

# PDFGuardian: An innovative approach to Interpretable PDF Malware Detection using XAI with SHAP Framework

by

Tahsinur Rahman

19101146

Nusaiba Ahmed

19101236

Shama Monjur

18201125

Fasbeer Mohammad Haque

19101269

Naweed Kabir

19101053

A thesis submitted to the Department of Computer Science and Engineering in  
partial fulfillment of the requirements for the degree of B.Sc. in Computer Science  
and Engineering

Department of Computer Science and Engineering  
School of Data and Sciences  
BRAC University  
January 2023

©2023. BRAC University  
All rights reserved.

# Declaration

It is hereby declared that


1. The thesis submitted is our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

## Student's Full Name & Signature:



---

Tahsinur Rafsan  
19101146



---

Nusaiba Ahmed  
19101236



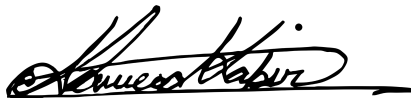
---

Shama Monjur  
18201125



---

Fasbeer Mohammad Haque  
19101269



---

Naweed Kabir  
19101053

# Approval

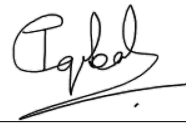
The thesis/project titled “PDFGuardian: An innovative approach to Interpretable PDF Malware Detection using XAI with SHAP Framework ” submitted by

1. Tahsinur Rahman(19101146)
2. Nusaiba Ahmed (19101236)
3. Shama Monjur (18201125)
4. Fasbeer Mohammad Haque (19101269)
5. Naweed Kabir (19101053)

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 26, 2023.

## Examining Committee:

Supervisor:  
(Member)



---

Dr. Muhammad Iqbal Hossain  
Associate Professor  
Department of Computer Science and Engineering  
BRAC University

Program Coordinator:  
(Member)

---

Dr. Md. Golam Rabiul Alam  
Professor  
Department of Computer Science and Engineering  
BRAC University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi, PhD  
Chairperson and Associate Professor  
Department of Computer Science and Engineering  
Brac University

# Abstract

As the world is moving more and more towards a digital era, a great majority of data is transferred through a famous format known as PDF. One of its biggest obstacles is still the age-old problem: malware. Even though several anti-malware and anti-virus software exist, many of which cannot detect PDF Malware. Emails carrying harmful attachments have recently been used in targeted cyber attacks against businesses. Because most email servers do not allow executable files to be attached to emails, attackers prefer to use non-executable files like PDF files. In various sectors, machine learning algorithms and neural networks have been proven to successfully detect known and unidentified malware. However, it can be difficult to understand how these models make their decisions. Such lack of transparency can be a problem, as it is important to understand how an AI system is making decisions in order to ensure that it is acting ethically and responsibly. In some cases, machine and deep learning models may make biased or discriminatory decisions or have unintended consequences. Hence, Explainable AI comes into play. To address this issue, this paper suggests using machine learning algorithms SGD(Stochastic Gradient Descent), XGBoost Classifier, and deep learning algorithms Single Layer Perceptron, ANN(Artificial Neural Network) and check their interpretability using Explainable AI (XAI)'s SHAP framework to classify a PDF file being malicious or clean for a global and local understanding of the models.

**Keywords:** malware, PDF, PDF-analysis, cybersecurity, SGD, machine-learning, detection, deep learning, artificial neural network, algorithm, Single Layer Perceptron, Extreme Gradient Boosting, Explainable artificial intelligence, Shapley Additive Explanations, ANN, SHAP, XAI, XGBoost, classifiers

## **Dedication**

We would like to dedicate this thesis to our parents, who have supported us in every step of our lives, our dearest supervisor who has guided us through this entire thesis and motivated us to push forward in research and lastly to our faculties who have provided valuable responses and given time and effort into making the best versions of ourselves and the thesis.

## **Acknowledgement**

First and foremost, glory be to the Great Allah, with whose help we were able to finish writing our thesis without any issues.

Secondly, we would like to thank our supervisor, Dr. Muhammad Iqbal Hossain, for his kind advice and support. He came to our aid anytime we needed it.

Thirdly, we would like to thank Faisal Ahmed Sir for his assistance and guidance during the whole research study. He never hesitated to provide a hand and generously donated his time to make suggestions and insightful criticism.

And lastly, our parents, without whom it could not have been possible. We are currently preparing to graduate thanks to their kind prayers and support.

# Table of Contents

Declaration	i
Approval	ii
Abstract	iii
Dedication	iv
Acknowledgement	v
Table of Contents	vi
List of Figures	viii
List of Tables	ix
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.2.1 Static Analysis: . . . . .	2
1.2.2 Dynamic Analysis: . . . . .	2
1.3 Objective and Contributions . . . . .	3
<b>2 Literature Review and Related Work</b>	<b>4</b>
2.1 Literature Review . . . . .	4
2.1.1 Malware Injection Techniques in PDF Files . . . . .	4
2.1.2 PDF Structure and its Vulnerable Features . . . . .	5
2.2 Algorithms/Models/Existing Techniques . . . . .	7
2.3 Explainable AI and Its Significance . . . . .	9
2.3.1 Choice of XAI Framework . . . . .	10
<b>3 Methodology</b>	<b>11</b>
3.1 Dataset Description . . . . .	11
3.1.1 Data Collection and Extraction . . . . .	11
3.1.2 Dataset Pre-processing . . . . .	14
3.2 Proposed Model/Workplan . . . . .	17
3.2.1 Work Plan . . . . .	18
3.3 Model Implementation Review . . . . .	20
3.3.1 Perceptron Model . . . . .	20
3.3.2 Artificial Neural Network . . . . .	23

3.3.3	XGBoost Classifier . . . . .	27
3.3.4	Stochastic Gradient Descent(SGD) . . . . .	30
<b>4</b>	<b>Results and Discussion</b>	<b>33</b>
4.1	Model Evaluation . . . . .	33
4.2	Model Classification Analysis . . . . .	34
4.3	SHAP Analysis . . . . .	37
4.3.1	Perceptron . . . . .	39
4.3.2	SGD . . . . .	41
4.3.3	XGBoost Clasifier . . . . .	43
4.3.4	ANN . . . . .	44
4.4	The Analogy of SHAP Analysis of Models . . . . .	45
<b>5</b>	<b>Conclusion and Future Work</b>	<b>46</b>
	<b>Reference</b>	<b>49</b>



# List of Figures

3.1	Pseudocode Part 1 . . . . .	12
3.2	Pseudocode Part 2 . . . . .	13
3.3	Pseudocode Part 3 . . . . .	13
3.4	Pseudocode Part 4 . . . . .	13
3.5	KDE plots before transformation . . . . .	14
3.6	KDE plots after transformation . . . . .	15
3.7	Correlation Heatmap . . . . .	16
3.8	Chi-square Test . . . . .	17
3.9	Work Plan . . . . .	19
3.10	Perceptron Confusion Matrix . . . . .	21
3.11	Perceptron ROC . . . . .	22
3.12	Model Accuracy of ANN . . . . .	24
3.13	Model Loss of ANN . . . . .	24
3.14	ANN Confusion Matrix . . . . .	25
3.15	ANN ROC . . . . .	26
3.16	XGBoost Confusion Matrix . . . . .	28
3.17	XGBoost ROC . . . . .	29
3.18	SGD Confusion Matrix . . . . .	31
3.19	SGD ROC . . . . .	32
4.1	Force Plot Perceptron . . . . .	39
4.2	Waterfall Plot Perceptron . . . . .	39
4.3	Feature Clustering Plot Perceptron . . . . .	40
4.4	Feature Importance Plot Perceptron . . . . .	40
4.5	Force Plot SGD . . . . .	41
4.6	Waterfall Plot SGD . . . . .	41
4.7	Feature Clustering Plot SGD . . . . .	42
4.8	Feature Importance Plot SGD . . . . .	42
4.9	Force Plot XGBoost . . . . .	43
4.10	Feature Clustering Plot XGBoost . . . . .	43
4.11	Feature Importance Plot XGBoost . . . . .	43
4.12	ANN Decision Plot . . . . .	44

# List of Tables

3.1	Perceptron Accuracy Scores . . . . .	21
3.2	Perceptron Classification . . . . .	21
3.3	ANN Accuracy Scores . . . . .	25
3.4	ANN Classification . . . . .	26
3.5	XGBoost Accuracy Scores . . . . .	28
3.6	XGB Classification . . . . .	29
3.7	SGD Accuracy Scores . . . . .	31
3.8	SGD Classification . . . . .	32
4.1	Model Computational Report . . . . .	34
4.2	Complete Classification Report . . . . .	35
4.3	Contingency Classification Report . . . . .	36

# Chapter 1

## Introduction

Portable Document Format (PDF), was created by Adobe in 1992 and is currently one of the most widely used formats for documentation. It is completely free to use and supports procedures that need encryption and digital signatures, as well as file attachments and metadata. Businesses as well as individuals use PDF files to distribute and save information in the United States and throughout the world. It is how invoices, contracts, and a variety of other business papers are frequently sent. E-commerce benefits greatly from modern technology such as PDF. It also consists of a broad variety of documents, including publications, catalogs, tax forms, and infographics. Because of this versatility, photos, text, e-Signatures, videos, website links, and a variety of other types of content are also used. With this wide array of usage, PDF files have become an easy tool for spreading malware. To detect such malware, ML, deep learning, and neural network are some effective tools that can be applied. The applications and results of these models can be explained further using the SHAP framework of XAI. With the popularity of data science-based solutions increasing, it has become an important tool in the field of artificial intelligence for understanding the behavior of machine learning models and for providing interpretable and transparent explanations of their predictions. A dataset of PDF files was collected and labeled as either malicious or clean to evaluate the effectiveness of using XAI with these machine and deep learning algorithms. The SGD, XGBoost, Single Layer Perceptron, ANN, and deep learning algorithms were then trained and tested on the dataset using the SHAP framework to provide explanations for their predictions. The results of the experiment showed that the use of XAI with these algorithms was effective in detecting PDF malware with a high degree of accuracy. Additionally, the white box SHAP explanations provided insight into the decision-making process of the models, allowing for a better understanding of their predictions and any potential biases or assumptions they may have made.

### 1.1 Motivation

With the rapid development of technology, people around the world have adapted to an extensive range of devices and software for their daily uses for business and educational purposes. Due to the COVID upsurge since 2019, the use of digital documentation software has increased. Businesses have shifted their information and work materials online [19], and educational institutions have started to take assignments and project presentations in the form of PDF files. Security bank

statements to form fill-up printing documents have been passed around from person to person using PDF files. Moreover, because it is one of the most widely used ways to share or transmit information, phishing and other sorts of assaults have also targeted it. Malware can cause great damage to a computer's network. It is used by hackers to steal passwords, destroy information, and disable machines. This malware can steal critical data, slow down the computer, interrupt everyday operations, restrict file access, and, if not stopped, rapidly propagate throughout the network. It is very difficult for antivirus systems to detect complex embedded malicious codes as it is hidden inside the functions of PDF. Thus, it is crucial to prevent malware in such a useful tool as PDF.

## 1.2 Problem Statement

Malware detectors, such as viruses and spyware scanners, are the most common kind of malware protection. Unfortunately, both researchers and malware developers have demonstrated that these scanners, which identify malware by matching patterns, may be attacked by simple code. These technologies depend on semantic signatures and static analysis techniques like model checking and theorem-proving transformations to detect threats. However, attackers are trying hard to hide the malicious program, resulting in dynamic signatures for their attacks. Because of the nature of PDF and its support for embedded JavaScript, attackers have a wide range of choices for developing dynamic exploit code, including techniques such as embedded objects, dynamic and complex code, encoding, and cryptography. As a result, it can distinguish two types of problems.

### 1.2.1 Static Analysis:

Binary obfuscation is a technique that aims to obscure the true application code to make it difficult for an outsider who does not have access to the sources to understand what the program is supposed to do [1].

- The bulk of past obfuscation methods has included working on the program's source code.
- It is not always easy to get hold of source code.
- Malware developers utilize NP-Hard problems and opaque constants. The use of opaque constants in a method may be developed to conduct a variety of modifications to obscure a program's control flow, data locations, and data use.
- Lack of skilled personnel to conduct the analysis effectively.

### 1.2.2 Dynamic Analysis:

- Malware may operate differently in the safe environment than it does in the real-world run time environment.
- To be monitored, the malware sample must be run in a secure environment for a specific length of time.

- The monitoring procedure takes time, and it must verify that execution malware does not infiltrate the platform [21].
- Difficult and time-consuming to pinpoint the precise location of an error [21].

In simpler terms, we would encounter both static and dynamic difficulties that must be resolved concurrently to protect PDF files against malware threats.

### 1.3 Objective and Contributions

Machine learning and deep learning models have been helping the human world greatly without any doubt. As past studies have suggested the use of certain algorithms, it is yet important to decide whether one should be relying on the models or not especially in such a field of malware detection in cybersecurity. Artificial intelligence (AI) systems that have the ability to give justifications for their choices and behaviors are referred to as explainable AI. This is crucial because it enables users to comprehend how an AI system makes decisions, boosting user confidence and ensuring that the system is behaving ethically and responsibly. Given that it enables programmers to comprehend how the system came at a given choice, explainable AI can also aid in the detection and correction of AI system flaws. Overall, explainable AI is essential to guaranteeing the justice, accountability, and transparency of AI systems. The purpose of this research is to develop a detection system for PDF malware, by using machine learning models: SGD, XGBoost Classifier, Single Layer Perceptron, and an ANN model which would be explained using Explainable AI. The aim is to use different models to figure out whether a file is malicious or not. The models' abilities to distinguish between clean and malicious PDF files are observed and compared. The main objectives are the following:

1. Understanding malware and how it works
2. How malware is embedded and how it can be prevented
3. Understanding dataset parameters for supervising the model for malware detection
4. Develop the Machine Learning model and evaluate the model based on its performance
5. Use SHAP Explainers to check each model's interpretation and its feature contributions.

# Chapter 2

## Literature Review and Related Work

### 2.1 Literature Review

A few basics of certain issues must be covered to properly understand the solution for the malware invasion. This includes how a PDF file works, where malicious writers can insert or disguise their code, and how an attacker employs the malware-producing technique. It's crucial to comprehend the benefits of applying machine learning and deep learning to the problem as well as how to deal with it utilizing the current static and dynamic techniques of malware insertion as well as how Explainable AI helps make better decisions.

#### 2.1.1 Malware Injection Techniques in PDF Files

Malware is a malicious software that infiltrates a computer by posing as a genuine program. Phishing emails, fraudulent installers, infected attachments, and phishing websites are the most prevalent ways for them to be installed [17]. To persuade consumers to install malware, hackers make it look appealing. Since the application appears to be legitimate, most people are unaware that it is malware. In fact, antivirus programs are unable to identify the problem, the hidden codes blend in seamlessly into the appearance of a typical PDF file. This may be accomplished by including seemingly harmless links and attachments that connect to their system. As attackers' techniques for creating malicious code have become more sophisticated, various studies have been conducted to detect and analyze malicious code. Malware is classified into two types: executables and non-executables [17]. There are many ways in which malicious files are injected. Therefore malware detection is required. Malware detection is a technique that is designed to detect malware. It is done in several ways such as Signature-Based Detection, Heuristic Analysis, Sandbox, etc [24].

In order to figure out and develop an effective mechanism it is important to consider how the malware interacts with the PDF formats. In general, PDF files are an easy target due to the fact that PDF files are generally non-executable files, i.e. without an active application or feature running. Most users consider PDF files to be static documents that contain only output from an executable program and not itself, for

example, the output of a JavaScript program in the background [18]. Most PDF attacks use built-in features or embedded codes, executed by the PDF reader, to exploit vulnerabilities. While there are many that fall under that criteria, the most prevalent scenarios are as follows:

An action is considered a regular feature in a PDF. This has led to the use of the OpenAction feature to set the malware or exploit to be triggered by opening the PDF, as it is disguised as a normal action. [4], [5].

Another method is to utilize the Launch action, which offers the operating system special commands that allow it to launch an application when the user clicks OK on the confirmation prompt [4].

Arguably, the most difficult to combat is the use of GotoEmbedded actions. This is the process of hiding encrypted, malware-affected PDF files or hyperlinks [3] within the main PDF. This will load the hidden PDF file as soon as the main PDF file is loaded. This type of threat is difficult to prevent as the main PDF is harmless and hence the embedded files slip through along with the main file [4], [5].

A final common scenario is the usage of Universal Resource indicators to point users to another infected source [9]. The most common use of it is to redirect users to malicious websites, or steal personal data from that particular device [18]. This is mostly done by using multiple functions like JavaScript and OpenAction.

### 2.1.2 PDF Structure and its Vulnerable Features

The overall structure of a PDF file must be comprehended to understand how information is stored inside it and which sections are vulnerable. The trailer, header, cross-reference, and object are the four main components. The header is followed by the body, which contains even more elements. Numbers, strings, streams, dictionaries, and booleans make up the majority of the data. In this case, streams are PDF files, which often contain simple text, fonts, and graphics, as well as JavaScript. A list of objectives may be found on the cross-reference board. The trailer section at the conclusion comprises PDF files and cross-reference table positioning data[11]. A PDF file has a lot of features that operate in layers. However, not all features are vulnerable. Features like header storage and unique address storage have an extremely low chance of being attacked. Therefore, only a few features with their susceptibility are researched.

- **JavaScript and JS-** A JavaScript action is an object that contains JavaScript code and either a string or a pointer to a string or stream object. The PDF standard enables the assessment of JavaScript operations at various times during the life of a PDF file within a PDF reader [6]. Obfuscation is a method of disguising the true content of data or the true functioning of program code. A PDF document can be obfuscated to conceal numerous of its attributes [6]. JavaScript code, on the other hand, maybe obfuscated to limit readability. This can, in turn, reduce the static analysis of the PDF files [14].

- **Open Action-** When creating a PDF document, OpenOffice and LibreOffice commonly include OpenAction. Its function is to display the first page of the document when you open it for the first time [22]. When files run, the open action can point to a JavaScript or JS file in order to run. An attacker can easily inject malicious code in this section using the “GO TO R” or “GO TO E” methods. The “GO TO R” method usually includes links that, when a file is opened, will lead the file into opening a link or URL from which the attacker can work. The “GO TO E” causes the file to go into an embedded file that runs hidden without the user noticing [20].
- **AA-** Using a Universal Resource Indicator, URI action provides access to external resources. By combining that functionality with Javascript, OpenAction, or PDF forms, an attacker can trick a user into visiting a hostile website or stealing data. This action is often started with AA which is an automatic action [22]. So, if an attacker uses the feature, he can automatically refer to any site or file that contains a virus without the user noticing. This is usually the case with PDF files that are opened via a web browser such as google chrome or firefox.
- **Object, EndObject, Objstream, and Trailer-** The PDF document is divided into header, body, cross-reference table, and trailer sections. The body contains objects that carry insight into the actual content [10], [3]. The trailer holds the root object as well as cross-reference table position data for the objects in the body section. PDF apps(for example, PDF readers) support executing JavaScript code inside the file which implies that any function can be executed dynamically using the JavaScript API [10]. Stream objects have no size limits, and some part of the stream performs malicious behaviors. Therefore, the attacks are usually hidden inside objects, which allows us the first identification of a possible malicious attack.
- **Startxref Xref-** The abbreviation Xreff stands for external references. A list of objects along with their location and status within the file is contained in the Xref table, also known as the cross-reference table [4]. The Xref tool, as the name suggests, enables you to add external references to your design. Other drawings, PDF files, photos, point cloud data, and other types of external references are also possible. A Startxref, the location of the final Xref table, and %%EOF, which should finish the file, are all included after the trailer [10]. The Startxref returns the URL to the specified process link. These can be checked for the presence of suspicious content and cut the search time down, as the Xref and Startxref tables will contain references of the entire layer system of PDF, much like the table of contents inside any document.
- **Stream, EndStream, and Encrypt-** The stream is a sequence of bytes of binary data that vary in length. Normally, the stream is expected to contain large image files or page composition objects [19]. It is harder to detect embedded code in files since streams can contain all kinds of data, including scripts and binary files, and they can also be encrypted and encoded. The name/filter specifies the compression method. More than one filter may be present in a stream. A PDF file may be encrypted for extra security, in which case a password is required to read or alter the contents. The encryption could



easily contain such passwords which could be used by the user for other fields. When attackers enable and edit passwords, they can not only view content but also prevent the user from accessing the PDF or modifying contents by injecting a new password. Shellcode encryption is one of the most important forms of encryption-based attacks [12]. The security of such PDF files is often overlooked as it easily hides under the layers that systems perceive to be their own.

- **Page**- Pages are the least susceptible to any malware injection. However, it can be a key indicator of whether a file contains malware or not. For instance, if a PDF file's page count is larger than the actual page numbers for its contents, it may indicate that some material has been concealed, which may be a sign of potential attacks inside subsequent layers.
- **XFA**- XML Forms Architecture (XFA) offers a template that enables fillable fields and establishes the form's design or looks [6]. It may be used to update forms, perform calculations, verify output, and preserve modifications for forms that have been filled out because it is embedded in PDF. In many cases, ignoring XFA resulted in false positives as identifiers could not recognize the format and hence previous works sometimes considered it to be suspicious when it was not.

One of the most prevailing attacks is against PDF files, which are more versatile than other document formats. The majority of malicious PDF documents contain binary or JavaScript code that exploits vulnerabilities and causes damage [19]. Experts have been making great efforts in PDF manipulation for more than a decade. The majority of talks focused on the worrying rise in these cyberattacks and how the perpetrators made use of a PDF file's structure. Only a few of the many recommended methods were implemented. New methods for a getaway have been found in more recent investigations. However, the field falls short in terms of precision and general considerations of many forms of attacks. In previous research, combining static and dynamic features increased the detection rate of hazardous Mobile Apps, and it is worth investigating in the context of PDF malware. This hypothesizes that combining static, dynamic, and hardware features can increase the evasion resistance of the classifier [18].

## 2.2 Algorithms/Models/Existing Techniques

The most basic and prevalent way of detecting malware is to compare files with traits of known malware and sniff out traces. While this leads to quick results, the trade-off is accuracy and it is not difficult to bypass this method. So, there are 4 methods used today that are relatively much more effective than the rest.

There are some well-known PDF-based attacks:

- OpenAction
- Launch action
- Embedded files

- GotoEmbedded action
- URI action

While there are other ways available, they tend to focus on 21 properties that are typically seen in malicious files (obj, endobj, stream, endstream, xref, trailer, startxref, /Page, /Encrypt, /ObjStm, /JS, /JavaScript, /AA, /OpenAction, /AcroForm, /JBIG2Decode, /RickMedia).

Some well-known methods used to detect such anomalies are explained below:

1. **Static Analysis:** This is a process that directly looks into the PDF content. It uses tools like PDFiD or Peepdf which scans the entire PDF and counts the occurrences of the features. This is commonly used for a quick overview of the condition of the PDF [22].
2. **Dynamic Analysis:** In contrast to the static analysis, the dynamic analysis is performed at run-time. In some instances, PDF files are not executed on PDF readers. For this reason, the analysis needs to run on a vulnerable machine [22] or in a virtual machine, for example, a malware analysis environment [3].
3. **Hardware Malware Detection (HMD):** Hardware Malware Detection is yet another modern way, and it differs from the previous two as it tries to detect malware at the hardware level. The micro-architecture features, for example, the frequency of opcodes, and comparisons of opcode signatures are vital to this method. The values of these features are collected and analyzed to check for any irregular behavior [22].
4. **Machine learning:** Machine learning is a very convenient tool when solving the problem of detection. The objective is to give it an idea of what harmful malware may seem like and use data sets and a test train split to train the machine. This gives the machine an idea of what to look for and thus detects data it may not have even encountered before. This is a very robust and flexible method as it is constantly adapting and evolving, and static analysis can be performed on it to give an even more accurate result.

While they can be individually implemented it is important to note that the best methods of PDF malware detection today usually consist of more than only one of these methods, and are usually implemented together with machine learning to create an effective system for the detection of PDF malware [22]. Pareek's technique was based on feature extraction and machine learning algorithms (LMT, Naive Bayes, Bayes Net, and J48) without JavaScript emulation techniques [10]. Schmitt, Gassen, and Gerhards-Padilla use both static and dynamic strategies to spot malicious activity in a simulated environment [6]. Jeong, Woo, and Kang detected malware on PDF file byte streams using convolutional neural networks(CNN) [17]. Corum suggested a creative, learning-based approach to identifying PDF malware that makes use of image processing and visualization methods [15]. Cuan, Damien, Delaplace, and Valois used SVM to detect malware [13]. S. R. Gopaldinne, H. Kaur, P. Kaur,

G. Kaur, and Madhuri suggested the use of Artificial Neural Networks (ANN), and K-Nearest Neighbors (KNN) [23].

There are currently some malware analysis tools present. The first notable tool is FileMon which detects changes to the file system efficiently, as well as detects questionable searches performed. However, a deficiency is that it picks up a large number of file changes even when the system is resting or idle [11]. Alternative tools are Norman Sandbox and JoeBox, both of which are dynamic analysis tools. Norman Sandbox runs the program in a secure, controlled virtual environment that is designed to act similarly to a host computer and the malware affects it accordingly. JoeBox on the other hand maintains a log of all performed actions regarding file system, registry, and more. [11].

Despite the fact that these methods exist, the majority of them have significant rates of false positives and negatives, making it impossible to utilize the models' accuracy to forecast whether or not a PDF is infected in the end. The earlier study was more static-based, which meant that run-time flaws could not be detected. The dynamic analysis that was later studied exhibited increased redundancy and false alarm rates. The majority of recent talks center on widely used features that hackers may exploit, failing to take into account rare instances of serious assaults on other features. This raises the worry that any unusual attack method utilized by the attackers might soon gain popularity as an injection method utilizing disregarded characteristics. Hence, it is crucial to have a comprehensive detection system that will examine both static and dynamic-based detections, as well as any feature with a high risk of infiltration. By applying higher advanced models for accuracy and hyper-tuning parameters, this may be successfully accomplished with machine learning and deep learning models. Although previous works were successful in making a good model with optimal accuracies and f1 scores, no research was found talking about its reliability and assurance. As research has been made in the past on the detection of malware, many fields of cyber security had implemented Explainable AI for a better understanding of the models. However, this sector was completely lacking in PDF malware detection specifically. Thus, there was a need for such implementations in this sector as well. As studies had been made it was determined that this paper will aim to use the SHAP framework from the Explainable AI (XAI) algorithms to investigate the models which are considered to be more optimal than models used in the past using machine learning algorithms and deep neural networks.

## 2.3 Explainable AI and Its Significance

Explainable artificial intelligence (XAI) refers to the development and use of artificial intelligence (AI) systems that can provide interpretable and transparent explanations of their behavior. It includes both the development of AI algorithms and models that are interpretable, as well as the development of techniques for explaining the predictions and decisions made by more complex, non-interpretable models [26].

The target of XAI is to provide insights into how AI systems are making their predictions and decisions and to identify any potential biases or assumptions that the systems may be making. It can be important when AI systems are being used to

make decisions when the predictions of the AI system need to be understood and explained, or when the use of AI systems is regulated [15]. Essentially, using XAI techniques can help to identify any biases or assumptions in an AI system that may be causing the system to make poor predictions. This can help to improve the system's performance by allowing the system to make more accurate and reliable predictions.

White box artificial intelligence (AI) methods are methods that provide interpretable and transparent explanations of the behavior of an AI system. These methods are also known as White Box explainable AI (XAI) methods.

### 2.3.1 Choice of XAI Framework

There are multiple interpretable frameworks to use for the XAI classification interpretation which include LIME, GRAD-CAM, LORE, and SHAP [26]. Choosing the right model for the work at hand was another crucial step. From this, SHAP was the most suitable framework which is further discussed.

SHAP(Shapley Additive Explanations) is a method for explaining the output of a machine learning model by decomposing the model's prediction into the contributions of each feature in the input data. It uses Shapley values, which are based on cooperative game theory, to assign a fair "payoff" to each feature based on its contribution to the model's output [25].

One key difference between Shapley Additive Explanations and other frameworks is their way to approach model interpretation. SHAP is a global interpretation method, which means that it can be used to explain the model's output for any specific input data point, as well as for the model as a whole. Whereas, other frameworks such as LIME are local interpretation method, which means that it is designed to explain the model's behavior for a specific input data point, but may not provide a global understanding of the model's behavior. Moreover, while computing individual features to the model's output prediction, SHAP uses Shapley values, which are based on cooperative game theory and provide a fair "payoff" to each feature based on its contribution to the model's output. LIME uses a weighted linear model to approximate the behavior of the original model in the vicinity of the specific input data point being explained. SHAP does not make any assumptions about the algorithm used in black-box models while LIME works on the assumption that all machine-learning models will behave linearly over a few local data points. Since SHAP is based on the Shapley values considering every case of a contribution of each feature it has a solid mechanism which LIME lacks [25]. LORE works with synthetic data generated through genetic algorithm while GRAD-CAM is used on visual data [16], [27]. Hence, out of all frameworks SHAP was chosen to be the best fit for a global interpretation of the entire classification of the models and its feature contributions.

# Chapter 3

## Methodology

### 3.1 Dataset Description

#### 3.1.1 Data Collection and Extraction

Using Contigo Dump [8], a total of 21095 files have been collected, of which 12095 are marked as malicious while the rest 9000 are clean. With the use of the PDFid module, the features of each PDF have been extracted.

The PDFid module is used on both bulks of clean and malicious PDF files to extract the features of each PDF. The PDF files contain a specific feature list which mostly defines how a PDF is structured and how many objects are present in correspondence to each feature.

The PDFid module is used on both bulks of clean and malicious PDF files to extract the features of each PDF. The PDF files contain a specific feature list which mostly defines how a PDF is structured and how many objects are present in correspondence to each feature. The information on the various functions keywords that describe the actions the item takes are indicated by the `"/` tag [5].

The program PDFiD will search a PDF file for a specified set of strings and count the total and disguised occurrences of each word:

```
obj
endobj
stream
endstream
Xref
trailer
startxref
/Page
/Encrypt
/ObjStm
/JS
/JavaScript
/AA
/OpenAction
```

```

/JBIG2Decode
/RichMedia
/Launch
/XFA
/Acroform
/EmbeddedFile
/Colors > 224

```

## Formulate The Dataset

The log file contains the unique identifier of each PDF which is a redundant element of the log file for the dataset. The log file has been converted to a CSV format with a custom converting module in Figure 3.1.

```

df ← Read individual dataset #malicious log & cleaned log
skipLine ← Skip unwanted line(s) of the log instance
  - regex formula used to identify lines with “MALWARE” and “CLEAN”

empty_array[ ] ← numpy array of (0,22)
uIdRow[ ]
value[ ]

For every instance of index, row in df.iterrows():
  IF regex skipLine matches first instance of row[0]:
    Ignore line
  ELSE:
    newRow ← row[0] with quotations removed
    append newRow to uIdRow
    match ← regex match in newRow where r ⇐ ([/a-z>(224)]+)([0-9]+)
    IF match:
      items ← group the matched items
      col, val ← items
      append val to value
    IF length of value is 21:
      insert uIdRow to value
      empty_array ← append empty_array with numpy array of value on axis 0
      clear value[ ]
      clear uIdRow[ ]

convert empty_array to dataframe using pandas, setting feature names as per the log
convert dataframe to csv setting name for dataframes
- malicious
- cleaned

```

Figure 3.1: Pseudocode Part 1

This module in Figure 3.1 has taken the log file of each malicious and clean file and converted two individual datasets, one with 12095 rows and the other with 9000 rows respectively. The unique identifier feature has been removed and a new target feature has been introduced to the dataset, using another module in Figure 3.2.

```

df ← Read individual dataset #malicious csv & cleaned csv
df ← drop 'PDF Header' with parameters inplace being true on axis 1

df ← adding feature 'isMalicious' to dataframe
    - 0 for clean 1 for malicious

convert dataframe to csv setting name for each dataframes
    -malicious
    -cleaned

```

Figure 3.2: Pseudocode Part 2

After the modifications of dropping the redundant feature and adding the target feature, the clean and malicious files have been combined into a single dataset. This has been achieved by constructing another module that concatenated and shuffled the datasets in Figure 3.3.

```

dfClean ← Read cleaned dataset #cleaned csv
dfMalicious ← Read cleaned dataset #malicious csv
dataset ← instantiate empty dataframe

dataset ⇐ concatenate dfClean and dfMalicious
dataset ⇐ randomize the clean and malicious data rows
convert "dataset: dataframe" to csv

```

Figure 3.3: Pseudocode Part 3

A total of 9000 clean PDF files and 12095 malicious PDF files with 21 characteristics make up the dataset. The majority of the dataset's files are malicious, hence an oversampled, balanced dataset has been created in order to make up for this. To create this dataset, another module was utilized as shown in Figure 3.4.

```

overSampleData ← read newly converted dataset #concatenated dataset
split feature data and target
    -X ⇐ overSampleData while dropping target column
    -y ⇐ overSampleData['isMalicious']
ros ← initiate RandomOverSampler with random_state 0
X_resampled, y_resampled ⇐ fit_resample X and y of ros
X_resampled['isMalicious'] ← y_resampled

convert X_resampled to csv

```

Figure 3.4: Pseudocode Part 4

The final dataset has been taken into consideration as the dataset with which the authors continued to classify the features stated in the dataset after forming the following modules on the log files.

### 3.1.2 Dataset Pre-processing

The dataset taken had a positively skewed distribution for each feature with no null values. Each feature was an integer type. However, there were a few redundant features that had little to no contribution to making a PDF file malicious. Hence there was a need to fix the dataset distribution as well as remove a few redundant features.

#### Dataset Distribution Fix

Skewness is a measure of the asymmetry of a distribution. A distribution is skewed to the right if it has a long tail on the right side, and is skewed to the left if it has a long tail on the left side. A distribution is symmetrical if it is not skewed [2]. Kurtosis is a measure of the "peakedness" of a distribution. A distribution with a high kurtosis has a more peaked shape, while a distribution with a low kurtosis has a more flat shape. In other words, high kurtosis data sets are more likely to have huge outliers. The normal distribution has a kurtosis of 3 [2]. Since the dataset was created using original malicious and clean PDF files, there was an issue with the dataset being skewed. The data's skewness and kurtosis were checked to ensure that they were distributed normally. Therefore, a kurtosis value less than or equal to 3 and a threshold value for skewness ranging from -0.5 to +0.5 has been considered. Any feature with a greater kurtosis value (Leptokurtic) and an out-of-range skewness value was corrected using the Box-Cox transformation, which uses mathematics to give non-normal dependent variables a normal shape.

In order to achieve a normal distribution for the data in the dataset, the data for each feature had been normalized, and any irregularities had been addressed after maintaining skewness and kurtosis threshold values. One such feature KDE plot has been shown in Figure 3.5 and Figure 3.6 for before and after any transformations.

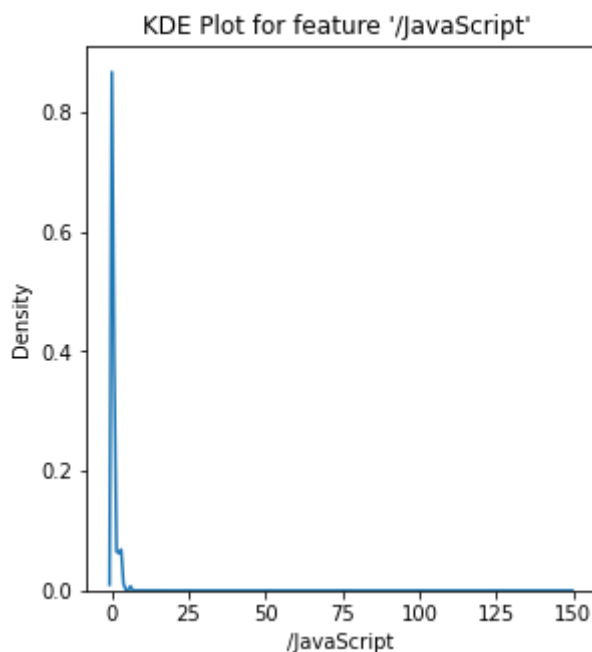


Figure 3.5: KDE plots before transformation



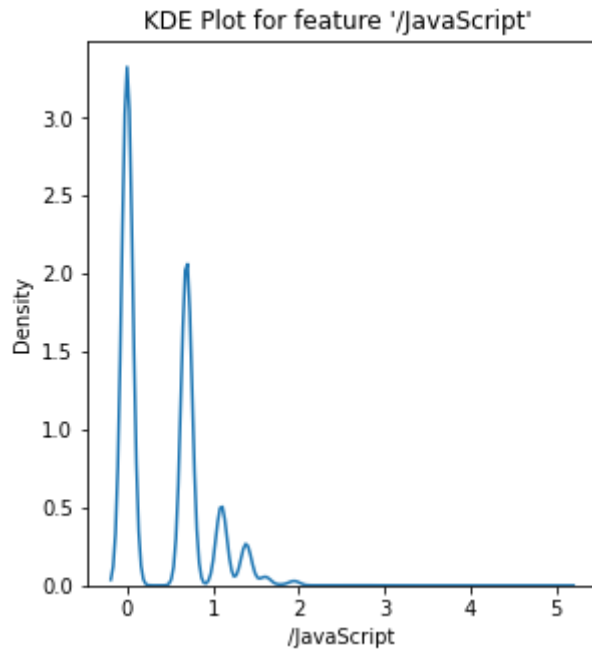


Figure 3.6: KDE plots after transformation

As it is visible that the distribution is highly positively skewed with a skewness value of 41. The kurtosis value was also high at 2713.87. In order to have an accurate analysis model, feature transformation was required to ensure the data of all features were normally distributed. Logarithmic, Reciprocal, and Square root transformations were not able to arrange the data into a normal distribution very accurately. However, Box-Cox Transformation gives a different way of normalizing data, by checking the standard deviation and whether it is the smallest or not. Furthermore, the variation of Box-Cox used was the `boxcox1p`. Because the dataset has zero values as well, the `boxcox1p` function gave better normal distributions compared to the original variation. The new skewness value is 0.9338, which is better than the initial value. The new kurtosis value is 1.253 which is a good value considering the threshold to be  $k=3$ .

### Feature Exclusion

The data is composed of features that describe how each PDF is produced and what components are operating inside them. Each component is seen as a layer of the PDF, and while they are kept apart from one another, they are linked to one another to work in sync. This feature list is frequently used by malicious writers to create their harmful scripts, which are programmed to launch whenever a PDF is downloaded or accessed in an attempt to spread viruses. The association between the traits to validate a PDF as malicious, however, is extremely broad; in actuality, not all features are necessary to carry out a specific assault. Therefore, the most prevalent traits are separated to see if they are the only factors contributing to a PDF's maliciousness.

**Correlation Check:** A correlation check using the `.corr()` function was also made and a heatmap was generated as shown in Figure 3.7 which gave insight into how

each feature had been affecting the classification.

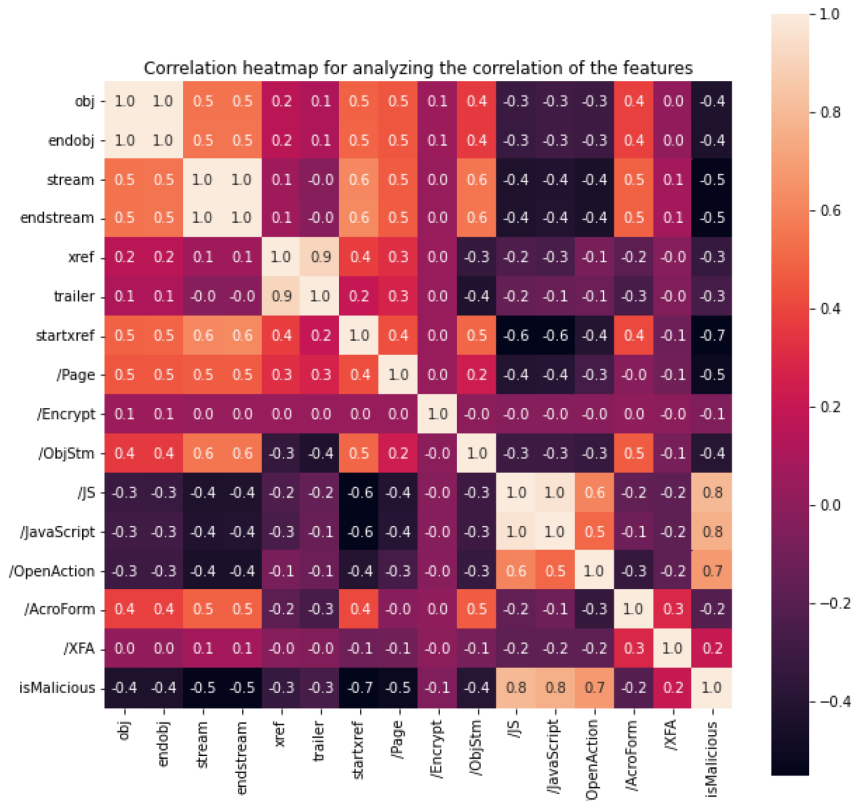


Figure 3.7: Correlation Heatmap

According to the heatmap generated in Figure 3.7, the features  $/Colors > 2^{24}$ ,  $/Launch$ ,  $/RichMedia$ ,  $/EmbeddedFile$ ,  $/AA$  have a correlation of 0.0 with the label "isMalicious." In addition, the feature JBIG2Decode has a correlation of -0.1. These features appear to have the least contribution to the analysis, so it was removed.

**Chi-square Test Validation:** To validate the modifications made to the dataset and the exclusion of certain features, a Chi-square test for Independence was conducted. The Chi-square test for Independence is a statistical method used to assess whether there is a significant relationship between two variables [7]. It is based on the Chi-square statistic, which quantifies the discrepancy between the expected and observed frequencies in a sample. The purpose of the Chi-square test for Independence is to examine the null hypothesis that there is no association between the two variables and to determine whether the observed association is statistically significant. If the p-value obtained from the test is lower than the predetermined level of significance, the null hypothesis can be rejected and it can be concluded that there is a significant association between the two variables.

The goal of this test was to determine whether any differences between actual and predicted data were due to chance or a correlation between the variables. The dataset, which had 21 features, was examined to determine its potential impact on the determination of whether a file was malicious or clean. The p-values obtained from the Chi-square test approximated the degree of independence, while the Chi-square scores reflected the degree of dependence.

Any features with p-values greater than the threshold of 0.001 were discarded, as they supported the null hypothesis for the Chi-square test. Chi-square scores were then generated for the remaining features based on the removal of features according to their p-values. The results of the Chi-square scores, as shown in the generated chart in Figure 3.8, were consistent with the removal of features based on correlation and heatmap generation. This suggests that the modified dataset, after the removal of redundant features, will contain relevant data that can be used in predictive models.

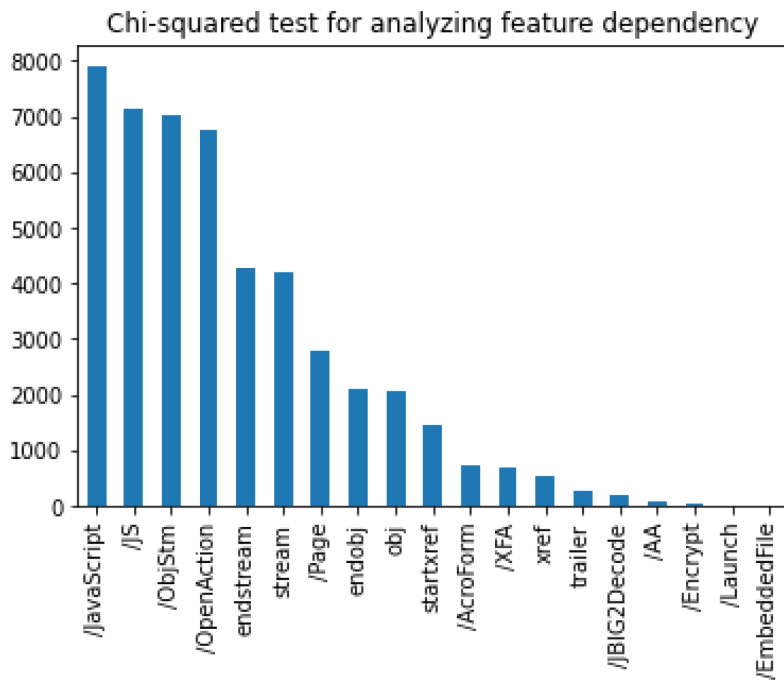


Figure 3.8: Chi-square Test

## 3.2 Proposed Model/Workplan

The processed dataset has a balanced combination of clean and malicious files. This could have been considered a generic binary classification. However, the malicious files are formulated in such a way that no antivirus or firewall can detect its executions and eventually able to bypass the operating system of a user. Considering that case, the models heavily depend on primary features and restricted identification of both variants. In order to address that problem, the implementation of models such as Stochastic Gradient Descent (SGD), XGBoost Classifier, Single Layer Perceptron, and Artificial Neural Network has been selected for classifying the dataset.

### 3.2.1 Work Plan

The aim is to use different models to figure out whether a file is malicious or not. For this we have divided our work into the following stages which have been shown in Figure 3.9:

1. The first step is to acquire malicious and clean files. For this, Contagio Malware Dump has been used. Via the use of the Pdfid model, further extraction of the features of the PDF files has been done.
2. The dataset then has to be processed and converted from log to CSV form using a converting module.
3. The unwanted header information feature of unique ids for the PDF files is removed.
4. The individual clean and malicious datasets are then combined to form one dataset combining both to be used for training and testing.
5. To get a balanced dataset, oversampling has been done on the data as required
6. Dataset was pre-processed through EDA
7. We use the train/test split and use several different machine learning algorithms individually on the split.
8. Currently used are Stochastic Gradient Descent, XGBoost Classifier, Linear Model Perceptron, and an Artificial Neural Network model.
9. Keep false positive occurrences and the size of the dataset into consideration while generating and analyzing results.
10. The results of each algorithm are gathered, charted, and compared to results of the other algorithms
11. SHAP framework for XAI is used for checking each feature's contribution to the models.
12. The best-fit algorithm will be identified and applied.
13. Classification of PDF files will be accurately completed using this.
14. The end product will be able to detect PDF malware using this algorithm/algorithms and be able to allow or prevent a PDF file from opening if proven by the model.

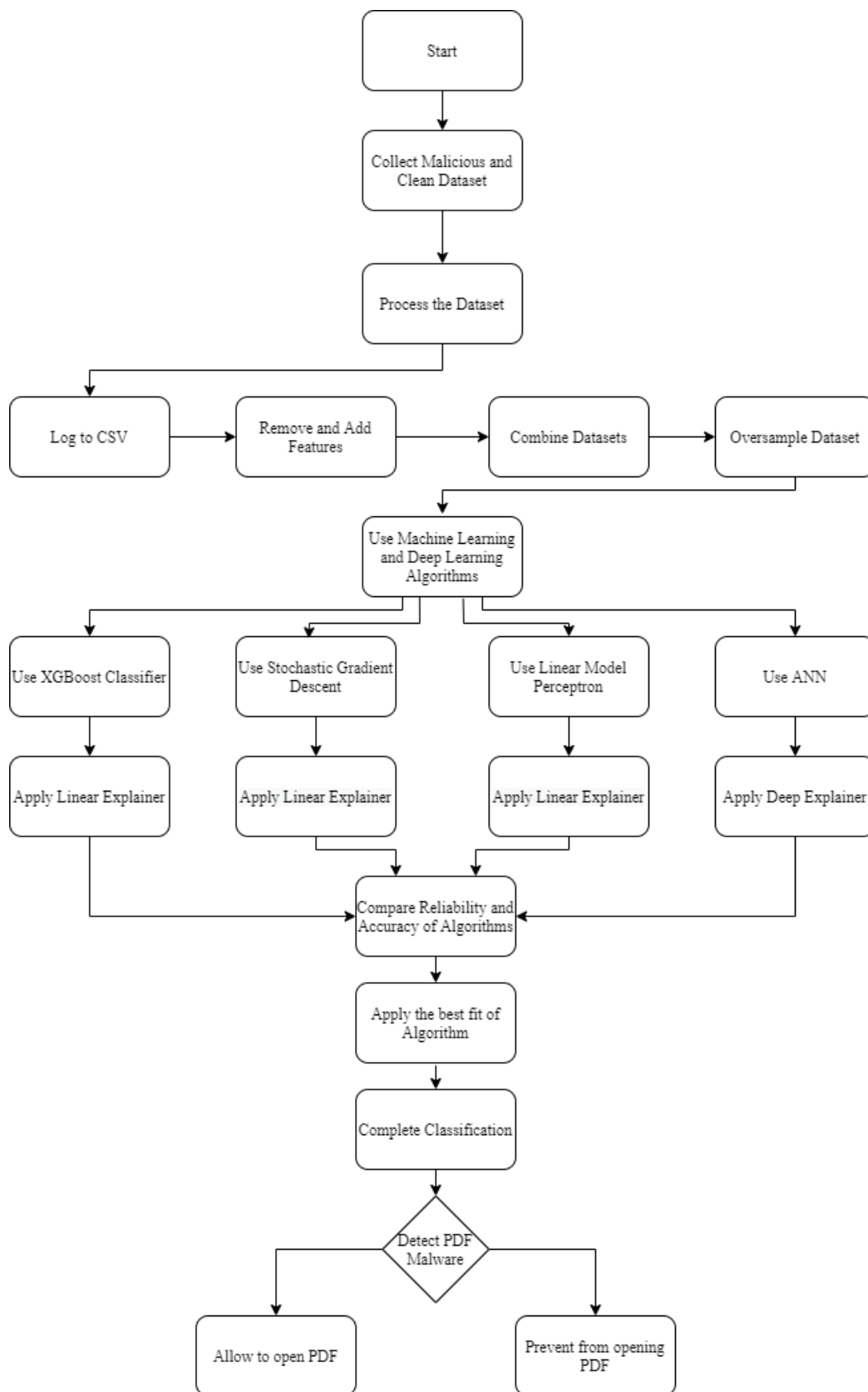


Figure 3.9: Work Plan

## 3.3 Model Implementation Review

### 3.3.1 Perceptron Model

A single-layer perceptron is a type of artificial neural network that consists of a single layer of interconnected "neurons." It is a simplified version of a multi-layer. The neurons in a single-layer perceptron are connected to the inputs through weighted connections, and each neuron produces an output based on a linear combination of the inputs and their weights. The output of the neuron is then passed through an activation function, which determines the final output of the neuron explained in Equation (5.1).

$$activation = weights * inputs + bias \quad (3.1)$$

In the context of linear binary classification, a single-layer perceptron can be used to classify data points into two classes based on a linear decision boundary. It does this by learning a set of weights that define the direction and slope of the decision boundary.

One advantage of using a single-layer perceptron for linear binary classification is that it is a simple and efficient algorithm. It requires minimal training data and has a low computational cost, making it well-suited for problems with limited resources. Additionally, single-layer perceptrons are easy to implement and understand, making them a popular choice for students and practitioners learning about neural networks. For the following dataset, the perceptron model has been implemented with some hyperparameter tuning. The penalty value has been set to 'l1', and the alpha has been considered along the fit-intercept term. Regularization (i.e. penalty) uses high-valued regression coefficients as a means of punishment to prevent overfitting. Simply put, it compresses the model and decreases its parameters. Predictions will probably perform better with this simplified, more economical approach. L1 regularization introduces an additional L1 penalty equal to the amount of the coefficients in absolute terms. In other words, it restricts the coefficients' size. Some coefficients can be deleted and reduced to zero. The regularization term's alpha parameter, also known as the penalty term, prevents overfitting by limiting the size of the weights. A more complex decision boundary may come from lowering the alpha, which might be used to address excessive bias (an indication of underfitting). The fit-intercept is defaulted to true with which the model will learn the intercept can be thought of as a learned bias unit. Since the data is already scaled, the fit-intercept is assumed False.

A confusion matrix was obtained to check for the results of the model in Table 3.1 and Figure 3.10 :

Table 3.1: Perceptron Accuracy Scores

Perceptron				
	precision	recall	f1-score	support
benign	0.99	0.98	0.99	4838
malware	0.98	0.99	0.99	4838
accuracy			0.99	9676
macro avg	0.99	0.99	0.99	9676
weighted avg	0.99	0.99	0.99	9676

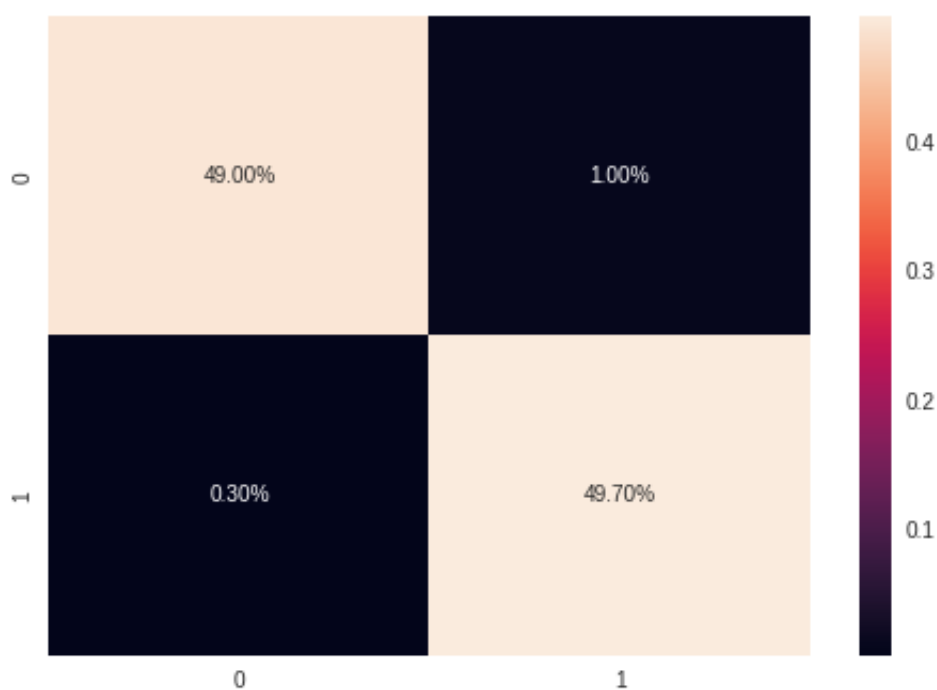


Figure 3.10: Perceptron Confusion Matrix

A ROC graph was generated as seen in Figure 3.11 to visualize how well the model was learning and it gave the following results. The table 3.2 shows the True Positives and False Negative scores:

Table 3.2: Perceptron Classification

Perceptron	
True Negative	4741
False Positive	97
False Negative	29
True Positive	4809

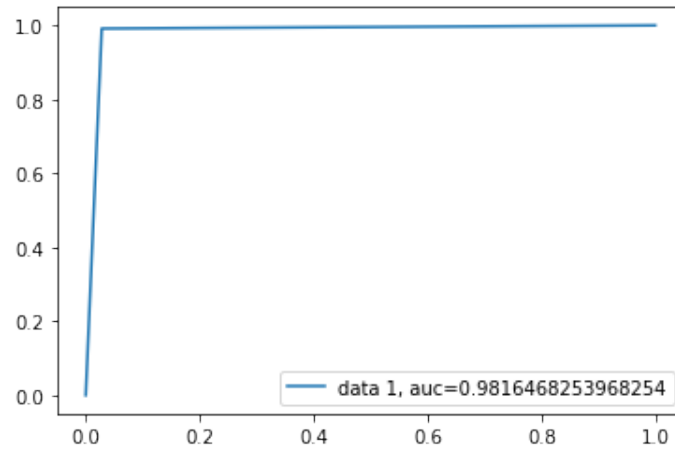


Figure 3.11: Perceptron ROC

From Figure 3.10 and Figure 3.11 with Table 3.1 and Table 3.2, it is seen that the Perceptron model has a precision of 0.99 and a recall of 0.98, resulting in an F1 score of 0.99. Its False Negative value is very low, indicating that it is a reliable model. The area under the curve (AUC) of the model's ROC curve is 0.987, which is a high value that further demonstrates the accuracy of the Perceptron model.



### 3.3.2 Artificial Neural Network

Artificial neural networks (ANNs) are computational models that mimic the structure and function of the brain. They are used for tasks such as pattern recognition, regression, and classification, and are trained using backpropagation to adjust weights based on the error between predicted and desired output. ANNs classify new data points by applying learned weights to input features and comparing the predicted output to the desired output to evaluate performance. The number of neurons and layers, activation function, and weight values all affect the performance of an ANN.

The ANN model was created with the Tensorflow library and Keras framework. It used HeNormal initializers for its hidden layers and Glorot Normal initializers for its output layer. The model consisted of four hidden layers with 5, 10, 5, and 10 units respectively, all using the ReLU activation function. A Dropout layer has been used in between the hidden layers to control the overfitting of the data. The output layer had a single unit and used the sigmoid activation function. The model was compiled with a binary cross entropy loss function, Adam optimizer, and an 'accuracy' metric. It was designed to prevent overfitting and improve the model's performance.

HeNormal is a kernel initializer that is often used for layers with ReLU activation functions. It initializes the weights of the layer's kernels using a normal distribution with a mean of 0 and a standard deviation of  $\sqrt{2/n}$ , where  $n$  is the number of input units in the layer. Glorot normal (short for Glorot and Bengio normal) is a kernel initializer that is often used for layers with sigmoid or tanh activation functions. It initializes the weights of the layer's kernels using a normal distribution with a mean of 0 and a standard deviation of  $\sqrt{2/(n_{in} + n_{out})}$ , where  $n_{in}$  is the number of input units in the layer and  $n_{out}$  is the number of output units in the layer.

Both Henormal and Glorot normal are designed to initialize the weights of the layer's kernels in a way that helps the model converge faster and perform better. They are often used as a replacement for the default initializer, which is usually a uniform distribution. The HeNormal initializers were used inside the hidden layers while the output layer used the Glorot Normal initializer.

The idea behind dropout is to randomly "drop out" or deactivate a portion of the units in a neural network during training. This is done by setting the activations of the dropped-out units to zero, which effectively removes them from the network. The dropout rate is the fraction of units that are dropped out. The main purpose of dropout is to prevent overfitting, which occurs when a model is too complex and has too many parameters relative to the amount of training data. By randomly dropping out units during training, dropout forces the model to learn multiple independent representations of the data, which can help improve generalization.

Finally, the model is compiled with a `binary_crossentropy` loss function and the Adam optimizer, with the metric set to 'accuracy' to evaluate the model.

A total of 25 epochs had been run. After an epoch is completed, the model's weights and biases are updated based on the error calculated on the training data. Then,

the second epoch will begin, and the model's weights and biases will be updated based on the error calculated on the next set of samples, which in our case is 100. Training the model for 25 epochs means that the model will see the training data 25 times and its weights and biases will be updated accordingly helping it to learn and provide good classification. After training the model, predictions had been made based on accuracy and loss shown by the curves in Figure 3.12 and 3.13.

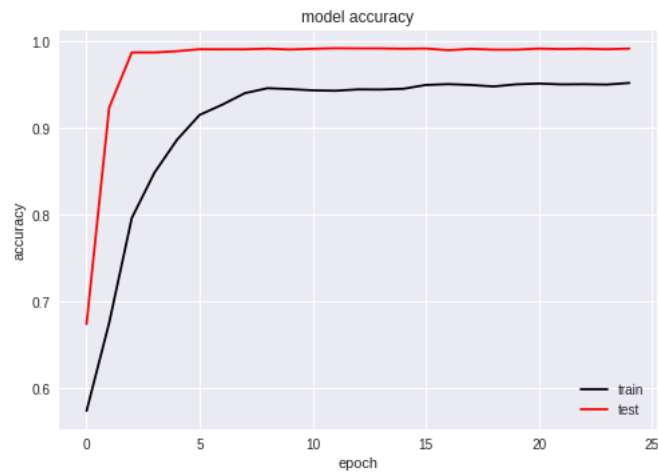


Figure 3.12: Model Accuracy of ANN

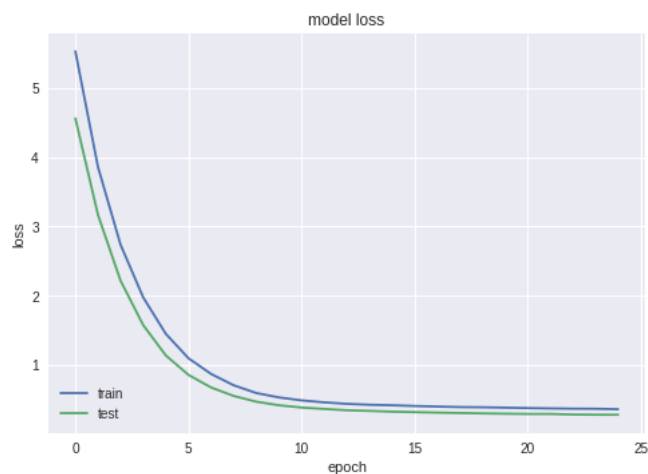


Figure 3.13: Model Loss of ANN

From the model accuracy curve in Figure 3.12 it is seen that the training curve starts from roughly 50-60% while testing accuracy starts from 60-70% which is an indication that the model is training and testing quite consistently. The testing accuracy graph shows us that testing accuracy is higher than training accuracy for all epochs. It can be explained due to the presence of a dropout function which was used in the layers of the ANN model. The dropout function works on training the model during forward propagation. However, in testing no nodes are intentionally dropped, thus resulting in higher test accuracy. This model loss curve in Figure 3.13 shows that the testing and training curves were able to drastically reduce their loss as the number of epochs increased and went to a slow change after the 10th epoch.

Moreover, the test curve is lower than the training curve in the model loss figure which indicates that the model has learned and optimized appropriately. The ANN confusion matrix shows the following in Table 3.3 and Figure 3.14:

Table 3.3: ANN Accuracy Scores

ANN				
	precision	recall	f1-score	support
benign	0.99	0.99	0.99	4838
malware	0.99	0.99	0.99	4838
accuracy			0.99	9676
macro avg	0.99	0.99	0.99	9676
weighted avg	0.99	0.99	0.99	9676

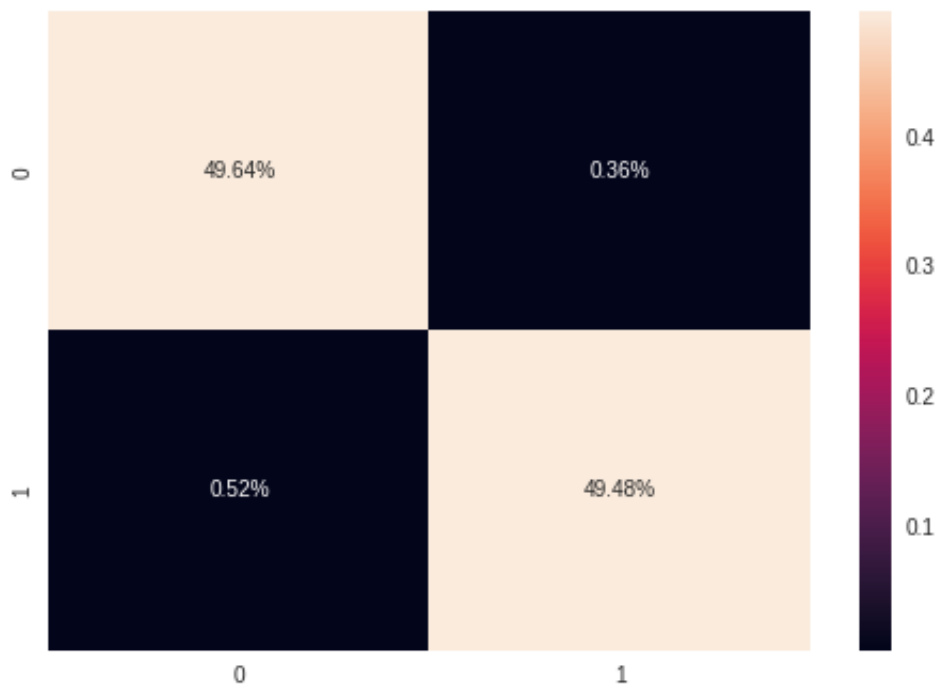


Figure 3.14: ANN Confusion Matrix

The ROC curve gave the following results in Figure 3.15 and Table 3.4 shows the True Positives and False Negatives:

Table 3.4: ANN Classification

ANN	
True Negative	4803
False Positive	35
False Negative	50
True Positive	4788

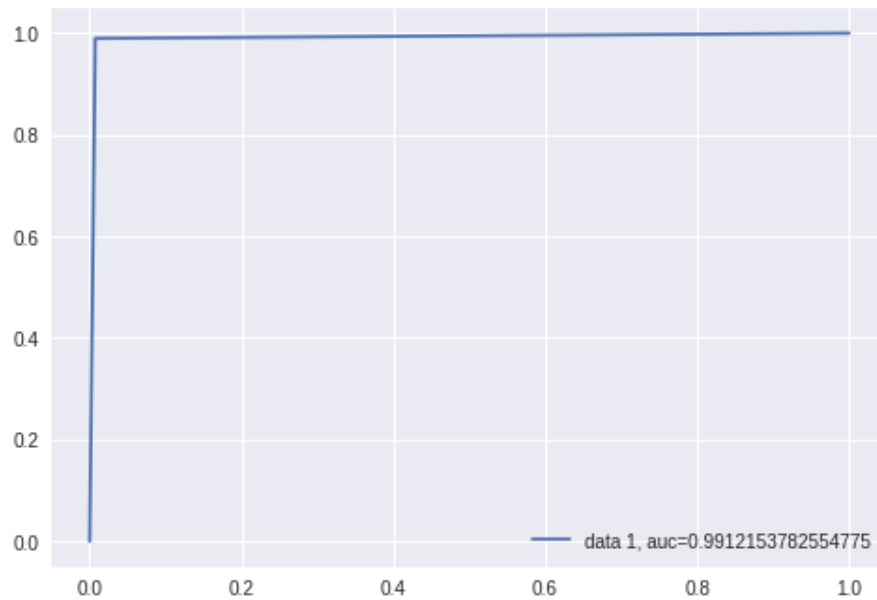


Figure 3.15: ANN ROC

From Figure 3.14 and Figure 3.15 with Table 3.3 and Table 3.4, it is seen that the F1 score is very high at 0.99. This is because the precision and recall values for this model are 0.99. This score is further proved accurate by the presence of both very few False Negatives and False Positives which is significantly less than 100. The ROC curve has an ideal shape, with maximum sensitivity and the AUC=0.991 is obtained which is almost completely ideal.

### 3.3.3 XGBoost Classifier

XGBoost is a popular open-source machine-learning library that provides efficient implementations of gradient-boosting algorithms. One of the algorithms implemented in XGBoost is binary logistic regression, which is a linear model used for binary classification tasks.

In the context of binary classification, the XGBoost binary logistic regression model can be used to learn a decision boundary that separates the positive and negative examples in the training data. It does this by fitting a linear model to the input features and using a gradient descent algorithm to optimize the model weights. The model produces a predicted probability for each class, which can be thresholded to obtain a binary prediction.

One advantage of using the XGBoost binary logistic function for binary classification is that it is fast and efficient, as it uses a linear model as the base learner. It also has good scalability, as it can handle large datasets and is capable of parallelization. Additionally, the XGBoost library provides a range of hyperparameters that can be fine-tuned to improve the performance of the binary model. The Logistic transformation in the binary setting is simply referring to the sigmoid function over the output probability of the model. The correct parameter tuning has a great influence on the model's accuracy during prediction. Moreover, implementing the parameters reduced the errors while increasing performance as well.

The model was iterated with the variations of values and concluded on a certain learning rate where it learns the best and the mean square error is appropriate. A booster known as gblinear was used. It is a linear booster that is trained using gradient descent to minimize a loss function. The gblinear booster is similar to linear regression, but it is trained using the gradient boosting framework, which involves building an ensemble of weak learners and adding them together to form a strong learner. The gblinear booster was used with the loss function logistic which is used for binary classification tasks. The logistic loss function is defined as in Equation (5.2):

$$loss = \log(1 + \exp(-y * f)) \tag{3.2}$$

where  $y$  is the true label (either 0 or 1) and  $f$  is the predicted probability of the positive class. The maximum depth was set which provided the optimal XGBoost model for classification.

The confusion matrix gave the following results in Figure 3.16 and Table 3.5:

Table 3.5: XGBoost Accuracy Scores

XGBoost				
	precision	recall	f1-score	support
benign	0.91	0.89	0.90	4838
malware	0.89	0.91	0.90	4838
accuracy			0.90	9676
macro avg	0.90	0.90	0.90	9676
weighted avg	0.90	0.90	0.90	9676

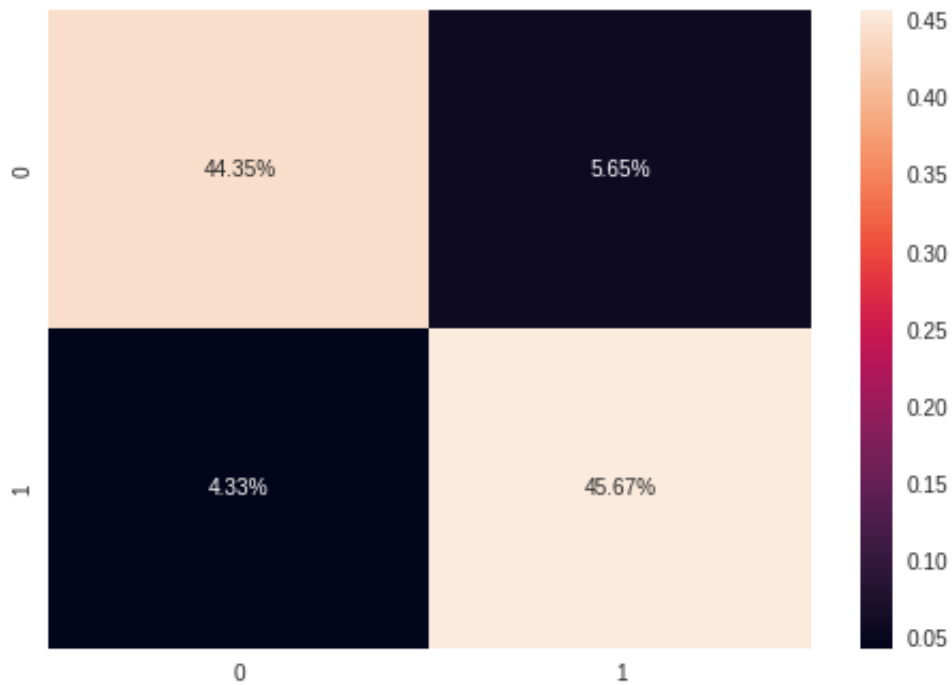


Figure 3.16: XGBoost Confusion Matrix

The ROC curve gave the following results in Figure 3.17 and Table 3.6:

Table 3.6: XGB Classification

ANN	
True Negative	4291
False Positive	547
False Negative	419
True Positive	4419

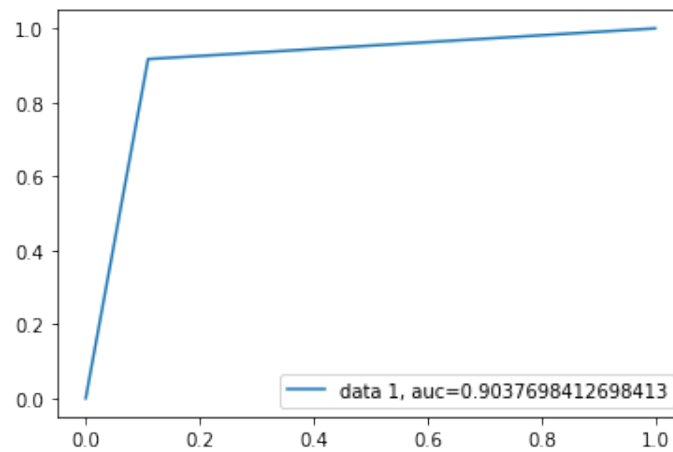


Figure 3.17: XGBoost ROC

From Figure 3.16 and Figure 3.17 with Table 3.5 and Table 3.6, it is seen that the F1 score of XGBoost was 0.90. The false negative and false positive values were comparatively quite large. The AUC calculated via the ROC is 0.900. This model also has a good classification score. From the ROC curve it can be seen that after a point, the curve is increasing slightly and shows that the sensitivity of the model is high as it has greater true positive rate in proportion to false positive rate.

### 3.3.4 Stochastic Gradient Descent(SGD)

The Stochastic Gradient Descent (SGD) classifier is an iterative algorithm that can be used for learning linear support vector machines (SVMs). It is a variant of the gradient descent algorithm that updates the model weights using only a single training example at a time, rather than using the entire training dataset. In the context of linear SVMs, the SGD classifier is used to find the hyperplane that maximally separates the positive and negative examples in the training data. It does this by minimizing the hinge loss function, which measures the misclassification of training examples. The SGD classifier updates the model weights in the direction of the negative gradient of the loss function with respect to the weights.

One advantage of the SGD classifier is that it can be used to learn, which means that it can update the model weights incrementally as new data becomes available. This makes it well-suited for handling large datasets that do not fit in memory. Another advantage is that it is relatively efficient, as it only requires a single pass over the training data to update the model weights.

The model of SGD Classifier had its parameters tuned to overcome the overfitting and underfitting of the models. The penalty specifies the type of regularization to use. In this case, the value is 'l1', which indicates that L1 regularization (also known as Lasso regularization) will be used. The alpha is the regularization strength. A higher value means more regularization, which can help prevent overfitting. The learning rate was set to 'optimal' which will use an adaptive learning rate that is calculated based on the data. The class weight was set to 'balanced' means that the model will adjust the class weights so that the classes are balanced. This can be helpful if the classes are imbalanced in the training data.



The confusion matrix obtained gave the results in Figure 3.18 and Table 3.7:

Table 3.7: SGD Accuracy Scores

SGD				
	precision	recall	f1-score	support
benign	0.99	0.97	0.98	4838
malware	0.97	0.99	0.98	4838
accuracy			0.98	9676
macro avg	0.98	0.98	0.98	9676
weighted avg	0.98	0.98	0.98	9676

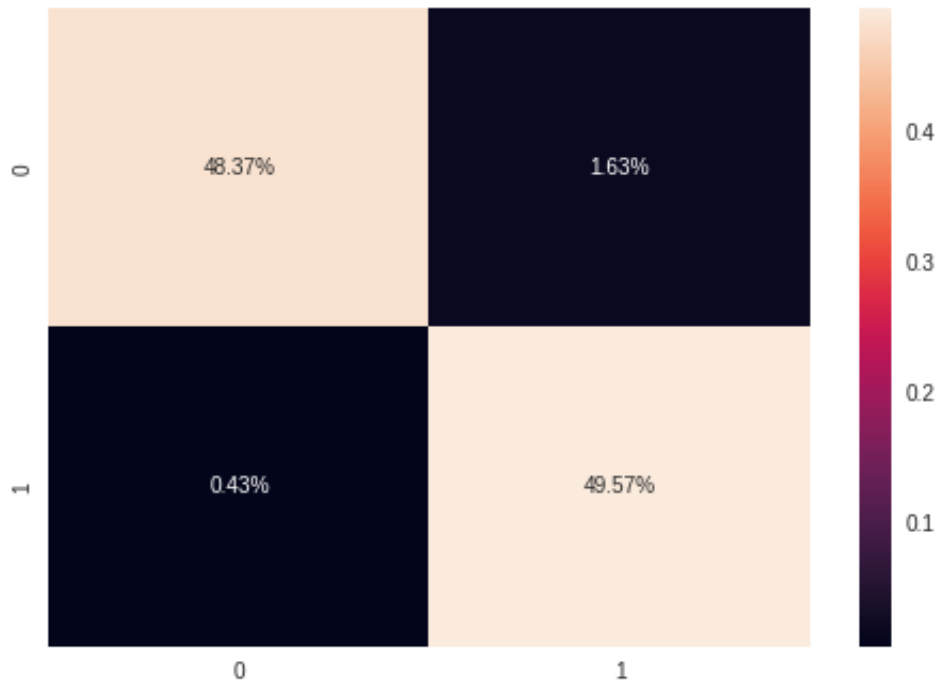


Figure 3.18: SGD Confusion Matrix

The ROC curve was generated which showed in Figure 3.19 and Table 3.8:

Table 3.8: SGD Classification

SGD	
True Negative	4680
False Positive	158
False Negative	42
True Positive	4796

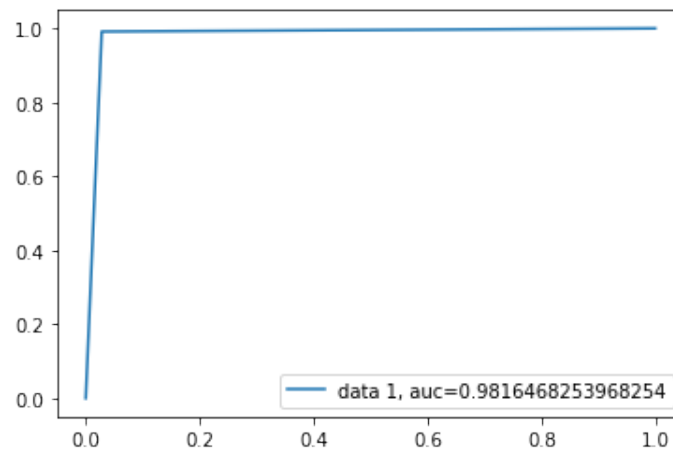


Figure 3.19: SGD ROC

From Figure 3.18 and Figure 3.19 with Table 3.7 and Table 3.8, it is seen that the recall score for benign in SGD and precision for malware is 0.97, and the recall score for malware in SGD and precision for benign is 0.99. SGD gives a decent F1 score of 0.98. The False Negative score is decent, but False Positive is significantly more. The ROC shows that the AUC is 0.979. So the accuracy of this model is also satisfactory.

# Chapter 4

## Results and Discussion

The data were randomly divided into train and test parts once the dataset had been processed to fit the models. Every model was trained using the train section, and tests were run to see if the models could tell a malicious file from a clean one. The accuracy of these results, as well as any false negatives and positives, were then evaluated by comparison. A thorough analysis of each model was done considering many aspects of its learning and prediction which has been discussed below:

### 4.1 Model Evaluation

For SGD, XGBoost, and Perceptron models, k-fold cross-validation was done. It is a technique for evaluating the performance of a machine-learning model. It involves randomly dividing the dataset into k folds (where k is a user-specified number), training the model on k-1 folds, and evaluating it on the remaining fold. This process is then repeated k times, with a different fold being used as the test set each time. The performance measure is then averaged across all k iterations to give an overall estimate of the model's performance. This allows the model to be trained and evaluated on different subsets of the data, which can provide a more robust estimate of the model's performance. Therefore, the technique can tackle the overfitting issues of the model. In the models, the value of k was set to be 10 which indicates it is a 10-fold cross-validation. With this, the model's adaptations to the dataset were evaluated. For the deep neural network model, k-fold validation was not needed as it already uses epochs to learn and evaluate its loss function and overfitting is tackled with the use of dropout functions. The evaluation was based on the "root mean square(RMSE)" metric. The RMSE is calculated as the square root of the mean squared error (MSE), which is defined as the average of the squared differences between the predicted values and the actual values. The MSE is calculated as:  $MSE = (1/n) * (y_{pred} - y)^2$  where n is the number of examples in the dataset,  $y_{pred}$  is the predicted value for each example, and y is the actual value for each example. The RMSE is a measure of the magnitude of the error, with a lower RMSE indicating a better fit.

Each model was compared based on test and train accuracies, computational timings, k-fold evaluations, and RMSE scores. The Table 4.1 below shows the values for each model:

Table 4.1: Model Computational Report

	XGBoost	SGD	Perceptron	ANN
Training Time(ms)	442.32	26.43	74.84	21353.38
Testing Time(ms)	3.99	0.92	0.75	0.73
Training Accuracy(%)	90.34	98.06	98.94	99.43
Testing Accuracy (%)	90.02	97.93	98.7	99.68
RMSE	0.316	0.144	0.114	0.069
Kfold[10](%)	90.3	98.1	99.2	

Compiling all results and accuracies, the above Table 4.1 is obtained. It can be seen that among all tested models, the least accurate in terms of both training and testing accuracy is the XGBoost Classifier even though it has decent accuracy at 90 percent. This assumption is further strengthened by the lowest K-Fold Cross-Validation and highest Root Mean Square Error among all the tested models. In regards to training time, it also falls behind SGD and Perceptron making it the least valuable model among those tested. SGD and Perceptron give similar results in all categories, with Perceptron giving slightly better accuracies in all sectors than SGD. So while it does take slightly more training time, the testing time is better than SGD and has better accuracies thus leading us to the conclusion it is preferred over SGD under most circumstances. However, the model yielding the best accuracies is the Artificial Neural Network model, as it has both the highest training and testing accuracies, as well as a very low RMSE value, indicating that it has the best-fit model among all of the models and predicts the response more accurately. However, a major drawback is the long duration of the training time, making it difficult to use in scenarios where training speed is most important. Thus the best two models are Perception and Artificial Neural Network, with each of them being more suitable for different scenarios; Perceptron when training time is of most concern, and Artificial Neural Network when the defining factor is solely accuracy.

## 4.2 Model Classification Analysis

The model classification report is a summary of the performance of the classification models. It is based on the confusion matrix, which is a table that describes the number of true and false positive and negative predictions made by the model. This classification report also includes a receiver operating characteristic (ROC) curve, which is a graphical representation of the model's performance. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at different classification thresholds. The area under the ROC curve (AUC) is a measure of the model's overall performance, with a higher AUC indicating a better-performing model. The classification report also includes other performance metrics, such as precision, recall, and the F1 score. These metrics can be used to evaluate the model's performance and identify areas for improvement. The reports for all models have been described below:

**True Positive:** A true positive is an outcome where the model correctly predicts the positive class.

**True Negative:** A true negative is an outcome where the model correctly predicts the negative class.

**False Positive:** A true negative is an outcome where the model incorrectly predicts the positive class.

**False Negative:** A false negative is an outcome where the model incorrectly predicts the negative class.

**Precision:** Precision is a measurement of how many positive predictions were predicted correctly.

**Recall:** Recall is a measurement of how many positive class samples present in the dataset were correctly identified, by the model.

**F1 Score:** F1 score is a machine learning accuracy metric that computes how many times a model made a correct prediction across the whole dataset. It combines the precision and recall scores of a model and gives a score out of 1.

**AUC:** The area under the ROC curve (AUC) is a measure of the model’s overall performance, with a higher AUC indicating a better-performing model.

The most important aspect was the False Negative, as it defines when a malware-containing PDF is incorrectly claimed as malware free. A high value of that (greater than 100) will be considered bad. False Positive is the second important aspect but slightly less important as it is not directly harmful to a user. These metrics can be used to evaluate the model’s performance and identify areas for improvement. The reports for all models have been described in Table 4.2 and Table 4.3 below:

Table 4.2: Complete Classification Report

Models	Class	Benign	Malware	Accuracy	Macro Avg	Weighted Avg	AUC
Perceptron	Precision	0.99	0.98	-	0.99	0.99	0.98
	Recall	0.98	0.99	-	0.99	0.99	
	F1 Score	0.99	0.99	0.99	0.99	0.99	
	Support	4838	4838	9676	9676	9676	
XGBoost	Precision	0.91	0.89	-	0.90	0.90	0.900
	Recall	0.89	0.91	-	0.90	0.90	
	F1 Score	0.90	0.90	0.90	0.90	0.90	
	Support	4838	4838	9676	9676	9676	
SGD	Precision	0.99	0.97	-	0.98	0.98	0.979
	Recall	0.97	0.99	-	0.98	0.98	
	F1 Score	0.98	0.98	0.98	0.98	0.98	
	Support	4838	4838	9676	9676	9676	
ANN	Precision	0.99	0.99	-	0.99	0.99	0.991
	Recall	0.99	0.99	-	0.99	0.99	
	F1 Score	0.99	0.99	0.99	0.99	0.99	
	Support	4838	4838	9676	9676	9676	

Table 4.3: Contingency Classification Report

Models	Class	Values
Perceptron	True Negative	4741
	False Positive	97
	False Negative	29
	True Positive	4809
XGBoost	True Negative	4291
	False Positive	547
	False Negative	419
	True Positive	4419
SGD	True Negative	4680
	False Positive	158
	False Negative	42
	True Positive	4796
ANN	True Negative	4803
	False Positive	35
	False Negative	50
	True Positive	4788

Comparing all the models, we can see ANN and Perceptron have better F1 scores and a less False Negatives and False Positives compared to the other models. XGBoost gives us values of False Negatives and False Positives both higher than the threshold which was considered(100). SGD lies comfortably in the middle of the pack. A NN's False Negative and False Positive scores are better than those of the Perceptron model as they are both significantly less than the threshold while the False Positive of the Perceptron model was close to 100. Comparing all models, ANN had the best F1 score, the least numbers of False Positive and False Negatives with the most value for AUC. Therefore the conclusion can be reached that ANN is the most accurate model followed by Perceptron and SGD respectively, and the least accurate out of the models it implemented is XGBoost.

## 4.3 SHAP Analysis

SHAP framework provides a way to decompose the output of a machine learning model into the contributions of each feature in the input data, which can be useful for understanding the model's behavior and identifying any potential biases or assumptions that the model may be making. There are many explainers of the SHAP framework of which the Linear Explainer and the Deep Explainer have been used to explain the machine learning models and deep neural models respectively.

There are two types of importance measures in SHAP: global importance and local importance.

Global importance measures the overall contribution of each feature to the model's output. It is computed by summing the Shapley values of all the feature's interactions. Global importance can be plotted using SHAP summary plots, which provide a global view of the model's behavior.

Local importance measures the contribution of a single feature to the model's output for a specific input or dataset. It is computed by summing the Shapley values of the feature's interactions with all the other features. Local importance can be plotted using SHAP dependence plots, which show the relationship between a single feature and the model's output.

After fitting the explainers and achieving the Shapley values, force plots, importance plots and waterfall plots of each model had been taken into consideration.

**Force plot:** A force plot in SHAP is a visual explanation of the output of a machine-learning model for a specific input or prediction. It shows the contribution of each feature to the model's output, as well as the overall effect of all the features. The force plot is composed of a horizontal bar chart and a vertical axis. The horizontal bar chart shows the contribution of each feature to the model's output. The length of each bar represents the magnitude of the contribution, and the direction of the bar indicates whether the feature is increasing or decreasing the model's output. Positive values indicate that the feature is increasing the output, while negative values indicate that the feature is decreasing the output.

**Cluster plot:** A SHAP cluster plot provides a visual summary of the patterns and relationships in a dataset based on the Shapley values of the features. It can be used to identify groups of similar data points or to explore the structure of the data. The SHAP cluster plot with bar representation shows the data points as horizontal bars, and the clusters are indicated by different colors or shapes. The plot may also include lines or other visual elements to indicate the boundaries between the clusters or the relationships between the data points.

**Waterfall plot:** A waterfall plot in SHAP is a visual representation that shows the contribution of each feature to the model's output for a specific input or prediction. It is similar to a forced plot in SHAP, but instead of a single bar chart, it shows a series of stacked bars, one for each feature. The waterfall plot is composed of a

vertical axis that represents the model's output and a series of horizontal stacked bars, one for each feature. The length of each bar represents the magnitude of the feature's contribution to the model's output.

**Feature Importance Plot:** In SHAP, a feature importance plot is a visual representation that shows the overall importance of the features in a model. This shows the mean absolute Shapley values of the features as a bar chart. The plot will show the features on the y-axis and the Shapley values on the x-axis, with the length of the bars representing the magnitude of the feature's importance. The bars will be sorted in decreasing order of importance.

**Decision plot:** A decision plot in SHAP is a visual representation which here is explaining the global importance that shows the decision boundary of a classification model and how it is influenced by the input features. It is a type of scatter plot that shows the predicted class probabilities on the y-axis and the input features on the x-axis. The decision plot is created by sampling a set of input points across the feature space and computing the class probabilities for each point using the classification model. The points are then plotted on the scatter plot according to their class probabilities and feature values. The decision plot includes a horizontal line at the point where the class probabilities are equal (i.e., 0.5). This line represents the decision boundary of the model, and points above the line are predicted to belong to one class, while points below the line are predicted to belong to the other class. The decision plot also shows the Shapley values of the features as contour lines or regions of color. The Shapley values indicate the relative importance of the features in determining the model's output, with higher values indicating more important features.

Depending on the outputs of each model's predictions, the suitable SHAP plots with the suitable Explainers used have been plotted for each model indicating its global and local importance for the features.



### 4.3.1 Perceptron

#### Local Importance Analysis



Figure 4.1: Force Plot Perceptron

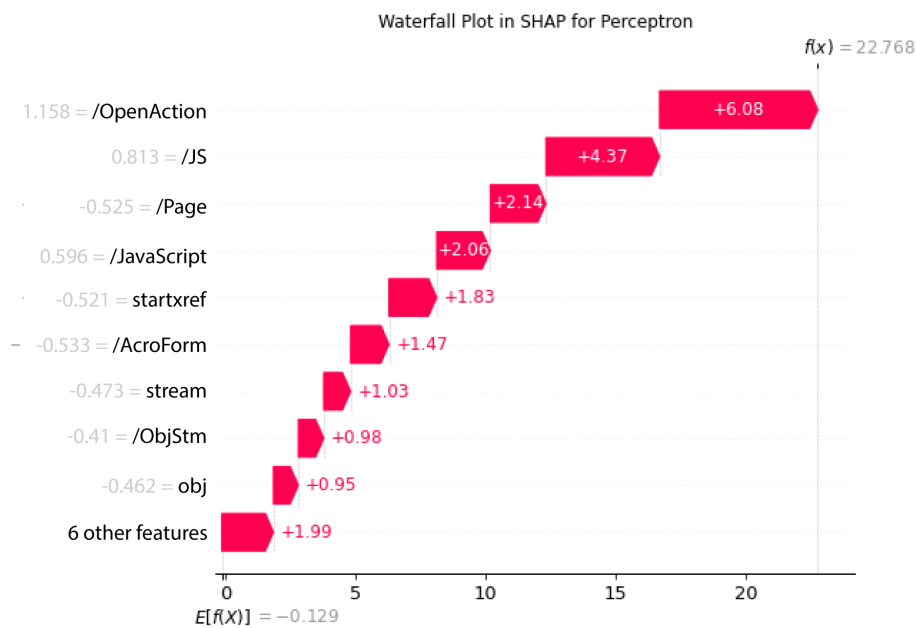


Figure 4.2: Waterfall Plot Perceptron

The force and waterfall plots in Figure 4.1 and Figure 4.2 both define the feature contributions on a single PDF data which was malicious. Here as we see from the force plot that the Perceptron model was able to detect it as malicious with /OpenAction, /JS, /startxref, and /JavaScript having the most contribution for this particular PDF file which is indicated by the red arrows going on the right direction showing it has higher values. However, /Xref, /XFA, and /trailer has a negative impact on making a file malicious which is indicated by the blue arrows going in the left direction. The waterfall plot is also showing a similar result with similar values but better visuals.

## Global Importance Analysis

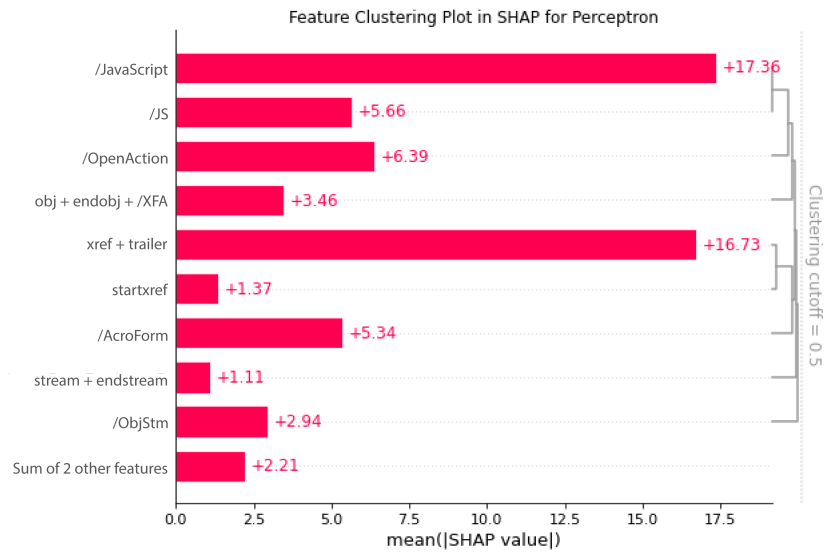


Figure 4.3: Feature Clustering Plot Perceptron

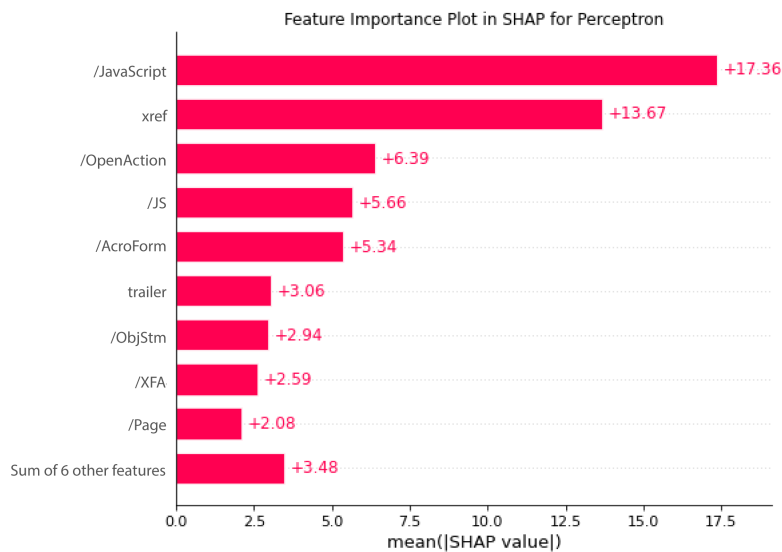


Figure 4.4: Feature Importance Plot Perceptron

Here, the entire testing data was taken into consideration for an overall analysis of the contribution of each feature in the Cluster Plot of Figure 4.3 and Importance Plot of Figure 4.4. From this, we can see that /JavaScript, /Xref, /Js, and /OpenAction are the features with the highest contribution and the cluster plot also shows which features combined have the highest probability of making a file malicious. This also shows that /JavaScript, /JS along with /OpenAction and a combined prediction of obj, endobj, and /XFA would probably result in a file being malicious according to Perceptron.

The local analysis of one instance and the global analysis does suggest similar feature contributions which suggests the interpretations are consistent.

### 4.3.2 SGD

#### Local Importance Analysis

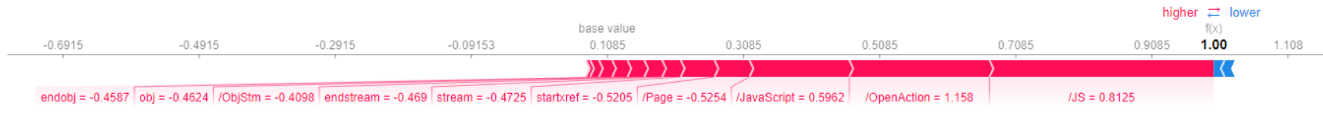


Figure 4.5: Force Plot SGD

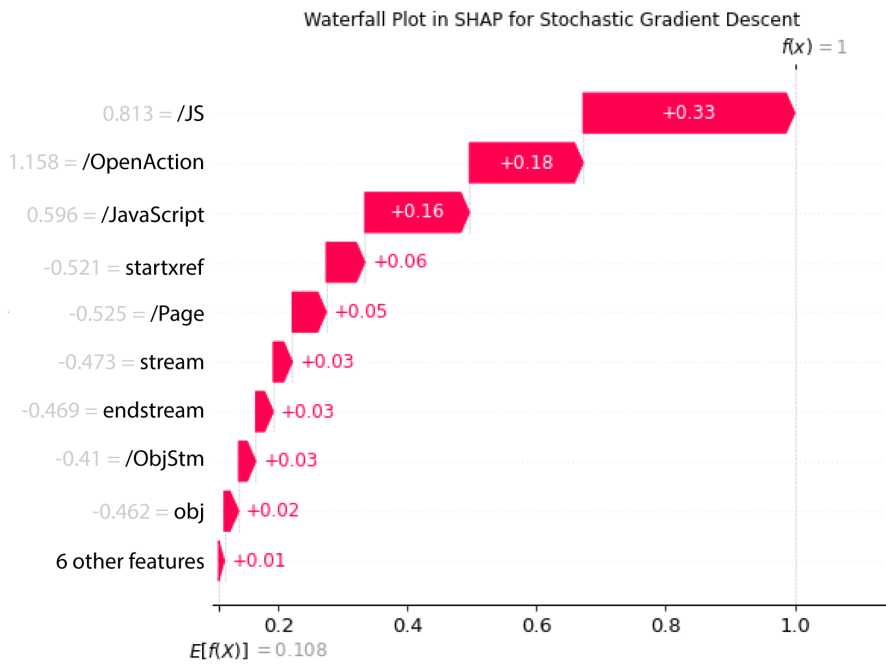


Figure 4.6: Waterfall Plot SGD

The force and waterfall plots in Figure 4.5 and 4.6 for the same pdf file used in the Perceptron model's local importance analysis were also used to interpret that of the SGD model. This shows that the SGD model shows /JS is the highest contributor with /OpenAction and /Javascript following along. /XFA and Acroform were thought to be the two features having a negative contribution in making it malicious. The waterfall plot also suggests the same.

## Global Importance Analysis

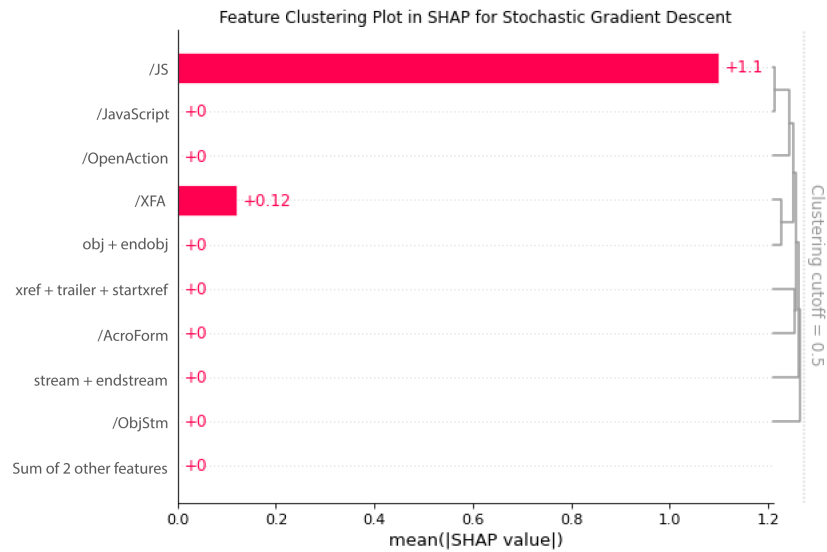


Figure 4.7: Feature Clustering Plot SGD

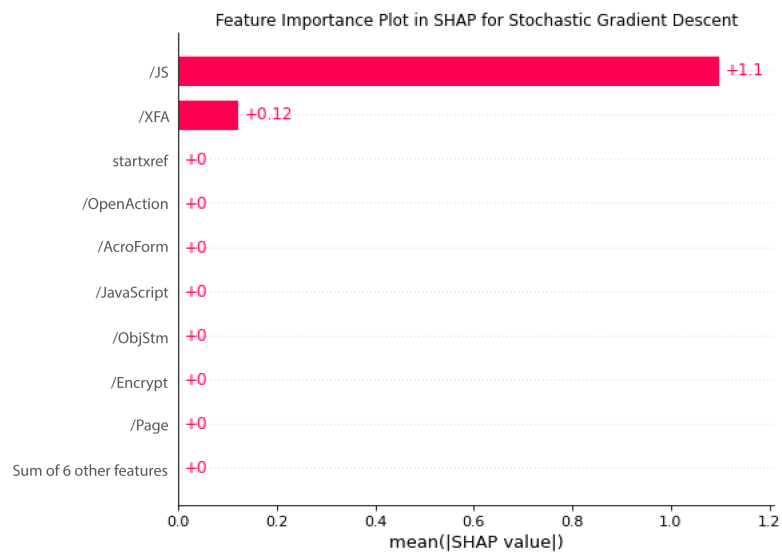


Figure 4.8: Feature Importance Plot SGD

Considering all the test files the model did a mediocre job in analyzing which features gave the most contribution as it only recognized and prioritized /Js and /XFA ignoring all other features as shown in the Cluster and Importance plots in Figure 4.7 and Figure 4.8. This is because these two features had the most occurrences which got prioritized over the other features. This suggests that even though the SGD model can learn one instance at a time if a chunk of data is provided, it will do a very poor classification of the files.

### 4.3.3 XGBoost Classifier

#### Local Importance Analysis

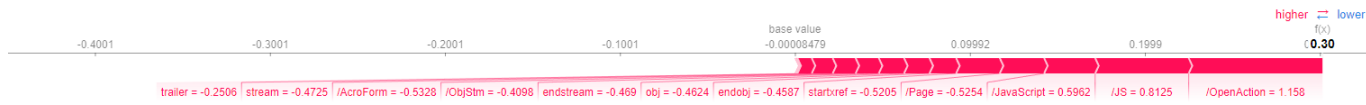


Figure 4.9: Force Plot XGBoost

The XGBoost classifier takes /OpenAction, /Js, /Javascript, and /Page to have the highest contribution. One notable difference from the other models is that XGBoost did not notice any feature having a negative impact on the file as malicious in the Force Plot of Figure 4.9.

#### Global Importance Analysis

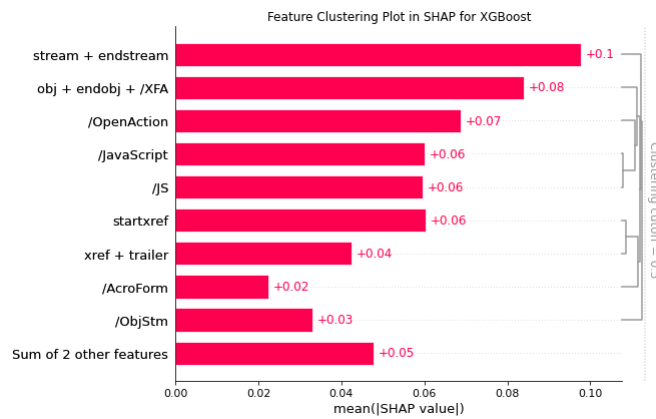


Figure 4.10: Feature Clustering Plot XGBoost

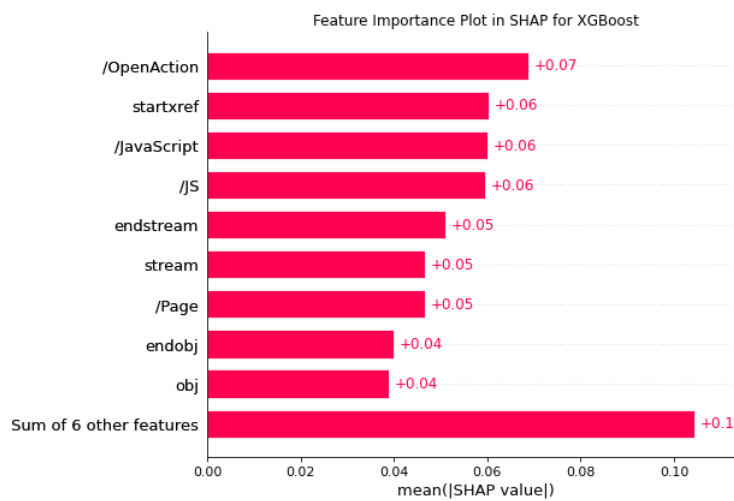


Figure 4.11: Feature Importance Plot XGBoost

The Cluster Plot in Figure 4.10 of the XGBoost model shows that this model takes combinations of features to be responsible for contribution. While each feature alone in the Importance Plot of Figure 4.11 has a little contribution, taking summations of a few features provides the overall contribution. This is understandable as XGBoost is a tree classifier and it takes combinations of features to come to a classification result. Hence the importance plot and cluster plot shows the same interpretation. The local and global plots are also consistent in their analysis so it would be a good model for the classification of multiple files.

### 4.3.4 ANN

#### Global and Local Analysis

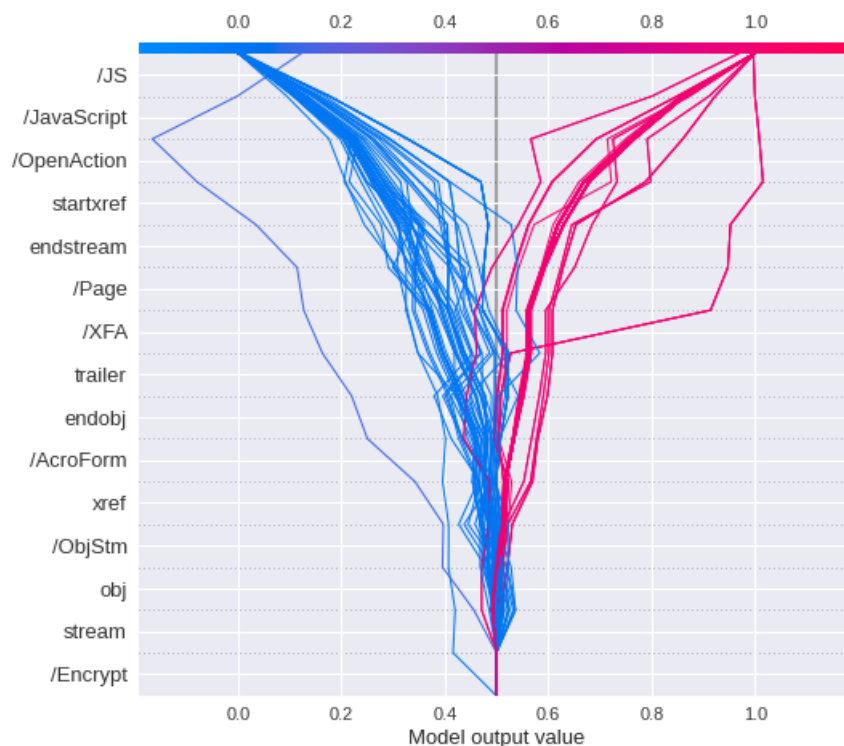


Figure 4.12: ANN Decision Plot

As ANN is a deep neural network, its output of predictions is probabilities in a two-dimensional array. This is different from machine learning models and hence is a more complicated black-box model. Therefore, Deep Explainer is used for SHAP value interpretation. The global and local analysis of such a model can be found through the Decision Plot of Figure 4.12 where the overall contribution of the features are shown through the x-axis in a decreasing order while each line represents each pdf test file which is classified in either 1 or 0. From this, we can see that /JS, /Javascript, /OpenAction, and startxref have the highest contribution.

## 4.4 The Analogy of SHAP Analysis of Models

Taking all plots into consideration of every model, it is clear that although the classification report and evaluation of the SGD model was better than the XGBoost model, its interpretation of the features and the working mechanism is done poorly and is less robust. If different data are provided. SGD will not perform as well as the other models despite its f1, precision, and accuracy. XGBoost will work better than SGD but perceptron and ANN have the most reliable results in interpreting the data and learning. The total contribution according to the force plots is 1.00,0.3,22.77 for the SGD, XGBoost, and Perceptron respectively. From there it suggests that perceptron is the best model while XGB is the worst in terms of interpretation for that specific pdf. However, the force plots cannot decide the best model as ANN has a different form of explanation and is model specific for one data. Hence, a closer look at the global analysis suggests that each model has the same set of features shown as the most contributed which also matches with the research on how PDF malware is injected in a general sense. Hence it reflects that all models are capable of classification of malware if the data consists of a javascript attack. However, it is seen that SGD will not perform as well for other sorts of attacks while XGBoost, Perceptron, and ANN will still be able to detect it. On a closer look, XGBoost is a tree classifier and learns through the combination of a few features together suggesting that if there is a different combination of features it will not work as well and has less accuracy over the Perceptron and ANN model. This comes to the point that the best working models are Perceptron and ANN in terms of interpretation of feature contributions where ANN works slightly better than Perceptron.

# Chapter 5

## Conclusion and Future Work

This research aimed to detect PDF malware using machine learning algorithms and deep neural networks, with the aid of XAI's SHAP framework. The dataset was analyzed using machine learning algorithms (Stochastic Gradient Descent and XGBoost) and the neural networks (ANN and Perceptron) model. The models were evaluated from various perspectives to determine which model would perform the best. It was found that the ANN model had the best overall accuracies and F1 scores. The least accurate model was the XGBoost, with the lowest F1 scores. However, when examining how each model was prioritizing different features using Explainable AI, it was discovered that the XGBoost model was more versatile in identifying malware with a range of different features, while the SGD model focused more on identifying JavaScript-based obfuscation attacks. Therefore, it was concluded that the SGD model was the least versatile. This explains that SGD is more scoped than the other models. Therefore, it would not be accurate to determine that the XGBoost model is the worst based on its lowest accuracy alone. In terms of interpreting Shapley values, the Perceptron and ANN models provided the most reliability, trust, and usability. Therefore, based on all forms of evaluation, the ANN model was determined to be the optimal choice, with the best accuracies, the least true positives and false negatives, and the highest level of trust in its ability to detect various types of malware attacks. Additionally, it should be noted that the results of this study are specific to the training of the data and that the machine learning models had to be adjusted and tuned for the specific dataset. The fact that the ANN model can automatically adjust its biases through forward and backward propagation makes it the most robust model that does not require specific hyperparameter tuning. One factor that could make the Perceptron model more appealing than the ANN model is its shorter training time, even though it has slightly lower accuracy.

Overall, it is recommended to consider both computational time and explainability when choosing the best model, especially in sensitive areas such as cybersecurity. This paper concludes that neural networks have good potential for classifying PDF malware and inspiring further research into more complex neural networks, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). It is hoped that, in the future, when the tensor library is compatible with the SHAP library of Deep Explainers, it will be possible to test the interpretability of RNNs and CNNs using SHAP explainers.



# References

- [1] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, IEEE, 2007, pp. 421–430.
- [2] L. Wang and J. Ma, “A kurtosis and skewness based criterion for model selection on gaussian mixture,” in *2009 2nd International Conference on Biomedical Engineering and Informatics*, IEEE, 2009, pp. 1–5.
- [3] P. Laskov and N. Šrندیć, “Static detection of malicious javascript-bearing pdf documents,” in *Proceedings of the 27th annual computer security applications conference*, 2011, pp. 373–382.
- [4] C. Ulucenk, V. Varadharajan, V. Balakrishnan, and U. Tupakula, “Techniques for analysing pdf malware,” in *2011 18th Asia-Pacific Software Engineering Conference*, IEEE, 2011, pp. 41–48.
- [5] D. Maiorca, G. Giacinto, and I. 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.
- [6] F. Schmitt, J. Gassen, and E. Gerhards-Padilla, “Pdf scrutinizer: Detecting javascript-based attacks in pdf documents,” in *2012 tenth annual international conference on privacy, security and trust*, IEEE, 2012, pp. 104–111.
- [7] Y. Tang and S. N. Srihari, “Efficient and accurate learning of bayesian networks using chi-squared independence tests,” in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, IEEE, 2012, pp. 2723–2726.
- [8] *Contagio: 16,800 clean and 11,960 malicious files for signature testing and research*, <https://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html?fbclid=IwAR0itQst8iovFZG-5ZPsryKe-QWIVc36Rcpum>, 2013.
- [9] V. Hamon, “Malicious uri resolving in pdf documents,” *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 2, pp. 65–76, 2013.
- [10] H. Pareek, P. Eswari, and N. S. C. Babu, “Malicious pdf document detection based on feature extraction and entropy,” *International Journal of Security, Privacy and Trust Management*, vol. 2, no. 5, pp. 31–35, 2013.
- [11] N. Bhojani, “Malware analysis,” *Malware Analysis*, pp. 1–5, 2014.
- [12] G. Brinda and G. George, “Detection and analysis of shellcode in malicious documents,” 2016.

- [13] B. Cuan, A. Damien, C. Delaplace, and M. Valois, “Malware detection in pdf files using machine learning,” in *SECRYPT 2018-15th International Conference on Security and Cryptography*, 2018, 8p.
- [14] M. Elingiusti, L. Aniello, L. Querzoni, and R. Baldoni, “Malware detection: A survey and taxonomy of current techniques,” *Cyber threat intelligence*, pp. 169–191, 2018.
- [15] A. Corum, D. Jenkins, and J. 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.
- [16] R. Guidotti, A. Monreale, F. Giannotti, D. Pedreschi, S. Ruggieri, and F. Turini, “Factual and counterfactual explanations for black box decision making,” *IEEE Intelligent Systems*, vol. 34, no. 6, pp. 14–23, 2019.
- [17] Y.-S. Jeong, J. Woo, and A. R. Kang, “Malware detection on byte streams of pdf files using convolutional neural networks,” *Security and Communication Networks*, vol. 2019, 2019.
- [18] J. Müller, F. Ising, V. Mladenov, C. Mainka, S. Schinzel, and J. Schwenk, “Practical decryption exfiltration: Breaking pdf encryption,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 15–29.
- [19] *Why pdf is considered the world’s most important file format*, <https://www.bbntimes.com/companies/why-pdf-is-considered-the-world-s-most-important-file-format>, 2019.
- [20] J. Lindenhofer, R. Offenthaler, and M. Pirker, “A curious exploration of malicious pdf documents.,” in *ICISSP*, 2020, pp. 577–584.
- [21] P. Singh, S. Tapaswi, and S. Gupta, “Malware detection in pdf and office documents: A survey,” *Information Security Journal: A Global Perspective*, vol. 29, no. 3, pp. 134–153, 2020.
- [22] N. Fleury, T. Dubrunquez, and I. Alouani, “Malware: An overview on threats, detection and evasion attacks,” *arXiv preprint arXiv:2107.12873*, 2021.
- [23] S. R. Gopaldinne, H. Kaur, P. Kaur, G. Kaur, *et al.*, “Overview of pdf malware classifiers,” in *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, IEEE, 2021, pp. 337–341.
- [24] *Malware detection - a simple guide in 3 easy points*, <https://www.jigsawacademy.com/blogs/cyber-security/malware-detection/>, 2021.
- [25] A. Bhattacharya, *Applied Machine Learning Explainability Techniques: Make ML models explainable and trustworthy for practical applications using LIME, SHAP, and more*. Packt Publishing, 2022, ISBN: 9781803234168. [Online]. Available: <https://books.google.com.bd/books?id=Kal3EAAAQBAJ>.
- [26] N. Capuano, G. Fenza, V. Loia, and C. Stanzone, “Explainable artificial intelligence in cybersecurity: A survey,” *IEEE Access*, vol. 10, pp. 93 575–93 600, 2022.

- [27] K. Duvvuri, S. Chethana, S. S. Charan, V. Srihitha, T. Ramesh, and K. Srikanth, “Grad-cam for visualizing diabetic retinopathy,” in *2022 3rd International Conference for Emerging Technology (INCET)*, IEEE, 2022, pp. 1–4.