Explainable Machine Learning Models for Detecting PDF Malware

A report submitted in partial fulfillment of the requirements for the

award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Anish Akode (CSE19U001)
M. Anunay Reddy (CSE19U011)
Sudam Eliya (CSE19U021)

under the guidance of

Dr. Anoop Jacob Thomas

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY TIRUCHIRAPPALLI SETHURAPATTI, TIRUCHIRAPPALLI – 620012.

BONAFIDE CERTIFICATE

This is to certify that the project work titled "Explainable Machine Learning Models for Detecting PDF Malware" is a bonafide record of the work done by

Anish Akode (CSE19U001)
M. Anunay Reddy (CSE19U011)
Sudam Eliya (CSE19U021)

in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of the **Indian Institute of Information Technology Tiruchirappalli** during the year 2022-2023. The contents of this report, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Dr. Anoop Jacob Thomas	**Dr. R. Dhanalakshmi**
Assistant Professor Supervisor	Head of the Department

Project viva-voce held on _____

Internal Examiner **External Examiner**

ABSTRACT

Over the past few years, the Portable Document Format (PDF), which is commonly referred to as PDF, has become a popular and widely accepted method for sharing and distributing documents. Its flexibility and ability to be used across various platforms make it a highly effective means of launching malicious code attacks. There exist multiple methods for examining malicious software files, including both static and dynamic techniques, that can identify distinguishing characteristics between malware and non-malicious files. When dealing with zero-day vulnerabilities, traditional analysis methods may be insufficient, and ML models use training data to automatically detect PDF malware. These techniques are vulnerable to evasion attacks, in which a malicious PDF is modified to appear benign. In this study, we aimed to evaluate the efficacy of a machine learning and deep learning model using a dataset we proposed. We conducted experiments using various machine learning classifiers and analyzed a dataset consisting of 28,102 PDF documents, including 11,266 malicious files and 16,836 benign files. Our results showed that the proposed approach was successful, achieving an accuracy rate of 99.06%. Our project presents a novel method to address the sustainability problem in malware classification by creating an explainable machine learning model which selects features that stand the test of time without requiring frequent retraining.

Keywords: Malware classification, Random Forest, LightGBM, Deep Neural Network, Explainable Artificial Intelligence (XAI), and SHAP.

ACKNOWLEDGEMENT

We wish to record my deep sense of gratitude and profound thanks to our project supervisor **Dr. Anoop Jacob Thomas,** Assistant Professor, Department of Computer Science & Engineering, Indian Institute of Information Technology Tiruchirappalli for his keen interest, inspiring guidance, constant encouragement with my/our project work during all stages, to bring this project report into fruition.

We thank **Prof. NVSN. Sarma,** Director, Indian Institute of Information Technology Tiruchirappalli and **Dr. R. Dhanalakshmi**, Associate Professor and Head, Department of CSE, Indian Institute of Information Technology Tiruchirappalli for providing us all the facilities to complete our project work.

We also thank the faculty and non-teaching staff members of the Department of CSE, Indian Institute of Information Technology Tiruchirappalli for their valuable support throughout the course of our project work.

ANISH AKODE

M. ANUNAY REDDY

SUDAM ELIYA

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

PDF - Portable Document Format

XAI - Explainable AI

ML - Machine Learning

AI - Artificial Intelligence

XGBoost - Extreme Gradient Boosting

LightGBM - Light Gradient Boosting Machine

DNN - Deep Neural Networks

RF - Random Forests

DT - Decision Trees

KNN - K-Nearest Neighbor

SVM - Support Vector Machine

CNN - Convolutional Neural Network

LSTM - Long Short-Term Memory Networks

SHAP - SHapley Additive exPlanations

TP - True Positives

TN - True Negative

FP -False Positive

FN - False Negative

DLL - Dynamic link library

CHAPTER 1
INTRODUCTION

1.1 Introduction

The use of PDF (Portable Document Format) [1] files is ubiquitous in modern society. They are used for exchanging documents over the internet, and their use is prevalent across various domains such as healthcare, finance, and government. However, PDF files are also a popular target for malware attacks due to their widespread use and the difficulty in detecting malicious content within them. Malware can cause significant damage to computer systems, leading to data breaches, financial loss, and system downtime. Therefore, there is a demand for productive techniques to discover PDF malware and protect computer systems from potential damage.

The application of machine learning algorithms for malware detection has been on the rise prevalent in recent times, as a result of its effectiveness in detecting and preventing cyberattacks [7]. However, the deficiency of visibility and interpretability of machine learning models has hindered their widespread adoption in the field, as it is often difficult to understand how the models are making decisions. In the process of PDF malware detection, the challenge is further exacerbated by the complexity and diversity of PDF files, which makes it challenging to develop effective and reliable models.

1.2 Problem Statement

The problem we address in this project is the development of effective and interpretable machine learning models for detecting PDF malware. Existing methods for PDF malware detection often lack transparency and interpretability, making it difficult to understand how the models are making decisions. Additionally, the complexity and diversity of PDF files make it challenging to develop reliable models that can accurately detect instances of malware.

1.3 Aims and Objectives

The aim of our work is to develop explainable machine learning models for detecting PDF malware that are both effective and interpretable. Our objectives are to boost the exactness and trustworthiness of PDF malware detection, and to provide knowledge about the decision-making process of the models, thereby increasing transparency and interpretability. To achieve this aim, we have the following objectives:

- To carry out a review of existing literature to determine the current state-of-the-art in machine learning models for detecting malware in PDF files. This objective will enable us to gain a thorough comprehension of the latest techniques being used and the constraints associated with them.
- Collect and preprocess data to train and test our models. This involves gathering and preparing data to train and evaluate our models.
- Develop and evaluate machine learning models that can detect PDF malware and provide explanations for their decisions. This involves building models that can detect PDF malware accurately and provide human-readable explanations for their decisions.
- Compare our models' performance and evaluate their transparency and interpretability.

1.4 Background

PDF malware is a growing threat to computer systems, and its detection is challenging due to the format's complexity. PDF files contain various objects, including images, fonts, and metadata, that can be used to embed malicious content. Additionally, the format's cross-platform compatibility and ubiquitous use make it an attractive target for attackers.

Machine learning models [2] have shown potential results in detecting various types of malware, including PDF malware. However, traditional machine learning models lack transparency and interpretability, which limits their usefulness in security applications.

Explainable machine learning models aim to address this limitation by providing insights into the features that contribute to their decision-making process.

1.5 Research Framework

Our research framework is built on a combination of supervised machine learning techniques and explainable AI methods. To achieve our objectives, we employed static and dynamic analysis techniques combinedly and trained several machine learning models.

Specifically, we used the PDFiD [16] tool for static analysis to extract features from PDF files, and the Cuckoo sandbox [17] for dynamic analysis to capture runtime behavior. We then used the extracted features as input to train four different machine learning models, including random forest, XGBoost, LightGBM, and DNN. Each model was trained on a dataset of categorized PDF files to detect instances of malware.

We evaluated how each model performs using several metrics, including accuracy, precision, recall, and F1-score, and compared the results to determine the most effective model for our purposes. Additionally, we utilized explainability techniques, such as SHAP, to gain insights into how the models arrive at their decisions and to ensure their transparency.

1.6 Summary

Our project aims to develop explainable machine learning models for finding PDF malware that are both effectual and interpretable. We employed both static and dynamic analysis techniques and trained different machine learning models and deep learning models, evaluating their performance and using explainability methods such as SHAP to gain insights into their decision-making process. The significance of our research stems from its potential capabilities to improve the current state of PDF malware detection and enable organizations to better protect themselves against cyber threats.

CHAPTER 2

LITERATURE REVIEW

2.1 Background Study:

Malware [1] refers to a category of harmful software that is specifically intended to infiltrate and cause harm to computer systems, networks, or devices. There are several forms of malware, such as viruses, worms, trojans, spyware, ransomware, and adware. These malicious programs serve a range of purposes, such as stealing sensitive data, interrupting computer operations, or demanding money from victims.

The detection of malware involves the identification and elimination of harmful software from computer systems. It is a crucial process for safeguarding computer systems against the detrimental impacts of malware. Malware detection techniques can be categorized into two types: signature-based detection and behavior-based detection.

Signature-based detection is regarded as the most commonly used malware detection technique. Signature-based detection relies on the use of malware signatures, which are unique patterns of code that are associated with known malware. Signature-based detection functions by matching a file or program's signature against a collection of established malware signatures stored in a database. When a match is detected, the file or program is identified as malware and removed.

Behavior-based detection is a more advanced malware detection technique that uses machine learning models to identify suspicious behavior that may be associated with malware. Behavior-based detection works by analyzing the behavior of a file or program and comparing it to known patterns of malware behavior. If the behavior of a file or program is similar to known malware behavior, it is identified as malware and removed.
However, both signature and behavior based detection techniques have limitations. Signature-based detection can only detect known malware and may be ineffective against new or unknown malware. On the other hand, behavior-based detection can be complex to implement and may produce false positives, leading to the removal of legitimate files or programs.

Explainable Machine Learning (XAI) [8] models have been developed as a solution to the limitations of traditional malware detection techniques. XAI models can provide insights into how a model reaches its decision, increasing transparency and trust in the model's output. XAI models can be used to detect both known and unknown malware and can provide valuable insights into the characteristics of malware. XAI models are capable of improving the accuracy and efficiency of malware detection, making them an important area of research in the field of cybersecurity.

2.2 Existing models and methods

2.2.1. Efficient malware detection

In recent times, PDF malware has emerged as a major risk to information security. To identify and remove PDF malware, different detection methods have been put forward, which include signature-based, heuristic-based, and machine learning-based approaches [10].

Efficient and effective malware detection is a pivotal task in the area of cybersecurity. Malware is a kind of malevolent software that can cause harm to computer systems, networks, and devices by stealing sensitive information, disrupting computer operations, or extorting money from victims. Traditional malware detection techniques, such as signature and behavior detection, have limitations in terms of their accuracy and efficiency. To enhance the accuracy and efficiency of malware detection, machine learning and deep learning models have been developed.

One approach to efficient and effective malware detection is the use of ensemble learning [12]. The concept of ensemble learning entails the fusion of several machine learning models to enhance the system's overall accuracy and resilience. Ensemble learning can reduce the risk of producing incorrect detection results, including false positives and false negatives, leading to more accurate and effective malware detection. Several ensemble learning techniques have been proposed for malware detection, including stacking, boosting, and bagging.

Another approach to efficient and effective malware detection is the use of deep learning models. Malware detection has exhibited encouraging outcomes with the utilization of deep learning models [9]. Deep learning models' ability to learn complicated patterns in data, including the behavior of malware, leading to improved accuracy and efficiency in malware detection.

Furthermore, explainable machine learning models (XAI) can provide transparency and insight into how a model makes decisions, increasing trust and understanding of the model's output. XAI models can help identify potential biases and errors in the model and provide explanations for the predictions made by the model.

Overall, efficient and effective malware detection requires the use of advanced machine learning models, such as ensemble learning and deep learning models, combined with effective feature engineering techniques. These techniques can enhance the accuracy and efficiency of malware detection, leading to improved cybersecurity and protection of computer systems, networks, and devices. The use of XAI models can also provide valuable insights into the characteristics of malware, leading to improved understanding and detection of malware in the future**.**

2.2.2 Malware analysis techniques

Malware analysis techniques are used to identify and understand the behavior of malicious software [6]. There are several malware analysis techniques that can be employed to develop machine learning models that are both efficient and effective for detecting malware.

- Static Analysis: It is a malware analysis technique that examines the file or code without executing it. Static analysis techniques include examining the binary code, disassembling the code to examine its functionality, and identifying the code's behavior.
- Dynamic Analysis: It is a method of malware analysis that involves running the malware to monitor and understand its behavior in a controlled environment.

Dynamic analysis techniques include observing system calls, monitoring network activity, and detecting changes in the system's behavior.

- Hybrid Analysis: It is the combination of static and dynamic analysis techniques. It involves analyzing the malware file statically and dynamically to identify the malicious behavior.
- Memory Analysis: Memory analysis is a malware analysis technique that examines the memory of a system to identify malicious behavior. Memory analysis techniques include examining process memory, analyzing system data structures, and identifying rootkit behavior.
- Sandbox Analysis: Sandbox analysis is a malware analysis technique that executes the malware in a controlled environment. It simulates the behavior of a real system and identifies the malicious behavior of the malware.
- Signature-based Analysis: Signature-based analysis is a malware analysis technique that identifies known malware by matching up to a database of known signatures. This method is commonly used by antivirus software to identify and remove known malware.

By using these malware analysis techniques, machine learning models can be trained to accurately find and prevent malicious infections. The data obtained from malware analysis can be used to identify and extract features that can be fed into machine learning models to improve their accuracy and efficiency.

2.2.3 Comparative Analysis of Research Papers

Table 1. Comparative Analysis of Research Papers

S.NO	Title	Tools used	Algorithms used	Significance
1	Interpreting Machine and Deep Learning Models for PDF Malware Detection using XAI and SHAP Framework (2023) [8]	Python, scikit-learn,TensorFlow,Lime, SHAP	Random Forest, CNN	The authors find that both the Random Forest and CNN models are effective in detecting PDF malware, with the CNN model performing slightly better. Overall, this research is significant because it demonstrates the usefulness of XAI techniques in interpreting the predictions made by machine learning models for PDF malware detection.
2	PDF Malware Detection Based on Fuzzy Unordered Rule Induction Algorithm (FURIA) (2023) [9]	Python, scikit-fuzzy, Adobe Reader	Fuzzy Unordered Rule Induction Algorithm (FURIA), Decision Tree	The authors find that the FURIA model is able to achieve higher detection rates and lower false positive rates than the decision tree model. They also find that the FURIA model is more robust and less prone to overfitting than the decision tree model.

3	Design and Analysis of Machine Learning Based Technique for Malware Identification and Classification of Portable Document Format Files (2022) [10]	Python, scikit-learn, Weka, Adobe Reader	Decision Tree, Random Forest, Gradient Boosting, k-NN, SVM, Naive Bayes	The authors find that the Random Forest algorithm achieves the highest accuracy for both malware identification and classification tasks. They also find that the Gradient Boosting algorithm performs well for malware identification, while the SVM and Naive Bayes algorithms perform well for classification.
4	Increased evasion resilience in modern PDF malware detectors: Using a more evasive training dataset (2022) [11]	Python, scikit-learn, Adobe Acrobat Reader	SVM, Random Forest, XGBoost, Gradient Boosting, Neural Network	The authors train various machine learning algorithms on both the evasive and benign datasets and compare the performance of the models to those trained on only the benign dataset. They find that training on the evasive dataset significantly improves the evasion resilience of the models, with the SVM algorithm achieving the highest performance.
5	PDF Malware Detection based on Stacking	Python scikit-learn	Decision Tree, Logistic	The authors use a dataset of benign and malicious PDF

	Learning (2022) [12]	Weka Adobe Acrobat Reader	Regression, Random Forest, Gradient Boosting, Stacking Ensemble Learning	files to train and test their model, comparing the performance of the stacking model to that of individual base classifiers. The authors find that the stacking model is able to achieve higher detection rates and lower false positive rates than the individual base classifiers.
6	PDF Malware Detection Based on Optimizable Decision Trees (2022) [13]	Python scikit-learn Weka Adobe Acrobat Reader	Optimizable Decision Trees (ODT), Random Forest	The authors use a dataset of benign and malicious PDF files to train and test the ODT model, and compare its performance to that of the RandomForest algorithm.The authors find that the ODT model is effective in detecting PDF malware, achieving high detection rates while maintaining low false positive rates.
7	EvadeRL: Evading PDF malware classifiers with deep reinforcement learning(2022) [14]	Python TensorFlow scikit-learn Malware Analysis Sandbox	Deep Reinforcement Learning (DRL), CNN	The authors use a dataset of benign and malicious PDF files, as well as a pre-trained malware classifier, to train and test their model. The authors find that their DRL

				model is able to generate adversarial PDF files that are able to evade the malware classifier with high success rates.
8	HAPSSA: Holistic Approach to PDF malware detection using Signal and Statistical Analysis (2021) [15]	Python scikit-learn MATLAB	PCA, SVM, k-NN, Random Forest	The authors use various features extracted from PDF files, including image-based, text-based, and layout-based features, to create a comprehensive feature set. They then apply signal processing techniques to the feature set to extract relevant information and reduce the dimensionality of the data.
9	PDF-Malware: An Overview on Threats, Detection and Evasion Attacks (2021) [1]	Feature Extraction using PDFid	Artificial Neural Network and SVM implementation	Evasion attacks are to fool the classifier by changing the features of the infected PDF files so that the classifier considers them as clean.
10	Malware Analysis on PDF (2020) [6]	Analyzing the pdf using PeePDF	Injecting the malicious code into the pdf	Demonstrated pdf structure and some of the properties of PDF.

11	Robust PDF Malware Detection with Image Visualization and Processing Techniques (2020) [2]	Image visualization (PDF files to grayscale images), Image Feature extraction. Byte plot and Markov plot	Random Forests (RF), Decision Trees (DT) and K-Nearest Neighbor (KNN)	The proposed method is more robust than the state-of-art learning-based method in resisting reverse mimicry attacks.
12	Malware Detection on Byte Streams of PDF Files Using convolution neural network (2019) [5]	Drop-out, batch normalization	Making the convolutional neural network by using different layers using Adam's optimizer.	Demonstrated the performance of the proposed network by experiments using manually labeled PDF documents.
13	Towards Adversarial Malware Detection: Lessons Learned from PDF-based Attacks (2019) [7]	Python PyTorch TensorFlow scikit-learn	Generative Adversarial Networks (GANs), CNN, SVM	The authors use a dataset of benign and malicious PDF files, as well as adversarial PDF files generated using a generative adversarial network (GAN), to train and test their models.They found that the GAN-generated adversarial PDF files are able to bypass traditional PDF malware detection methods, demonstrating the

				need for more robust and resilient detection methods.
14	Malware Detection in PDF Files Using Machine Learning (2018) [4]	Static Analysis using PDFID	SVM Classifier.	The proposed method blocks various attacks and removes features that are vulnerable to PDF detection.
15	Malicious PDF Detection Using Metadata and Structural Features (2012) [3]	Feature Selection by considering the robustness of the model	Naive Bayes Classifier, RandomForest, SVM	Mentioned the feature description of the PDF document which is used to effectively identify malwares.

In our project, we put up a procedure for detecting PDF malware using explainable machine learning models. We combine the strengths of both static and dynamic analysis techniques to draw out a comprehensive set of attributes that capture the behavior of PDF files. We then train and evaluate several machine learning models, including Random Forest [19], XGBoost [20], LightGBM [21], and DNN [22], to categorize PDF files as either clean or malicious. To interpret the models and gain insights into their decision-making process, we use the SHapley Additive exPlanations (SHAP) [23] method for explainability.

Our proposed approach is inspired by several related papers in the field of PDF malware detection. The paper [1] highlights the need for effective and efficient detection techniques for PDF malware. The paper [2] proposes an approach for visualizing PDF files as images and using image processing techniques to detect malware.

The paper [3] suggests using metadata and structural features for malware detection. The paper [4] proposes utilizing machine learning models for classification of PDF files as either malicious or benign. The paper [5] proposes deep learning has been proposed for detecting malware, which involves the analysis of byte streams contained in PDF files.

Finally, the paper [6] provides an overview of different types of PDF malware and their analysis techniques.

We build upon the insights and techniques proposed in these papers to develop a comprehensive approach for detecting PDF malware using explainable machine learning models. By using a mixture of static and dynamic features and interpreting the models using SHAP, we aim to provide a transparent and accurate way of detecting PDF malware, which can be used by both researchers and practitioners in the field of information security.

CHAPTER 3
PROPOSED METHODOLOGY

3.1 Architecture of the Proposed Method

The Architecture of the Proposed Method is a crucial component of a project that outlines the design and implementation of a new approach or solution to a specific problem. Below show the architecture of our proposed method:

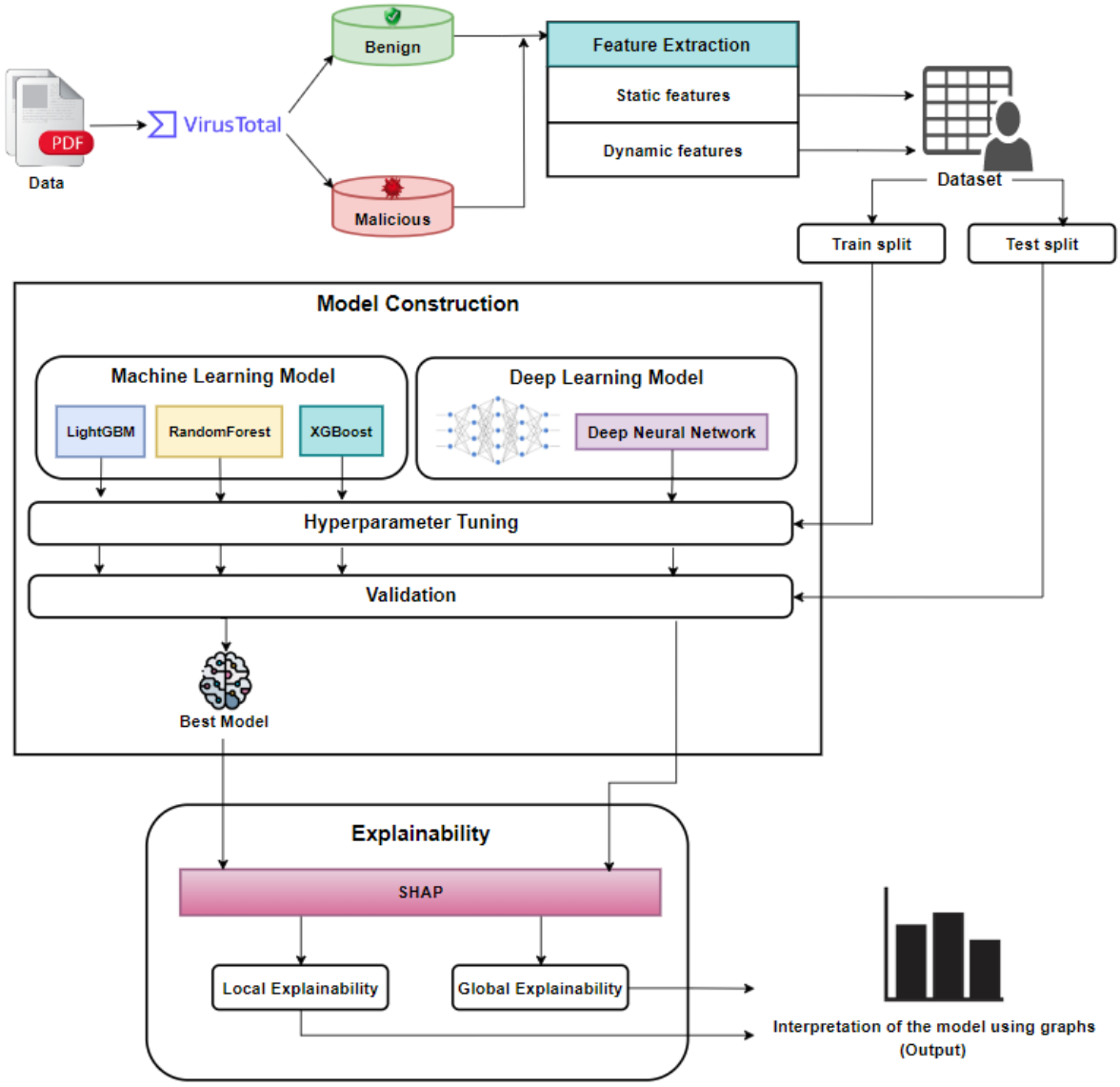

Fig. 1 Proposed Architecture diagram

This is the architecture diagram flow implemented in our project as shown in Fig 1. Below is the brief description of the section:

Data: This section is focused on the data that will be utilized to train the machine learning model. It might include information about the data sources, how the data is collected, and any pre-processing or cleaning that is required to prepare the data for feature extraction and model training.

VirusTotal: VirusTotal is a website that aggregates information about malware and other suspicious files. In this section, you may be integrating with the VirusTotal API to retrieve information about potentially malicious files and incorporate that information into the model.

Feature Extraction: In this section, you'll be extracting features from the data that will be utilized to train the machine learning model. These features may include file size, file type, the presence of specific keywords, and other relevant characteristics that can help distinguish between malicious and benign files.

Dataset: This section is focused on the creation and management of the dataset that are going to be used to train the machine learning model. It includes information about the process of dividing the data into training, validation, and test sets.

Model Training: This section involves the actual training of the machine learning model using the features extracted from the dataset. You may be using various algorithms and techniques to train the model, such as supervised, unsupervised, or deep learning.

Explainability: In this section, you'll be working on interpreting the model's predictions to gain a better understanding of how the model is making decisions. This may involve generating explanations for individual predictions or creating visualizations to help explain how the model is weighting different features.

Interpretation of Model using Graphs: This section is focused on creating visualizations that help explain how the model is making predictions. This may include plotting feature importance scores, generating confusion matrices, or creating other types of visualizations that help explain how the model is performing.

3.2 Data collection

The first step in our research was to collect a dataset of PDF files that contained malware and benign samples. We collected data from multiple sources including Contagio Dump [24], Malware Bazaar [25], and various Ebooks. Contagio Dump provided data from the period between 2015 to 2020, while Malware Bazaar provided data from 2020 to 2023. We also collected clean PDF files from Ebooks for comparison purposes.

Contagio Dump is a well-known repository of malware samples, maintained by a malware researcher. The data available on Contagio Dump includes various types of malware, including PDF malware. Our dataset included 11,098 malicious and 9,194 benign PDF files from Contagio Dump.

Malware Bazaar is another source of malware samples that provides a collection of malware samples contributed by researchers and organizations from around the world. Our dataset includes 168 malicious PDF files from Malware Bazaar.

We also collected a set of clean PDF files from Ebooks for comparison purposes. Our dataset includes 7,734 clean PDF files from Ebooks.

In total, our proposed dataset consists of 28,102 PDF files, including 11,266 malicious PDF files and 16,836 benign PDF files. This proposed dataset was used to train and evaluate our machine learning models for the classification of PDF files as malware or benign.

After collecting the data we gave all the files to VirusTotal [26]. VirusTotal analyzes files and URLs by running them through multiple antivirus engines, which check for known signatures and behaviors of malware.

3.3 Feature Extraction

In order to develop an effective and accurate model for detecting PDF malware, it is necessary to extract relevant features from the PDF files. This is done using both static and dynamic analysis techniques.

3.3.1 Static Analysis

Static analysis is a technique used to analyze a PDF file without executing it. It

involves extracting features from the PDF file's structure, metadata, and content. The features obtained from static analysis can be used to determine whether a PDF file is malicious or benign. In the context of our project, we used static analysis to extract features from PDF files that are malicious and benign. In Fig2 the structure of PDF is shown.

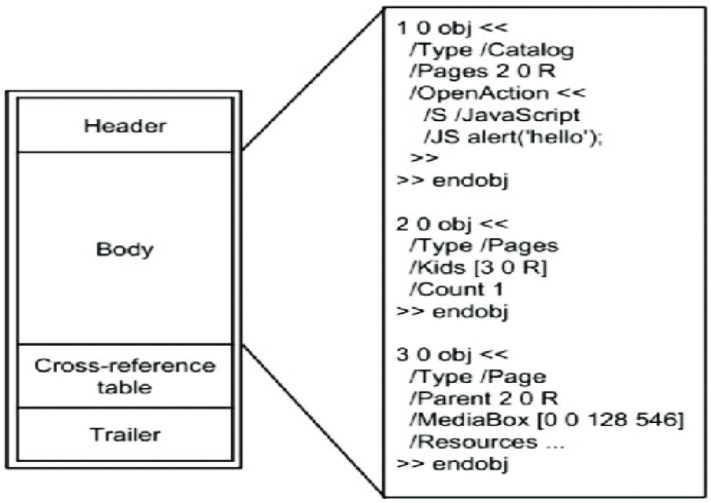

Fig. 2 Structure of a PDF file

One of the tools we used for static analysis is PDFID [16], which is a script that analyzes a PDF file and provides a report of its PDF object structure. PDFID extracts the following features:

- /Page
- /Encrypt
- /ObjStm
- /JS
- /JavaScript
- /AA
- /OpenAction
- /AcroForm
- /JBIG2Decode
- /RichMedia
- /Launch
- /EmbeddedFile

- /XFA
- /URI
- /Colors > 2^24
- obj
- endobj
- stream
- endstream
- xref
- trailer
- startxref

3.3.2 Dynamic Analysis

Dynamic analysis is a crucial part of the feature extraction process, as it enables us to extract information about the behavior of the PDF file when executed in a sandboxed environment. For our study, we used the Cuckoo sandbox environment to analyze the behavior of the PDF files. The sandbox environment enables us to open the PDF files in a quarantined environment and monitor their behavior

Fig. 3 Work flow of cuckoo

Steps to install Cuckoo [18],

1.	Installation of Ubuntu 20 Desktop Go to https://ubuntu.com/download/desktop and download the .iso.

2. We need to install python, cuckoo requires python2.
3. We need to install the Python supporting tools, mongodb.
4. Cuckoo’s recommended database is PostgreSQL.
5. Now we will install Virtual Box 6.1.
6. Download and install all the plugins needed for cuckoo to work. So now we start with installing Volatility, Distorm3.4.4 and Yara-python3.6.3
7. Then install some other dependencies- openpyxl, ujson, jupyter, tcpdump.
8. Then install cuckoo with 'sudo -H pip install -U cuckoo'.
9. Create our Windows 7 Virtual Machine and make some changes to make the Virtual Machine vulnerable on purpose.
10. Now install Adobe Reader,python,pillow Flash, Java and chrome in the virtual machine.
11. Upload the agent.py file from your Ubuntu host to windows7 guest at startup.
12. In virtualbox manager Select Network and then change Attached to: Host-only Adapter and select vboxnet0 in the Name.
13. We want to take a snapshot of the Virtual Machine.
14. We need to finish off the configuration of the Cuckoo services on our Ubuntu 20 workstation.
15. To use Cuckoo you first need to update Cuckoo’s scoring signatures so open a Terminal and type 'cuckoo community'. Then type 'cuckoo' to use the cuckoo sandbox.

During the dynamic analysis process, we extract various features connected to the behavior of the PDF file as shown in Fig. 3, such as the list of system calls, registry keys accessed, mutexes created, and DLLs loaded. We used the open-source Cuckoo sandbox [17] to perform dynamic analysis of the PDF files. Cuckoo provides an environment to execute the PDF files and monitor the system calls, network traffic, and file system activities performed by the PDF files.

We used the following features extracted from the dynamic analysis of PDF files:

- DLLs
- Mutex

- System calls
- Registry keys

Table 2 Static Analysis features Explanation

Features	Explanation
obj	Indicates the beginning of a new object in the PDF file.
endobj	Indicates the end of an object in the PDF file.
stream	Indicates the beginning of a stream object, which can contain arbitrary data.
endstream	Indicates the end of a stream object.
xref	Cross-reference table that maps object numbers to byte offsets within the file.
trailer	Contains information about the document, such as the root object, the page count, and the document ID.
startxref	Byte offset of the cross-reference table.
/Page	Indicates the presence of a page object in the PDF file.
/Encrypt	Indicates the presence of encryption in the PDF file.
/ObjStm	Indicates the presence of object streams in the PDF file.
/JS	Indicates the presence of JavaScript in the PDF file.
/JavaScript	Indicates the presence of JavaScript in the PDF file.
/AA	Indicates the presence of additional actions in the PDF file.
/OpenAction	Specifies an action to be performed when the document is opened.

/AcroForm	Indicates the presence of an AcroForm in the PDF file.
/JBIG2Decode	Indicates the presence of JBIG2Decode filter in the PDF file.
/RichMedia	Indicates the presence of RichMedia annotations in the PDF file.
/Launch	Specifies a command to be executed when a link is clicked.
/EmbeddedFile	Indicates the presence of embedded files in the PDF file.
/XFA	Indicates the presence of XFA forms in the PDF file.
/URI	Specifies a URI to be launched when a link is clicked.
/Colors > 2^24	Indicates the presence of images with high color depth in the PDF file.
DLL	Indicates the presence of Dynamic Link Libraries (DLLs) that are loaded during execution of the PDF file.
Mutex	Indicates the presence of named synchronization objects (mutexes) that are created during execution of the PDF file.
Registry Keys	Indicates the presence of modifications to the Windows registry that are made during execution of the PDF file.
System Calls	Indicates the presence of system calls made during execution of the PDF file.

By extracting the features given in Table 1, we aim to capture relevant characteristics of PDF malware that can be used to differentiate it from benign PDF files.

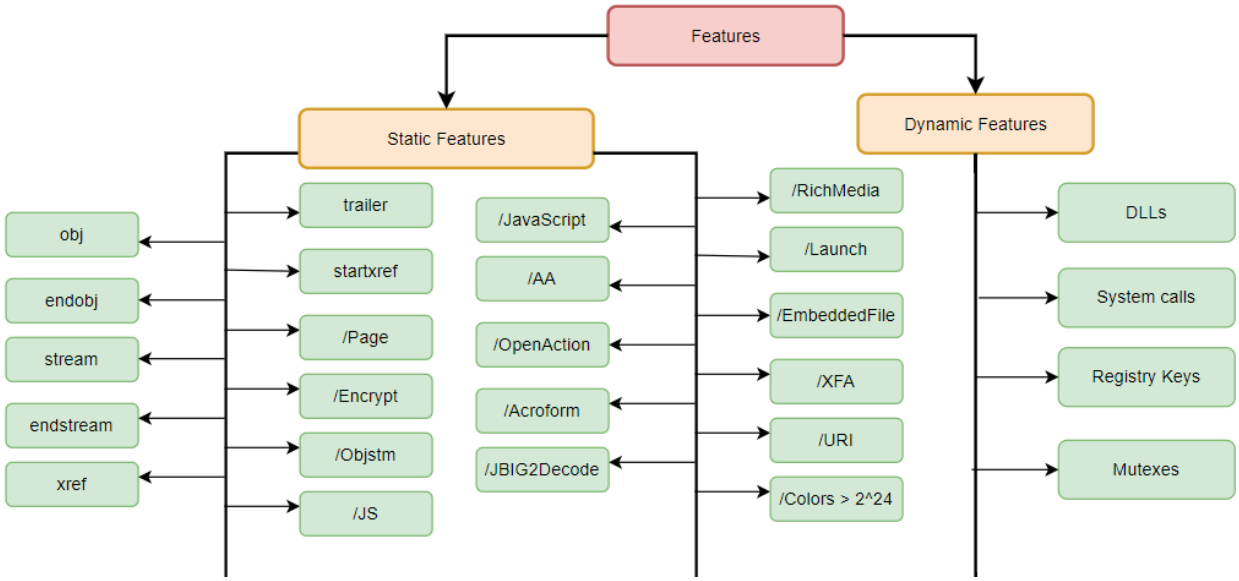

Fig. 4 Features Extracted

3.4 Proposed Dataset Construction

We have collected PDF documents from various databases like Contagio, Malware Bazaar, and E-Books. In total, our dataset consists of 28,102 PDF files, including 11,266 malicious PDF files and 16,836 benign PDF files. Using the PDFiD tool for static analysis and Cuckoo for Dynamic Analysis we generated our proposed dataset .

In the referred papers they have considered 10,000 to 13,000 PDF files from a single database and have considered only one type of analysis. We combined all the files from different databases, which gave our dataset more feature vectors. We have extracted 22 features using PDFiD tool and using Cuckoo Sandbox we have extracted DLLs, Mutex, System calls, Registry keys.

Our Proposed Dataset consists of static features extracted from all the 28,102 files and dynamic features from 1,003 files. The static feature dataset consists of 28,102 rows and 24 columns.

3.5 Feature Selection

3.5.1 What is Feature Selection?

Feature selection is the process of identifying a subset of relevant features from a larger set of potential features for use in a machine learning model. The goal of feature selection is to improve model performance by reducing overfitting, reducing the time and resources needed for training, and improving the interpretability of the model.

In a machine learning model, features are the input variables used to make predictions or classifications. In a PDF malware detection model, potential features could involve various characteristics of the PDF file, such as file size, page count, or metadata information.

3.5.2 Why Feature Selection?

Feature selection is an important step in machine learning because it allows us to identify the most relevant features for a given task and reduce the dimensionality of the dataset. Optimizing machine learning models' accuracy, efficiency, and interpretability can be achieved by exclusively selecting pertinent features.

Including a detailed description of the feature selection process in a project report is important for several reasons:

- Reproducibility: By providing a clear and detailed description of the feature selection process, other researchers can reproduce the results and build upon the work.
- Transparency: Feature selection is a critical aspect that can greatly impact the performance of a machine learning model, and it's important to disclose the techniques and criteria used for transparency.
- Interpretability: Feature selection can also improve the interpretability of a machine learning model by reducing the number of input variables and identifying the most important features for making predictions.
- Efficiency: Feature selection can help to reduce the dimensionality of the dataset, which can lead to faster training times and lower computational requirements.

3.5.3 Types of Feature Selection

There are several techniques for feature selection, each with its own advantages and disadvantages. Here are some common types of feature selection techniques that you could describe in a project report:

- Filter methods: Filter methods involve selecting features based on their statistical properties, such as correlation with the target variable or mutual information with the target variable. These methods are often fast and simple to implement, but they may not consider the interactions between features.
- Wrapper methods: Wrapper methods involve training and evaluating a model with different subsets of features and selecting the subset that results in the best performance. These methods can be computationally expensive, but they can potentially capture interactions between features and result in higher accuracy.
- Embedded methods: Embedded methods involve selecting features as part of the model training process, such as regularization methods like L1 regularization. These methods can be computationally efficient and result in higher accuracy, but they may not be as interpretable as filter or wrapper methods.
- Principal Component Analysis (PCA): PCA is a dimensionality reduction method that involves projecting data onto a lower-dimensional space while preserving the maximum variance. This technique can be employed as a feature selection approach by selecting the principal components with the highest scores.
- Recursive Feature Elimination (RFE): RFE is a wrapper method that involves recursively removing the least important features and retraining the model until the desired number of features is reached. This can be a computationally expensive method, but it can potentially result in higher accuracy and improve the interpretability of the model.
- Genetic Algorithms: Genetic algorithms are optimization techniques that use principles of evolution to search for an optimal subset of features. These methods

can potentially result in higher accuracy, but they may be computationally expensive and difficult to interpret.

3.5.4 Random Forest Feature Selection

In this research, we used random forest feature selection to identify the most important features for detecting PDF malware. Random forest is an ensemble learning method that combines multiple decision trees to improve model performance and reduce overfitting. This is known as "embedded feature selection" since it is built into the algorithm itself rather than being a separate step in the analysis.

To perform random forest feature selection, we first trained a random forest model on the entire dataset, including all potential features. We then used the feature importance scores generated by the random forest model to rank the features based on their importance for detecting PDF malware. We then selected the top n features based on their importance scores and used them in our machine learning model for detecting PDF malware.

The feature importance score in random forest is calculated based on the decrease in impurity achieved by splitting on a particular feature. The more a feature decreases impurity, the higher its importance score. We used the Gini importance metric, which is a measure of how often a particular feature is used to split the data in the random forest model.

3.6 Proposed Model Training

3.6.1 Why Machine Learning Techniques?

Machine learning has been increasingly used for malware analysis and detection due to the sheer volume of malware that is produced every day. Traditional signature-based approaches are not sufficient for detecting new and unknown malware, which is why machine learning-based approaches have become popular.

One of the key advantages of using machine learning for malware detection is its ability to identify and learn from new and unknown malware. Machine learning models

can be trained on large datasets of known malware to identify common patterns and features that can be used to detect new and unknown malware. These models can then be used to classify new software and determine whether it is malicious or not.

Another advantage of using machine learning for malware detection is its ability to detect polymorphic malware. Polymorphic malware is malware that can change its form and behavior in order to evade detection by traditional signature-based approaches. Machine learning algorithms can detect patterns in the code of polymorphic malware that indicate its malicious intent, even when its form and behavior have changed.

Deep learning, a subset of machine learning, has also been used for malware detection. Deep learning algorithms can analyze large datasets of malware and extract complex features and patterns that may not be detectable by traditional machine learning approaches. This can improve the accuracy and effectiveness of malware detection.

3.6.2 Algorithms

3.6.2.1 Random Forest

Random Forest is an ensemble learning technique for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Random forest works by creating a set of decision trees, where each tree is built using a subset of the dataset and a subset of the features. During the training phase, each decision tree is trained on a different subset of the data and features, which ensures that the trees are diverse and not highly correlated. During the prediction phase, the final prediction is made by averaging the predictions made by all the trees in the forest.

3.6.2.2 XGBoost

Extreme Gradient Boosting (XGBoost) is a widely used machine learning algorithm

in malware detection and analysis. It is a variant of gradient boosting algorithm that uses decision trees as weak learners. XGBoost has become a standard in machine learning competitions due to its accuracy and speed.

XGBoost is widely used in malware detection because it can handle high-dimensional and sparse data effectively. Malware detection is a classification problem where the intent is to classify whether a given file is a malware or not. It can be used to train on binary classification dataset of features extracted from files. The features can include static and dynamic analysis attributes such as system calls, API calls, and opcode sequences.

3.6.2.3 LightGBM

LightGBM is an open-source gradient boosting framework that was developed by Microsoft. It is anticipated to be fast, extensible, and productive in handling large datasets. LightGBM uses a tree-based learning algorithm that constructs trees in a leaf-wise manner rather than the traditional depth-wise manner used by other gradient boosting frameworks. This approach reduces memory usage and improves training speed.

LightGBM has become popular in the area of data science and machine learning because it provides superior performance in terms of accuracy and training time compared to other gradient boosting frameworks. It has been used in a variety of applications, including image recognition, natural language processing, anomaly detection, malware detection due to its high accuracy and fast training speed.

In malware detection, LightGBM is used to classify malicious samples into different categories comparing their features such as size and type of file, API calls, and behavior. The features are extracted from the samples using various techniques such as static, dynamic and hybrid analysis. The extracted features are then fed into the LightGBM model which learns the structure in the data and fabricates the predictions on new samples.

3.6.2.4 Deep Neural Network

A Deep Neural Network (DNN) is an artificial neural network (ANN) that possesses several masked layers connecting the input and output layers. The hidden layers enable the network to study and extract increasingly composite and abstract attributes from the input data. DNNs are often used in supervised learning tasks such as speech recognition, natural language processing (NLP), and regression problems.

The architecture of a DNN typically consists of an input layer, numerous hidden layers, and an output layer. Each and every layer includes multiple neurons, also called as nodes that receive inputs, perform computations, and produce outputs.

Deep Neural Networks (DNNs) are used for malware detection and analysis. Complex patterns and relationships between inputs and outputs can be learned by DNNs through training with a large amount of data. In the context of malware detection, the input to the DNN is typically a set of features that describe a particular file, and the output is a binary classification indicating whether the file is malicious or benign.

3.7 Evaluation Metrics

Evaluation metrics are an essential part of any project report, especially in the sphere of machine learning and data science. They help in assessing the performance of a model or a system and comparing it with other competing models. In our work we have considered Accuracy, Recall, Precision and F1 Score metrics.

3.7.1 Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. It is a matrix of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions.

In binary classification, a confusion matrix consists of a pair of rows and columns. The rows represent the actual class of the samples, while the columns represent the predicted class. The four possible combinations of actual and predicted classes are:

- True positive (TP): The model predicted the sample as positive, and it is actually

positive.

- False positive (FP): The model predicted the sample as positive, but it is actually negative.
- True negative (TN): The model predicted the sample as negative, and it is actually negative.
- False negative (FN): The model predicted the sample as negative, but it is actually positive.

The confusion matrix allows us to calculate several performance metrics for the model, such as accuracy, precision, recall, and F1 score. It is a useful tool to diagnose the strengths and weaknesses of a classifier and to determine which classes the model is having difficulty predicting correctly.

3.7.2 Accuracy

Accuracy is widely used evaluation metrics in machine learning. It estimates the amount of correct predictions made by a model over the total number of predictions. It is defined as follows:

Accuracy = (Number of correct predictions) / (Total number of predictions)

Accuracy is simple to calculate and easy to interpret. The suitability of using this metric to assess the performance of a model may not always be the best approach, especially when dealing with imbalanced datasets, where the number of instances of one class is much larger than the other.

3.7.3 Recall

It is referred to as sensitivity or true positive rate, and is a commonly used performance metric in machine learning classification tasks. It is used to measure the proportion of actual positive cases that were correctly identified by the classifier.

Recall = True Positives / (True Positives + False Negatives)

A high recall value stipulates that the model is good at recognizing instances of the

target class and a low recall value indicates that the model is not finding some instances of the target class. The recall metric is particularly relevant in imbalanced datasets where the target class is underrepresented and it is essential to detect all instances of the target class.

3.7.4 Precision

Precision is a metric used in machine learning to compute the measure of true positive results (i.e., correctly identified positive results) among all the positive results predicted by a model. In other words, precision computes the model's ability to correctly recognize the positive class (e.g., detecting a disease) without falsely identifying too many negative examples as positive (i.e., avoiding false positives).

Precision = True Positives / (True Positives + False Positives)

A high precision value shows that the model is correctly guessing most of the positive instances and is less likely to generate false positives. However, a high precision value does not necessarily imply a high recall score, which measures the proportion of true positive instances that are rightly identified by the model.

3.7.5 F1 Score

F1 score is a metric used in machine learning to evaluate the performance of a classification model. It is the harmonic mean of precision and recall, and provides a single score that balances the trade-off between precision and recall.

F1 score = 2 * (precision * recall) / (precision + recall)

The F1 score ranges from 0 to 1, with a higher value indicating better performance. A value of 1 represents perfect precision and recall, while a value of 0 represents the worst performance possible.

3.8 Explainability

3.8.1 Introduction

Machine learning models are great tools to solve those complex problems not easily handled by conventional programming methodologies. The ability to solve large-scale

problems using just historical data comes at the cost of transparency in terms of the results. The models are “black boxes” to the ML teams and end-users. Explaining why a model makes a prediction can be as important as the model accuracy itself in many applications.

3.8.2 Black Box Model

This refers to machine learning models that are difficult to interpret due to their complex internal workings. These models are trained to make accurate predictions based on input data, but their decision-making process is often opaque to humans. As a result, it can be challenging to understand how and why the model arrived at a certain prediction. This lack of transparency can be problematic in situations where the stakes are high, such as in healthcare or finance, where incorrect predictions can have serious consequences. In order to put up trust in the models and ensure that they are making decisions in an ethical and fair manner, it is essential to have a way to interpret and explain their predictions.

3.8.3 Explainability using SHAP

Explanation or interpretability in AI aims at creating a set of techniques that produce explainable models that are easily understood and trusted. Highly interpretable models make it easier to comprehend why certain decisions or predictions were made. This is particularly important for black box models, where the internal workings are not readily apparent.

One approach to explainability is to use a technique called SHAP (SHapley Additive exPlanations), The SHAP method is based on the concept of Shapley values from cooperative game theory. In the context of machine learning, the "game" is the prediction of a model, and the "players" are the input features used to make the prediction. It applies the Shapley values to feature attribution in machine learning models by computing the marginal contribution of each feature to the prediction, and then aggregating these contributions across all possible feature combinations.

In the case of PDF malware analysis, explainability becomes even more important

as PDF malware attacks are becoming increasingly complex and sophisticated. PDF malware can use multiple techniques to evade detection, such as encrypting malicious payloads or using code obfuscation techniques. This drives the difficulty for traditional signature-based detection procedures to find and prevent PDF malware attacks.

Therefore, using explainability techniques such as SHAP can help to provide insight into how machine learning models are making their predictions for PDF malware detection. By understanding the features that are contributing to a particular prediction, analysts can gain a better understanding of the characteristics of malign PDF files, which can interpret the enhancement of effective detection and prevention strategies.

In our study, we use SHAP to analyze the feature importance of our machine learning models, in order to gain insight into how they are making their predictions. This allows us to better understand the prediction of the models and identify any potential issues or biases. By doing so, we can ensure that our models are making predictions in an ethical and transparent manner, and that they can be trusted to provide accurate and fair results.

3.8.3.1 Local Interpretability

Local interpretation methods aim to help users understand how the model arrived at a particular prediction for a specific input instance, by identifying which features of the input were most influential in the prediction. To provide a local interpretation using SHAP, the method computes the contribution of each feature to the prediction of the model for a specific input instance. This is done by comparing the prediction of the model for the input instance to a baseline prediction, which is typically the average prediction of the model for the entire dataset or a similar reference point.

The SHAP method would then provide a visualization, such as a SHAP plot, that shows the contribution of each feature to the prediction for the specific data sample. The plot shows the positive and negative contributions of each feature, relative to the baseline prediction.

3.8.3.2 Global Interpretability

Global interpretation methods aim to help users understand how the model as a whole arrived at its predictions and what features of the input data are most important for the model's decision-making process. SHAP provides a way to decompose the overall prediction of a machine learning model into contributions from individual input features, allowing us to identify the most important features and their impact on the model's behavior.

To provide a global interpretation using SHAP, the method computes the contribution of each feature to the overall prediction of the model across all input instances. This is done by comparing the average prediction of the model for the entire dataset to a baseline prediction, which is typically the average prediction of a null model that randomly assigns predictions.

The SHAP method would then provide a visualization, such as a SHAP summary plot, that shows the overall impact of each feature on the model's behavior across all the data samples. The plot shows the positive and negative contributions of each feature, relative to the baseline prediction.

CHAPTER 4
EXPERIMENT SETUP

4.1 Dataset Description

To train and evaluate the machine learning models for detecting PDF malware, we built a custom dataset of 28,102 PDF files. We collected the PDF files from various sources, such as Contagio Dump, Malware Bazaar and E-books.

To extract features from PDF documents, we used both dynamic and static analysis techniques. After extracting the features, we pre-processed the data to remove any sensitive information, and then converted the PDF files format that will be used for machine learning analysis.

We divided the dataset into train and test sets, with a split of 70:30. Overall, the custom dataset that we built for this project is diverse, balanced, and includes both static and dynamic features, making it suitable for training and evaluating the proposed machine learning models for detecting PDF malware.

4.2 Hardware Specification

Table 3. Hardware Specification

Hardware	Specification
Processor	12th Gen Intel Core i5-12500, 3000mhz, 6 core(s), 12 logical processors
Graphics Card	2 GB, NVIDIA GeForce MX570/570A, GDDR6
Memory	16 GB, DDR4, 3200 MHz
Display	40.6-cm, Full HD (1920X1200)

Storage	512 GB SSD + 1TB HDD

In *Table 3*, we can see all the hardware specifications of the system which we have used for the project.

4.3 Software Specification

Table 4. Software Specification

Software Type	Specifications
Operating System	Dual Boot - ● Windows 11 Home Single Language ● Ubuntu 20.04 LTS
Software Installed	● Cuckoo Sandbox 2.0.7 ● Visual Studio Code 1.77.3 ● Python 3.11.2 ● Keras 2.12.0 ● Scikit-learn 1.2.1

In *Table 4*, we can see all the software specifications of the system which we have used for the project.

CHAPTER 5
RESULTS AND DISCUSSION

5.1 Static Dataset

In this study, we extracted static features from the PDF document files using the PDFiD tool. The dataset included various features of PDF documents.

	filename	obj	endobj	stream	endstream	xref	trailer	startxref	/Page	/Encrypt	...	/OpenAction	/AcroForm	/J
0	students_cword_puzzle_090809.pdf	210	210	15	15	3	3	3	1	0	...	2	2	
1	capecom.pdf	11	11	3	3	2	2	2	1	0	...	0	0	
2	OOO-36691-1.pdf	40	40	8	8	1	1	1	1	0	...	0	0	
3	LIBRE_OFFICE-67689-0.zip-1.pdf	13	13	3	3	1	1	1	1	0	...	1	0	
4	LIBRE_OFFICE-55958-0.pdf	21	21	4	4	1	1	1	1	0	...	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
28097	32da24c0304076d609d2986c8f97cbfb078a27d6	9	9	3	3	1	1	1	1	0	...	1	0	
28098	0fc253e794c5e93e0743a24f8cb602caefbcf92f	22	22	6	6	1	1	0	3	0	...	2	0	
28099	7364243598bbdea27766986fd32580286976be34	8	8	2	2	1	1	1	1	0	...	1	0	
28100	f5a55b344edb6c2290a1b430ac28eaa0b2675925	14	14	2	2	1	1	1	1	0	...	1	1	
28101	eb7eea498e68c3920d151b935c8a727c3fd492ba	9	9	3	3	1	1	1	1	0	...	1	0	

28102 rows × 24 columns

Fig. 5 Static Dataset

The above Fig. 5, shows the static feature extracted from PDF files using the PDFiD tool. We have a total data size of 28,102 files with 24 features for each file.

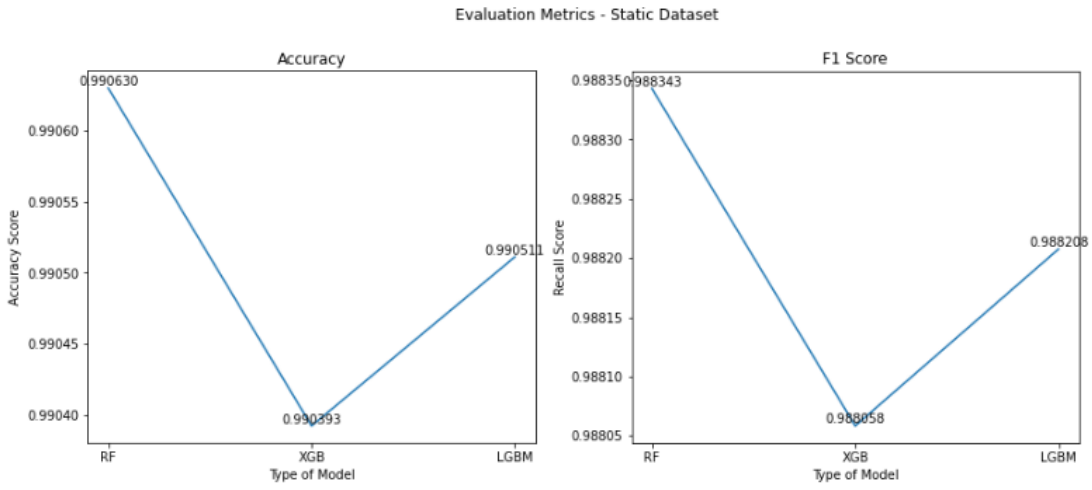

Fig. 6 Accuracy and F1 Score of Machine Learning Models on Static Dataset

As indicated in the above Fig. 6, it shows the comparison of the evaluation metrics on the static dataset. We can see that RandomForest is giving good results compared to XGBoost and LightGBM models.

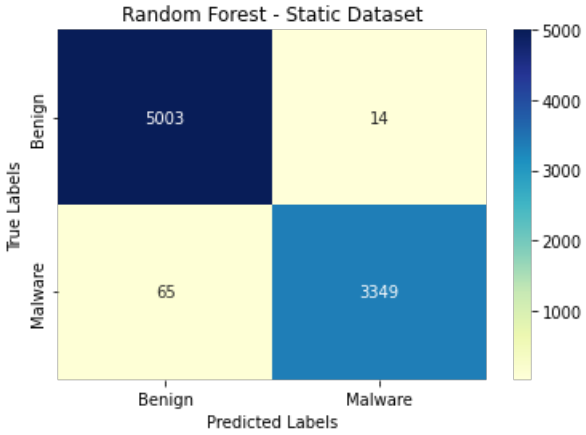

Fig. 7 Confusion matrix of Random Forest on Static Dataset

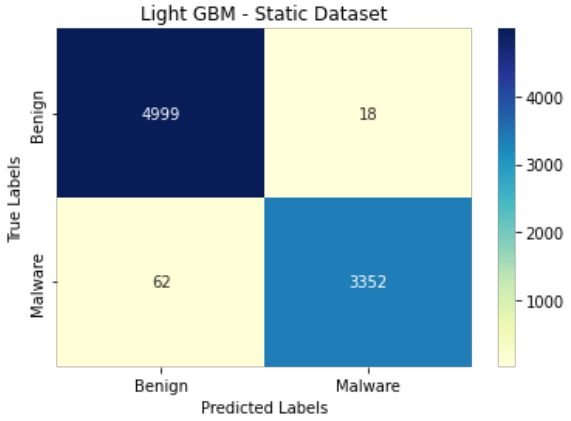

Fig. 8 Confusion matrix of LightGBM on Static Dataset

The above Fig. 7,describes the confusion matrix for the RandomForest trained model on a static dataset. Fig. 8, describes the confusion matrix for the LightGBM trained model on the static dataset. We can see most of the data is well classified.

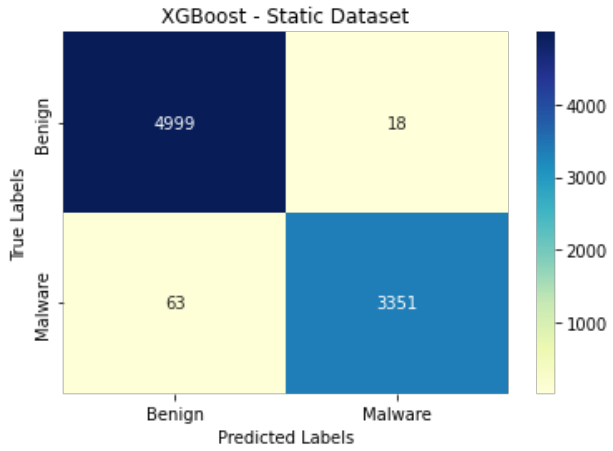

Fig. 9 Confusion matrix of XGBoost on Static Dataset

As can be observed in the figure above Fig. 9, describes the confusion matrix for the XGBoost trained model on a static dataset. Here we can see most of the data is well

classified by XGBoost.

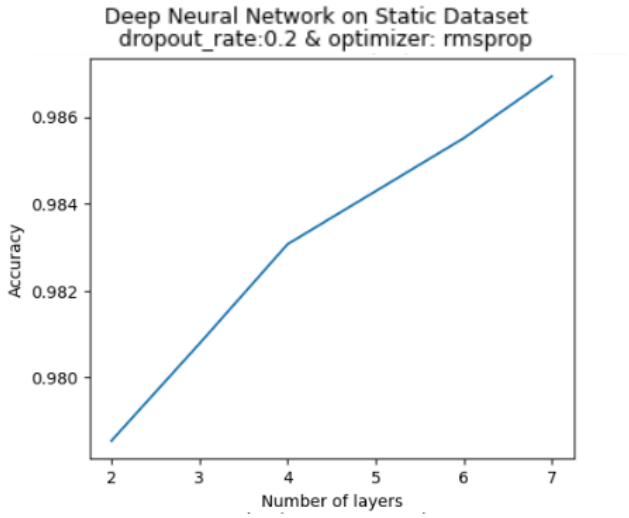

Fig. 10 Best Accuracy DNN Model on Static Dataset

In Fig. 10, we can see the comparison of the Accuracy and Number of Layers in DNN with {'dropout_rate': 0.2, 'optimizer': rmsprop} parameters. Here with 7 hidden layers we are getting the highest accuracy.

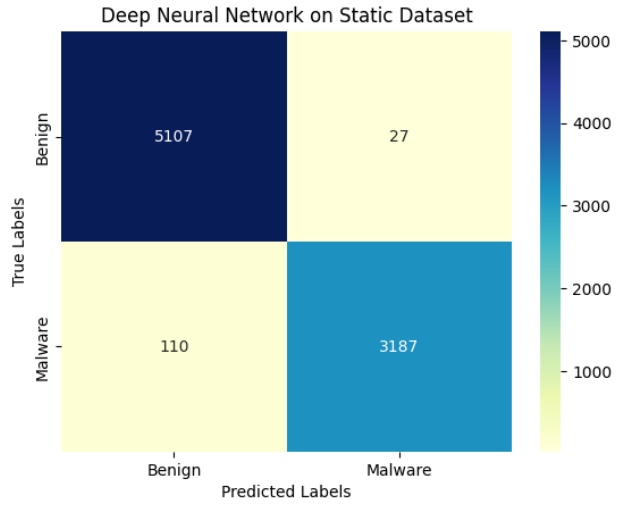

Fig. 11 Confusion Matrix of Best DNN Model on Static Dataset

In the above Figure Fig. 11, describes the Deep Neural Network (DNN) confusion matrix trained on the static Dataset. Here we can see most of the data is well classified by Deep Neural Network.

5.2 Mixed Dataset

In this study, we extracted static and dynamic features from the PDF document files using the PDFiD tool and Cuckoo Sandbox. The dataset included various features of PDF documents.

	obj	endobj	stream	endstream	xref	trailer	startxref	Page	ObjStm	JS	...	servicesexe	conhostexe	636fcaff78c6433d89caa0cf93a5270esqlce_se_lck1
487	9	9	2	2	0	1	0	1	0	1	...	1	0	0
639	8	8	2	2	1	1	1	1	0	1	...	0	0	0
443	9	9	3	3	1	1	1	1	0	1	...	0	0	0
8	22	22	7	7	1	1	1	6	0	1	...	1	0	0
854	9	9	2	2	0	1	0	1	0	2	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
763	4	4	1	1	1	1	1	0	0	1	...	1	0	0
194	74	74	36	36	2	2	2	3	2	0	...	0	0	0
949	74	74	68	68	0	0	3	0	9	0	...	0	0	0
714	8	8	2	2	1	1	1	1	0	1	...	1	0	0
579	9	9	3	3	1	1	1	1	0	1	...	0	0	0

971 rows × 252 columns

Fig. 12 Mixed Dataset

The above Fig. 12, shows the static and dynamic features extracted from PDF files using the PDFiD tool and Cuckoo. Here we have a dataset with 971 malicious and benign files with 252 feature vectors after the feature selection from random forest.

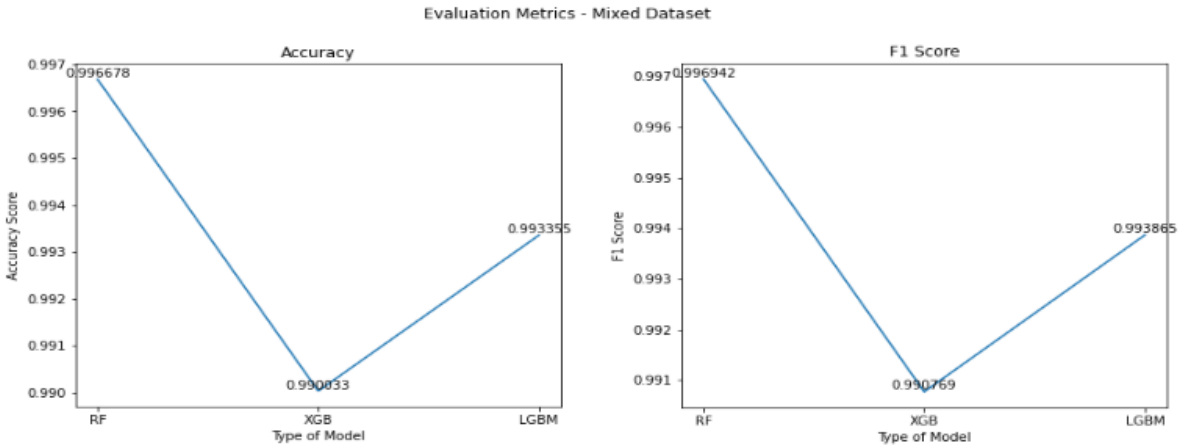

Fig. 13 Accuracy and F1 Score of Machine Learning Models on Mixed Dataset

As indicated in the above Fig. 13, it shows the comparison of the evaluation metrics on the combined dataset. We can see that RandomForest is giving good results compared to XGBoost and LightGBM models.

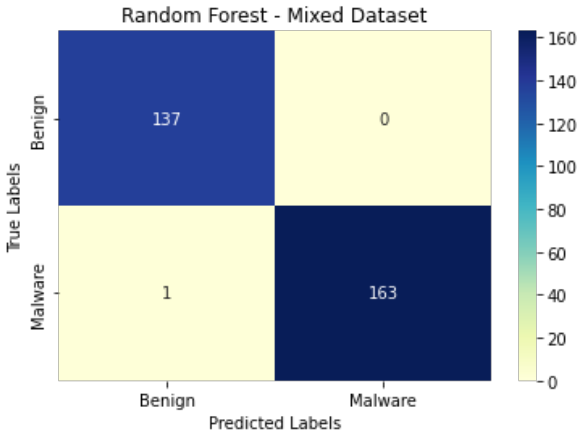

Fig. 14 Confusion matrix of Random Forest on Mixed Dataset

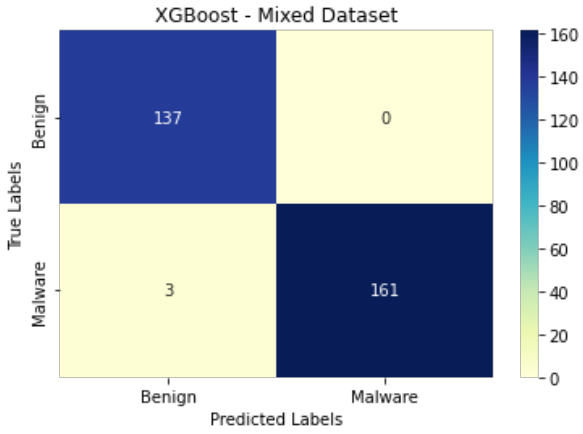

Fig. 15 Confusion matrix of XGBoost on Mixed Dataset

The above Fig. 14, describes the confusion matrix for the RandomForest trained model on mixed dataset. Fig. 15, describes the confusion matrix for the XGBoost trained model on the static dataset.We can see most of the data is well classified

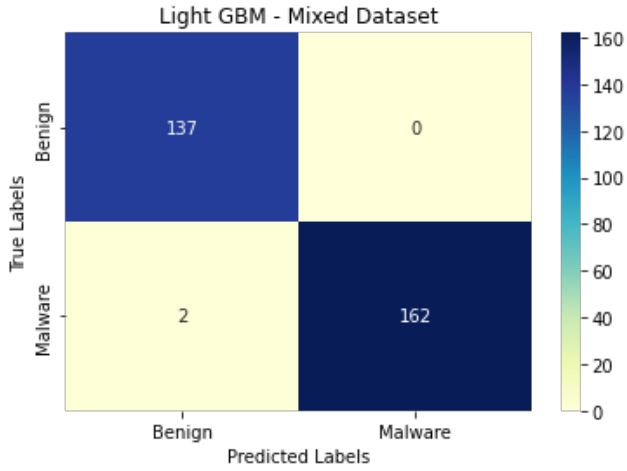

Fig. 16 Confusion matrix of LightGBM on Mixed Dataset

As depicted in the above Figure Fig. 16, describes the confusion matrix for the LightGBM trained model trained on a mixed dataset. Here we can see most of the data is well classified by LightGBM.

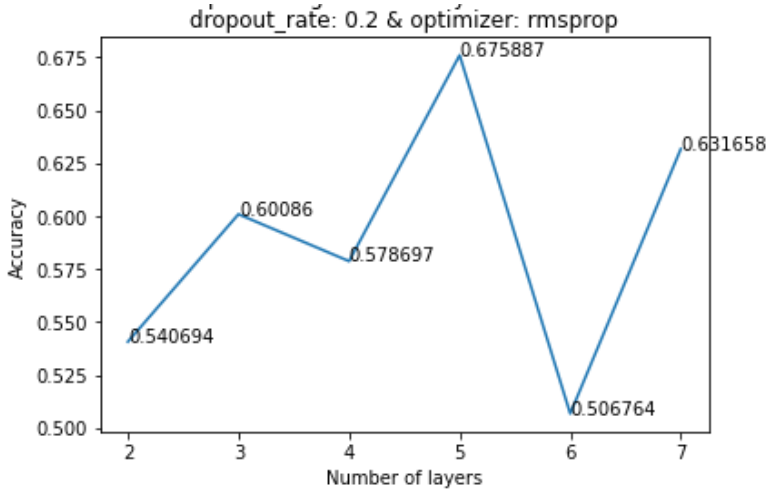

Fig. 17 Best Accuracy DNN Model on Mixed Dataset

In Fig. 17 we can see the comparison of the Accuracy and Number of Layers in DNN with {'dropout_rate': 0.2, 'optimizer': rmsprop} parameters. Here with 5 hidden layers we are getting the highest accuracy.

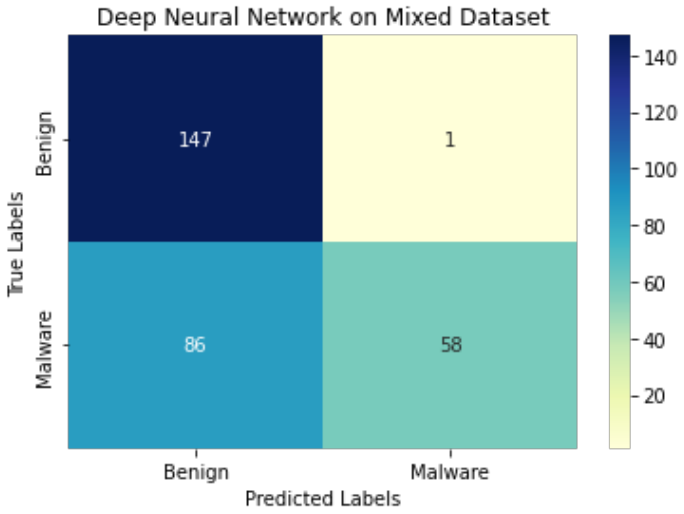

Fig. 18 Confusion Matrix of Best DNN Model on Mixed Dataset

In the above Figure Fig. 18, describes the Deep Neural Network (DNN) confusion matrix trained on the mixed Dataset. Here we can see most of the data is well classified by Deep Neural Network.

5.3 Explainability

5.3.1 Waterfall Plots

Waterfall plot is a type of visualization used in the SHAP framework, which is a popular method for explaining the predictions of machine learning models. In a waterfall plot, each bar represents the value of a single feature added to all model predictions. The bars are arranged in descending order according to their magnitude, with the most important feature at the top and the least important at the bottom.

The plot is split into two sections: the positive and negative contributions. The positive section shows the contributions of the features that improve the prediction value, while the negative section shows the contributions of the features that minimize the prediction value. Each bar height represents the magnitude of contribution, and the color indicates the direction of the contribution (red for positive, blue for negative).

5.3.1.1 SHAP Local Interpretability for Static feature Dataset

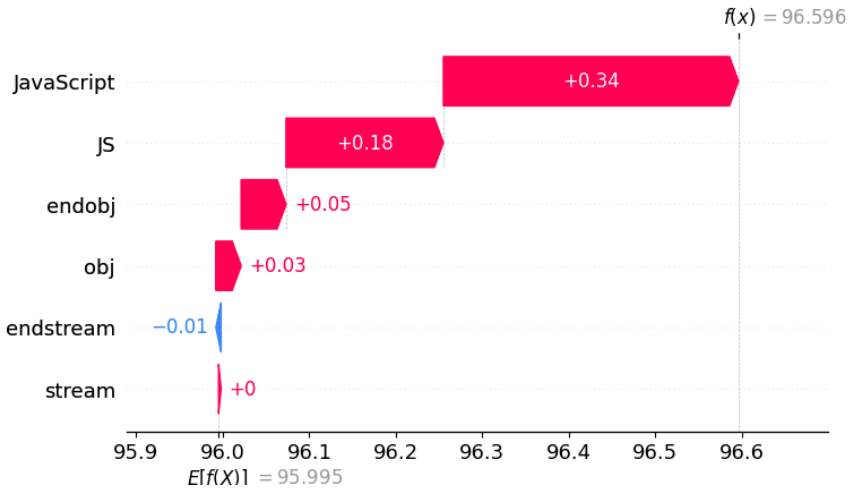

Fig. 19 Local interpretation of a static feature data sample labeled as malicious

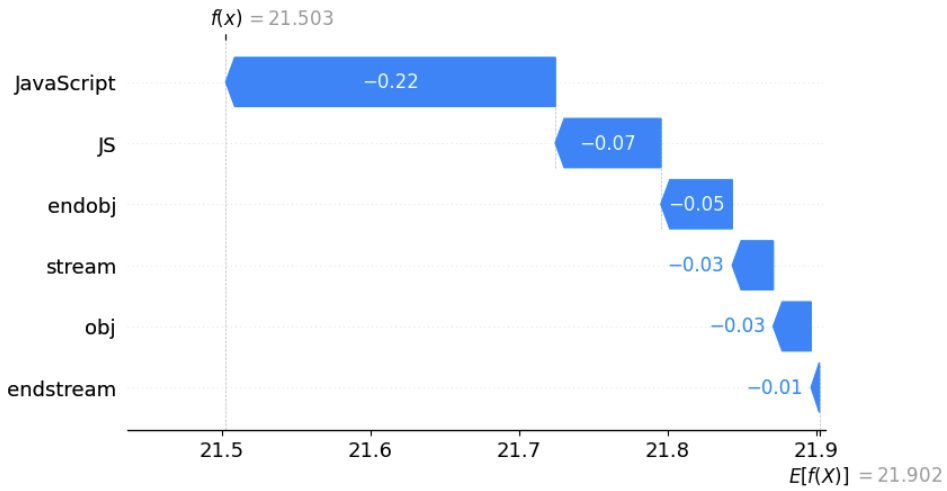

Fig. 20 Local interpretation of a static feature data sample labeled as benign

Local interpretation means the process of explaining predictions of a machine learning model and deep learning model for a single, specific instance or observation. It involves identifying which features or variables are most important for that particular prediction, and understanding how they are contributing to the final result.

In fig 19,20 waterfall plots f(x) and E[f(x)] can be used to determine between malicious and clean labeling of data samples where E[f(x)] is the mean value of classification.

In fig 19 where f(x) >E[f(x)] the data sample is labeled as Malicious on the other hand in fig 20 f(x) < E[f(x)] indicates the data sample is labeled as Benign. The values of each feature indicates the magnitude of contribution towards the classification.

5.3.1.2 SHAP Local Interpretability for Static and Dynamic feature Dataset

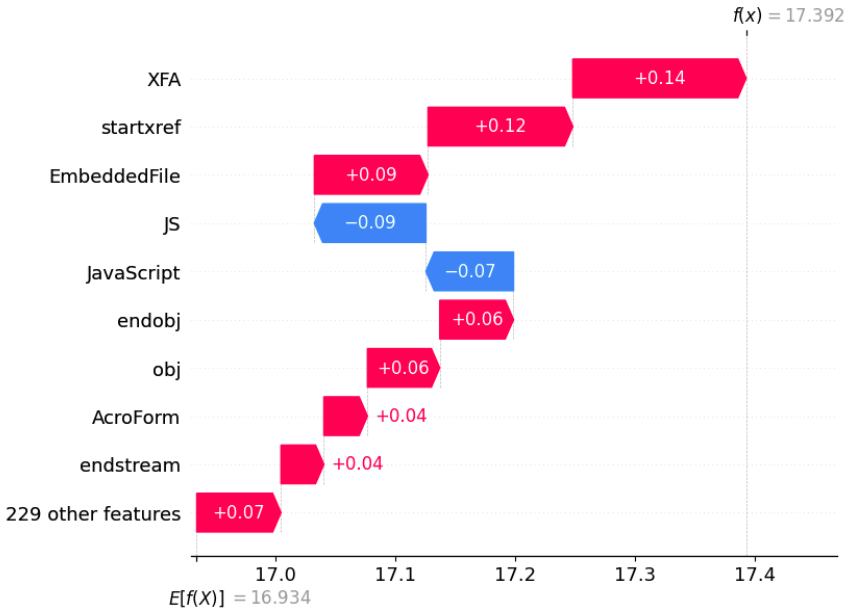

Fig. 21 Local interpretation of a mixed feature data sample labeled as Malicious

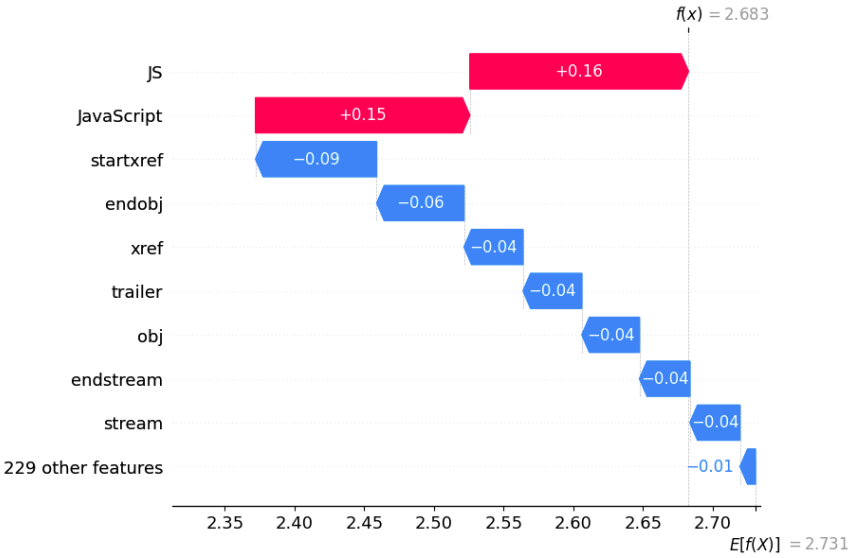

Fig. 22 Local interpretation of a mixed feature data sample labeled as Benign

The two waterfall plots in fig 21,22 are similar to figures 19,20. Here, the waterfall plots are plotted on the combined dataset of static and dynamic features.

5.3.2 Force plots

A force plot is another type of visualization used in the SHAP framework, which is

a popular method for explaining the predictions of machine learning models.

In a force plot, each observation in the dataset is represented by a horizontal line. The line is split into two parts, with the left side indicating the base value (or expected value) of the model, and the right side showing the final model prediction.

The bars are color-coded to indicate whether they are increasing or decreasing the prediction (red for positive, blue for negative).

The length of each bar represents the magnitude of the contribution, with longer bars indicating more important features. The bars are ordered sidewise based on their magnitude, so it is easy to see which features are having the greatest impact on the prediction.

5.3.2.1 SHAP Local Interpretability for Static feature Dataset

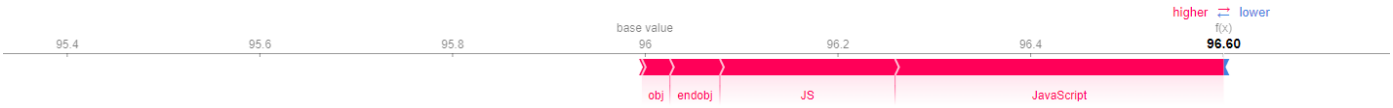

Fig. 23 Local interpretation of a static feature data sample labeled as malicious using force plot

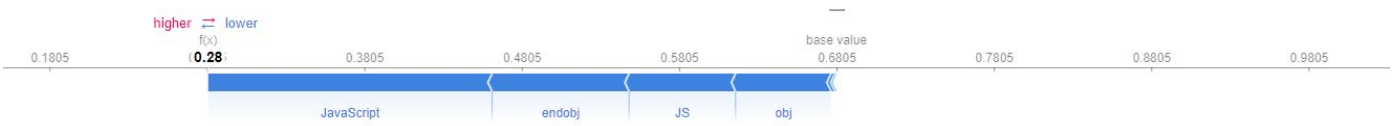

Fig. 24 Local interpretation of a static feature data sample labeled as Benign using force plot

In Fig. 23 the f(x) value greater than the base value (expected value) indicates it is a malicious labeled sample whereas in Fig. 24 f(x) value is less than the base value indicating the data sample is labeled as Benign.

5.3.2.2 SHAP Local Interpretability for Static and Dynamic feature Dataset

Fig. 25 Local interpretation of a mixed feature data sample labeled as Benign using force plot

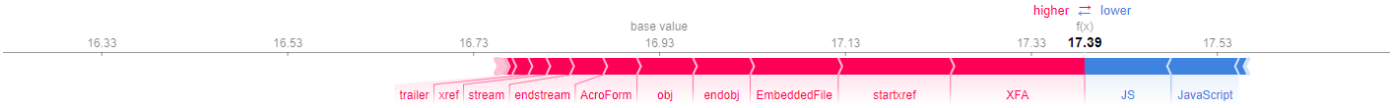

Fig. 26 Local interpretation of a mixed feature data sample labeled as malicious using force plot

Fig. 25, 26 are force plots of data samples from the combination of static and dynamic feature dataset. Fig. 25 is of a Benign labeled data sample in which f(x) value is less than the original value. In Fig. 26 the data sample is malicious labeled in which f(x) value is higher than original value.

5.3.3 SHAP Global Interpretability

Global interpretability refers to the ability to understand and explain the machine learning and deep learning model behavior, and to identify the key factors that drive the model's predictions.

Global interpretability using a summary plot is a technique for understanding the relative importance of different features in machine learning and deep learning models. A summary graph typically expresses the impact of every feature on the output, and can be used to gain insights into the overall behavior of the model.

5.3.3.1 Global interpretability for Static feature Dataset

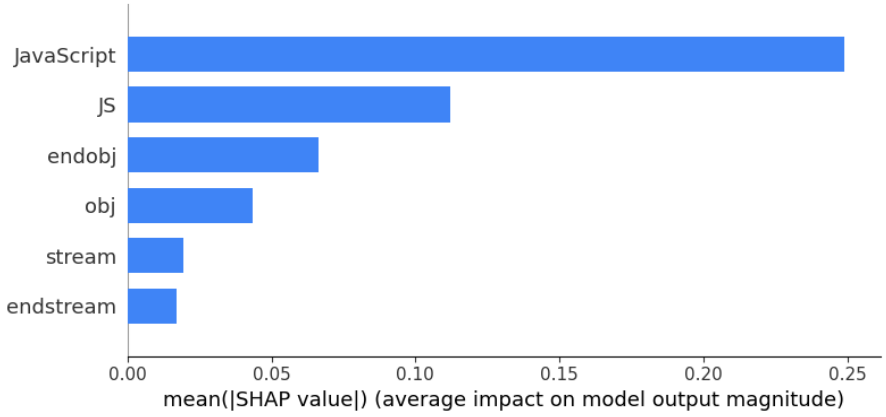

Fig. 27 Global interpretation of the static feature dataset using summary plot

In Fig 26 the plot shows the features' contributions towards the overall behavior of the model. The JavaScript feature stands out to be the prominent feature which contributes the most towards the classification of the model.

5.3.3.2 Global interpretability for Static and Dynamic feature Dataset

Fig. 28 Global interpretation of the mixed feature dataset using summary plot

Fig. 27 shows the contribution of features towards the overall behavior of the model and Javascript stands out to be the prominent feature among all others.

5.4 Discussions

Using PDFiD we have extracted 22 features from 28,102 files and with Cuckoo we have extracted features from 1003 files which contained 3136 features. Cuckoo extracted different features such as DLL's, System Calls, Mutex, Registry Keys and Check Strings. The RandomForestClassifier is called with the SelectFromModel function.

The machine learning and deep learning models built are very well hyper parameter tuned using GridSearchCV. The GridSearchCV will train the model on every possible combination of the parameters and select the best parameters with the scoring parameter as accuracy. Finally we get the Best Score and Best Parameters from the GridSearch results.

We have done the training on 2 datasets - static and mixed datasets. In the mixed dataset we combined the static and dynamic features. After training with these datasets, Random Forest is performing well on both the datasets compared to XGBoost, LightGBM and DNN. The RandomForest has an accuracy of 0.9906 on static dataset and 0.9966 in mixed dataset.

The best trained model has been used to develop the explainability. The trained model is given to SHAP which then produces an explainable model and generates feature importance values for each feature which are known as Shapley values. These shapley values are then used to interpret the model and features' importances.

The explainability is done in two ways: Local Explainability and Global Explainability. In local explainability, for each different data sample the prominent feature that contributes more to make it either malicious or benign may differ. In global explainability, for the static feature dataset and the mixed features dataset the JavaScript stands out to be the most prominent feature towards the overall behavior of the model.

CHAPTER 6
CONCLUSION

Malware has become a major security threat in recent years, and traditional methods of defense, such as anti-virus software that relies on malware detection based on signatures, are no longer effective against new types of malware. To address this challenge, we have approached malware analysis and classification as a machine learning issue, using state-of-the-art techniques to develop our models.

We compared the models built using Random Forest, XGBoost, LightGBM and DNN in detecting PDF malware that has been shown to be effective in both static and dynamic analysis. The combination of PDFiD and Cuckoo Sandbox for static and dynamic analysis respectively, provided a comprehensive analysis of the PDF malware.

Based on the results in terms of accuracy, precision, and recall, Random Forest outperforms all the models in malware detection. We achieved the highest accuracy of 99.06% on static feature set and 99.66% on static and dynamic feature set with Random Forest. In addition to our analysis and classification of malware using machine learning and deep learning techniques, we have also conducted a comparison of various models. This comparison serves as a valuable contribution to the field and will be allowed to identify more sophisticated types of malware in the coming days.

The explainability of the models was ensured by using methods such as feature importance ranking and Shapley values. This allowed for the identification of the key features contributing to the classification decision, providing transparency and trust in the decision-making process.

Overall, the use of Explainable Machine Learning models in classification of PDF malware has shown great promise in the field of Malware Analysis. The ability to provide transparency and trust in the decision-making process is crucial, especially when dealing with such critical systems.

REFERENCES

1. Fleury, Nicolas, Theo Dubrunquez, and Ihsen Alouani. "PDF-Malware: An Overview on Threats, Detection and Evasion Attacks." arXiv preprint arxiv:2107.12873 (2021).
2. Corum, Andrew, Donovan Jenkins, and Jun Zheng. "Robust PDF malware detection with image visualization and processing techniques." 2019 2nd International Conference on Data Intelligence and Security (ICDIS). IEEE, 2019.
3. Smutz, Charles, and Angelos Stavrou. "Malicious PDF detection using metadata and structural features." Proceedings of the 28th annual computer security applications conference. 2012.
4. Cuan, Bonan, et al. "Malware detection in pdf files using machine learning." (2018).
5. Jeong, Young-Seob, Jiyoung Woo, and Ah Reum Kang. "Malware detection on byte streams of pdf files using convolutional neural networks." Security and Communication Networks 2019 (2019).
6. https://scholarworks.sjsu.edu/etd_projects/683/
7. Maiorca, Davide, Battista Biggio, and Giorgio Giacinto. "Towards adversarial malware detection: Lessons learned from PDF-based attacks." ACM Computing Surveys (CSUR) 52.4 (2019): 1-36.
8. Rahman, Tahsinur, et al. "Interpreting Machine and Deep Learning Models for PDF Malware Detection using XAI and SHAP Framework." *2023 2nd International Conference for Innovation in Technology (INOCON)*. IEEE, 2023.
9. Mejjaouli, Sobhi, and Sghaier Guizani. "PDF Malware Detection Based on Fuzzy Unordered Rule Induction Algorithm (FURIA)." *Applied Sciences* 13.6 (2023): 3980.
10. Alshamrani, Sultan S. "Design and Analysis of Machine Learning Based Technique for Malware Identification and Classification of Portable Document Format Files." *Security & Communication Networks* 2022 (2022).

11. Ekholm, Oscar. "Increased evasion resilience in modern PDF malware detectors: Using a more evasive training dataset." (2022).
12. Issakhani, Maryam, et al. "PDF Malware Detection based on Stacking Learning." *ICISSP*. 2022.
13. Abu Al-Haija, Q., A. Odeh, and H. Qattous. "PDF Malware Detection Based on Optimizable Decision Trees. Electronics 2022, 11, 3142." (2022).
14. Mao, Zhengyang, et al. "EvadeRL: Evading PDF malware classifiers with deep reinforcement learning." *Security and Communication Networks* 2022 (2022).
15. Mohammed, Tajuddin Manhar, et al. "HAPSSA: Holistic Approach to PDF malware detection using Signal and Statistical Analysis." *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. IEEE, 2021.
16. Didier Stevens. PDF Tools. blog.didierstevens.com/programs/pdf-tools. Accessed 17 Feb. 2023.
17. Cuckoo SandBox. cuckoosandbox.org. Accessed 1 Mar. 2023.
18. James. Cuckoo Installation on Ubuntu 20. utopianknight.com/malware/cuckoo-installation-on-ubuntu-20. Accessed 1 Mar. 2023.
19. Breiman, Leo. "Random forests." Machine learning 45 (2001): 5-32.
20. Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.
21. Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in neural information processing systems 30 (2017).
22. Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." Neural networks 61 (2015): 85-117.
23. Anurag Singh Waliya. Explainable AI for Ransomware Resiliency With SHAP. www.druva.com/blog/explainable-ai-for-ransomware-resiliency-with-shap. Accessed 19 Mar. 2023.
24. Mila. Contagio Malware Dump. contagiodump.blogspot.com. Accessed 12 Feb.

2023.
25. Malware Bazaar. bazaar.abuse.ch. Accessed 13 Feb. 2023.
26. VirusTotal. www.virustotal.com/gui/home. Accessed 1 Mar. 2023.

APPENDIX 1: Code Implementation

This appendix contains the detailed code snippets to implement the proposed method discussed

in this project.

```
def extract_features_PDFiD(file_path):
    # Call the subprocess to execute the pdfid.py
    pdfid_output = subprocess.check_output(["pdfid.py", file_path]).decode("utf-8")
    # Create a dictonary to store the features
    features = {}
    # split the output w.r.t to '\n'
    lines = pdfid_output.split("\n")
    # loop over every line of output from pdfid
    for line in lines:
        if "PDF Header:" in line or "PDFiD" in line:
            pass
        elif "Colors" in line:
            features["/Colors > 2^24"] = int(line.strip().split()[-1])
        else:
            lst = line.strip().split()
            if len(lst) != 0:
                features[lst[0]] = lst[1]
    return features
```

Code for Extracting static PDF features using PDFID

```
obj: 9  endobj: 9      stream: 3      endstream: 3    xref: 1 trailer: 1     startxref: 1    /Page: 1       /Encrypt: 0     /ObjStm: 0      /JS: 1
/JavaScript: 1  /AA: 0  /OpenAction: 1  /AcroForm: 0    /JBIG2Decode: 0 /RichMedia: 0   /Launch: 0      /EmbeddedFile: 0        /XFA: 0 /URI: 0 /Colors > 2^24: 0
```

Output for the code for PDFID

```
# Read the .csv using pandas
df = pd.read_csv("final_staticPDFID.csv")
# Rename the columns
df= df.rename(columns = lambda x:re.sub('[^A-Za-z0-9_]+', '', x))
# Return a random sample of items
df = df.sample(frac=1)
df
```

Reading the Dataset using pandas

```
# Split the data into train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=42)
# Returns the shape of all the train and test samples
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Train and test set splitting

```
# Selecting features based on importance weights
sel = SelectFromModel(RandomForestClassifier())
# Train the feature selection model
sel.fit(X_train, y_train)
```

Calling the Random Forest Classifier for feature selection

```
# Dlls Extractions
def dll_extract(report):
    dll_process={}
    if 'signatures' in report:
        signatures = report['signatures']
        for sig in signatures:
            for marks in sig['marks']:
                if 'dll' in marks:
                    x=marks['dll']['process_name']
                    if x in dll_process:
                        dll_process[x]+=1
                    else:
                        dll_process[x]=1
    return dll_process
```

Code for extracting the DLL's

```
# System Calls Extraction
def syscall_extract(report):
    syscall_process={}
    if 'memory' in report:
        memory=report['memory']
        try:
            for data in memory['ssdt']['data']:
                x=data['syscall_name']
                if x in syscall_process:
                    syscall_process[x]+=1
                else:
                    syscall_process[x]=1
        except:
            pass
    return syscall_process
```

Code for extracting the System Calls

```
# Mutex Extraction
def mutant_extract(report):
    mutant_names={}
    if 'memory' in report:
        memory = report['memory']
        try:
            for data in memory['mutantscan']['data']:
                if data['mutant_name']:
                    x=data['mutant_name']
                    if x in mutant_names:
                        mutant_names[x]+=1
                    else:
                        mutant_names[x]=1
        except:
            pass
    return mutant_names
```

Code for extracting the Mutex

```
# Registry Keys Extraction
def handle_extract(report):
    handle_names={}
    if 'memory' in report:
        memory = report['memory']
        try:
            for data in memory['handles']['data']:
                x=data['handle_name']
                if x and x.startswith('MACHINE\\'):
                    if x in handle_names:
                        handle_names[x]+=1
                    else:
                        handle_names[x]=1
        except:
            pass
    return handle_names
```

Code for extracting the Registry Keys

```
# Check Strings for /URL and IP addresses
def checkstrings(report):
    dic={}
    if 'strings' in report:
        strings = report['strings']
        ip_pattern = r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"
        search_char = '/URL'
        URL=0
        for string in strings:
            if search_char in string:
                URL=1

        ip_address=0
        for line in strings:
            ips = re.findall(ip_pattern, line)
            if ips:
                ip_address=1
        dic={'/URL':URL,'IP_address':ip_address}
    return dic
```

Code for extracting the Check Strings

```
# Read the .csv using pandas
df = pd.read_csv("small_df_mix.csv")
# Rename the columns
df= df.rename(columns = lambda x:re.sub('[^A-Za-z0-9_]+', '', x))
# Return a random sample of items
df = df.sample(frac=1)
df
```

Reading the Dataset using pandas

```
# Split the data into train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=42)
# Returns the shape of all the train and test samples
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

Train and test set splitting

```
# Selecting features based on importance weights
sel = SelectFromModel(RandomForestClassifier())
# Train the feature selection model
sel.fit(X_train, y_train)
```

Calling the Random Forest Classifier for feature selection

```
'''
Parameter Description:
max_depth - controls the maximum depth of each decision tree in the forest
criterion - controls the quality of the split at each node of the decision tree
max_features - controls the number of features that are taken for each split
min_samples_leaf - controls the minimum number of samples required to form a leaf node in the decision tree
n_estimators - controls the number of decision trees in the forest
'''
# Define the parameters for tuning
parameters ={'max_depth': [10, 30, 50, 70, 90],
    'criterion' : ['gini', 'entropy'],
    'max_features': [0.3,0.5,0.7,0.9],
    'min_samples_leaf': [3,5,7,10,15],
    'n_estimators': [100,200,400,600,800]}

# Call the classifier
clf = RandomForestClassifier()
# Call the Grid SearchCV
grid_search=GridSearchCV(clf, parameters, scoring="accuracy", n_jobs=-1, verbose=1)
# Train every instance with the X_train and y_train
grid_search = grid_search.fit(X_train, y_train)

# Print the Best Score and Best Parameters
print('Best Score: ', grid_search.best_score_*100)
print('Best Params: ', grid_search.best_params_)
```

Random Forest model

```
'''
Parameter Description:
max_depth - controls the maximum depth of each decision tree in the boosting process
learning_rate - controls the step size shrinkage that is applied to the weights of the trees
n_estimators - controls the number of decision trees that are added to the boosting process
'''
# Define the parameters for tuning
parameters = {
    'max_depth': [1,2,3,4,5,6,7,8,9],
    'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],
    'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
}

# Call the classifier
clf = xgb.XGBClassifier()
# Call the Grid SearchCV
grid_search = GridSearchCV(clf, parameters, scoring="accuracy", n_jobs=-1, verbose=1)
# Train every instance with the X_train and y_train
grid_search.fit(X_train, y_train)

# Print the Best Score and Best Parameters
print('Best Score: ', grid_search.best_score_*100)
print('Best Params: ', grid_search.best_params_)
```

XGBoost model

```
'''
Parameter Description:
learning_rate - controls the step size shrinkage that is applied to the weights of the trees
num_leaves - controls the maximum number of leaves in each decision tree
min_child_samples - controls the minimum number of samples required to form a new leaf in the tree
subsample - controls the fraction of samples used for training each tree
colsample_bytree - controls the fraction of features used for training each tree
'''
# Define the parameters for tuning
parameters ={
    'learning_rate': [0.01, 0.05, 0.1],
    'num_leaves': [10, 30, 50],
    'min_child_samples': [20, 40, 60, 80, 100],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
}
# Call the classifier
clf = lgb.LGBMClassifier()
# Call the Grid SearchCV
grid_search=GridSearchCV(clf, parameters, scoring="accuracy", n_jobs=-1, verbose=1)
# Train every instance with the X_train and y_train
grid_search = grid_search.fit(X_train, y_train)

# Print the Best Score and Best Parameters
print('Best Score: ', grid_search.best_score_*100)
print('Best Params: ', grid_search.best_params_)
```

LightGBM model

```
# Define the function to create the model
def create_model(units=[16], dropout_rate=0.2, optimizer='adam'):
    # Calling the sequential function
    model = Sequential()
    # adding dense input layer [size of the features]
    model.add(Dense(units=units[0], activation='relu', input_dim=X_train.shape[1]))
    model.add(Dropout(dropout_rate))
    # add layers depending on the number of layers
    for i in range(len(units) - 1):
        model.add(Dense(units=units[i+1], activation='relu'))
        model.add(Dropout(dropout_rate))
    # define the output layer
    model.add(Dense(units=1, activation='sigmoid'))
    # Compile the model
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
```

Deep Neural Network (DNN) Model Definition

```
'''
Define the grid search parameters
units = number of neurons in each layer
dropout_rate = randomly dropping out a fraction of the inputs to each layer (Rgularization to avoid over fitting)
optimizer = specific algorithm that is used to update the parameter.
'''
units = [[32, 16], [64, 32, 16], [128, 64, 32, 16], [256, 128, 64, 32, 16], [512, 256, 128, 64, 32, 16], [1024, 512, 256, 128, 64, 32, 16]]
dropout_rate = [0.2, 0.3, 0.4]
optimizer = ['adam', 'rmsprop', 'sgd']

# Create the parameter grid
param_grid = dict(units=units, dropout_rate=dropout_rate, optimizer=optimizer)

# Create the KerasClassifier object
model = KerasClassifier(build_fn=create_model, verbose=1)

# Perform the grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, verbose=1)
grid_result = grid.fit(X_train, y_train)

# Print the results
print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, std, param in zip(means, stds, params):
    print('%f (%f) with: %r' % (mean, std, param))
```

Deep Neural Network (DNN) Model Tuning