MALWARE ANALYSIS USING MACHINE LEARNING

Major project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

GAUTAM GUPTA (191311)

UNDER THE SUPERVISION OF

Dr. Deepak Gupta

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan-173234, Himachal Pradesh

DECLARATION

I hereby declare that this project has been done by me under the supervision of (Dr Deepak Gupta, Associate Professor, Deptt. Of CSE & IT), Jaypee University of Information Technology. I also declare that neither this Project nor any part of this Project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

Dr. Deepak Gupta

Associate Professor

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology

Submitted by:

Gautam Gupta - 191311

Computer Science & Engineering Department Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the Project report titled "**Malware Analysis using Machine Learning**" in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** and submitted to the Department of Computer Science & Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by "Gautam Gupta (191311)" during the period from August 2022 to May 2023 under the supervision of Dr. Deepak Gupta, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

(Student Signature)
Gautam Gupta, 191311

The above statement made is correct to the best of my knowledge.

(Supervisor Signature)

Dr Deepak Gupta

Associate Professor

Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat

PLAGIARISM CERTIFICATE

ACKNOWLEDGEMENT

First, I express my gratitude to god who provided me with the courage and fortitude to complete the project.

I am grateful and wish my profound indebtedness to Supervisor Dr Deepak Gupta, Associate Professor, Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "Machine Learning" to carry out this Project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this Project.

I would like to express my heartiest gratitude to Dr Deepak Gupta, Department of CSE, for his kind help to finish my Project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a success. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Gautam Gupta
191311

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Numpy	Numerical Python
SVM	Support Vector Machine
KNN	K- Nearest Neighbors
ANN	Artificial Neural Networks
Pandas	Panel Data
Sklearn	Scikit-learn
GPU	Graphics Processing Unit
TPU	Tensor Processing Unit
ML	Machine Learning
APK	Android Application Package
LightGBM	Light Gradient-boosting Machine
PNN	Probabilistic Neural Network
ROC	Receiver Operating Characteristic curve
ROC-AUC	Area under the ROC Curve
PE	Portable Executable
OneR	One Rule
VFI	Voting Feature Interval
MARS	Multivariate Adaptive Regression Splines
SGC	Simplifying Graph Convolutional Networks
OSS	Open-Source Software
DARPA	Defense Advanced Research Projects Agency
API	Application Programming Interface
RAM	Random Access Memory
GB	GigaBytes

CSV	Comma-separated Values
CPU	Central Processing Unit
SHA	Secure Hash Algorithm
CFS	Completely Fair Scheduler
RR	Round-robin
NVCSW	Number of voluntary context switches
NIVCSW	Number of involuntary context switches
CNN	Convolutional neural network
OLS	Ordinary least squares
RELU	Rectified linear activation function
TP	True Positives
FP	False Positives
FN	False Negatives
TN	True Negatives
MSE	Mean Squared Error
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
DDoS	Distributed Denial of Service Attack

ABSTRACT

Malware poses a significant threat to today's infrastructure. Malware is a computer code designed to gain unauthorized access, exploit vulnerabilities and cause overall harm to digital systems all around the world. Today, malware poses a big threat to any country's critical infrastructure such as banks, defense systems, stock markets, etc. Although working in the digital space, the consequences of its actions can reflect in the physical world too. In order to detect and prevent malware from affecting infra, many techniques such as signature-based detection are used but with the advancements in technology, these old strategies are rendered obsolete by ever-evolving malware threats.

Here machine learning has emerged as a powerful agent for detecting and analysing malware, semi-automating the process on a large scale. By training algorithms on a dataset of known malware files, Machine learning models can learn to recognize patterns and features that distinguish malware from a legit file. Machine learning models learn constantly and evolve with each training set, thus countering the evolving threat of malware to some extent.

This project aims to explore the application of machine learning to malware analysis. The report first provides an overview of what malware is and how it affects infrastructure, and then it would introduce machine learning and its potential in malware detection. By training the models on a dataset containing malware and benign files, models would learn to analyze patterns and features these files have and finally we evaluate the effectiveness of the machine learning-based malware detection methods.

Chapter 01: INTRODUCTION

1.1 INTRODUCTION

Malware is a short form for "Malicious software" which refers to any piece of software which is designed with harmful intent. The general purpose of malware is to find vulnerabilities within a system and then exploit it. Generally, malware is deployed on a computer where it can exploit the vulnerabilities and then provide the controls of the computer to the intruder. Malware may work incognito as spyware, or in a loud way like ransomware, in both cases, giving control of the computer to the malware and its developer.

Malware can be classified into many categories based on their roles. Some are viruses, worms, trojan horses, ransomware, spyware and adware. Each type of malware has a unique way of operating and infecting a device.

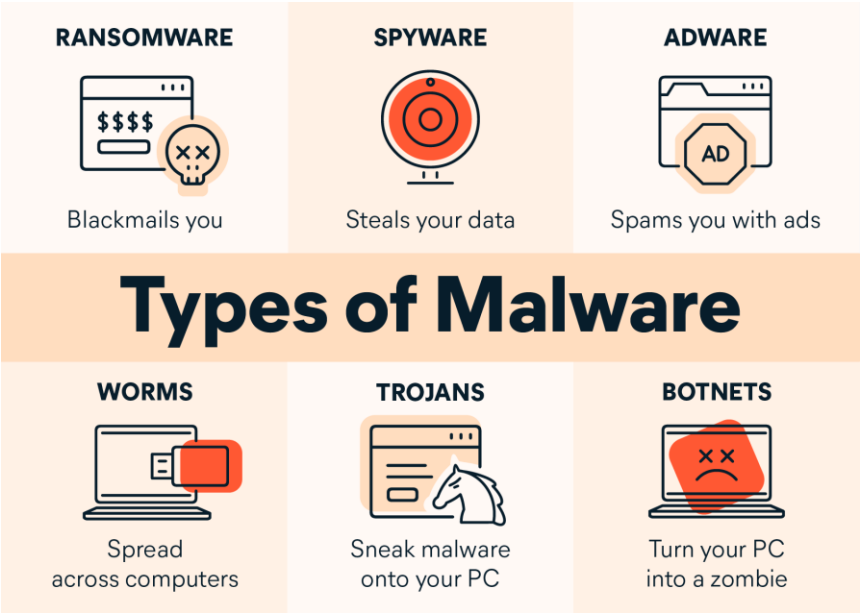


Figure 1.1 - Types of malware [21]

The history of malware dates back to the 1970s when the first computer viruses were created. These early malware were relatively harmless and generally designed for research purposes. However, with the rise of the Internet, malware evolved into a more significant threat. In the late 90s and early 2000s, worms like the Melissa virus and ILOVEYOU virus caused millions of dollars in damages [1].

The threat of malware is ever growing and today, it poses an ever-increasing threat to our modern infrastructure. From national banking systems and military and defence systems to regular computers, every system is vulnerable. For example, in 2017, WannaCry Ransomware infected over 200,000+ computers around the world, causing billions of dollars in damage [2].

Another notable example of malware being used to exploit the bank is the 2016 Bangladesh Bank heist. Allegedly the North Korea-funded Lazerous group used malware to gain access to one of the employee's computer system and then into the mainframe computer of the bank with careful planning where the bank was closed because of holidays, they were able to steal over $700+ million from the bank's Federal Reserve Bank of New York account. Although most of the money was recovered, $81 million was stolen. These hacker groups target third-world countries and their financial infrastructures which are poorly maintained and secured and cause hundreds of millions of dollars in heists every single year.

Despite the increasing threat of malware, many individuals and organizations continue to ignore security practices. This ignorance stems from a lack of awareness. However, the cost of a big heist can be far more significant than the cost of implementing security measures.

1.2 PROBLEM STATEMENT

Malware is one of the biggest threats to computer systems and networks, an ever-evolving one with its destructive capabilities increasing every day. Therefore, studying malware analysis using machine learning can help tackle these threats. The traditional methods of malware detection are still powerful but they are not scalable. On a large scale, analysts cannot check every file while malware is constantly evolving, and here machine learning comes to take the baton of responsibility by automating the process and analysing a large amount of data and detecting patterns in the malware to identify unknown ones. The amount of malware being developed is growing at an alarming rate, and traditional ways of analysis are time-consuming

and not efficient. Machine learning can automate the process and can help speed up the process and provide more accurate results and learn from its mistakes.

1.3 OBJECTIVES

The goals of this project are

1. To identify and understand the behaviour and capabilities of different types of malware.
2. To build machine learning models to automatically detect and classify malware with high accuracy.
3. Improving the overall understanding of the cyber threat space and enhancing cybersecurity practices.

1.4 METHODOLOGY

The project's methodology involves using a dataset of behavioural analysis of both malware and benign Android applications which is created in a controlled environment. The dataset contains system calls made by both Malware and Benign files. After pre-processing, to train and test the models, the dataset is split into training and testing splits at the ratio of 80-20. The training data is used to train four machine learning models: Support Vector Machine, Logistic Regression, Random Forest, and Linear Regression. These models are chosen because they work well for binary classification tasks like dividing unknown data into malicious and benign types.

Following training, the performance of each classifier is compared. Then A voting classifier is developed upon three base classifiers, Support Vector Machine, Random Forest, and Logistic Regression. These base classifiers are trained on a loop on subsets of the training data and then stored in an estimator list. This list is then used to train the voting classifier. Then the voting classifier and the previously trained individual base classifiers are tested on the testing data and their performance is evaluated and compared.

1.5 ORGANIZATION

To achieve the goal of malware analysis with high accuracies, the requirements to make the project work have been mentioned below:

1.5.1 Python

Python is a garbage-collected, interactive, dynamically typed programming language. Developed by Guido Van Rossum between 1985-1990. It is a high-level language that is used for a variety of works including automating tasks, data analysis, etc. The features of python have been mentioned below:

- Readability- Python was designed to be more user-friendly, and easy to code but also allows us to make intelligent models.
- Versatility - Python is a very versatile language as it can be used for a variety of tasks. Python helped us in creating and training machine-learning models.
- Robust variety of support libraries - Because Python is open source, the developers have created countless helpful libraries that help in increasing the scope of usability of Python. In our project, Libraries like Numpy, Seaborn, TensorFlow, matplotlib, and Pandas.

1.5.2 NumPy “Numerical Python”

- NumPy was developed by Travis Oliphant in 2005. It is an open-source Python library which is used to manipulate arrays and provide other functions for working with linear algebra, matrix operations, and the Fourier transform.
- Numpy is a Python library that handles large datasets efficiently. It has support for large, multi-dimensional arrays and matrices.
- It provides researchers with a wide range of mathematical functions that are essential for Machine learning tasks that may include Linear algebra, matrix evaluation, and many more advanced mathematical functions.

1.5.3 Pandas (Python data analysis package)

- Pandas is an open-source Python library which was developed by Wes McKinney in 2008 in response to the need for a trustworthy and adaptable tool for doing quantitative research. Since then, Pandas has developed into one of the most well-known Python libraries.
- Pandas is a Python library that is used for data manipulation and analysis. It is capable of handling and organizing large amounts of data.
- Pandas provides two classes for storing and manipulating data, The data frame and the series.
- Pandas can be used for data cleaning, grouping, filtering, merging, etc.

1.5.4 Sklearn

- Sklearn short form for Scikit-learn is a machine learning open-source library in Python that includes a wide range of supervised and unsupervised learning algorithms as well is capable of pre-processing, feature extraction, model selection and evaluation tools.
- Sklearn is a library that takes the aid of other scientific computing libraries like NumPy, SciPY, matplotlib and hence works like a charm when used along with other machine learning tools.
- Sklearn consists of a range of supervised learning algorithms like Linear Regression, Logistic regression, K Nearest Neighbor, Random Forest, and Support vector machines to name a few. It also supports a range of unsupervised learning algorithms too such as clustering, cross-validation, etc.
- Sklearn enabled us to train supervised learning algorithms like Support Vector Machine, Logistic Regression, Random Forest, and Linear Regression in the project.

1.5.5 Google Collaboratory

Researchers and developers can write, run, and collaborate on Python code for machine learning and data analysis tasks using Google Colaboratory (Colab), a cloud-based Jupyter Notebook environment provided by Google. Colab offers a variety of potent tools and services, including free use of GPUs, TPUs, and other hardware accelerators, pre-installed libraries like

TensorFlow and PyTorch, and integration with other Google services like Drive and Sheets. Some benefits of using google colab are mentioned below:

- Colab is simple to set up and use, and it enables real-time collaboration between multiple users on a single notebook.
- Free GPU Access: Colab offers free access to GPUs, which can greatly accelerate the development of machine learning models.
- Pre-installed Libraries: TensorFlow and PyTorch, two pre-installed libraries that are frequently used in machine learning research, are included with Colab.
- The seamless integration of Colab with other Google services, such as Drive and Sheets, makes it simple to access and manage data.
- Reproducibility: Colab notebooks are simple to share and duplicate, which is crucial for verifying research results and fostering openness in the industry.

1.6 DELIVERABLES OF THE MAJOR PROJECT

An application with the following key features:

- Machine Learning Models: Developed machine learning models to use them in predicting whether a given file is malware or not.
- A comprehensive report detailing the research and findings of the project, including a literature review, methodology, analysis and conclusion

Chapter 02: MAJOR PROJECT SDLC

2.1 FEASIBILITY STUDY

Researchers have found that there is a need to defend highly dependent societies on software from malware. All the military, and strategic, finances of the country are closely linked with a different type of software. Recently, a lot of research has been carried out to find and recognize the existence of malware, as well as the amount of risk that they pose to software assets and operations performed using them. Security experts and analysts have prepared a set of security tools. Systems nowadays come with pre-installed antivirus and firewalls. Antivirus detects particular malware uniquely based on the hashes this is called signature-based analysis, another method used by antivirus is a heuristic method in which malware is analysed based on a predefined set of rules which are determined by analysts that how the malware behaves in the infected device. Malware is created in such ways that they bypass detection and analysis. As the number of malware is increasing exponentially, There is a need of automating the detection of such malicious codes. This is what this project aims at.

2.2 LITERATURE SURVEY

When it comes to Malware detection, many Machine Learning and Data Mining techniques have been proposed by researchers all around the world. Each methodology proposes methods for a specific purpose, some methods proved to have performed better than others. Some of these methodologies have been discussed below.

Hani et al.[3] their research showed a similar view of machine learning algorithms for Android malware. They used a large dataset consisting of 11,598 APK collected from several sources and provided by CLCMalDroid2020, a repository by the Canadian Institute for Cybersecurity at the University of New Brunswick. The paper shows that algorithms like K-NN achieve a much better F1 score when the dataset used for training is balanced and that the LightGBM algorithm is the best-performing algorithm in the research with an F1 score of 95.47%.

Zhang et al. [4] proposed a malware classification method based on the n-gram feature. To explain the N-gram, one can say that it is simply a sequence of n-words that occur together in a sentence or a document. It is a way to represent data in a form such that a computer can analyze and process it effectively. The researcher first used information gain methods for selecting the best n-gram features among all. Then Probabilistic Neural Network (PNN) is used to build the classifier followed by combining the individual predictions made by PNN classifiers through Dempster-Shafer Theory. The methodology was evaluated on a dataset containing 450 Malware and 423 Benign samples and results show better ROC scores for the ensemble of PNNs.

Menahem et al. [5] in their research used three categories of features which are n-gram, function-based and portable executable(PE) features. For evaluation, they constructed five different datasets and considered five base classifiers, namely, OneR, VFI, KNN, Naive Bayes, and C4.5 decision trees to process these datasets. The experimental results showed that troika and stacking which were used to combine the base classifiers, outperformed the base classifiers.

Mukkamalaa et al. [6] proposed a methodology that made use of a majority voting method to ensemble the prediction of various classifiers for detecting malware in network traffic. The base classifiers used by the researchers were SVM, MARS and three types of Artificial Neural Networks (RP, SCG and OSS). The dataset used was taken from Defense Advanced Research Projects Agency (DARPA) for evaluation purposes. The results proved that the majority voting method implemented by the researcher improves the accuracy of the detection of malware.

Landage and Wankhade [7] in their research used opcode sequences of malware samples to train three base classifiers and combined the predictions using majority and veto-based voting. Their research show that veto based voting method had a better detection rate when compared to the majority voting method, but increased the false positive rate as a result.

Ye et al. [8] in their methodology used API calls and strings as features. By constructing eight base classifiers on combinations of features and combining them through a simple voting method, the results proved the usage of the voting classifier as the method outperformed the base classifiers.

Guo et al. [9] used API calls where categorized them into seven classes and trained their base models upon them. The predictions made by these base models were combined and results are compared.

Ozedemir and Sogukpinar [10] proposed a method for the detection of Android malware. Based on API calls made by the malware, they build different base classifiers whose predictions are combined using the majority voting method. The results reveal that the proposed ensemble learning method has improved the accuracy of detecting malware.

For the detection of Android malware, another method [11] is proposed where the researchers used permissions and API calls made by these Android files. Six base classifiers are then trained upon the prepared dataset and then the predictions are combined by a collaborative decision fusion method. The authors in their research claim that this method of ensemble learning gave better results as compared to traditional methods like Adaboost and Bagging.

Sheen et al. [12] used API calls and features extracted from the PE header to build a set of heterogeneous base classifiers. In their research, they proposed two ensemble methods that were used to select and combine a set of base classifiers. These methods attained a 99.7% malware detection rate which was better compared to traditional methods of bagging, boosting and stacking.

Yerima and Sezer [13] proposed a multilevel architecture technique called "Droidfusion" for combining the predictions of base classifiers. In their proposed methodology, they used four ranking algorithms to rank the base classifiers and later combined the results of base classifiers. They used four different datasets to prove that Droidfusion showed better results than traditional ensemble methods.

Kuncheva [14] shed light on the importance of having diversity in the classifiers to make ensemble learning more effective. Diversity is achieved by making use of different kinds of classifiers. Since each classifier has an explicit or implicit bias in its prediction, The combination of such types of classifiers achieves better accuracy than that of an individual classifier.

Krawczyk et al. [15] proposed an approach features are divided into subspaces and each one of these subspaces is used to train a base classifier. A voting method is used to combine the

based classifiers which were selected by an evolutionary algorithm. The results show that this method provides better results.

The aforementioned discussed studies point out that combining multiple Machine learning models outperform a single ML model, by producing a strong model that benefits from the strengths of all the base classifiers it is built upon.

2.2.1 Malware Definition

Malware is an abbreviation for malicious software, that refers to any software or code that is specifically designed to cause harm or damage to a targeted computer system. It is designed by criminals to gain unauthorized access to confidential information, steal data, or extort money.

Malware can be classified into various types -

Type of Malware	Description
Worm	It spread itself from one infected host to others through the OS interface, for instance system call interface.
Botnet	It employs a user's computer as a member of infected computers with a help of central malicious agency.
Rootkit	It hides its existence from other users and applications. Also, it masks the activity of other malicious software.
Trojan	It acts maliciously once installed and masquerade as nonmalware.
Adware	It can deal with the unwanted advertisement.
Spyware	It secretly observes and reports about the usage of user's computer and information.
Virus	It attaches itself on the running programs and spreads through the user's interactions with different systems.
Polymorphic virus	It attaches to a new target by altering its payload to evade detection.
Metamorphic virus	It alters both the functionality and payload that includes the framework for generating future changes.

Figure 2.1 - Classification of malware [17]

- **Adwares** are created to serve unwanted advertisements to users in many ways like popups, browser redirects, hyperlinks, etc on unusual pages to promote malicious or fraudulent products. Adware creators gain revenue from the number of visitors or clicks. Sometimes they act as Spyware.
- **Botnets** are a network of computers which are used to carry out a number of malicious activities. Generally, computers that are part of a Botnet are used to perform DDoS

(Distributed Denial of Service Attack). They can also be used auto Render and Click on ads to benefit its creator.

- **Ransomware** is a type of malware which used to encrypt victim computer's sensitive data by strong encryptions and then demanding a ransom for decryption key. It' is created in such a way that makes it impossible to decrypt the files without the decryption key. WannaCry is one of the famous ransomwares in modern cybercrime history.
- **Rootkits** are malware whose purpose is to provide remote access of victim computer to the attacker without getting detected. Rootkits can further be used to disarm the victim computer by changing system settings and are also used to steal sensitive data.
- **Spyware** is a type of malware which is used to monitor user activity without the knowledge of the victim. It actively monitors users' actions, records browsing history, search history, financial details, Bank login details, and credit card details. Keylogger is a subtype of spyware as it records users' keystrokes which are then sent to its makers.
- **Trojan Horse** enters the user's computer by masquerading itself as a normal file and then performs malicious tasks set by its developer. A Trojan Horse then can be used to perform other malicious activities such as taking control over, downloading other malware, and in general compromising the computer's security.
- **Virus** is a malware that copies itself and spreads to other computers. Once it takes control, it spreads and replicates itself by attaching itself to other files. A Virus needs to be executed by the user to begin it's actions. It can corrupt or delete system files and make the computer unusable.
- **Worms** are similar to viruses, but they don't need user to execute its code to spread itself. It can spread by exploiting network or system vulnerabilities and is designed to overload servers by consuming bandwidth.

2.2.2 Malware Analysis Phases

Malware analysis can be divided into the following phases

Discovery phase: The discovery phase of malware analysis is the stage where the new and previously unknown malware samples are identified and analyzed. One of the critical phases, the Discovery phase is where the malware's behavior is analysed, its purpose, and the impact it may have on the system. It is within this phase, that means such as static analysis, and dynamic analysis are used. The sample is taken either from a malware repository or identified within the targeted system through analysis of the process running. It can also be acquired by means of network traffic analysis, email attachments, and infected files. This phase is taken place within a safe and controlled environment to understand the behaviour of the malware.

Forensic analysis phase: This is the phase where the analysis of a captured malware sample is done. The goal of this phase is to gain a deep understanding of the malware's behaviour, including how it enters a system, how it spreads, and what data it targets. This information can help analysts identify the intents of the malware, its potential source and the damage it may have caused. Through this analysis, the analysts get to know more about vulnerabilities in the system to prevent future malware infections.

Feature extraction - The feature extraction phase of malware analysis involves identifying and extracting relevant features of the malware that can be used for further analysis. Later on, while training machine learning models, the data combined by feature extraction is used. There are two types of techniques which are static analysis where the binary code and metadata of the malware sample are analysed without executing it, and dynamic analysis where the analysis is done on a running sample within a controlled environment.

In their research [16] the authors make use of multiple features obtained through static and dynamic malware analysis. After performing dynamic and static malware analysis in an automated environment, the generated reports are stored and then processed using Apache Spark to extract malware features. They put emphasis on various features such as File metadata, Packer detection, Dynamically Linked Libraries, Windows API Calls, and Registry activities, to name a few.

2.2.3 Challenges of Malware Detection

Malware evolution rate grows exponentially which means there is so much malware that could not be detected and classified due to which signature-based detection failed very badly [18]. Then behavioural analysis was introduced, and some malware even evaded it by countering the traditional detection techniques. One of the most challenging hurdles it posed was when it started masquerading its behaviour in an analysis environment. Furthermore, A small data set can generate biased predictions. If the dataset doesn't have diversity, it may not be able to train the models accurately with modern malware behaviour and features. Also, If the same dataset is used for both training and testing, the results show abnormally high accuracies which might be an indicator of overfitting. To fix this, separate datasets must be used for training and testing [19].

Chapter 03: IMPLEMENTATION

3.1 DESIGN OF THE PROJECT

The Malware Analysis project follows a structured development process, which is divided into several key stages. The primary objective is to develop a system that can accurately classify files into malware and benign by behavioural analysis. This chapter provides an overview of the methodology used with a focus on the dataset used, the flowgraph of the project, and the evaluation of the results.

3.1.1 Dataset

The dataset is a behavioural analysis of Android files in a controlled environment that contains system calls made by malware and benign processes. It contains the observation data of over 100,000 Android files and 35 features which is present in the form of a CSV file. The feature columns include Process ID, and Classification Label, and the rest columns are system call columns, which represent the different types of system calls made by the processes. The system calls provide an insight into how the malware executes its code on the victim machine. The columns provide detailed information about the processes, including the number of child processes that were spawned, the amount of CPU and memory resources that were used and the number of system calls that were made.

The dataset [20] used contains a total of 100,000 data points corresponding to different processes and the details associated with them. The files contained in the dataset are Android files and are studied in a controlled environment and analysed by Cuckoo Sandbox. The dataset features are as follows:

1. **Hash:** Hash column contains SHA256 hash value. The hash function has a special feature that makes it one of the most essential tools to ensure integrity across the internet. Basically, a hash function produces a fixed and unique output for a given input. Given the input isn't changed or altered; the generated hash is exactly the same. The hash value can be used to uniquely identify the files, and it provides a way to verify the integrity and authenticity of

the files. By computing the hash value of an suspected file, in some cases, we can directly compare it to a database of well-known and updated malicious hashes to determine the file type and intent. This process is called Hash-Based Detection and is the first basic step in malware detection.

2. **Millisecond:** The millisecond column in the dataset represents the time stamp of each system call made by the processes in milliseconds. It is calculated from the time the system has started and the time when the call has been made. The timestamp is provided in sequential/chronological order of system calls made by each process. Unusual system call times can be a factor through which checkers can find anomalies. The Significance of this parameter is that it can be used to correlate the system calls made by a process with other significant events or activities happening on the system.
3. **Classification:** The Classification column depicts the category of each process. The entire dataset is divided into two possible values, 1 and 0, where 1 means the process is malware and 0 denotes that it is a benign process. The classification column is critical because it is used as the target variable when training the ML models for Malware detection. Its relevance is solely for the researchers to train and evaluate ML models for malware detection.
4. **State:** The state column in the dataset shows the current state of the process at the time of the system call. In an operating system, Linux in our case, a process can have many different states, including running, sleeping, stopped and terminated. The state column provides us with the behaviour of the processes and their interactions with the OS. Malware present can execute their codes on different occasions say an event, and for the time being they can disguise as a sleeping process, etc. If the process is in a "running" state then we can determine that the process code is being executed on the CPU. For example, '0' as a state indicates that the process is in the TASK_RUNNING state. '4096' Process state means that the process is in the TASK_DEAD state.
5. **usage_counter:** Theusage_counter column in Linux/Unix-based Systems shows the number of times a process has been used. It is a metric that can help us in tracking the frequency with which a process is executed which in turn could help us in identifying processes that are executed more frequently than others. Although malware developers try masquerading the executable code into some benign process and keep the actual usage_counter of malware low, still the param can help us in identifying the frequency.

6. **dynamic_prio:** The 'dynamic_prio' column in the dataset refers to the dynamic priority of the process. In operating systems such as Linux, dynamic_prio reflects the process's current scheduling priority for execution on the CPU. In our dataset, the dynamic priorities mentioned are the ones that were at the time the analysis data was taken. The lower the value, the higher the priority.
7. **static_prio:** The static_prio column in the dataset represents the static priority of the process. Static priority is a fixed priority value assigned to a process that remains constant throughout the process' lifetime. It can be used to analyze the scheduling behaviour of the system and to identify the processes that may be using resources more who might be causing performance issues.
8. **normal_prio:** The normal_prio refers to the priority of the process without considering any priority inheritance. It is the base priority of the process and determines the order in which the scheduler chooses to run the process. If a process has a higher normal_prio, it will be executed before a process with a lower normal_prio.
9. **policy:** Policy refers to the scheduling rules that a process uses. There are several different scheduling procedures, including Round Robin, Completely Fair System, and Real-Time. The policy field contains integer values, such as 0 and 1, to indicate which policy the process is using.
10. **vm_pgoff:** The offset of the process memory page in the virtual memory area of the process is contained in the dataset's vm_pgoff column. By dividing the value of "vm_pgoff" by the page size, it is used to determine the address of the page in the memory. This specific feature aids in our analysis of process memory consumption patterns. A researcher can determine how a process is allocating and managing memory by looking at the values in this column. The more memory it uses, the larger the vm_pgoff.
11. **vm_truncate_count:** The vm_truncate_count is a metric that tells us how many times a process's virtual memory has been reduced in size. It is a useful metric for understanding how a process is utilizing its virtual memory. A high-value count may show that a process is repeatedly using and truncating the memory, which might be an indicator of poorly optimized and inefficient use of the memory.
12. **task_size:** the task_size column in the dataset represents the virtual memory size of the process. Adding more to the previous points about Virtual memory, the virtual memory size of a process includes the memory used by the process itself and by any shared libraries it

uses. The Task_size feature provides us with information about the virtual memory that is allocated to a particular process at a given time. The value is measured in KiloBytes (KB).

13. **cached_hole_size:** cached_hole_size column in the dataset refers to the amount of memory, in kilobytes the process is allocated but is currently not in use. It shows the size of the hole of the process's cached memory. Cached data is stored in memory by the system to be accessed quickly in future if needed. This Metric helps us in understanding the memory usage patterns of a process and identify inefficiencies in the program.
14. **free_area_cache:** free_area_cache column in the dataset represents the amount of memory in KB, that is available in the memory management's free memory cache for a specific process. Based on this metric one can know how much memory is available to the kernel for allocation if in case a process requests memory. Generally, the information received by this metric is used to optimize the performance of a system. In general, the higher the value of this metric, the more memory is available for allocation.
15. **mm_users:** the mm_users column in the dataset provides information regarding the number of users of the memory management structure associated with the process. In the Linux kernel, each process has its memory management structure, and the structure contains information about the process's memory mappings, page tables and other details related to memory management. A high value of mm_users may indicate that a process is using a lot of memory or that there are multiple references to the same memory structure in others.
16. **map_count:** map_count column in the dataset refers to the number of virtual address space areas that are currently mapped by the process. Each process in Linux during execution has its own virtual address space that it uses to store its code and data. This particular metric provides us with an insight into the amount of virtual address space that is currently being used by the process.
17. **hiwater_rss:** the hiwater_rss column represents the peak resident set and the size of the process in KB. It shows the highest amount of memory a process has held in the RAM and how much physical memory is being used by it during its lifetime.
18. **total_vm:** the total_vm metric here represents the total size of the Virtual memory for a given process. In Linux/Unix systems, the total_vm column is used to keep track of the virtual memory used by each process. Used by the kernel of the Operating system, if a process exceeds a set limit, the kernel then terminates the process or takes action to prevent

the system from running out of available memory. Abnormally high usage of memory can be a sign of malicious activity which can be analysed using this metric.

19. **shared_vm:** shared_vm column in the dataset represents the total size of the shared memory in bytes that is used by the process. Shared memory is the memory part that can be accessed by multiple processes. For example, malware may inject its code and masquerades itself in a benign process, in such cases, the shared_vm metric may show unusually high amounts of shared memory being used by a seemingly benign process.
20. **exec_vm:** the exec_vm metric in the dataset is indicating the size of the process's virtual memory area that is used for the executable code. When the malware is executing its code, the executable code's virtual memory area would increase and therefore it can be used to detect the presence of malware in the system. If the exec_vm is suddenly increasing without any particular event, then it is an indication of malware.
21. **reserved_vm:** the reserved_vm metric represents the total size of virtual memory reserved for the process. Oftentimes, Malware uses process injection to allocate virtual memory within another process's address space for execution. How this helps is that the reserved_vm metric can identify abnormal memory allocation that could potentially indicate the presence of malware. Along with other metrics, Reserved_vm can be used to determine the presence of malware.
22. **nr_ptes:** the nr_ptes metric is an abbreviation of the Number of page table entries. This metric indicates the number of page table entries used by a process. Since malware has unusual memory usage, they are bound to have significantly high page table entries, which indicates that it is trying to evade detection techniques by manipulating its memory manipulation. Hence nr_ptes aids us in finding suspicious behaviour.
23. **end_data:** the end_data metric used in the dataset indicates the virtual address right after the end of the data segment of the process. To avoid detection, malware may try to modify or execute code outside of their assigned memory segments, including the data segment. Thus any suspicious activity beyond the authorized limits may aid the malware identification process.
24. **last_interval:** the last_interval column in the dataset shows the time interval between the last time the process was scheduled and the current time. Malware may execute very actively and this metric may help us filter out some processes which are scheduling unusually high as they may have very short last_interval values.

25. **nvcsw:**nvscw means many voluntary context switches, performed by the process. Context switching is a scenario where the Kernel of the operating system, Linux in our case, switches the CPU from executing one process to executing another. A voluntary context switch may occur when a process explicitly calls for example the 'sleep()' command. This may be useful because some malware may use this command to avoid detection while the antivirus is actively working and usually may have a high nvscw count. Along with other aspects, nvscw can help us in identifying anomalies.
26. **nivcsw:** nivcw stands for the number of involuntary context switches, which measure the number of times a process has been forced to give up the CPU before it was finished with its time slice. A malware running may have high context switches which is a sign that a process is causing a lot of involuntary context switches. a high nivcsw value may be a sign that the process is taking up resources and causing starvation for other processes.
27. **min_flt:** min_flt means minor page faults. A minor page fault occurs when a process tries to access a page memory that is not currently loaded into the physical memory and may need a page fault to retrieve it from the disk. A malware that tries to hide its activity by using multiple memory pages and rapidly switching between them thus increasing the probability of page faults and increasing the min_flt.
28. **maj_flt:** maj_flt means major page faults that have occurred for a process. A major page fault is an expensive operation that requires the CPU to fetch the missing memory page in the memory from the disk. Unusually high major page faults for a process can be an indication of malicious activity. Malware may try to load a large amount of data from a hard disk or execute malicious code that requires a lot of memory, thus increasing the major page fault for the process. Malware may process itself into small chunks in the memory while attempting to execute a large amount of code. This may lead to higher page faults new pages are loaded from the disk continuously.
29. **fs_excl_counter:** fs_excl_counter is a metric that represents the count of times the process has been granted exclusive access to a file system object. If malware attempts to manipulate or modify important system files, this metric may come in handy as it may result in the process (malware in our case) holding exclusive excess to these files. By monitoring fs_excl_counter, security analysts can detect abnormal behaviour such as processes gaining exclusive access to important system files, which can then be further investigated.

30. **lock:** The lock metric refers to the number of locks held by any process. A lock is used by a process to prevent multiple processes from accessing a shared resource simultaneously, thus avoiding race conditions. Malware may often use locks to control access to resources or to prevent other applications/processes from interfering with its operations say an antivirus. A process with a high number of locks is suspicious. If along with other suspicious activities, a process has a high lock count, it is worth investigating.
31. **utime:** utime is a metric that shows us the amount of CPU time consumed by a process. Malware that performs extensive actions, may consume a high amount of user CPU time. A process with high utime value (in jiffies in Linux kernel) which actually is running for a short duration, might come as a suspicious activity as this may be a signal for high computation being done by CPU for the process.
32. **stime:** The stime metric refers to the number of times (in jiffies) that a process has been scheduled in kernel mode. If a process has a high stime value relative to utime, this may indicate that process is spending a lot of time in the kernel mode or performing actions on a system level, performing malicious activities. If multiple processes have a high stime this may indicate a swarm of malware that has infected the entire system.
33. **gtime:** the gtime metric in the dataset refers to the cumulative time spent by all the threads in the process executing in the kernel mode. A process that is consuming a high amount of kernel resources, may have high gtime. If in case a seemingly benign process has a high gtime, it is concerning as it may be performing unauthorized activities on the kernel level. Although one cannot declare a process as malware based on gtime alone, it surely can be a factor.
34. **cgtime:** cgtime is a short form of cumulate group time or cumulative system CPU time of a process in Jiffies (1/100th of a second) spend on executing and waiting for other processes to complete. It includes both kernel and user-space execution time. A process with a sudden spike in cgtime over a short period of time could indicate a sign of malicious activity.
35. **signal_nvcsw:** the signal_nvcsw metric indicates the number of times a process was context-switched involuntarily due to a signal being received. This is a very important metric for malware detection as a process that is trying to evade detection by constantly interrupting its execution with signals. Malware may have an abnormally high number of context switches by signals.

3.1.2 Flowchart of the Major Project

First, the dataset is imported and contains information on various malicious and benign software instances. The initial data-processing is done on the data, with the most important one being mapping data entries into malware and benign. This enabled us to convert this classification metric into a numeric form where malware is given 1 and benign is given. The data is then shuffled randomly. To evaluate the performance of models on unseen data, the next step is to split the training data into training and testing sets. By using an entirely unseen set of testing data, we can get an estimate of the model's future performance on new data.

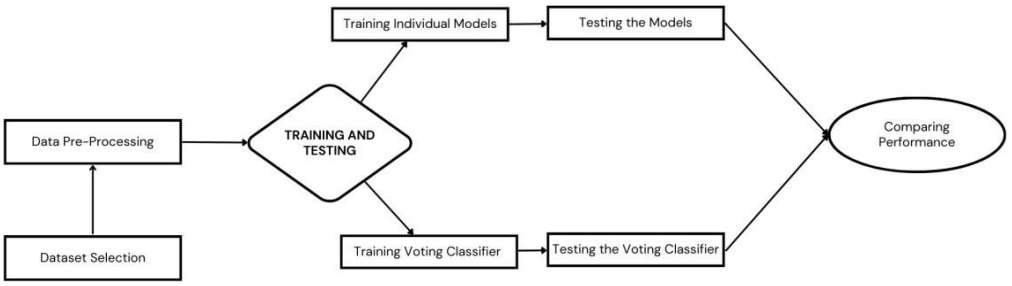


Figure 3.1: Flowchart of the project

Then four different Machine learning models, a Random Forest classifier, Support Vector Machine, and Logistic Regression along with a Linear regression model are trained on the training dataset. Their performances are evaluated using metrics such as Accuracy, Precision, F1 score, etc.

Additionally, A Voting Classifier is trained on the data. The three base classifiers selected for the voting classifiers include the Random Forest classifier, Support Vector Machine, and Logistic Regression. The training dataset is sub divided into four subsets and these base classifiers are trained on in a loop on each subset of the training data. A list of estimators is created, which contained the trained base classifiers. The voting classifier is then created which takes the estimators list as training input to combine the predictions of individual classifiers. Then the individual classifiers are trained on the same training data. The voting classifier is then tested on the testing data along with the individual classifiers and the performance of each individual classifier is compared to that of the voting classifier.

3.2 TRAINING AND TESTING OF MODELS

This section of the report discusses the models implemented in this project along with the motivation to choose these particular models and the parameters taken for each one of them. The detailed explanation for each model is mentioned below:

3.2.1 Random Forest Classifier

To learn what a random forest classifier is, we need to learn what a decision tree is. To put it in normal terms, a decision tree is a model which uses a tree-like model of decisions and decides an outcome. In our case, based on the dataset it is trained on, it decides during testing that weather if a data entry is malware or benign. A Random Forest classifier is an algorithm that has a large number of decision trees. The predictions made by the individual trees are combined to provide a final prediction. Each decision tree is trained on a random subset of the training data and a random subset of features, which improves generalization.

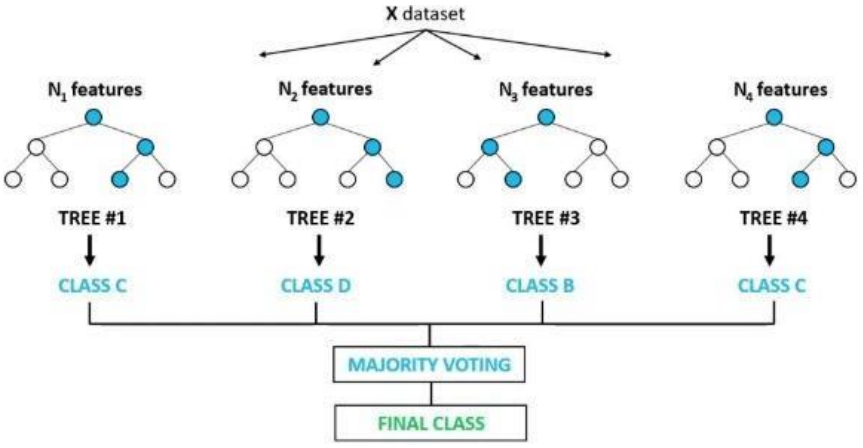


Figure 3.2: Depiction of a random forest classifier. [22]

In the context of malware analysis, a random forest classifier can be used to classify dataset entries as either malware or benign based on features present in the training data. If the model is trained on a large dataset of known malware and benign, it can learn to identify patterns and features that differentiate both classes and use this knowledge to classify the test data provided into the aforementioned two classes of malware and benign. Random forest classifiers have several advantages for malware analysis, as they are relatively fast, and scalable even on large datasets with many features.

Implementation of the Random Forest classifier begins by defining a set of parameters to try for the model including the number of parameters like n_estimators, max_depth, min_samples_split, min_samples_leaf. Each one of these parameters is important for the model to work.

To give a basic explanation, n_estimators define the number of trees in the forest, max_depth is the maximum depth of the tree, min_samples_split is the number of samples required to split an internal node, min_samples_leaf is the minimum number of samples required to at a leaf node.

To fine-tune the model, one can use GridSearchCV to find the best combination of input parameters for the Random Forest classifier by running a number of combinations of the parameters provided in the parameters dictionary. Once the best possible combination is found, it is then fed into the final random Forest classifier which outputs the best possible accuracy.

3.2.2 Support Vector Machine model

Support Vector Machine is a machine learning algorithm that is used for classification and regression tasks. To make predictions, it finds the optimal decision boundary called hyperplane that maximizes the margin between the two classes in the dataset. To keep it simple, SVM finds a line separating the two classes in the dataset, like in our case malware and benign, then when new data is to be classified, it is classified into either group based on their features.

As we just discussed, SVM can make really accurate predictions when it comes to binary classification. One of the best features of SVM is that it can work with imbalanced datasets. An example of an imbalanced dataset can be a dataset containing more benign files than malware and vice-versa. It also works efficiently while handling large and complex datasets. All these and more characteristics make SVM one of the most powerful tools for detecting new and unknown types of malware just based on their behaviours and features.

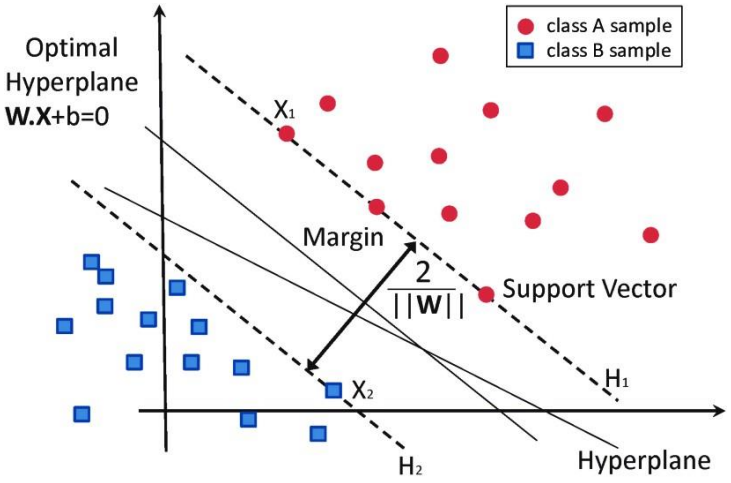


Figure 3.3: Scientific diagram of SVM. [23]

We implemented the SVM model using the Scikit-learn library in Python. The model uses the SVM algorithm with a linear kernel and a regulation parameter C set to 1.0. Roc_auc_score function from sklearn's metrics library is used to calculate the accuracy of the model.

3.2.3 Logistic Regression Model

Logistic Regression is a binary classification algorithm that predicts the probability of an input belonging to two possible categories. To explain it, Logistic Regression is like a Yes or No machine. Logistic regression in our case would classify a sample input into benign or malware. The model uses a math formula to calculate the probability. Logistic Regression works by modelling the relationship between the input variables and the binary output. It is a supervised learning algorithm and requires labelled training data to train and then estimate the model parameters.

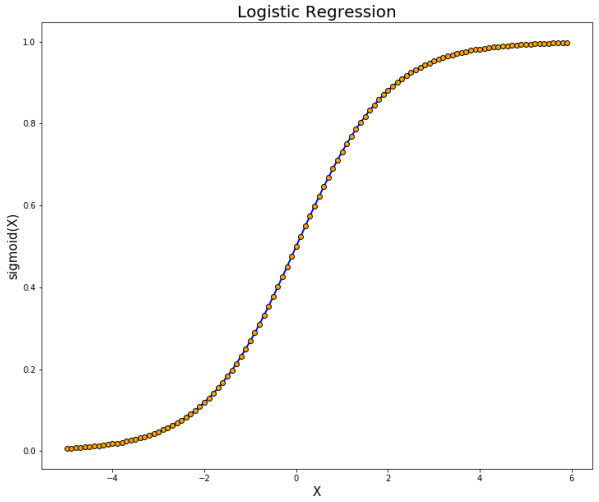


Figure 3.4: Logistic Regression Sigmoid Function Curve. [24]

Logistic Regression in the context of Malware Analysis can be used to build a model which predicts whether a given file is malicious or benign. It trains on many features such as system calls, size, type etc, and calculates the probability of the file being malicious. In the research, Logistic Regression using the SkLearn library in Python. It begins by importing the classes and then instantiating a Logistic Regression object with a regulation parameter C of 0.2. The Regularization parameter C helps us to prevent overfitting by shrinking the model

coefficients towards zero. The Model is then trained on the training data and then makes predictions on test data. The accuracies are then calculated and a ROC score is generated.

3.2.4 Linear Regression Model

Linear Regression is a regressor type model that predicts the value of a variable based on one or more independent variables. It helps us in understanding the relationship between the dependent variable and the independent variables by finding the best-fit line that explains the variation in the dependent variable. To get the basic idea behind linear regression is to find the line that minimizes the distance between the predicted values and the actual values of the dependent variable. The aforementioned line is called the regression line which is defined by the equation y=mx+b. where y is the dependent variable and x is the independent variable, m is the slope of the line and b is the y-intercept. Here, to find the best values for m and b, the regressor uses an Ordinary least squares (OLS) method.

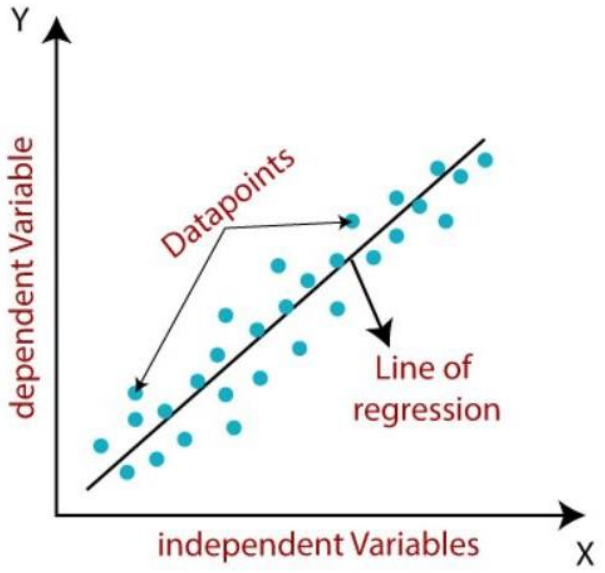


Figure 3.5: Best fit line as devised under linear regression

In the context of Malware analysis, a Linear Regression model can be used to analyse the relationship between different variables and to predict whether a file is malware or benign.

To implement linear regression, we use sklearn's Python library. The necessary classes are imported and then a Linear Regression object is created which is then fitted with the training data and is then fed test data to make predictions. The accuracy is calculated using the R-squared value, which is a statistical measure that represents the proportion of variance in the dependent variable that is explained by independent variables.

3.2.5 Ensemble Learning

Ensemble learning is a machine learning technique that combines the prediction of multiple models to improve the overall performance of the system. Ensemble learning is mainly of two types: Bagging and Boosting.

Bagging: Bagging is a method of creating multiple models, each of which is trained on a random subset of the training data. The output of the base classifiers are averaged to find final prediction.

Boosting: In boosting multiple ML models are created and each of them are trained on a modified version of the training data. Boosting reduces the bias of the model.

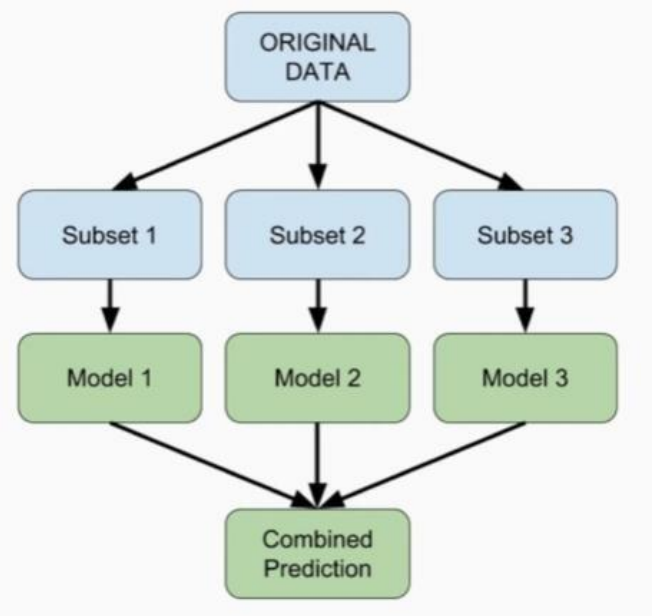


Figure 3.6: Ensemble learning voting model depiction

Since malware is a constantly evolving threat, it can be difficult to build a model that is accurate over a long period of time. By combining the output of multiple models, ensemble learning can help us create a more robust system of prediction. Each base classifier used in Ensemble learning has its own strength and weaknesses, but combining the outputs of all the models helps us in creating a more accurate and reliable system.

To implement ensemble learning, we decided to use three different base classifiers namely Random Forest, K-Nearest Neighbor, and Logistic Regression. We decided to split the data into training and testing subsets and then the classifiers were created and trained on different subsets of the training data. We created subsets to improve the accuracy of the predictions and reduce overfitting.

A voting Classifier is created which combines the prediction of all three classifiers. This is done using VotingClassifier() which takes in the list of tuples where each tuple contains a unique name and a classifier object. An estimator list is created by looping through each subset of training data and creating a new classifier object for each subset.

When the voting classifier is trained, the individual classifiers are evaluated on the test set along with the voting classifier to see how well they performed, and their accuracy is compared.

Chapter 04: RESULTS

4.1 DISCUSSION

On the Internet, various forms of malware are distributed. Every computer connected to the Internet is vulnerable to Malware infection. To fight this vulnerability, it is crucial to practice information security practices. And with the growing density of the cyber world, it is important to automate the process of Malware detection. For this, machine learning is very crucial to be optimized for the task.

In the project, we used different machine learning algorithms whose results varied in a range from 54% accuracy to 95% for different models.

4.2 EVALUATION

Evaluation is one of the crucial steps in machine learning projects that involves assessing the performance of a trained model. The purpose of this step is to see how well a model is making predictions on new data.

The evaluation process in our project is done by splitting the available data into training and testing data. The models are initially trained using the training data, and their performance is then assessed using the testing data. Evaluation is a key stage since it gives us important knowledge about the weaknesses, biases, and overfitting that the models may be experiencing. Utilising criteria like accuracy, precision, recall, and F1-score, we assessed the performance.

The evaluation procedure makes use of a variety of metrics which are:

Accuracy: Accuracy is determined by dividing the proportion of cases that were properly predicted by the overall number of instances. The accuracy calculation formula is

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

Precision: We divide the number of real positive values by the total number of expected positive values when calculating precision. The precision calculation formula is

$$Precision = \frac{(True\ Positive)}{(True\ Positive + False\ Positive)}$$

Recall: The percentage of genuine positives and the overall number of actual positives are found when computing the recall score. The Recall formula is

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

F1 Score: This is the harmonic mean of precision and recall, and it provides a balanced measure of both parameters. The formula for calculating F1 Score is

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Area Under the Receiver Operating Characteristics (ROC) Curve (AUC) Score: This tells us how effective the model is in predicting the values. The higher the ROC AUC, the better the model's performance at classifying between the positive and negative classes.

Confusion matrix: A matrix that visualize the number of true positives, false positives, true negatives, and false negatives. These parameters of the confusion matrix are used to evaluate other metrics such as precision, recall and f1 score etc. A confusion matrix for our project may look like this

Table 4.1: Confusion matrix illustration**.**

True Positive (TP): Actual: Malware ML model predicted: Malware Values:	**False Positive (FP):** Actual: Benign ML model predicted: Malware Values:
False Negative (FN): Actual: Malware ML model predicted: Benign Values:	**True Negative (TN):** Actual: Benign ML model predicted: Benign Values:

Mean Squared Error (MSE): Measures the average of the squared differences between predicted and actual values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Mean Absolute Error (MAE): measures the average of the absolute differences between predicted and actual values.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Root Mean Squared Error (RMSE): This is the square root of MSE and gives a measure of how close the predicted values are to the actual values. Smaller the value the better.

$$RMSD = \sqrt{\frac{\sum_{i=1}^{N} (X_i - \hat{x_i})^2}{N}}$$

4.2.1 Visualized Results

Table 4.2: Evaluation Metrics of Models.

Metrics	**Random Forest (%)**	**SVM (%)**	**Logistic Regression (%)**
Accuracy	94.55	94.62	93.75
Precision	99.33	94.00	92.98
Recall	89.82	95.46	94.79
F1 Score	94.34	94.62	93.74
ROC-AUC Score	94.60	94.62	93.74
TP Rate	89.82	95.46	94.79
FP Rate	0.62	6.23	7.31
FN Rate	10.10	4.54	5.21

Table 4.1 depicts the evaluation metrics (i.e., Accuracy, Precision, Recall, F1 score, ROC Score, TP rate, FP rate, FN rate) of the models that are used in the analysis. Table 4.2 shows the evaluation scores of the linear regression model i.e., MSE, MAE, RMSE.

Table 4.3: Evaluation Metrics of Linear Regression Model.

Evaluation Metrics	**Linear Regression Model**
Accuracy achieved	55.276%
Mean Squared Error (MSE)	11.32%
Mean Absolute Error (MAE)	28.87%
Root Mean Square Error (RMSE)	33.64%

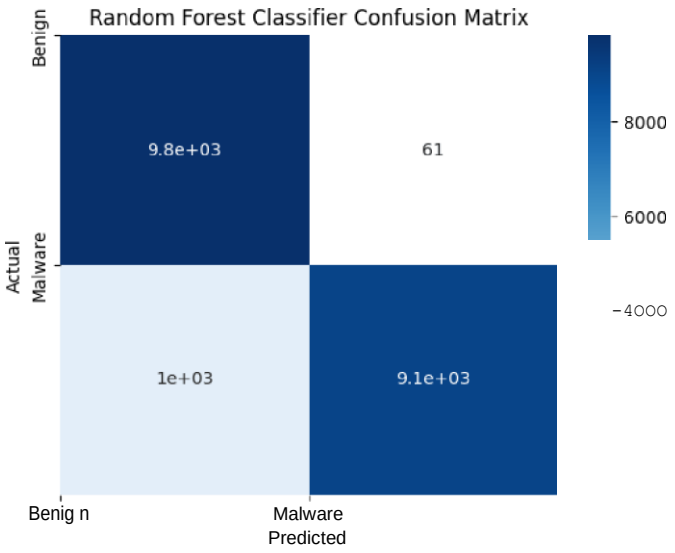


Figure 4.1: Confusion matrix of random forest**.**

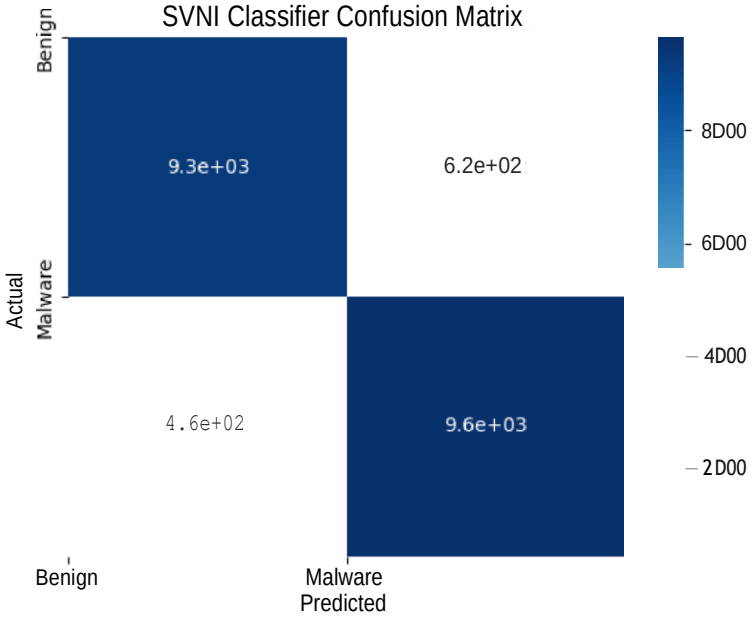


Figure 4.2: Confusion matrix of SVM.

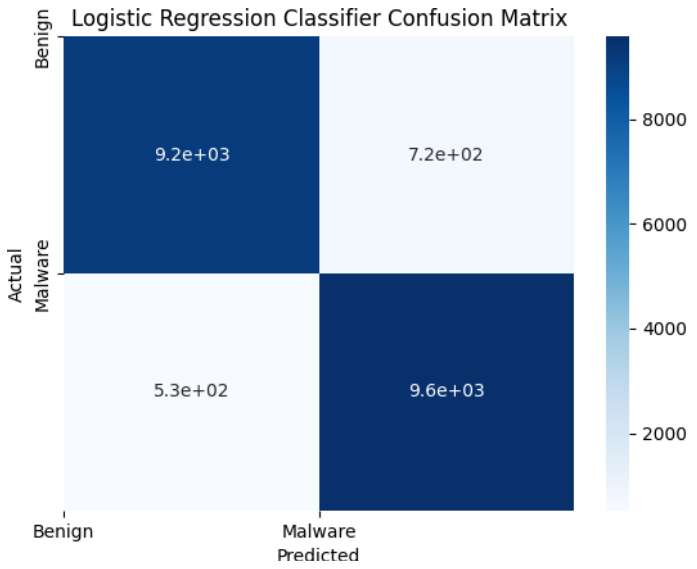


Figure 4.3: Confusion matrix of logistic regression.

Fig. 4.1, 4.2, and 4.3 are the visual representation of the confusion matrices for the classifiers. Fig. 4.1 depicts the confusion matrix for the random forest classifier, followed by the Confusion matrix of SVM in Fig 4.2, and finally the confusion matrix of Logistic regression in Fig 4.3

4.2.2 Accuracy of the Base Classifiers

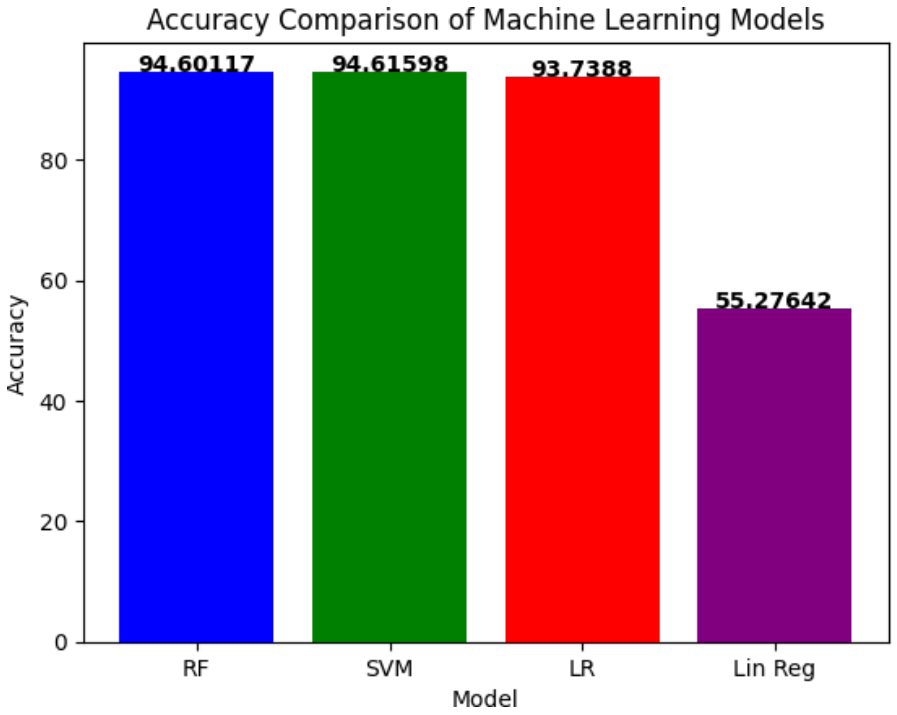


Figure 4.4: Accuracies of the Models

Fig. 4.4 is the visual representation of the prediction accuracies for the classifiers Random Forest, SVM, Logistic Regression and Linear Regression. The figure clearly shows that among the base classifiers, SVM shows the highest prediction accuracy of 94.61% followed by Random Forest at 94.60% and Logistic Regression and Linear Regression at 93.73% and 55.27%. One of the reasons for the lower accuracy of Linear Regression is that, unlike classifiers which are excellent in binary classification tasks, Linear Regression is incapable of outputting binary outputs. It is excellently suited for cases where output is a continuous numerical value for example predicting the prices of houses, or revenue prediction etc.

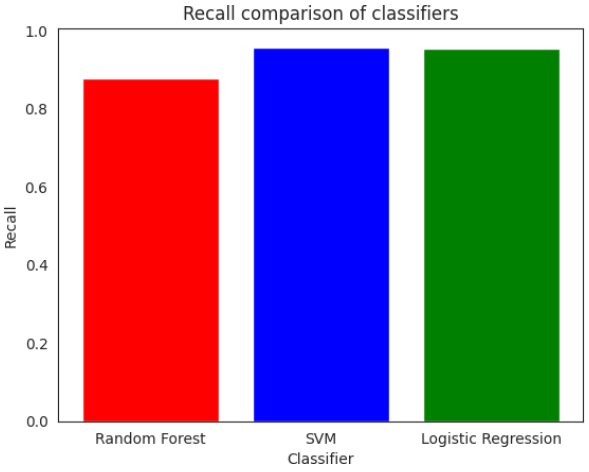


Figure 4.5: Recall Score comparison

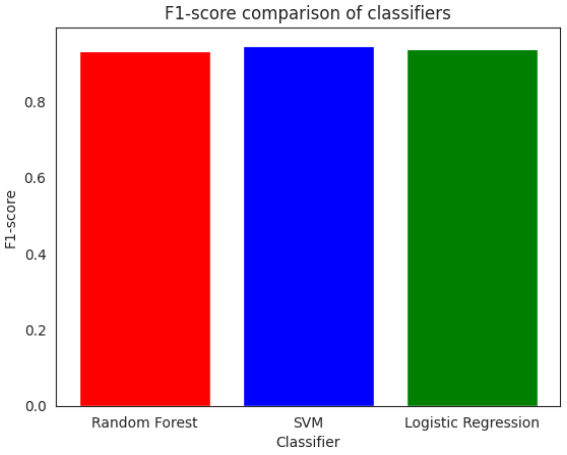


Figure 4.6: F1 Score comparison

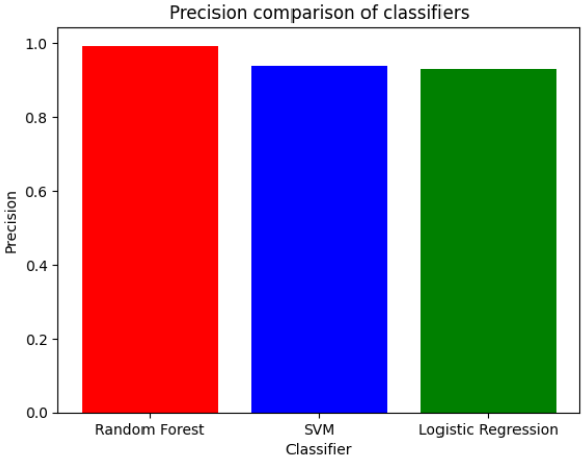


Figure 4.7: Precision Score comparison

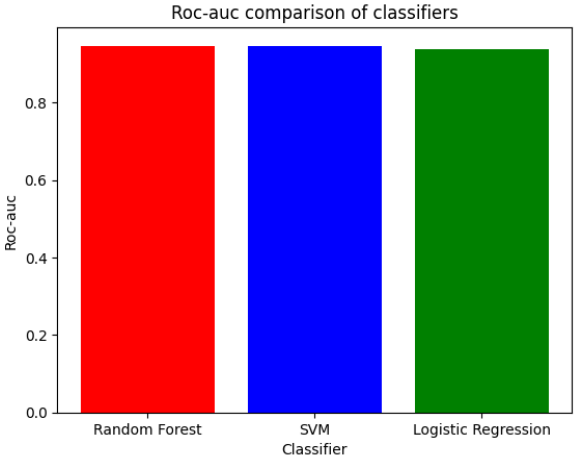
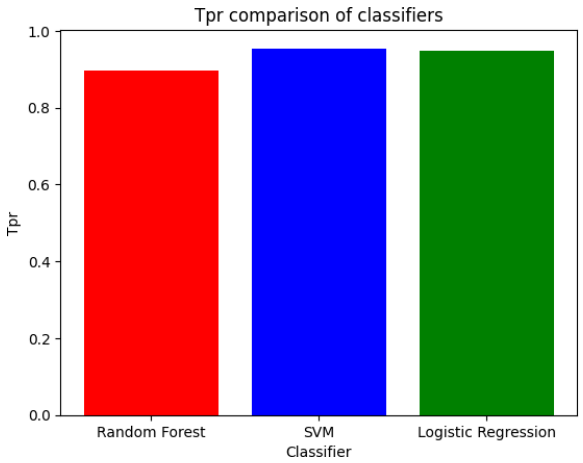


Figure 4.8: ROC-AUC Score comparison

Fig. 4.7-4.11 are the evaluation metrics of classifiers visualized. The red bar represents the scores of Random Forest, the blue one represents the scores of SVM and the Green one represents the score of Logistic Regression, Figure 4.5 are the Recall score of the classifiers with SVM having a Recall score of 95.46%, followed by Logistic Regression at 94.79%. and random forest at 89.82%. Figure 4.6 represents the F1 score of the classifiers where SVM has the highest score of 94.62% followed by Random Forest at 94.34% and then Logistic regression at 93.74%. Figure 4.7 represents the Precision score visualization for the classifiers where the

classifier with the highest precision score is Random Forest at 99.33% followed by SVM at 94% and logistic regression at 92.98%. Figure 4.8 shows the ROC score comparison for the classifiers, where SVM has the highest score of 94.62%, followed by Random Forest at 94.34% and Logistic Regression at 93.74%.



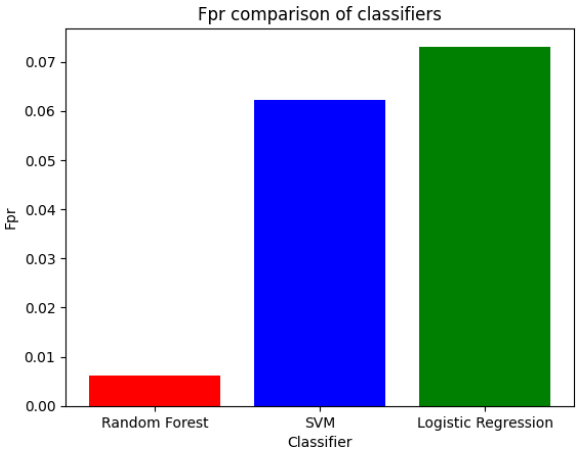


Figure 4.9: TPR comparison of Classifiers

Figure 4.10: FPR comparison of Classifiers

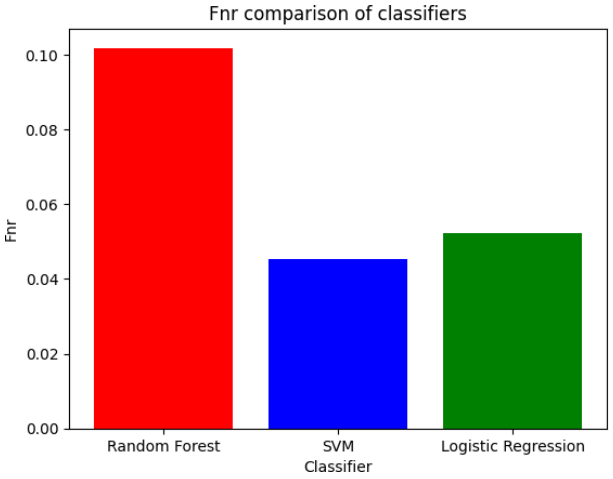


Figure 4.11: FNR comparison of the Classifiers

Fig 4.9 is the representation of TPR of the classifiers where SVM has the highest true positive rate of 95.46% followed by Logistic regression at 94.79% and Random Forest at 89.82%. Figure

4.10 represents the False positive rate for the classifiers where Random Forest excels at 0.62% followed by SVM at 6.23% and logistic regression at 7.31%. Figure 4.11 is the visual representation of the False negative rate of the classifiers, where FNR of Random Forest, SVM and logistic regression is 10.10%, 4.52% and 5.21% respectively.

4.2.3 Accuracy of the Voting Classifier

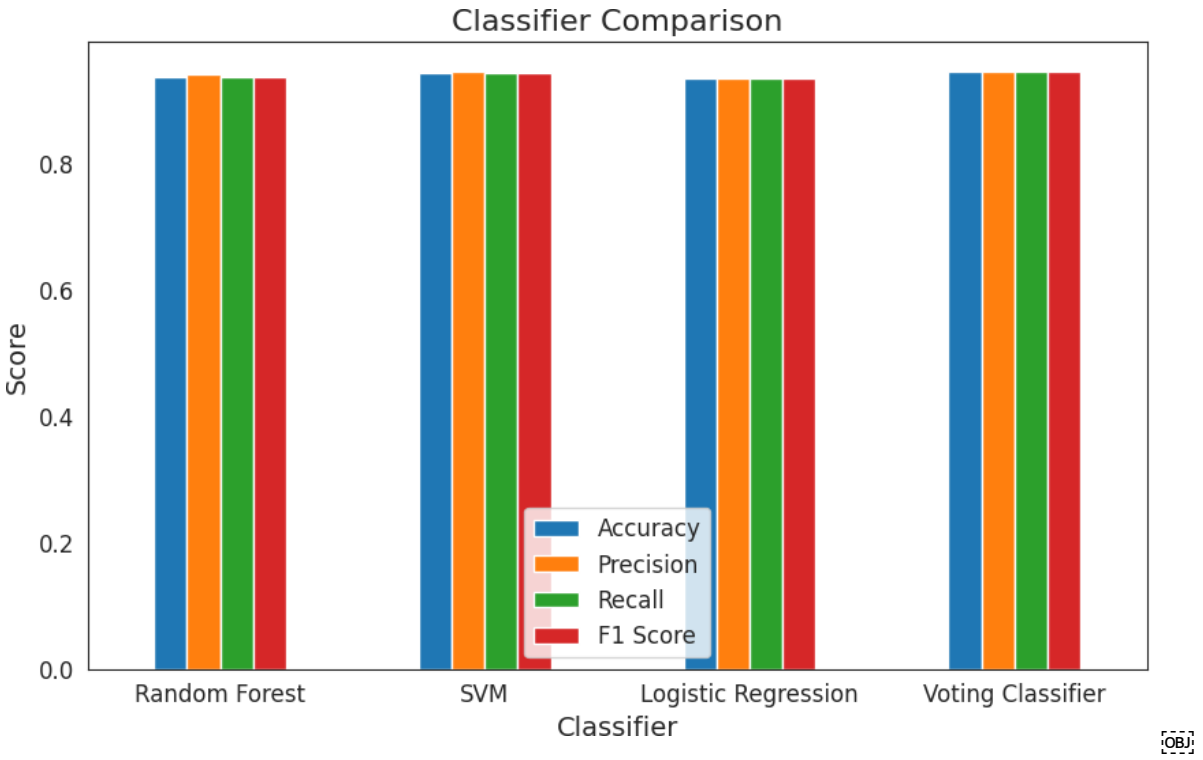


Figure 4.13: Accuracies of Ensemble learning model and its Base Classifiers

Table 4.3 shows the evaluation metrics of the Ensemble learning classifier when compared to the base classifier. It is clearly shown that the Voting classifier has higher scores when compared to the base classifiers. Figure 4.13 is the visual representation of the evaluation metrics of all three base classifiers compared to the voting classifier. We can clearly see that ensemble learning increases the performance when compared to the base classifiers, but the score is slightly better than the base classifiers. One of the reasons can be that the base classifiers are already very diverse and highly accurate, or the high degree of randomness in the data.

Table 4.3: Accuracies of Ensemble learning model and its base classifiers

Classifiers	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**
Random Forest	94.01	94.69	94.27	94.20
SVM	94.62	94.64	94.62	94.62
Logistic Regression	93.75	93.77	93.74	93.75
Voting Classifier	**95.74**	**94.85**	**94.93**	**94.74**

Chapter 05: CONCLUSION

5.1 CONCLUSION

In conclusion, this project provided an opportunity to get hands-on experience in malware detection and machine learning. The project was made possible by the use of publicly accessible datasets using well-known classifiers. The dataset used in the project helped in training four different classifiers and a voting classifier to achieve the maximum possible prediction accuracy.

Overall, the results demonstrate the effectiveness of machine learning when it comes to malware analysis and detection by behavioural analysis. This project provides a foundation for future research and development of more sophisticated models for malware detection and analysis in Android systems.

5.2 APPLICATION OF MAJOR PROJECT

There are several practical applications that can be used, some of these are as mentioned below:

- Malware Detector: Increasing the scope of the project, a malware detector can be designed to work on real-time data.
- Anti-viruses: Anti-virus software is quite popular among computer enthusiasts as they can work on a variety of domains, including the internet, local, etc. An Anti-virus is an even more powerful version of a Malware Detector which can perform multiple tasks.
- Reverse engineering Malware: Using analysers like these can be used to study malware's characteristics and also learn how malware masquerades within the tasks.

5.3 LIMITATIONS

- The biggest limitation of the project was a lack of diverse and representative datasets. The accuracy and reliability of malware prediction depend largely on the quality and

diversity of the dataset used for training the models. In our case, the dataset caused overfitting because of the low-quality dataset.

- Overfitting is another limitation where a model learns to fit the training data too well and becomes overly sensitive to noise and outliers.
- Evolving Malware: Malware is constantly evolving and adapting to new security measures. Newer malware can render the current prediction models obsolete.

5.4 FUTURE WORK

- This project has provided us with valuable information in the domain of Cyber security. It opens the doors for research in the aforementioned field as a domain. This small project helped in building the fundamentals of Machine learning and malware analysis.
- Malware analysis can be used to improve the effectiveness and accuracy of anti-viruses and anti-malware software.
- This project can lay grounds for malware research, as one of the critical steps involved in malware analysis is threat analysis through static and dynamic analysis. Malware research can help in strengthening our countermeasures for cyber threats.

REFERENCES

[1] S Mohurle, M Patil, "A brief study of Wannacry Threat: Ransomware Attack 2017," International Journal of Advanced Research in Computer Science (Volume 8, No. 5, May-June 2017).

[2] Peter Knight, “ILOVEYOU: Viruses, Paranoia, and the Environment of Risk,” Sage Journals(Volume 48, Issue 2_suppl, 2000).

[3] Hani AlOmari, Qussai M. Yaseen, Mohammed Azmi Al-Betar, A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection, Procedia Computer Science, Volume 220, 2023, P. 763-768, ISSN 1877-0509.

[4] Zhang B, Yin J, Hao J, Zhang D, Wang S. “Malicious codes detection based on ensemble learning.” In: Proceedings of the international conference on autonomic and trusted computing. Berlin, Heidelberg: Springer; 2007 Jul 11. p. 468–77.

[5] Menahem E, Shabtai A, Rokach L, Elovici Y. “Improving malware detection by applying multi-inducer ensemble.” Comput Stat Data Anal 2009 Feb 15;53(4):1483–94.

[6] Mukkamalaa S, Sunga AH, Abrahamb A. “Intrusion detection using an ensemble of intelligent paradigms”. J Netw Comput Appl 2005;28:167-82.

[7] Jyoti Landage, MP Wankhade. “Malware detection with different voting schemes.” Compusoft 2014;3(1):450–6.

[8] Ye Y, Li T, Jiang Q, Han Z, Wan L. “Intelligent file scoring system for malware detection from the gray list.” In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining; 2009 Jun 28. p. 1385–94. ACM.

[9] Guo S, Yuan Q, Lin F, Wang F, Ban T. “A malware detection algorithm based on multi-view fusion.” In: Proceedings of the international conference on neural information processing. Berlin, Heidelberg: Springer; 2010 Nov 22. p. 259–66.

[10] Ozdemir M, Sogukpinar I. “An android malware detection architecture based on ensemble learning.” Trans Mach Learn Artif Intell 2014;2(3):90–106.

[11] Sheen S, Anitha R, Natarajan V. “Android based malware detection using a multi feature collaborative decision fusion approach.” Neurocomputing 2015 Mar 5;151:905–12.

[12] Sheen S, Anitha R, Sirisha P. “Malware detection by pruning of parallel ensembles using harmony search.” Pattern Recognit Lett 2013 Oct 15;34(14):1679–86.

[13] Yerima SY, Sezer S. Droidfusion: a novel multilevel classifier fusion approach for android malware detection. IEEE Trans Cybern 2018 Jan 3;99:1–4.

[14] Kuncheva LI. Diversity in multiple classifier systems (Editorial). Inf Fusion 2005;6(1) 2005.

[15] Krawczyk B., Woźniak M. Evolutionary cost-sensitive ensemble for malware detection. In Proceedings of the international joint conference SOCO’14-CISIS’14-ICEUTE’14 2014 (pp. 433–442). Springer, Cham.

[16] Deepak Gupta and Rinkle Rani, ‘‘Improving malware detection using big data and ensemble learning,’’ Computers & Electrical Engineering, 86:106729, 2020.

[17] M. Asha Jerlin & K. Marimuthu, “Malware Detection System Using Machine Learning Techniques for API Call Sequences”, Applied Security Research.

[18] Omar N. Elayan & Ahmad M. Mustafa, “Android Malware Detection Using Deep Learning”, International Workshop on Data-Driven Security (DDSW 2021) March 2021, Warsaw, Poland.

[19] Sashie Dilhara, ”Malware Classification using Machine Learning and Deep Learning”, International Journal of Computer Applications.

[20] Burak Ergenc, “Malware detection,” 2016. [Online]. Available: https://github.com/mburakergenc/Malware-Detection-using-Machine-Learning/blob/master/data.csv.

[21] Avast, “What is Malware & How Does it Work? | Malware Definition | Avast,” 2019 https://academy.avast.com/hs-fs/hubfs/New_Avast_Academy/what_is_malware_academy_refresh/Types-of-Malware-EN.png?width=1684&height=1200&name=Types-of-Malware-EN.png

[22] Davis D 2020, Random Forest Classifier, digital image, "Random Forest Classifier Tutorial: How to Use Tree-Based Algorithms for Machine Learning"

https://www.freecodecamp.org/news/content/images/2020/08/how-random-forest-classifier-work.PNG

[23] Hard-Rock Stability Analysis for Span Design in Entry-Type Excavations with Learning Classifiers - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Classification-of-data-by-support-vector-machine-SVM_fig8_304611323

[24] James T 2020, Logistic Regression Sigmoid Curve, digital image, "Logistic Regression Explained" https://miro.medium.com/v2/resize:fit:720/format:webp/1*QY3CSyA4BzAU6sEPFwp9ZQ.png

GG02

ORIGINALITY REPORT

11%	6%	6%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Deepak Gupta, Rinkle Rani. "Improving malware detection using big data and ensemble learning", Computers & Electrical Engineering, 2020 Publication	4%
2	Submitted to Middle East College of Information Technology Student Paper	1%
3	www.mdpi.com Internet Source	1%
4	Hani AlOmari, Qussai M. Yaseen, Mohammed Azmi Al-Betar. "A Comparative Analysis of Machine Learning Algorithms for Android Malware Detection", Procedia Computer Science, 2023 Publication	1%
5	pubs.rsc.org Internet Source	<1%
6	www.researchgate.net Internet Source	<1%

7	www.dtic.mil Internet Source	<1%
8	arrow.tudublin.ie Internet Source	<1%
9	Submitted to Middlesex University Student Paper	<1%
10	Submitted to Jaypee University of Information Technology Student Paper	<1%
11	dokumen.pub Internet Source	<1%
12	essay.utwente.nl Internet Source	<1%
13	www.freedomtoascend.com Internet Source	<1%
14	Submitted to Kuwait University Student Paper	<1%
15	Submitted to University of Greenwich Student Paper	<1%
16	pub.towardsai.net Internet Source	<1%
17	Submitted to Bournemouth University Student Paper	<1%

18	Submitted to American Public University System Student Paper	<1%
19	soe.rutgers.edu Internet Source	<1%
20	www.ijraset.com Internet Source	<1%
21	Submitted to Loughborough University Student Paper	<1%
22	www.irjmets.com Internet Source	<1%
23	Jinrong Bai, Junfeng Wang. "Improving malware detection using multi-view ensemble learning", Security and Communication Networks, 2016 Publication	<1%
24	Submitted to University of Salford Student Paper	<1%
25	Jean Petric, David Bowes, Tracy Hall, Bruce Christianson, Nathan Baddoo. "Building an Ensemble for Software Defect Prediction Based on Diversity Selection", Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '16, 2016 Publication	<1%

26	github.com Internet Source	<1%
27	www.cse.uoi.gr Internet Source	<1%
28	Submitted to Nanyang Technological University Student Paper	<1%
29	Submitted to University of Bradford Student Paper	<1%
30	Submitted to University of Carthage Student Paper	<1%
31	Submitted to University of Edinburgh Student Paper	<1%
32	Submitted to WorldQuant University Student Paper	<1%
33	digitalcommons.njit.edu Internet Source	<1%
34	threatpost.com Internet Source	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		