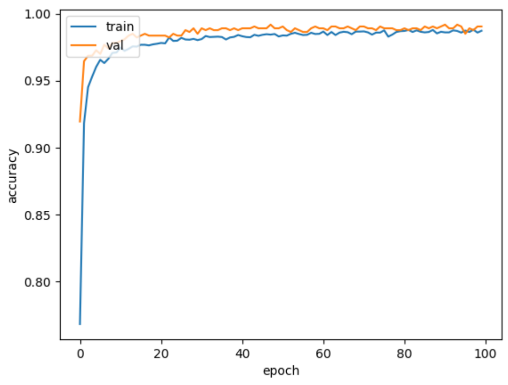
Figure 5.7: Confusion matrix and ROC curves of the two hybrid models

5.4 Result Analysis of the Hybrid Solution

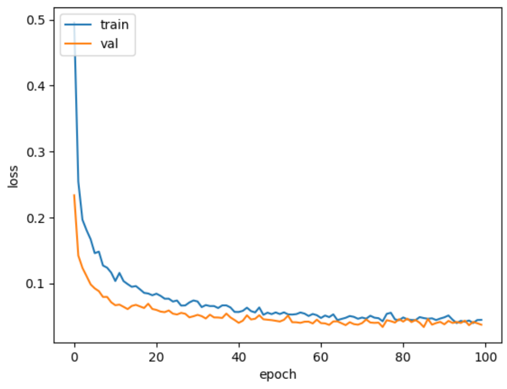
Improvising an ANN model performing a shade below 99% is a gigantic task. This thesis proposes a novel hybrid approach, the architectural specifications of which is found in section 4.5. The results generated by these hybrid models are delineated in table 5.4. The two hybrid models are manufactured by layering the ANN model M3 intended exclusively for feature engineering and ensemble techniques like RF and AdaBoost for the classification. The first hybrid model Hybrid-1 assembled with RF on top of ANN model M3 generates a test accuracy of 98.53%. Consequently, the Hybrid-1 model outperforms RF and AdaBoost separately but falls short of the ANN model M3. Nonetheless, the outcomes generated by Hybrid-1 is still admissible.

However, the Hybrid-2 model combining the strenghts of ANN and boosting, has outperformed all the previous techniques discussed in this thesis previously. With a flawless training accuracy of 100%, this hybrid model achieves the best validation accuracy and test accuracy so far, which are 99.73% and 99.51% respectively. The confusion matrix and ROC curves of both the hybrid models are shown in figure 5.7. The false positive and false negative for model Hybrid-2 are optimized to 0.27% and 0.22% only. The AUC produced by Hybrid-2 is 99.50% outshining Hybrid-1, also presiding the ROC curve by a cut above.

The accuracy and loss curves for the Hybrid-2 model are displayed in figure 5.8. Both the curves exhibit an immaculate and seamless pattern with no sudden spikes or drops in values. The curves demonstrates the immunity to overfitting clearly though the training accuracy is 100%.



(a) Accuracy curve for training and validation set



(b) Loss curve for training and validation set

Figure 5.8: Feature extraction using ANN for hybrid models

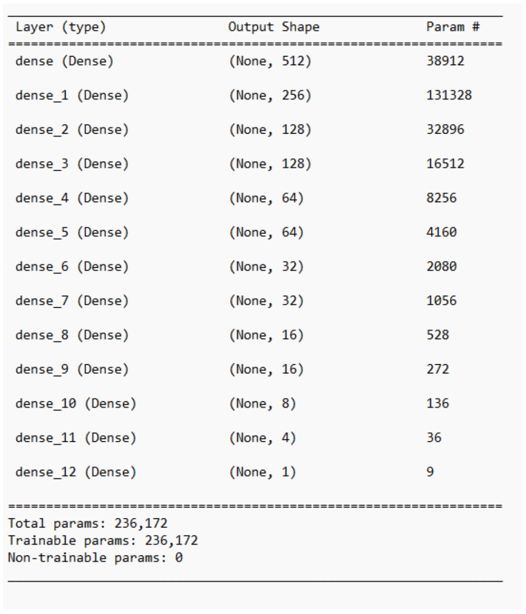
5.5 Result Analysis of PE Dataset

In this section, the result of the PE dataset will be analyzed. For convenience, the results have been accumulated for all the different approaches together in table 5.5. The visualization of confusion matrix of the different techniques is depicted in figure 5.10. And the ROC curves of all the techniques deployed is shown in figure 5.11. Both the ensemble techniques: RF and AdaBoost performed almost the same having test accuracy 97.62% and 97.25% respectively. RF has produced 2.04% of FP and 0.33% of FN. The classification performance of both the model can be assured by the satisfactory F1-score values 98.42% and 98.17% for RF and AdaBoost respectively. The ROC curve of these two techniques is found in figure 5.11a from the AUC values can be obtained which are 95.79% for RF and 95.08% for AdaBoost. 5.11.

Table 5.5: Results for evaluation metrics on PE dataset

	Training Acc.	**Val. Acc.**	**Test Acc.**	**Prec.**	**Recall**	**F1**	**AUC**
RF	97.7%	97.70%	97.62%	97.32%	989.55%	98.42%	95.79%
AdaBoost	97.0%	96.75%	97.25%	96.86%	99.52%	98.17%	95.08%
ANN	99.26%	97.89%	97.78%	98.49%	98.52%	98.51%	97.07%
Hybrid-3	99.99%	98.09%	97.94.%	98.46%	98.76%	98.61%	95.79%
Hybrid-4	99.18%	98.79%	98.45%	98.60%	99.31%	98.95%	97.62%

Then, ANN has been applied with a deeper architecture corresponding with the number of the features and size of the dataset. This time utilizing the previous knowledge on the domain only a single ANN model is applied. The architecture is shown in figure 5.9. The model is constructed with a bigger number of neu-



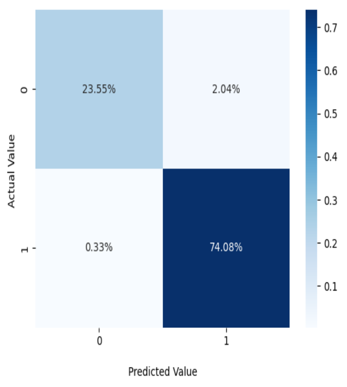
```
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               38912

dense_1 (Dense)              (None, 256)               131328

dense_2 (Dense)              (None, 128)               32896

dense_3 (Dense)              (None, 128)               16512

dense_4 (Dense)              (None, 64)                8256

dense_5 (Dense)              (None, 64)                4160

dense_6 (Dense)              (None, 32)                2080

dense_7 (Dense)              (None, 32)                1056

dense_8 (Dense)              (None, 16)                528

dense_9 (Dense)              (None, 16)                272

dense_10 (Dense)             (None, 8)                 136

dense_11 (Dense)             (None, 4)                 36

dense_12 (Dense)             (None, 1)                 9

=================================================================
Total params: 236,172
Trainable params: 236,172
Non-trainable params: 0
_________________________________________________________________
```

Figure 5.9: ANN model architecture for implemented PE dataset

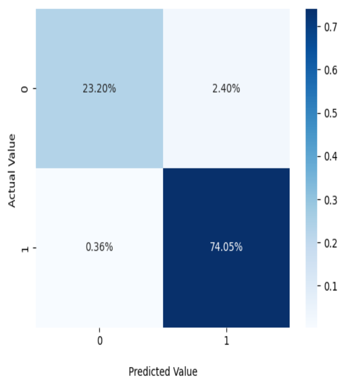
ron compared to ANN model deployed for the PDF dataset. The input layer has 512 neurons. There are 11 hidden layers with substantial number of nodes. For a binary classification output, the output layer has only one neuron. Analyzing the results, ANN has also produced slightly better test accuracy of 97.78%. The F1-score improved as well with a decent 98.51%. The ROC curve can be seen in figure 5.10c. The AUC value obtained from the ANN model is 97.07%.

Next, the proposed hybrid model is deployed for the PE dataset to verify that this model is indeed a fit for use solution for detecting malwares in files. Hybrid-3 model is constructed by combining ANN without the classification layer with RF for classification. The test accuracy yielded by Hybrid-3 is 97.94% that is current maximum accuracy achieved. Hybrid-3 model yields 98.61% of F1-score which confirms the quality of the classification for each classes of the dataset.

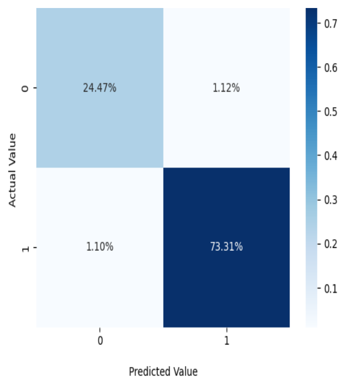
Finally, Hybrid-4 is the last model deployed, composed of putting AdaBoost on top of ANN feature extractor. Likewise, the PDF dataset, the hybrid model constructed with ANN and AdaBoost brings out the best results among all the models deployed with the test accuracy of 98.45%. The F1-score for Hybrid-4 is 97.62%. The ROC curves for the hybrid models are represented in the figure 5.11c. The AUC value obtained is 97.62%.



(a) Random Forest

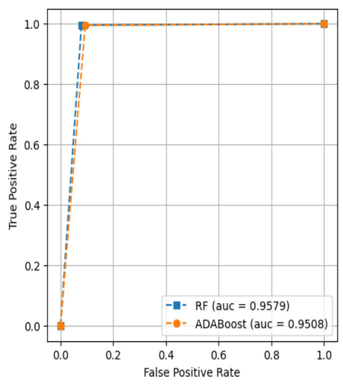


(b) AdaBoost Model

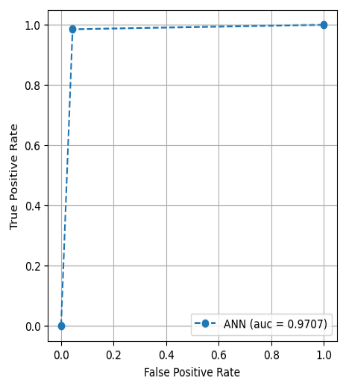


(c) ANN model

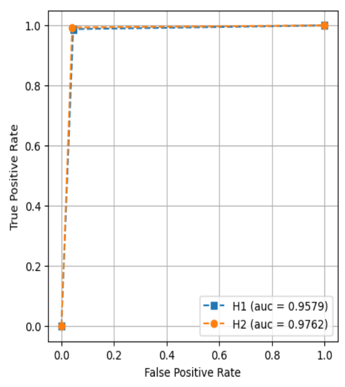
Figure 5.10: Confusion Matrix of different apporaches on PE dataset



(a) Ensemble Techniques



(b) ANN model



(c) Hybrid models

Figure 5.11: ROC curves for PE dataset

Table 5.6 represents a comparison of approaches of different models in the domain of malware detection. The techniques varies in the size of the dataset and detection approaches.

Table 5.6: Comparison table with other related works

Related Work	File	Dataset Size	Pre-processing	Model	Acc.
[19]	PDF	x	Static lexical Analysis	SVM	85%
[1]	PDF	4000	Convert binary data into 2-D image	NN based Clustering	94%
[23]	PDF	21,146	Keyword frequency, clustering	Random Forest	99%
[14]	PDF	Over 11,000	Data Labeling with clustering	Stacking Learning (SVM+RF+ MLP+AdaBoost) then (LR / KNN / DT)	99%
[17]	PE	5210	Merging of raw and boolean features	Random Forest	98%
[7]	PE	1800	Dynamic analysis to create JSON file of features and convert into binary	Artificial Neural Network	98%
This work	**PDF**	**10,000**	**Data transformation, Scaling, Handle Null values**	**Hybrid-2 (ANN + AdaBoost)**	**99%**
This work	**PE**	**20,000**	**Data transformation, Scaling, Handle Null Values**	**Hybrid-4 (ANN + AdaBoost)**	**99%**

5.6 Summary

Summarizing the entire picture, this project deploys different ML techniques like classical approaches, ensemble techniques, ANN and investigates the effectiveness of each techniques meticulously. ANN M3 model is discovered to be performing better than bagging and boosting approaches. Finally, hybrid models are engineered by integrating ANN with bagging and boosting techniques together surmounted all the other algorithms investigated in this project. The result obtained by this Hybrid-2 model is impeccable and can be refereed to as a potential solution for operation in malware detection.

At last, two ensemble techniques: RF and AdaBoost, ANN model, and two hybrid models composed of combining ANN and two ensemble techniques are applied on another dataset of PE data. The primary objective is to inspect whether the hybrid solutions can be used as a viable solution for different types of data. And it is experimentally found that, the Hybrid-4 model performed exceptionally and turned out to be the best performing algorithm among all the approaches.

Chapter 6

Conclusion

Diverse machine learning methods from several branches have been applied in this research to find malware in PDF and PE files. In pursuit of that, an extensive review of the existing work is carried out. The outcome of the literature review provides a broad overview of the domain including sources of the datasets utilized previously, performance of different models, metrics to evaluate the models. It becomes apparent that ensemble approaches like Random Forest and various variants of Neural Network perform remarkably well. To discover the superiority of the algorithms, at first, two classical techniques named Gaussian Naive Bayes and Logistic Regression have been applied. Next, two ensemble techniques named Random Forest (RF) and Adaptive Boosting (AdaBoost) have been applied to see if the results can be improved. And it is observed that the outcomes of the ensemble techniques have drastically improved from the classical approaches. Then, three variants of Artificial Neural Network have been deployed to improve the results even further. It has been a complex task to improve a system that has already performed with accuracy over 98%. However, the third ANN model M3 managed to produce better result that the previous best ensemble technique.

Finally, a novel solution is proposed combining the strengths of both ensemble techniques and ANN architecture. The novel approach extracts the features using ANN while the classification of the malwares is executed by the ensemble technique on the extracted features by ANN. Two hybrid models for each dataset have been deployed and all of the four hybrid models have generated satisfactory results.

For the PDF dataset, Hybrid-1 (ANN + RF) generates test accuracy and F1-score of 98.53% and 98.61% respectively. And the second hybrid model, Hybrid-2 (ANN + AdaBoost) achieves the best test accuracy of 99.51% and F1-score of 99.53%. For PE dataset, a deeper and more complex architecture is needed to train the model effectively. Interestingly, the result of Hybrid-4 (ANN + AdaBoost) also emerges better than Hybrid-3 (ANN + RF). Hybrid-3 has produced a test accuracy of 97.94% and F1-score of 98.61%. And, Hybrid-4 yields the best result for the PE dataset with a test accuracy of 98.45% and F1-score of 98.95%. The results clearly affirm that both the hybrid models can produce better results than individual execution of ANN and ensemble algorithms and also they are effective to learn essential features from the dataset and detect any new and unseen malwares efficiently.

Although, this project has been successful to generate satisfactory results. Still there are rooms for further improvements. The processing time of the models is not evaluated in this project. The processing time is also a good measure to evaluate the performance of a model. This can be calculated in future and can be further optimized by reducing the computation performed by the models.

Bibliography

[1] Irina Baptista, Stavros Shiaeles, and Nicholas Kolokotronis. "A novel malware detection system based on machine learning and binary visualization". In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE. 2019, pp. 1–6.

[2] Curtis Carmony et al. "Extract Me If You Can: Abusing PDF Parsers in Malware Detectors." In: *NDSS*. 2016.

[3] *Common Vulnerabilities and Exposures.* Accessed: 2023-03-11. URL: `https://cve.mitre.org/cve/search_cve_list.html`.

[4] *Contagio.* Accessed: 2023-07-15. URL: `https://contagiodump.blogspot.com/`.

[5] Gregory Conti et al. "A visual study of primitive binary fragment types". In: *White Paper, Black Hat USA* (2010).

[6] Andrew Corum, Donovan Jenkins, and Jun Zheng. "Robust PDF malware detection with image visualization and processing techniques". In: *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*. IEEE. 2019, pp. 108–114.

[7] Omid E David and Nathan S Netanyahu. "Deepsign: Deep learning for automatic malware signature generation and classification". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[8] Manuel Egele et al. "A survey on automated dynamic malware-analysis techniques and tools". In: *ACM computing surveys (CSUR)* 44.2 (2008), pp. 1–42.

[9] Raphael Fettaya and Yishay Mansour. "Detecting malicious PDF using CNN". In: *arXiv preprint arXiv:2007.12729* (2020).

[10] Dragos Gavrilut et al. "Malware detection using perceptrons and support vector machines". In: *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. IEEE. 2009, pp. 283–288.

[11] Valentin Hamon. "Malicious URI resolving in PDF documents". In: *Journal of Computer Virology and Hacking Techniques* 9.2 (2013), pp. 65–76.

[12] Nwokedi Idika and Aditya P Mathur. "A survey of malware detection techniques". In: *Purdue University* 48.2 (2007), pp. 32–46.

[13] Rafiqul Islam et al. "Classification of malware based on string and function feature selection". In: *2010 Second Cybercrime and Trustworthy Computing Workshop*. IEEE. 2010, pp. 9–17.

[14] Maryam Issakhani et al. "PDF Malware Detection based on Stacking Learning." In: *ICISSP*. 2022, pp. 562–570.

[15] Kazumasa Itabashi. "Portable document format malware". In: *Symantec white paper* (2011).

[16] Jagpreet Kaur and KR Ramkumar. "The recent trends in cyber security: A review". In: *Journal of King Saud University-Computer and Information Sciences* 34.8 (2022), pp. 5766–5781.

[17] Ajit Kumar, KS Kuppusamy, and Gnanasekaran Aghila. "A learning model to detect maliciousness of portable executable using integrated feature set". In: *Journal of King Saud University-Computer and Information Sciences* 31.2 (2019), pp. 252–265.

[18] Sumeet Kumar and Kathleen M Carley. "Approaches to understanding the motivations behind cyber attacks". In: *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*. IEEE. 2016, pp. 307–309.

[19] Pavel Laskov and Nedim Šrndić. "Static detection of malicious JavaScript-bearing PDF documents". In: *Proceedings of the 27th annual computer security applications conference*. 2011, pp. 373–382.

[20] Yuchong Li and Qinghui Liu. "A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments". In: *Energy Reports* 7 (2021), pp. 8176–8186. ISSN: 2352-4847. DOI: `https://doi.org/10.1016/j.egyr.2021.08.126`. URL: `https://www.sciencedirect.com/science/article/pii/S2352484721007289`.

[21] *Logistic Regression in Machine Learning*. Accessed: 2023-07-11. URL: `https://www.javatpoint.com/logistic-regression-in-machine-learning`.

[22] Davide Maiorca, Battista Biggio, and Giorgio Giacinto. "Towards adversarial malware detection: Lessons learned from PDF-based attacks". In: *ACM Computing Surveys (CSUR)* 52.4 (2019), pp. 1–36.

[23] Davide Maiorca, Giorgio Giacinto, and Igino Corona. "A pattern recognition system for malicious pdf files detection". In: *International workshop on machine learning and data mining in pattern recognition*. Springer. 2012, pp. 510–524.

[24] *Master the AdaBoost Algorithm: Guide to Implementing Understanding AdaBoost*. Accessed: 2023-07-11. URL: `https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/`.

[25] Akram M Radwan. "Machine learning techniques to detect maliciousness of portable executable files". In: *2019 International Conference on Promising Electronic Technologies (ICPET)*. IEEE. 2019, pp. 86–90.

[26] Rajeev DS Raizada and Yune-Sang Lee. "Smoothness without smoothing: why Gaussian naive Bayes is not naive for multi-subject searchlight studies". In: *PloS one* 8.7 (2013), e69566.

[27] Karthik Selvaraj and Nino Fred Gutierrez. "The rise of PDF malware". In: *Symantec Security Response* (2010).

[28] Ronghua Tian, Lynn Margaret Batten, and SC Versteeg. "Function length as a tool for malware classification". In: *2008 3rd international conference on malicious and unwanted software (MALWARE)*. IEEE. 2008, pp. 69–76.

[29] Ronghua Tian et al. "An automated classification system based on the strings of trojan and virus families". In: *2009 4th International conference on malicious and unwanted software (MALWARE)*. IEEE. 2009, pp. 23–30.

[30] Jose Torres and Sergio De Los Santos. "Malicious PDF documents detection using machine learning techniques". In: *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*. 2018, pp. 337–344.

[31] Dolly Uppal et al. "Malware detection and classification based on extraction of API sequences". In: *2014 International conference on advances in computing, communications and informatics (ICACCI)*. IEEE. 2014, pp. 2337–2342.

[32] Cristina Vatamanu, Dragoş Gavriluţ, and Răzvan Benchea. "A practical approach on clustering malicious PDF documents". In: *Journal in Computer Virology* 8.4 (2012), pp. 151–163.

[33] *Virus total.* Accessed: 2023-07-15. URL: `https://www.virustotal.com/gui/home/upload`.

[34] *What is a Neural Network?* Accessed: 2023-07-11. URL: `https://www.tibco.com/reference-center/what-is-a-neural-network`.

[35] *What is a Random Forest?* Accessed: 2023-07-11. URL: `https://www.tibco.com/reference-center/what-is-a-random-forest`.

[36] Muhammad Irfan Yousuf et al. "Multi-feature Dataset for Windows PE Malware Classification". In: *arXiv preprint arXiv:2210.16285* (2022).

[37] Jason Zhang. "MLPdf: an effective machine learning based approach for PDF malware detection". In: *arXiv preprint arXiv:1808.06991* (2018).

Appendix A

PDF Dataset

Table A.1: Name and Type of the features in the dataset

Sl. No	Feature Name	Type of Feature
1	Name of the PDF	General feature
2	Size of the PDF	General feature
3	Metadata size	General feature
4	No. of pages in the PDF	General feature
5	No. of Xref entries	Structural feature
6	No. of characters in the title	General feature
7	Encryption applied	General feature
8	No. of embedded files inside the document	General feature
9	No. of images in the document	General feature
10	Presence of text inside PDF	General feature
11	PDF header	General feature
12	No. of objects in the PDF	General feature
13	No. of keywords "endobj"	Structural feature
14	Average stream size	Structural feature
15	No. of keyword "endstream"	Structural feature
16	No. of Xref entries	Structural feature
17	No. of keyword "/Trailer"	Structural feature
18	No. of keyword "/Startxref"	Structural feature
19	No. of pages in the PDF	General feature
20	No. of keyword "encrypt"	General feature
21	No. of stream objects	Structural feature
22	No. of keyword "/JS"	Structural feature
23	No. of keyword "/JavaScript"	Structural feature
24	No. of keyword "ÄA"	Structural feature

Table A.1: Name and Type of the features in the dataset

Sl. No	Feature Name	Type of Feature
25	No. of keyword "OpenAction"	Structural feature
26	No. of keyword Acrobat forms	Structural feature
27	No. of keyword "/JBIG2Decode"	Structural feature
28	No. of keyword "/Richmedia"	Structural feature
29	No. of keyword "/launch"	Structural feature
30	No. of embedded files	General feature
31	No. of keyword "/XFA"	Structural feature
32	No. of colors used in the PDF	Structural feature

A.1 Correlation Matrix of the Features

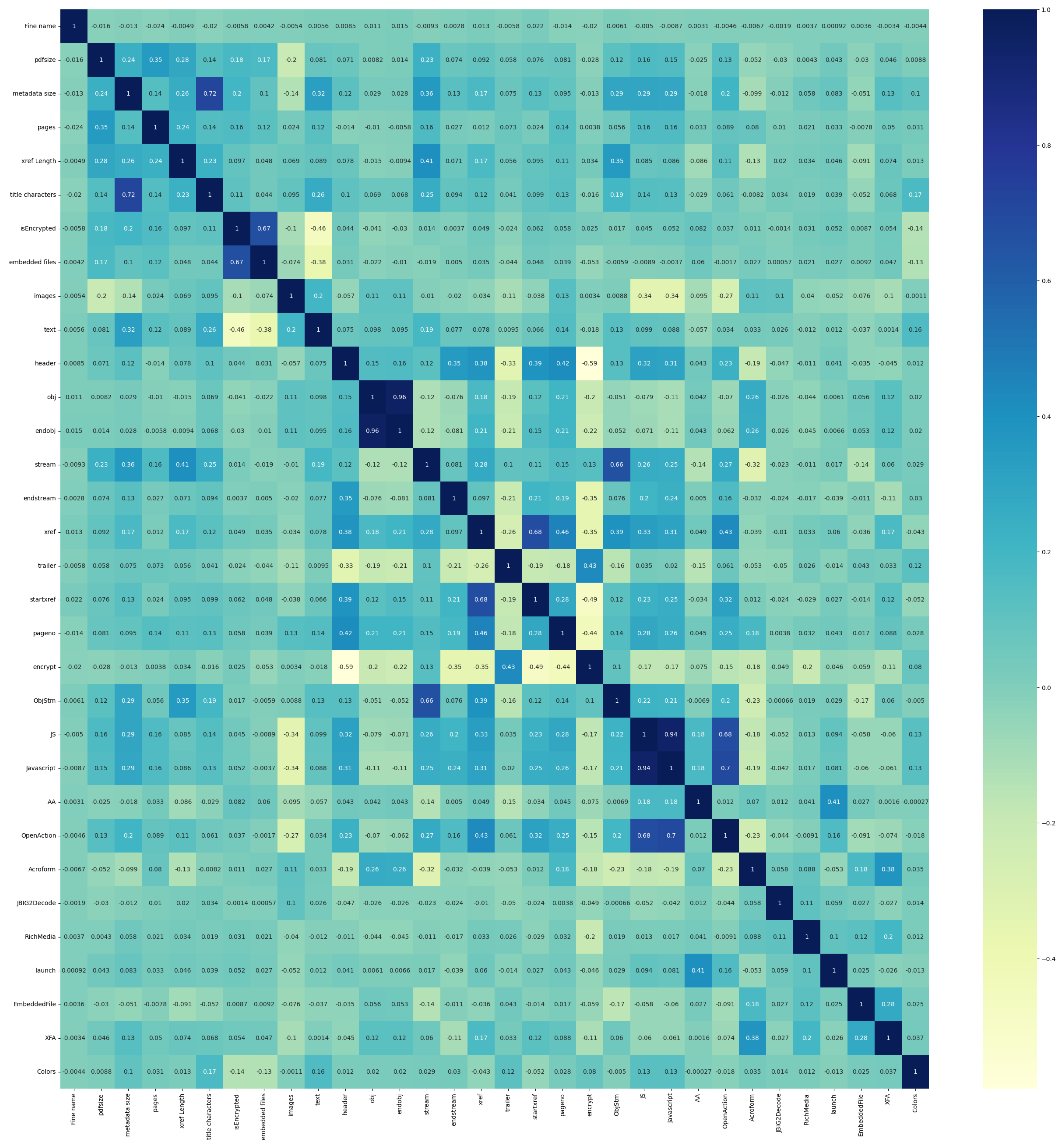
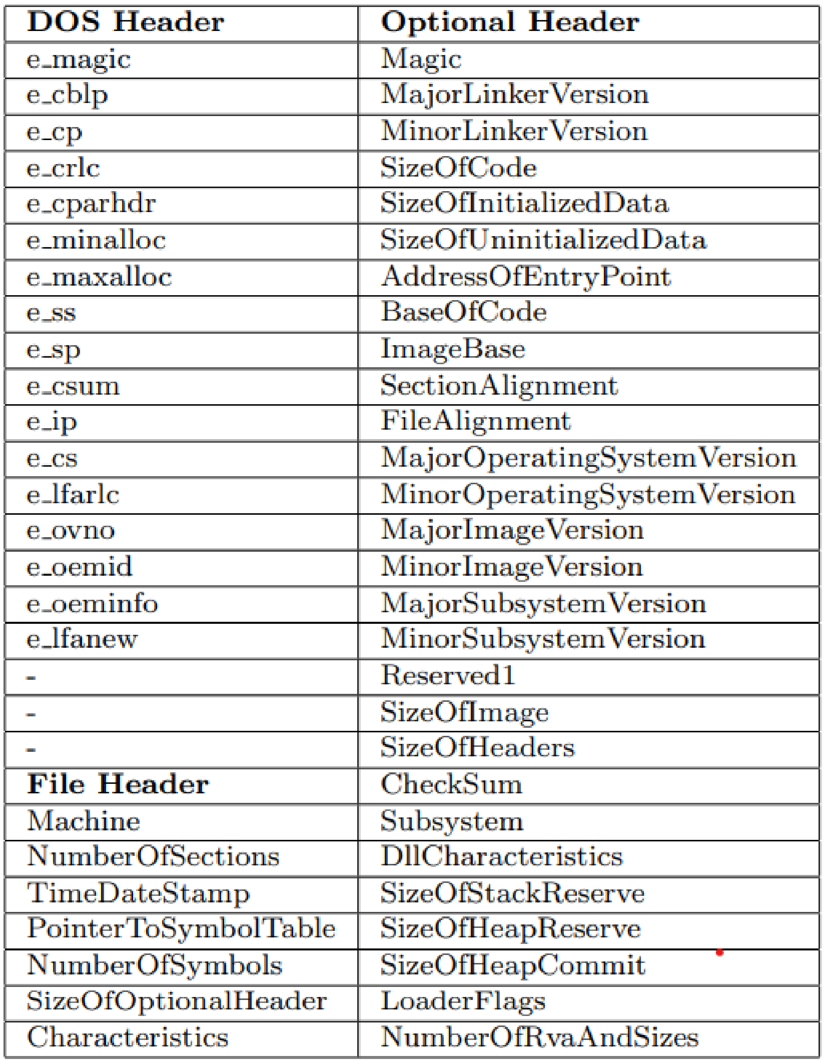


Figure A.1: Correlation of features

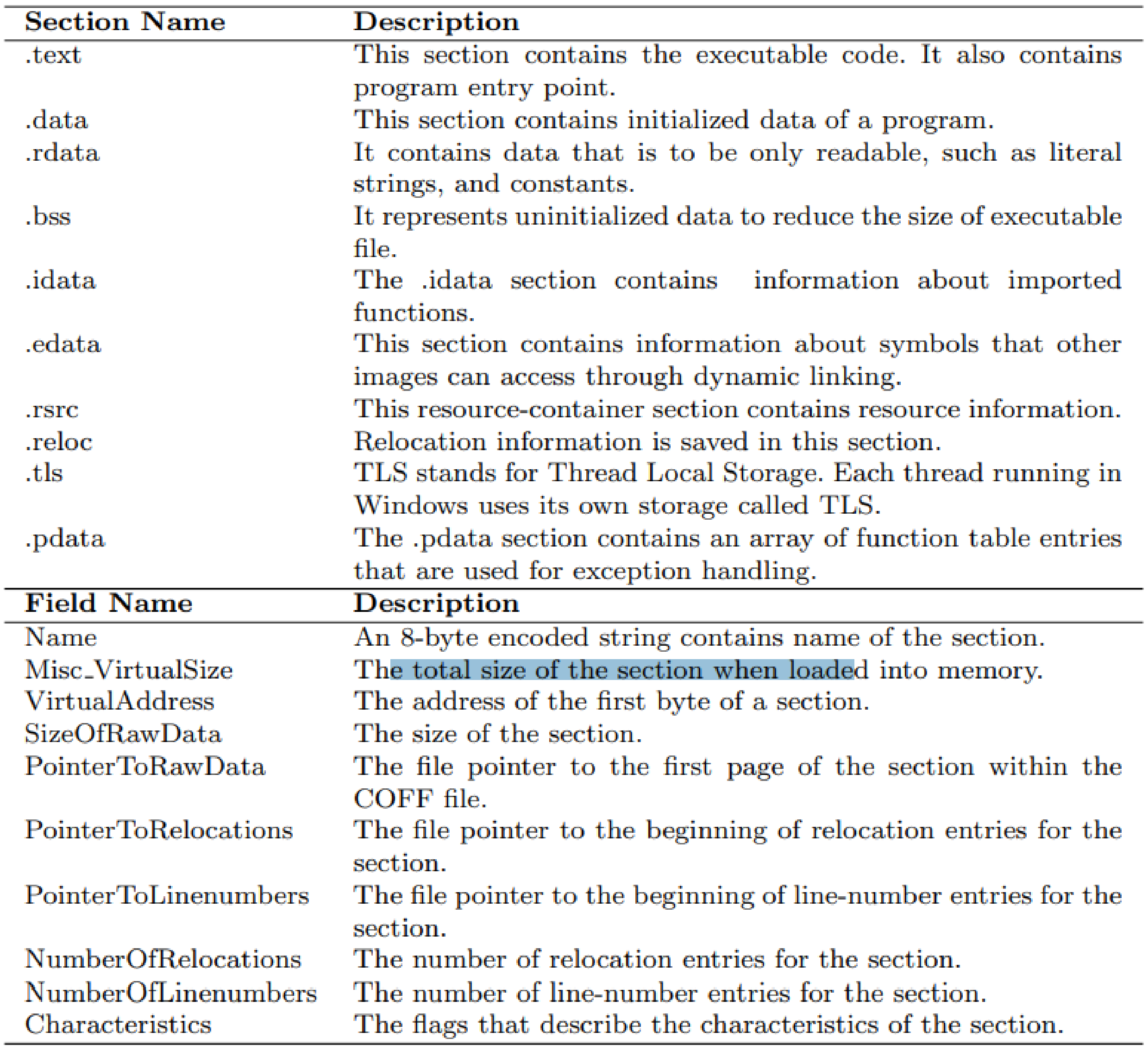
Appendix B

PE Dataset



DOS Header	**Optional Header**
e_magic	Magic
e_cblp	MajorLinkerVersion
e_cp	MinorLinkerVersion
e_crlc	SizeOfCode
e_cparhdr	SizeOfInitializedData
e_minalloc	SizeOfUninitializedData
e_maxalloc	AddressOfEntryPoint
e_ss	BaseOfCode
e_sp	ImageBase
e_csum	SectionAlignment
e_ip	FileAlignment
e_cs	MajorOperatingSystemVersion
e_lfarlc	MinorOperatingSystemVersion
e_ovno	MajorImageVersion
e_oemid	MinorImageVersion
e_oeminfo	MajorSubsystemVersion
e_lfanew	MinorSubsystemVersion
-	Reserved1
-	SizeOfImage
-	SizeOfHeaders
File Header	CheckSum
Machine	Subsystem
NumberOfSections	DllCharacteristics
TimeDateStamp	SizeOfStackReserve
PointerToSymbolTable	SizeOfHeapReserve
NumberOfSymbols	SizeOfHeapCommit
SizeOfOptionalHeader	LoaderFlags
Characteristics	NumberOfRvaAndSizes

Figure B.1: PE header fields in the dataset [36]



Section Name	Description
.text	This section contains the executable code. It also contains program entry point.
.data	This section contains initialized data of a program.
.rdata	It contains data that is to be only readable, such as literal strings, and constants.
.bss	It represents uninitialized data to reduce the size of executable file.
.idata	The .idata section contains information about imported functions.
.edata	This section contains information about symbols that other images can access through dynamic linking.
.rsrc	This resource-container section contains resource information.
.reloc	Relocation information is saved in this section.
.tls	TLS stands for Thread Local Storage. Each thread running in Windows uses its own storage called TLS.
.pdata	The .pdata section contains an array of function table entries that are used for exception handling.

Field Name	Description
Name	An 8-byte encoded string contains name of the section.
Misc_VirtualSize	The total size of the section when loaded into memory.
VirtualAddress	The address of the first byte of a section.
SizeOfRawData	The size of the section.
PointerToRawData	The file pointer to the first page of the section within the COFF file.
PointerToRelocations	The file pointer to the beginning of relocation entries for the section.
PointerToLinenumbers	The file pointer to the beginning of line-number entries for the section.
NumberOfRelocations	The number of relocation entries for the section.
NumberOfLinenumbers	The number of line-number entries for the section.
Characteristics	The flags that describe the characteristics of the section.

Figure B.2: PE sections and fields in the dataset [36]