Android Malware Detection Technique for IoT Devices

Using Machine Learning

Ephrem Elias Tsuka

A Thesis Submitted to the Department of Computer Science and Engineering School of Electrical Engineering and Computing

Presented in Partial Fulfilment of the Requirement for the Degree of Master's in Computer Science and Engineering

Office of Graduate Studies
Adama Science and Technology University

February, 2023
Adama, Ethiopia

Android Malware Detection Technique for IoT Devices Using Machine Learning

Ephrem Elias Tsuka

Advisor: Dr. Biruk Yirga

A Thesis Submitted to the Department of Computer Science and Engineering School of Electrical Engineering and Computing

Office of Graduate Studies
Adama Science and Technology University

February, 2023
Adama, Ethiopia

DECLARATION

I declare that this master thesis entitled “Android Malware Detection Technique for IoT Devices using Machine Learning” is my own work and has not been submitted to any university for similar purposes. The references used in this proposal are duly recognized by proper citations.

_____	_____	_____
Name of student	Signature	Date

RECOMMENDATION

I, the major advisor/supervisor of this thesis, hereby certify that I have closely advised/supervised the student while developing this master thesis entitled "Android Malware Detection Technique for IoT Devices using Machine Learning" prepared by Ephrem Elias. Therefore, I recommend the submission of the thesis to the department for further review and evaluation.

_____ _____ _____

Major Advisor/Supervisor Signature Date

APPROVAL SHEET

We, the undersigned, members of the Board of Reviewers of the thesis open defense by Ephrem Elias have read and evaluated the thesis entitled "Android Malware Detection Technique for IoT Devices using Machine Learning" and assessed the understanding of the candidate about the research. This is, therefore, to certify that the thesis is accepted and we recommend the implementation of the thesis.

_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Finally, approval and acceptance of the thesis is contingent upon the submission of its final copy to the Office of Postgraduate Studies (OPGS) through the Department Graduate Council (DGC) and

_____	_____	_____
Department Head	Signature	Date
_____	_____	_____
School Dean	Signature	Date
_____	_____	_____
Office of Postgraduate Studies, Dean	Signature	Date

ACKNOWLEDGMENT

First and foremost, I want to express my gratitude to the Almighty God for everything that has occurred, for all of his favors, and for helping me overcome all the obstacles and difficult times that I was unable to overcome on my own.

My advisor, Dr. Biruk Yirgu, whose sincere advice, helpful criticism, and dedication during the entirety of this study came to fruition, deserves the deepest gratitude from me. As my advisor, instructor, and postgraduate coordinator, he was the one who helped me every step of the way and maintained turning on the light over all of my challenges in a timely manner. I'm really grateful to him for giving me such valuable assistance.

Next, I want to sincerely thank my family and friend Andinet Asefa, a BHU instructor. I will never forget your aid, and God will always keep an eye out for you. I'm hoping you'll follow my every action.

Finally, I would like to express my heartfelt gratitude and best wishes to ASTU staffs and classmates for their prompt assistance and unwavering support in helping me finish my thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ALGORITHMS

LIST OF ACRONYMS

API	Application Programming Interface
APK	Android Package
AMD	Android Malware Dataset
ASTU	Adama Science and Technology University
COVID-19	Corona Virus Disease 2019
DLT	Distributed Ledger Technology
ETC	Ethiopian Telecommunication Corporation
Et al.	Other Authors
GPU	Graphics Processing Unit
ICT	Information Communication Technology
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineering
INSA	Information Network Security
ISP	Internet Service Provider
IOT	Internet of Things
KNN	K-Nearest Neighbor
ML	Machine Learning
MS	Micro-Soft
RAM	Random Access Memory
RNN	Recurrent Neural Network

OS	Operating System
SMS	Short Message Services
SVM	Support Vector Machine
LLE	Locally Linear Embedding
PCA	Principle Components Analysis
LDA	Linear Discriminant Analysis
AUC	Area Under Curve
ROC	Receiver Operating Characteristics
PUA	Potentially Unwanted Application

ABSTRACT

Internet of things (IoT) based Android devices is revolutionizing our world with its problem-solving applications in different aspects of life such as education, agriculture, healthcare, sensing and remote monitoring, and so on. Android applications are working hand to hand nearly to make IoT dreams real. The number and scope of Android devices keep on increasing. It is assessed that there will be approximately 6.2 billion smartphone users by 2020/2021. Therefore, malicious program can affect the working of many devices shared in a network. The recent studies focus on malware detection for Android based devices but many researchers use various techniques and limited malware families for analysis. The goal of this study is to improve the Machine Learning algorithm performance through training dataset with features that has high importance score. The machine learning classification algorithm is implemented to extract the various features of Android malware using enhanced Decision tree classifier and implemented in three ML algorithms. The machine learning model is trained with datasets from three popular data sources, those are google play store, virusshare and Kaggle data. Moreover, Decision Tree, Naïve Bayes and SVM algorithms are compared to find better detection rate and hybrid analysis method is used for extracting more important features to provide classification for achieving high accuracy and robustness. The algorithms are implemented on the selected features with different running time and different numbers of apk and features, and the SVM classifier performed with better accuracy than others that is up to 97%. While the Decision tree and Naïve Bayes classifier performs less accurate. Therefore, according to this study the SVM algorithm is the better to Android malware detection.

Keywords: *Algorithms, Android Application, Malware detection, Internet of Things, Machine Learning, Analysis, APK, ML Classifiers*

CHAPTER ONE

1. INTRODUCTION

1.1. Background of the Study

The mobile device has become a recently made growing trend in this digital age as many market holders have migrated their applications, products, and accessibility to this platform for improved productivity and interoperability. Android Operating System (OS) platform has become the fastest growing mobile operating system and have been an attraction to several illegitimate operations because of its open-source and popularity. By the 2021 Mobile Malware Evolution Report (Kaspersky Lab, 2021), "Kaspersky lab: 14,465,672 malware, adware and riskware attacks were prevented, 886,105 malicious installation packages were detected, of which: 24,604 packages were mobile banking Trojans; 3,623 packages were mobile ransomware Trojans, this shows high increasing rate of malware the from year to year". Although Google has created the Bouncer tool to perform malice detection on applications of Google Play, attackers could still bypass Bouncer to drive malware into Google Play (K. Shibija and R. V. Joseph, 2018).

Recently, the scope of the internet of things (IoT) is expanding with the introduction of a variety of applications such as healthcare, smart agriculture, smart home, smart shopping, etc. (R.Kumar et al. 2019) Most of the devices in a shared IoT network are based on the Android platform due to flexibility, robustness and hardware support, which is pivotal for sensors interface. Different Android devices provide several distinguished services related to monitoring, sensing and controlling tasks. Consequently, in our inexorably associated society, the number and scope of Android devices keep on increasing. It is assessed that there will be roughly 6.1 billion smartphone clients by 2020 (Kaspersky Lab, 2021). Therefore, malicious activity can affect the working of many devices connected in a network. The recent studies focus on malware detection for Android based devices.

In terms of strengths, Ethiopia is eager to absorb new technology, and the government is making aggressive plans. The Ethiopian Telecommunications Corporation (ETC) in charge of ICT industry is providing incentives to push the mobile industry. As a result, there is significant growth in both government and non-government entrepreneurship for the development of mobile applications. Specifically, there is strong development in the capital of Addis Ababa as well as the nearby towns Bishoftu and Adama. Promoting e-commerce will have a positive impact on the mobile industry. A notable lack of development resources is

one of the areas of vulnerability. Too many restrictions apply to customs. The issue of hackers and illegal goods is not addressed. According to the DataReportal Ethiopia 2022: only approximately, 25% of Ethiopia's population has access to the Internet, At the beginning of 2022, there were 58.54 million cellular mobile connections in Ethiopia, according to data from GSMA Intelligence. According to data from GSMA Intelligence, Ethiopia's mobile connections in January 2022 were equal to 49.1% of the country's total population, and only 5.3% (2022) of people utilize social media. (Simon Kemp, 2022) Due to the widespread use of smartphones, security issues with cellphones are starting to resemble those with personal computers. Malware on Android mobile can make the phone partially or entirely unusable; cause unknown billing; steal private information; send unnecessary SMS messages, or infect the contact of the users in the phone-book. Therefore, we need to continue to delve into the research of malicious code or malware detecting technology in Android-based devices.

The proposed framework detects malware using different features information to reflect various characteristics of applications in numerous aspects. The proposed methodology first distinguishing the malware and benign, and refines them using enhanced classifying methods. The classifying method calculates the feature importance scores of each feature set and iterative reduced the unnecessary features that can effectively distinguish malware and benign applications even though malware have many properties similar to benign applications. Moreover, the proposed classifying method handles multidimensional data by reducing the features using feature importance score learning procedure. This property enables the proposed method to fit for applying on many types of classifying problems.

The network of physical items, or "things," that are implanted with sensors, software, and other technologies for the purpose of communicating and exchanging data with other devices and systems through the internet is referred to as the Internet of Things (IoT). These gadgets include anything from common domestic items to high-tech industrial gear. Today, there are more than 7 billion connected IoT devices, and according to analysts, there will be 10 billion by 2020 and 22 billion by 2025. Although the concept of IoT has been around for a while, it has only just become a reality thanks to a variety of recent technological advancements.

- Access to low-cost, low-power sensor technology :- For more manufacturers Affordable and reliable sensors are making IoT technology possible.
- Connectivity :- It is made easy to connect sensors to the cloud and to other “things” for efficient data transfer through a network protocol.

- Cloud computing platforms :- Businesses and consumers may now get the infrastructure they need to scale up without having to manage it all thanks to the expansion of cloud platforms.
- Machine learning and analytics :- Businesses may acquire insights more quickly and easily thanks to improvements in machine learning and analytics, as well as access to diverse and enormous volumes of data stored in the cloud. The development of these complementary technologies pushes the limits of IoT, and the data generated by IoT feeds these complementary technologies.
- Conversational artificial intelligence (AI) :- Natural language processing (NLP) is now available on Internet of Things (IoT) devices, including digital personal assistants like Alexa, Cortana, and Siri. This has made IoT devices more appealing, practical, and inexpensive for usage at home.

1.2. Motivation of the Study

In this technological era, smartphone usage and its associated applications are rapidly increasing due to the convenience and efficiency in various applications and the growing improvement in the hardware and software on smart devices. By 2023, there are expected to be 4.3 billion smartphone users worldwide. The most popular smartphone operating system is Android (OS). Its market share was 71.86% as of July 2022. Apple iOS holds the second-highest market share with a share of 27.49%, while Samsung, KaiOS, and other small suppliers split the remaining 0.65% (Source : *Statcounter*). The prominence of the Android platform makes it a prime target for malware attacks. The statistics of Statista showed that the total number of Android malware instances was 26.61 million on March 2018 and that 482,579 new Android malware samples were captured per month as of March 2020. In 2020, more than 1.3 million new malware samples will be found each month, according to G Data security experts. Additionally, they stated that fresh malware was published every 1.3 seconds on average. The abundance of Android malware incidents has overwhelmed analysts who study it. We must categorize malware cases into malware families using a more precise classifier in order to analyze malware instances effectively and quickly.

Due to the complexity of their computations and their limited resources (such as cost, memory, and processing power), the majority of currently used malware detection techniques cannot be used directly on Internet of Things (IoT) devices (H. Chen *et al.,* 2018). Several techniques are developed for selecting the

features of malware, for instance information gain technique, but they are based on limited types of feature to detect the malware. To solve the problem, machine learning are posing a momentum around the world towards their use for intelligent model training. Machine learning automatically extracts the malware information using classifying and classification techniques.

Figure 1.1 Mobile OS market share Worldwide

According to secure list (by Kaspersky), in the first quarter (May 27, 2021):

we detected 1,451,660 mobile installation packages, of which:

- 25,314 packages were related to mobile banking Trojans.
- 3,596 packages were mobile ransomware Trojans.

Moreover, A new report forecasts that the size of digitally transacted economic activities in Ethiopia will grow to 3 trillion Birr by 2025 representing 39 percent of the country's GDP. According to this information the malware detection is the issue that should be focused on.

1.3. Statement of the Problem

Malware attacks are expanding due to the increased use of mobile devices, notably on Android phones, which hold 71.86% of the market. Hackers use a variety of techniques to attack smartphones, including credential theft, spying, and deceptive advertising. The ability to derive a classifier from a set of training instances means that machine learning (ML)-based methods, among other countermeasures, can effectively identify these attacks (Omar N. and M. Mustafa, 2021). This eliminates the need for explicit definitions of the signatures when creating malware detectors. The hacker exploits android application features to break the security and privacy of devices, which poses a serious threat towards leakage of data from the specific personal data to very sensitive Government departments such as military data, bank database, education

institutions and etc. There are many studies proposed for Android malware detection were based on the limited types of feature selection to detect malware (I. Dogru and K. Ömer, 2018). Additionally, because IoT devices have limited resources like cost, memory, and processing power as well as calculation complexity, the majority of existing malware detection algorithms cannot be used directly for IoT devices. Several techniques are developed for selecting the features of malware, for instance information gain technique, but they are based on limited types of feature to detect the malware.

To solve the problem, machine learning are posing a momentum around the world towards their use for intelligent model. In this structure, each type of latest malware can inspect by machine learning methods using our enhanced classifying and Multi-featured Naive Bayes classifier. Machine learning was used to detect and train the model for Android malware detection, creating a vast amount of malware database. Thus, this framework can identify the new type of malware for IoT devices. In addition, the proposed framework detects malware using different features information to reflect various characteristics of applications in numerous aspects. In this work, Methodology first distinguishing the malware and benign, and refines them using enhanced classifying methods. Our classifying method calculates the feature importance scores of each feature set and iterative reduced the unnecessary features that can effectively distinguish malware and benign applications even though malware have many properties similar to benign applications. Moreover, the classifying method handles multidimensional data by reducing the features using feature importance score learning procedure. This property enables the proposed method to fit for applying on many types of classifying problems. For example, it can be applied on image recognition, cell formation, topic identification, text summarization and many more (R. Kumar et al., 2019).

1.4. Research Questions

This systematic review aims to answer the following research questions.

RQ1: What are the significant feature extraction technique and data preprocessing methods in Android feature engineering?

RQ2: How to improve the performance of the Android malware detection by minimizing prediction deviation?

RQ3: How precise is the malware detection of recent Android malware by comparing models and analysis approaches?

1.5. The Objectives of the Study

1.5.1. General Objective

The general objective is to improve malware detection accuracy of android IOT devices based on Android feature analysis using machine learning approach.

1.5.2. Specific Objectives

To achieve the general objective, the following mentioned specific objectives are addressed:

- To study the recent research on Android malware detection technique and IoT technology evolution from popular repository.
- To extract features of the Android application hybrid analysis implemented.
- To evaluate the model different evaluation metrics are used.
- To reduce the irrelevant features importance score calculation.
- To reduce overfitting and underfitting the model validation.

1.6. Contributions of the study

This study proposes a multi-feature Naive Bayes algorithm for classification of Android malware. The input matrix for decision tree based Naive Bayes is the output matrix of the classifying process which includes feature importance scoring matrix and classifying labels. Among many useful classification algorithms, it's analyzed that the Naïve Bayes is the most suitable classification approach for various types of features. The advantage of this method is can be applied directly to the mobile device. This study's contribution includes major folds as follows:

- Minimize prediction deviation
- Enhance the classifying algorithm for feature selection by calculating the feature importance scores for each feature set.
- Propose Naive Bayes, Decision tree and SVM algorithm for classification to achieve high accuracy.
- The proposed work reduces the irrelevant features based on the decision tree algorithm and this minimizes computational time.

1.7. Scope and Limitation of the Study

1.7.1. Scope of the Study

This study focuses on detecting of the malware in Android OS rather than other platforms, in order to find the best solution that fits well the attention has to be given in the devices that has high market share. This work elaborate the dynamic analysis that is used to extract the training characteristics of the model and the classification models will be evaluated through well-known metrics such as accuracy, recall and precision to identify and distinguish Android malware effectively.

1.7.2. Limitation of the Study

Implementing a hybrid analysis is more challenging and complex than a single static approach.
Due to this fact using graph results from flow analysis is beyond the scope of this study. Beyond this study, do not address the very complex behavior of the Android applications due to different constraints like the schedule and resource limitations such as standard datasets. Because some information is missing or incomplete, so the review of the technology is based on the literature available.

1.8. Thesis Organization

The contents of the paper are discussed as follows:

- CHAPTER ONE: Discussed the introduction of the study, the motivation, and the statement of problems, the research questions that answered in the proposed solutions, the scope and limitation, the objective of the study, the methodology, and the beneficiaries of application results.
- CHAPTER TWO: Presents about literature review. It includes systematic literature review ,the overview of the mobile application, types of smartphone operating systems, mobile OS security, Basic concept of Machine learning, and related works.
- CHAPTER THREE: Discusses the research methodology, which includes a methodology for data collection method, Android malware detection methods, design proposed framework method, the framework implementation method, and the prototype evaluation method. It also describes the design and implementation tools used for the research work.
- CHAPTER FOUR: Discusses the proposed framework design, the analysis models, proposed framework components. The different techniques of feature extraction and features selection for the sample prototype is discussed.

- CHAPTER FIVE: Describes the results of the study , comparison and validation of models are presented. Discussion about the framework based on sample prototype implementations.
- CHAPTER SIX: Consists of a conclusion, recommendation, and identifies future works for further research direction on this research topic.

CHAPTER TWO

2. LITERATURE REVIEW AND RELATED WORK

2.1. Overview

The internet-based connectivity of various smart gadgets is known as the Internet of Things (IoT). IoT developments have significantly increased in recent years. These IOT gadgets can receive, gather, and transfer information thanks to features like sensors and internet connectivity. Market acceptance is rising, and IoT is still experiencing significant growth. Developers strive to link more electronic gadgets in homes and offices in order to meet demand. Android's rapid ascent as a software platform is largely due to Google's (the corporation that created it) decision to make it freely available to developers and device manufacturers. Today, Android is the operating system of choice for a large number of devices. Given how many devices use Android, it is simple to understand how Android acts as a front end for IoT. It is easy and cheap to develop devices for IoT making them even more affordable for consumers. The majority of Internet of Things (IoT) devices are developed in Java. Java is compatible with Android, which makes logical given that embedded Java requires tinyML devices. Understanding the IoT ecosystem and Android's place in it is necessary before understanding how Android may be used in IoT.

2.2. Android Devices

Android is a mobile operating system that has gained popularity and has been around for almost 15 years. It serves as the global standard operating system for smartphones and tablets. This mobile OS is currently owned by the Google company. In contrast to Apple's iOS, Microsoft's Windows, and Mac OS, all of which are closed-source platforms, Android is accessible to everyone and can be used for commercial purposes.

Currently, the most popular operating system in the world is Android. According to Statcounter from GlobalStats, Android has a 69.74 percent market share for mobile operating systems as of January 2022. With a market share of 29.49 percent, Apple iOS is ranked second. On Google Play, there are more than 2.6 million Android applications. Thanks to the plethora of programs available in the store, these phones are highly powerful and simple to use. Some programs are also susceptible to different virus and malware strains.

2.2.1. Android Things

A platform for an embedded operating system based on Android that was unveiled at Google I/O 2015 was released by Google as Android Things in 2018. Android Things Dashboard closure began on January 5, 2021. After January 5, 2022, the Android Things Dashboard will be completely shut down, and all of the data will be deleted. Originally intended for Internet of Things (IoT) devices with little memory and power, Android Things began supporting smartphones-class hardware in 2019 and has since discontinued supporting low-power hardware.

The Android ecosystem, developed by Google, helps to create IoT devices. Applications for these devices will be made by developers in a similar way to how they make apps for mobile devices. The integration with the entire Android ecosystem, including access to Google Services like Messaging, Voice, Authentication, Play, Assistant, etc., development using Android APIs on Android Studio, and general purpose and security updates delivered to the system through the typical Android channels, is the main improvement brought about by Android Things. Developers using the Android NDK will still be able to build programs in C/C++, but they now have the option to write programs in Java.

Weave, Google's protocol for connecting IoT devices, has been enhanced to more directly access cloud services, including the Assistant, which provides device control. In order to directly access more cloud services, such as the Assistant, which enables voice commands to manage devices, Weave, Google's IoT device communication protocol, has been upgraded. Google plans to enable new device types, including specialized ones, in addition to the standard smart lightbulbs, plugs, switches, and thermostats. Several manufacturers, including Belkin WeMo, First Alert, Honeywell, LiFX, TP-Link, and Wink, support Weave.

2.3. Mobile Operating System

Google created the Android operating system, a mobile operating system that is largely used for touchscreen gadgets, mobile phones, and tablets. Because of the way it is made, users may operate their mobile devices naturally using finger movements that mimic regular actions like pinching, swiping, and tapping. In addition, Google uses Android software in automobiles, watches, and televisions, each of which has an own user interface.

Figure 2.1 Mobile OS market-share, Ethiopia

2.3.1. Android OS

Google's open and free software stack for mobile devices, called the Android mobile OS, consists of an operating system, middleware, and application layers. Google is in charge of the OHA, a group of over 30 firms that created Android.

2.3.2. iPhone OS

A popular form of operating system (OS) for Apple's smartphone devices is called iPhone OS (iOS), and it was created by Apple Inc. This OS is now compatible with a variety of Apple products, including the iPhone, iPod Touch, iPad, and Apple TV.

2.3.3. Samsung OS

The Android Operating system is software that is developed by Google, and then customised for Samsung devices. The well-known Samsung Galaxy series of cell phones runs only on the Android platform. Phone developers can easily adapt this open-source operating system to their requirements. Numerous phones in various sizes and price points are part of the Galaxy line. The Rugby, Replenish, and Continuum are further Samsung mobile devices that run the Android operating system. The Windows Phone operating system is used on the Samsung Ativ series and the Focus. The Bada operating system is used by the Wave line.

2.3.4. Linux OS

The use of Linux-based operating systems on portable devices whose primary or only Human interface device (HID) is a touchscreen is known as "Linux for mobile devices" or "mobile Linux." The concept of mobile Linux was introduced relatively recently, with Google's Android operating system serving as its

pioneer. Even while Canonical Ltd. made an attempt to emulate this with Ubuntu Touch, the development of free Linux operating systems expressly for mobile devices wasn't really sparked until the middle 2010s, when a number of smaller businesses began working on open source phone development projects.

2.3.5. Symbian OS

Nokia introduced the smartphone operating system known as Symbian [11]. In the past, Symbian OS was developed by a number of hardware manufacturers, including Motorola, PSION, Ericsson, and Nokia. Only Nokia devices are used to run the Symbian underpinnings today.

2.3.6. WebOS

For smartphones and smart TVs, Web OS is a proprietary operating system built on the Linux kernel and created by Palm. HP Web OS and Palm Web OS are the names of this OS. HTML, CSS, and JavaScript code are used to implement both native and third-party programs.

2.3.7. Blackberry OS

For smartphones and smart TVs, Web OS is a proprietary operating system built on the Linux kernel and created by Palm. HP Web OS and Palm Web OS are the names of this OS. HTML, CSS, and JavaScript code are used to implement both native and third-party programs.

2.3.8. Windows OS

In 2010, Microsoft Corporation introduced this operating system under the name Windows Phone 7. Windows Phone devices are currently being developed by a number of mobile manufacturing companies, including HTC, Nokia, Samsung, and LG.

2.4. Android Applications

A software program that runs on the Android platform is known as an Android app. A typical Android app is created for a smartphone or tablet PC running the Android OS because the Android platform was created for mobile devices. A software program that runs on the Android platform is known as an Android app. A typical Android app is created for a smartphone or tablet PC running the Android OS because the Android platform was created for mobile devices. Although developers can make Android apps available through

their websites, the majority of Android apps are submitted to and published on the Android Market, a website devoted to these apps. Both free and paid apps are available in the Android Market. Java is the programming language used in Android apps, and Java core libraries are used. To execute on the Dalvik virtual machine, a virtual computer created especially for mobile devices, they must first be built to Dalvik executables. The Android website offers a software development kit (SDK) for download by developers. For building Android apps, the SDK comes with tools, sample code, and pertinent documents. If we divide mobile apps into three categories based on the coding language they were written in:

- Native apps are designed specifically for a given platform or operating system.
- Web apps,
- hybrid apps, and native applications are all created particularly for the operating system (OS) of a mobile device.

2.4.1. Android Components

Developers can download a software development kit (SDK) from the Android website. The SDK includes tools, example code, and relevant documents for creating Android apps.

1. Activities

Any individual screen on the user interface is represented by an Android activity. A straightforward login screen with a username and password, for instance, is one of the application's activities. Applications execute tasks in accordance with the process. This is the main element of the application's user interface. The Activity Java class is the source of all Android application screens. They are Views containers.

2. Services

A component that discreetly operates in the background to carry out specific tasks is known as an Android service. For instance, a service might collect some remote data while a user is engrossed in reading the content on a certain application page. These are background elements that operate similarly to Windows services and UNIX daemons. They operate silently and carry out continuous unsupervised processing.

3. Broadcast Receivers

In order for other programs to be aware of the event and maybe initiate some action, Android applications broadcast messages continuously. Responding to broadcast messages issued by other programs or the Android system requires the use of broadcast receivers.

4. Content Providers

On the basis of specific requests, content providers fulfill the task of providing data to other applications. One of the finest methods for cross-application data sharing is through content providers. The preferred method of transferring data between applications is through data managers. The content provider API is used by Android applications to expose data from their database when sharing data. For instance, the Android platform's contact content provider permits an infinite number of applications to reuse contact persistence. An application can incorporate access to a user's contacts that are synced with the Google cloud and saved locally by simply invoking this content provider.

Applications don't need to provide their own database manipulation code in order to read and write data in content providers. This is how content producers offer a significant feature that makes it simple for developers to design applications with complex data management; frequently, applications just write a little amount of their own data persistence code.

5. Manifest

This is an XML file that contains all configuration parameters of an Android application. In Android projects, a manifest file is included with the project's software and resources when the project is built. This file instructs the Android operating system on how to set up and operate the software found in the ZIP containing the completed project. The ADT plug-in offers a dedicated XML editor to edit the manifest file, which is in an XML format. Applications may need to store significant amounts of data to control their runtime behavior. Some of this data describes the application environment: the app name, the intents it registers, the permissions it needs and so on. All of an Android application's configuration parameters are contained in this XML file. When an Android project is built, a manifest file is added along with the project's resources and software. This file instructs the Android operating system on how to set up and operate the software found in the ZIP containing the completed project. The ADT plug-in offers a dedicated XML editor to edit the manifest file, which is in an XML format. To manage their runtime behavior, applications may need to store a lot of data. The name of the program, the registered intents, the required permissions, and other information are some of the facts that describe the application environment.

2.5. Android Malware

Malicious software that specifically targets Android devices is known as Android malware. Malware can flourish in an environment made less secure by Android's infrastructure, which includes its Play Store where users can download programs and side-load content from the internet. Among mobile devices, Android

devices are the most prone to malware infection and spread. You'll learn more about mobile malware and some of the ways it spreads in the Android ecosystem in this session. Malware for Android devices is essentially the same as malware for desktop or laptop computers, with a few minor exceptions. It only targets Android-powered devices. Malicious software or code that targets mobile devices, such as trojans, adware, ransomware, spyware, viruses, or phishing apps, is referred to as mobile malware.

What source does it have? A lot of locations As we've already covered, numerous sorts of software are hidden with malware in third-party app stores, where people go to get new games, for instance. Contrary to Apple users, Android users are also capable of side-loading apps. The owner of the device must modify the "unknown sources" security authorization in order to accomplish this. Following that, users can download material directly from the internet to their smartphone or PC, skipping the Play Store entirely.

Android malware is becoming more prevalent, therefore users of mobile devices need to be cautious about the apps they choose to download. Hackers can quickly access your personal information by using malicious spyware in the guise of apps.

2.6. Android Malware Categories

The malware categories listed below are based on our core conviction that consumers should be aware of how their devices are being used and should support a secure ecosystem that fosters thriving innovation and a reliable user experience.

1. **Adware*:*** Adware is virus that spreads advertisements. It is a malicious program that displays intrusive ads on the user's screen, especially when he or she is accessing web services. Adware entices users to click on flashing advertising for profitable products by presenting them with tempting offers. The creator of this annoying application receives money when a user clicks on the advertisement. Gexin, Batmobi, Ewind, Shedun, Pandaad, Appad, dianjin, Gmobi, Hummingbird, Mobisec, Loki, kyhub, and Adcolony are a few significant adware families.
2. **Backdoor*:*** Backdoors function as covert entry points into a smartphone. In other words, backdoors give an attacker the ability to access a smartphone whenever they want by circumventing authentication and granting them higher rights. Backdoors make it possible to launch remote attacks without physically being present at the target device. They could be components of an already-existing program or whole new ones. Mobby, kapuser, hiddad, dendroid, levida, fobus, moavt, androrat, kmin, pyls, and droidkungfu are a few typical families of backdoor Android malware.

3. **File infector*:*** Malware that affixes itself to APK files is known as a file infector. Android Package Kit, or APK, is a container for all the information needed to run an application. APK files are used to install the file virus. When APK files are installed, the malware is then put to use. Any Android application, including games, word processors, navigational tools, and other programs, can be contained in an APK file. These groups record the IMEI, device ID, and phone status.
4. **PUA*:*** PUAs, or potentially unwanted applications, are free, reputable programs that are packed with PUAs. Sometimes they are referred to as possibly undesirable programs (PUPs). PUAs don't always cause harm. Whatever their usage, it all depends. When the application it's bundled with is installed, a PUA also gets installed automatically. Adware, spyware, or hijackers are some examples of its forms. Adware is what happens when PUAs start to display advertising. PUAs use up memory, which slows down the device.
5. **Ransomware*:*** Ransomware is a type of malware that encrypts computer files and directories to prevent users from accessing them. It demands a sizable payment in exchange for the decryption key needed to open the data. Bitcoins are frequently exchanged as ransom. The fact that some customers were unable to recover their data even after paying the ransom has been corroborated by a number of occurrences. Some of them claimed to have received files with errors. Sometimes, files just disappeared. We cannot vouch for the usefulness of paying a ransom. Ransomware for Android has substantially changed, and new variations are appearing.
6. **Riskware*:*** A genuine program called riskware can potentially put the device's security holes at risk. Despite being a legitimate program, it is used to hijack devices, steal data, and drive users to dangerous websites. Alternative terminology for it is dangerous software that compromises device security in order to execute functions.
7. **Scareware*:*** Scareware is a fear-monger that induces people to download or purchase malicious programs by provoking their dread. For instance, persuading consumers to download a bogus program that claims to protect the device. Avpass, Mobwin, and Fakeapp are three well-known scareware families. The goal of the scareware families is to download dangerous software onto the device, gather device data, and track its GPS location.
8. **Spyware*:*** Once installed on a device, harmful software known as spyware has the ability to steal confidential data. The information that spyware collects is given to advertisements, outside agencies, or businesses. Later, this information is employed for nefarious purposes. Spyware is installed without the user's consent even when Android asks for permission to access information like as location, camera,

and settings. Spynote, spydealer, smsthief, spyagent, spyoo, smszombie, and smforw are examples of common spyware families.

9. **Trojan*:*** Trojans are crafty clones that act like authentic software. They can remain undetected while stealing data from the device. It is the largest malware category and includes the trojan-banker, trojan-dropper, trojan-sms, and trojan-spy malware types.

2.7. Machine Learning

Machine learning (ML) is a topic of study focused on comprehending and developing "learning" methods, or methods that use data to enhance performance on a certain set of tasks. It is considered to be a component of artificial intelligence. Without being explicitly programmed, machine learning algorithms create a model using sample data, or "training data," in order to generate predictions or judgments. The rapidly expanding discipline of data science includes machine learning as a key element. In order to provide classifications or predictions and uncover crucial insights in data mining projects, algorithms are trained using statistical approaches. Ideally, the choices taken as a result of these insights affect important growth metrics in applications and businesses. Data scientists will be more in demand as big data continues to develop and flourish. They will be expected to assist in determining the most pertinent business questions and the information needed to address them.

The majority of the time, machine learning algorithms are developed utilizing accelerated solution development frameworks like TensorFlow and PyTorch. A developing technology called machine learning makes it possible for computers to learn autonomously from historical data. Machine learning uses a variety of techniques to create mathematical models and make predictions based on previous information or data. When a machine learning system receives new data, it forecasts the outcome using the prediction models it has built using prior data. The amount of data used determines how well the output is anticipated, as a larger data set makes it easier to create a model that predicts the outcome more precisely. In the actual world, we are surrounded by people who are able to learn from their experiences thanks to their capacity for learning, and we also have computers or other robots that carry out our orders.

Given that "learning" is the main focus of the discipline of machine learning, there are numerous sorts that a practitioner can run into. Some forms of learning, such as "supervised learning," define entire subfields of study made up of numerous different kinds of algorithms. Others talk about effective methods you can apply to your tasks, such "transfer learning."

There are 4 types of Machine learning that you must be familiar with :

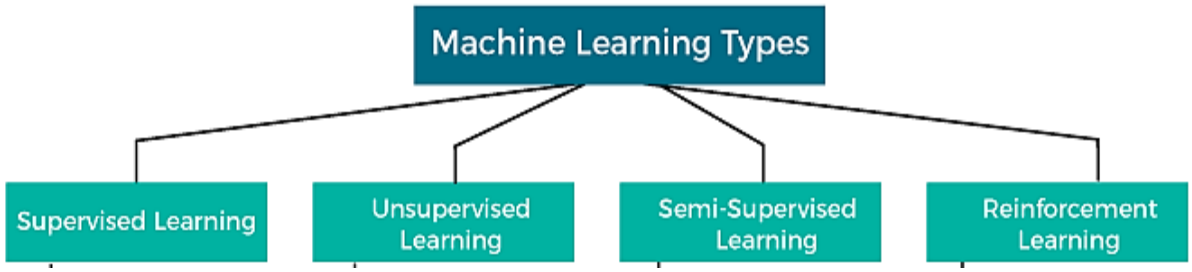


Figure 2.2 Types of Machine Learning

2.7.1. Supervised Machine Learning

Supervised machine learning is based on supervision, as its name suggests. In the supervised learning technique, this means that we train the machines using the "labelled" dataset, and then the machine predicts the output based on the training. Here, the labeled data indicates which inputs have already been mapped to which output. More precisely, we may state that after training the machine with input and related output, we ask it to predict the outcome using test dataset.

Let's use an illustration to clarify supervised learning. Assume we have a dataset of photos of dogs and cats as our input. To train the machine to understand photos, we will first show it examples like the size and shape of a cat's tail, dog, eye color, eye shape, height (dogs are taller than cats, who are shorter), etc. After training, we input a cat image and ask the computer to recognize the object and forecast the outcome. Now that the machine is educated, it will examine every characteristic of the thing, including height, form, color, eyes, ears, tail, and so on, and determine that it is a cat. As a result, it will be classified as a cat. This is the method the computer uses to recognize the objects in supervised learning.

A. Classification

When a classification problem has a categorical output variable, such as "Yes" or "No," Male or Female, Red or Blue, etc., classification methods are employed to solve the problem. The categories that are present in the dataset are predicted by the categorization algorithms. Spam detection, email filtering, and other examples of categorization systems in use today.

The following list of well-liked classification algorithms includes:

- Naïve Bayes Algorithm
- Random Forest Algorithm
- Decision Tree Algorithm
- Logistic Regression Algorithm
- Support Vector Machine Algorithm

B. Regression

Regression problems with a linear relationship between the input and output variables are solved using regression techniques. These are employed to forecast variables with continuous outputs, such as market trends, weather forecasts, etc.

The following list of popular regression algorithms includes:

- Simple Linear Regression Algorithm
- Multivariate Regression Algorithm
- Decision Tree Algorithm
- Lasso Regression

2.7.2. Unsupervised Machine Learning

Unsupervised learning is a type of learning where a computer picks up information without any human intervention. The algorithm must respond independently to the unlabeled, unclassified, or uncategorized data set used to train the machine. Unsupervised learning's objective is to reorganize the incoming data into fresh features or a collection of objects with related patterns. It can also be divided into two types of algorithms:

A. Classifying

When looking for the innate groups in the data, we employ the classifying technique. It is a method of classifying the items so that those that share the most similarities stay in one group and share little to none in common with those in other groups. Grouping clients based on their purchase habits is an illustration of the classifying method.

The following list of well-liked classifying algorithms includes

- K-Means Classifying algorithm
- Mean-shift algorithm
- DBSCAN Algorithm
- Principal Component Analysis
- Independent Component Analysis

B. Association

An unsupervised learning method called association rule learning identifies intriguing relationships between variables in a sizable dataset. This learning algorithm's primary goal is to identify the dependencies between data items and then map the variables in a way that maximizes profit. This algorithm is mostly used in continuous production, web usage mining, market basket analysis, etc. Popular association rule learning algorithms include:

- Apriori Algorithm
- FP-growth algorithm.

2.7.3. Semi-Supervised Learning

Semi-supervised learning is a type of machine learning technique that sits in between supervised and unsupervised learning. It falls between supervised learning (with labeled training data) and unsupervised learning (without labeled training data) techniques and uses a combination of labeled and unlabeled datasets during the training phase.

Although semi-supervised learning is a middle ground between supervised and unsupervised learning, the majority of the data it uses is unlabeled. It operates on data that contains a few labels. Although labels are pricey, there might not be many labels needed for corporate needs. It is fundamentally different from supervised and unsupervised learning, which depend on the presence or absence of labels, respectively.

2.7.4. Reinforcement Machine Learning

A learning agent in a reinforcement learning system receives a reward for each correct action and receives a penalty for each incorrect activity. With the help of these feedbacks, the agent automatically learns and performs better. The agent explores and engages with the environment during reinforcement learning. Reinforcement learning is categorized mainly into two types of methods/algorithms:

A. **Positive Reinforcement Learning:** By adding something, positive reinforcement learning refers to raising the likelihood that the desired behavior will occur once more. It strengthens the agent's behavior and has a favorable effect on it.

B. **Negative Reinforcement Learning:** The reverse of positive reinforcement learning is negative reinforcement learning. By avoiding the undesirable circumstance, it makes it more likely that the particular behavior would recur.

2.8. Malware Detections Techniques

The two stages of machine learning are training the model and fitting the model to the training data. putting a trained model to use on samples to get predictions. This training data is used in the training phase when we look for the model that, given the feature set X, will yield the proper label Y for previously undiscovered objects.

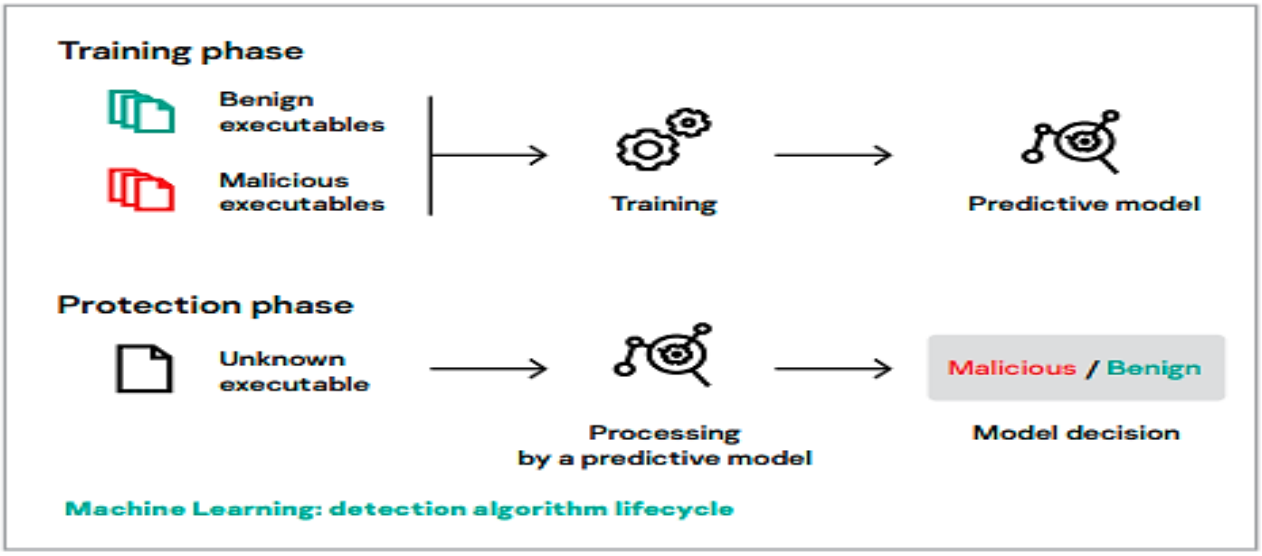


Figure 2.3 Machine Learning Phases

A aspect of a file's behavior or content, such as file statistics or a list of frequently called API calls, could be X in the context of malware detection. Labels Y could be classified as malware, innocuous software, or perhaps something more specific like a virus, Trojan-Downloader, or adware. We must choose a family of models for the training phase, such as neural networks or decision trees. Typically, the parameters of each model within a family decide it. When we train a model, we look for the model from the chosen family with a certain set of parameters that, when compared to the collection of reference objects, produces the best accurate results for the trained model. To put it another way, we "learn" the ideal characteristics that define a reliable X to Y mapping. The second step, applying the model to new objects, can be done when a model has been trained and its quality has been established. The model's type and its parameters stay the same during this phase. Only predictions are produced by the model. This is the protective phase in the case of malware discovery. Vendors frequently provide users with a trained model whose predictions are used by the product to make decisions on its own. Errors can have disastrous effects on a user, such as uninstalling an OS driver. The seller must make an informed decision when choosing a model family.

To select the model with a high detection rate and a low false alarm rate, the vendor must employ an effective training approach.

Table 2.1 Advantages and Disadvantages of classifiers

Algorithms	Advantages	Disadvantages
Naive Bayes	Despite the fact that the conditional independence assumption is rarely valid, NB models perform quite well in practice, especially considering how straightforward they are. They can scale with the dataset and are simple to build.	NB models are frequently compacted due to their extreme simplicity.
Support Vector Machines (SVM)	deal with major dimensions and non-linear problems with high precision and speed;	selecting a kernel function and frequently calling arguments are required because of heavy storage overhead;

Decision Tree	They naturally model non-linear decision boundaries and are resistant to scalability, outliers, and scaling.	Easy to overfitting;
Random Forest	Overcome overfitting, deal with high dimensional data	Dependent on the number of trees, sensitive to an unbalanced sample set
K-means	ability to self-optimize and converge quickly;	K must be predefined and is sensitive to outliers;
K-Nearest Neighbors	high accuracy and precision, nonlinear classification, and no feature assumption;	sensitive to an imbalanced sample collection and a significant computational overhead;
Logistic Regression	The outputs have a fantastic probabilistic meaning, and the process may be regularized to prevent overfitting.	weak to handle high dimensional data;

2.9. Related Works

This section deals with the analysis of existing literature on the area of Android malware detection with the objective of revealing contributions, weaknesses and gaps. Books, articles and thesis papers used to collect the necessary information for this study.

B. Tahtaci and B. Canbay (2020), in this work the trained models are combined with different feature extractionand selection methods using N-gram feature of decompiled android packages. T. Truong et al. (2020) have developed a real-time detection machine learning method using common features including API call and permission as an input to model.

F. Abdurhaman (2016) framework has been tested by analyzing both malicious and benign applications collected from different websites. The technique used is shown to be an effective means of detecting malware and alerting users about detection of malware, which suggests that it has the capability to stop the attack spread of detected malware. Experimental results show that the proposed framework has detection rate of 96.73% in RandomForest machine learning model which is used during the final development of our proposed framework as an Android application and low rate of false positive rate (0.01), performance impact on the Android system can also be ignored which is only 3.7-5.6% CPU overhead, 3-4% of RAM overhead and the battery exhaustion is only 2%.

Hyo-Sik and Mi-Jung (2018) in which they applied machine learning on selected features of Android applications. The weakness of their work is that they evaluated their proposed system with five malware families, whereas in this research, more than five malware families used, which in turn makes the result much reliable.

(Z. Wang et al.,2017) , used logistic regression, SVM, random forest and decision trees to analyses the use of vulnerable permissions for malware detection. M. Varsha, extract static features from the manifest and application executable apk files. DAPASA utilized sensitive sub-graphs to construct five features depicting the performance random forest algorithm achieved the best detection. (Z. Wang et al. 2018), proposed a paper in android malware detection approach using feature importance score-adjusted deep learning.

Both static (such as authorization and APICall) and dynamic features are available (SystemCall). Naive Bayes is used for dynamic analysis, while the SVM classifier is used for static analysis, and that is a labor-intensive operation. Additionally, by choosing both static (permission) and dynamic (important API, SMS, User activity System call) features, (A. Saracino et al., 2018) applies the KNN technique. Consortium Blockchain-based malware detection in mobile devices, to reduce a false-Positive and improve the detecting ability malware variants blockchain based android malware detection methods are proposed. Machine Learning based blockchain for COVID-19 contact tracing and vaccine support, the combination of blockchain technology with advanced cryptography techniques to guarantee security.

Table 2.2 Related works with methods used and their limitations

R.No	Title	Author & Year	Methodology	Tools	Research Gap	Repository
1.	Android Malware Detection Using Machine Learning	B. Tahtaci And B. Canbay (2020)	Decompile Android package to extract n-gram features (CNN, SVM)	Monkey tool & Emulator	Features are selected randomly, no feature_importance_score calculation and this affects the prediction accuracy.	IEEE
2.	A Machine Learning Approach for Real Time Android Malware Detection.	T. Truong et al. (2020)	Common features including API call and permission are extracted statically (DNN)	Android Asset packaging tool	It lacks the advantage of dynamic analysis, as dynamic malicious payloads cannot be detected	IEEE
3.	An SVM-based Malware Detection Mechanism for Android Devices	Y. Chen et al. (2018)	Both dynamic and static analysis for feature extraction (SVM)	LibSVM, API monitor	Android API call is used as a feature set and there is no comparison of different ML models.	Science Direct
4.	Research on data mining of permission-induced risk for android IoT devices	RU Khan et al. (2018)	Permissions as static analysis features (RF)	APK tool, WEKA	Doesn't use features other than Android permission and this hinders the malware detection accuracy	Google Scholar
5.	An android malware detection approach using feature importance score-adjusted deep learning	Wenjia & Wang et al. (2018)	Permissions as static and dynamic analysis features (KNN, DT, RF)	Monkey tool, WEKA	Dynamic and static analysis are used separately, No hybrid analysis advantage	Research Gate

CHAPTER THREE

3. THE RESEARCH METHODOLOGY

This chapter discusses the research methodology and the process of developing an Android malware detection framework using machine learning. It begins with a methodology for how the data were collected and processed to alternatively, in the Android malware classification. Finally, the following section clarifies the explanatory information about the detection method, classification, feature selection algorithms, development tools, and framework evaluation method.

3.1. Dataset Retrieval and Description

In this section, it is about how to retrieving the required applications information as input data. Android benign and malware files are collected from different datasets and stored them on storage device. First, the decision for the repository of Android malware is defined well, next to that by the benign application repository has followed. Because some datasets in *Drebin* and *Genome* Project repository contains outdated files, wouldn't used within this study as up-to-date information affects the malware detection accuracy. Instead virusshare and Kaggle repository has requested for this paper. The virusshare and Kaggle service request needs an invitation to access. And access granted with the help of personal email communication, with a short description of the research institute and use case. The whole research design procedures diagram for this study through experimental approach.

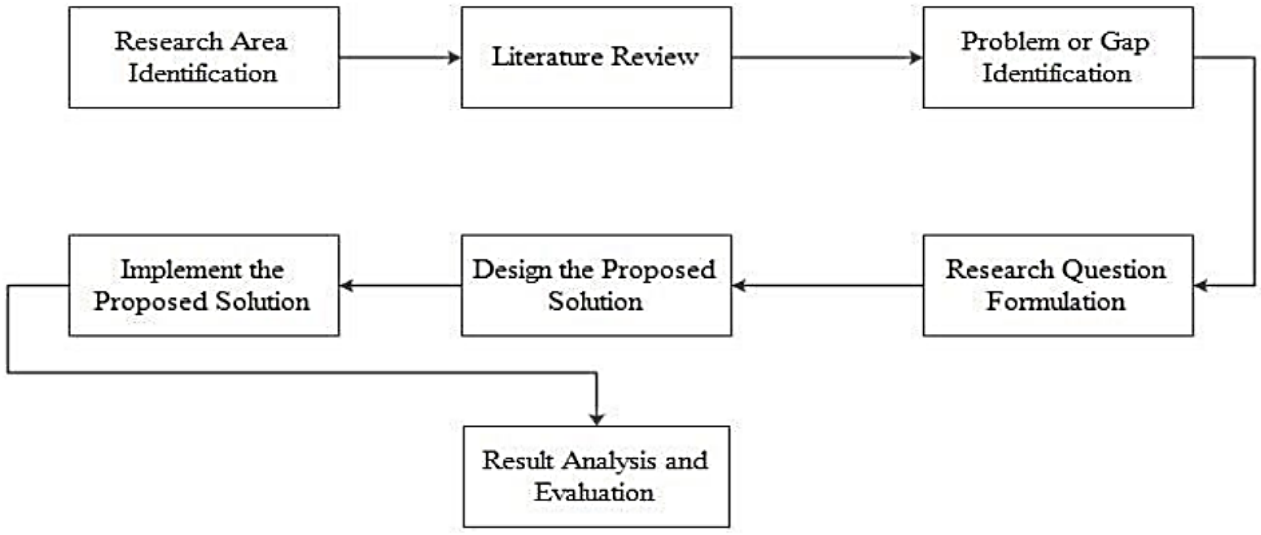


Figure 3.1 Research Design Procedure

3.1.1. Android Package Files

Android Package, Apk is a file format that is created by Google. This format is one of the group members of Executable Files, similar to .EXE, .CMD, .MST and .AIR. The Android file extension is the Open Handset which is a mobile device platform based on Linux. This Apk file is used to install and distribute an application on the Android operating system. In this study, various components of an Android installation file, known as Apk are referred. Those components can be easy access to the Apk itself as the information are stored in the AndroidManifest.xml via the Manifest Parser. The ordinary applications are mostly collected from the Google Play Store and Apps apk for evaluation purpose. Kaggle and virusshare samples are the repository of malware and benign samples to provide access for the scholars in the research to collect a sample of both malicious and benign applications.

3.1.2. Apk File Repositories

This subsection introduces app repositories used for this study in order to get benign and malware Android applications. An extensive collection named Google Play Store is described, followed by Virusshare repositories.

A. Kaggle Dataset is the world's largest and reliable data science community with csv format data, where the features are represented by the column and attributes of a dataset on Kaggle (e.g. file_name, Intent_name, each activity… etc.) and each row is a dataset on Kaggle repository. Every day a new datasets are uploaded on Kaggle in the way of appropriate way to the researchers use the data to train their models.

B. Google Play Store is the popular Android app storage for Android applications. Several apps are organized in categories. From this storage, downloading the apps directly in APK format may not be possible, but through the service *Android apps Downloader can* provides this functionality. Other Android app repositories allows a direct download of APKs file, for instance, Apkpure or *APKMirror*. Both of these websites offer downloads of benign as well as malware apps from different categories.

C. Virusshare , this is a malware repository that provides samples to security researchers, malware analysts and incident responders access to the datasets. Researchers can inspect, analyze and evaluate the behavior and features of a particular malwares.

3.2. Data Pre-processing

Data preprocessing is mining data through various technique and preparing for a model and meaningful data format to analyze the necessary data for processing. It is basics to step into machine learning development for modeling. In this study, the data are collected by the research scholars needs to be pre-processed since there are a lots of weaknesses during data collection. If there are a missing data occurrence and that significantly affect the desired output and Algorithm model accuracy. Data pre-processing have a steps with the following section, such as cleaning data, transformation data, and reduction data.

3.2.1. Data Cleaning

Recently, the researcher focused on the missing data for the model preparation in these steps. By replacing in missing data, in such way the data can be cleaned for a model. There are many options for preparing data depending on data type and the amount of data missing. Data cleaning in machine learning involve different techniques based on the problem category and the data type. Different methods with its own trade-offs can be applied. Such used techniques include mean, median, most frequent value, and regression. These listed ways of replacing or filling may affect data accuracy by filling a similar deal for every value of missing. In this study, a deterministic regression model for fill missing values and Decision tree model for accuracy prediction.

3.2.2. Data Reduction

Data reduction is part of data pre-processing that deal with various techniques, including dimensionality reduction, feature and attribute selection. Unnecessary data can be reduced and be preferable in the form that the model needs and as well as made simple for training, testing and visualizing the proposed model. The data is collected with various magnitudes. The data has been reduced based on their importance or degree of necessity for the model training and testing.

3.3. Proposed Design Solution

To enhance the significance of proposed work, the framework performance is compared with state-of-art detection systems. In this paper, the similar approaches with the previous methods have been investigated. A different kind of Android malware detection methods using machine learning has been studied before. By comparing the previous work this framework improves the performance of Android malware detection

method. The malware detection accuracy or the values F-measure of this framework were higher than the methods that was. Moreover, the number of benign and malware applications used in previous method and used in this work are different.

3.3.1. Design Tools

The available tools, platforms and devices for designing are used in machine learning. Using those tools make human beings to understand concepts through pictorial representation rather than text. To make the proposed solution more precise, the new found solutions have been designed in pictorial or diagrammatical representation. To do such kinds of works tools like, the google draw, Edraw_Max and draw.io are used.

3.3.2. Working Environment

This section deals about the objective that we have discussed in chapter one. To achieve the goal of this paper hardware tools, software tools, different techniques and programming language is used. The working environments in this framework are described as following.

3.3.2.1. Hardware Tools

Different hardware requirements are used for system development, running, and everything which software wants to integrate with the following specification:

- Processor : Corei5
- Computer : Desktop & Laptop.
- HDD : 1TB
- RAM : 8GB
- Smart phone : Android
- Computer Mouse : Wired
- Internet cable : Twisted-pair
- Data cable : USB cable

3.3.2.2. Software Tools

For implementing the source code before installing, the following software requirements are used to design, develop and implement the proposed framework :

Table 3.1 Software tools and their Description

R.No	Tools	Description
1.	Windows OS	Development Environment for all other applications
2.	MS Office 2021	For documentation & presentation
3.	Anaconda Python Version 3.8	To train and test data
6.	Android Studio Version 3.2-7.2	Android Application Development tool
7.	Foxit Reader Version: 9.7.0.2	To read PDF files
8.	Browsers	Chrome and Internet Explore, for browsing in the internet.
9.	Genymotion Version 3.2.1	Dynamic and static analysis tool

3.4. Evaluation technique and Model Improvement

3.4.1. Model Evaluation techniques

In this section, the evaluation methods of a machine learning approach are mentioned in addition to the way of minimizing and optimizing parameters used in the model training. All the data feature collected are stored in a matrix *_X as a CSV value*. The data rows have each malware or benign sample for training and testing, while the columns represents the basic feature set for the model.

Another vector *_Y* holds the labels value for data in the matrix *_X*. To evaluate supervised model the dataset is separated into training data and testing data to make it easy and clear. The data X, Y are separated into a training dataset of X_tain, Y_train and testing dataset of X_test, Y_test. In machine learning more time is spent in data training session to train the model for accuracy and performance improvement, so that in this modeling the size of data for training is 70% and the for testing the model is 30% only. The X_train and Y_train datasets are used to train the model, whereby the X_test, Y_test datasets are utilized for model testing. Therefore the labels are predicted by *X_test* dataset. Then, the *Y_test* is utilized in order to compare

the results and the test set are displayed to know the prediction accuracy. There folds that are the separation of the datasets in different subsets for cross-validation. In this framework datasets are named 10 folds, from those the 1 fold used to test the model while the other 9 folds are used to train the model. That means generally, 10 classes are used for cross-validation as shown in the fig. 3.1(folds) moreover as we have ten folds we can summarize that there are ten compositions and ten final results. When we use those features they are selected based on their depth to get prediction result as malware in decision tree and the feature with low depth is easily predicted malware. If it took long or deep depth to find that the feature represents that a malware as shown in the fig. 3.2 (tree), those feature are safe and not dangerous to use. Based on this the proposed model trains and tests data based on Naïve Bayes, Decision tree and SVM machine learning algorithm.

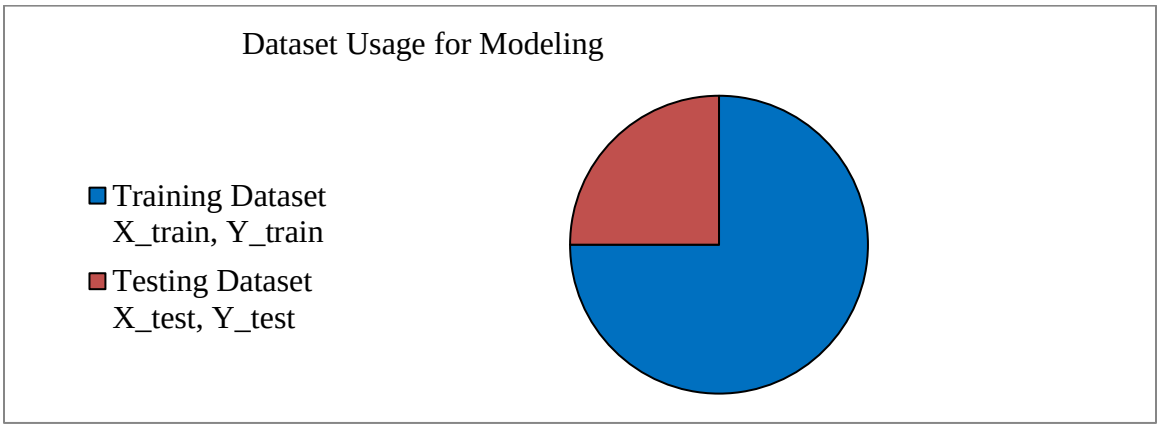


Figure 3.2 Training and Testing Dataset diagram

3.4.2. Model Evaluation Metrics

To evaluate Machine Learning model there are different kinds of metrics in different applications. Those evaluation metrics help the researcher evaluate their model's accuracy in addition to measure the performance of trained model. Identifying the appropriate evaluation metrics based on the problems you are going to solve through modeling is a vital step of any data science related researches because it is aimed that to generalize accuracy estimation of the model. If the model's response is binary, that takes only two values like **0** or **1, true** or **false, malware** or **benign, spam** or **non-spam**… etc. In such model, it is appropriate to use classification metrics rather than Regression that predicts the output. There are types of machine learning metrics such as :

A. **Classification :** in which the output value in the form of categories that is similar with attributes. And such models can provide two-valued or binary result mentioned above. Some of the most popular metrics for classification are :
   - Accuracy,
   - Precision,
   - Recall, and
   - F1 Score, etc.

B. **Regression :** this is also a kind of prediction in which the output result is numerical may be continuous value, not categorical (opposite to classification). as example, the model that predicts the length of stay for the patient in the hospital. The most popular metrics for regression are :
   - MSE (Mean Squared Error),
   - RMSE (Root Mean Squared Error), and
   - MAE (Mean Absolute Error).

3.4.3. Confusion Matrix

A confusion matrix is a core element in machine learning classification model and it is most commonly used to prediction results of binary testing data to describe the model performance in the classification. Naturally, it is described by two dimensional table of showing actual values and predicted values. There classes in confusion matrix of two dimensions as follow:

- True Positive (TP): a class is predicted true and is true in reality (data that are malware and predicted malware);
- True Negative (TN): a class is predicted false and is false in reality (data that are benign and predicted malware);
- False Positive (FP): a class is predicted true but is false in reality (data that are malware but predicted benign); and
- False Negative (FN): a class is predicted false but is true in reality (data that are benign but predicted malware).

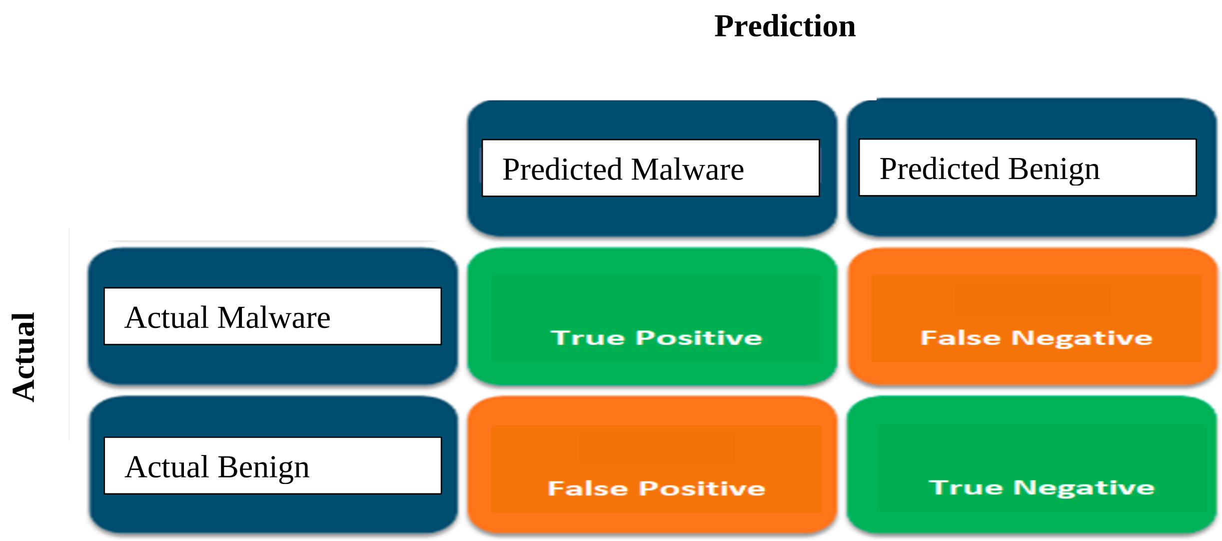


Figure 3.3 Confusion matrix

1. Accuracy

Accuracy is one of classification models metric for evaluating. Informally, **accuracy** is the predictions fraction of our machine learning model got right. actually, it has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct prediction}}{\text{Total number of prediction}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.1)$$

In case of binary classification, accuracy can be calculated as positives and negatives as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.1)$$

Where *TP* => True Positives, *TN* => True Negatives,

FP => False Positives, *FN* => False Negatives.

2. Precision

Accuracy can become an unreliable criteria for gauging our success when there is a class imbalance. As a result, we also need to consider class-specific performance indicators. One of these measurements, called precision, is defined as positive predicted values.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive + False Positive}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.2)$$

3. Recall

Another critical metric is recall, which measures the proportion of true positive cases that were correctly detected.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(3.3)}$$

4. F1-score

The F1 score is made up of two key error measurements, Precision and Recall. It can be thought of as the harmonic mean of the Precision and Recall error metrics for an unbalanced dataset in the context of binary data categorization.

$$\text{F1- Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision + Recall}} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(3.4)}$$

CHAPTER FOUR

4. PROPOSED MODEL OF MALWARE DETECTION

This chapter describes about the high-level architecture of the proposed solution is presented the components and their details of workflow within the proposed framework. The Android APK file features extraction, feature selection procedure of machine learning as well as the design of dataset analysis module are discussed.

4.1. Proposed Framework

In this section, we propose the malware detection framework which is based on two basic techniques, i) Classifying Algorithm ii) DT-Naive Bayes Classifier for Multi-Feature. Overall architecture of the proposed system is shown in Figure 4.1 A new proposed framework was presented to improve the performance of malware detection. The proposed machine learning technique provides an efficient approach to train the model. Using the previous history, we trained our model and detected malware efficiently and fast. Moreover, the first method based on a classifying algorithm, which reduces the high dimensional data and removes unnecessary features. Secondly, we use classification method based on Naive Bayes for multi-feature classification. The proposed framework runs in recurring way described in following three steps as follows :

1) Hackers create a new type of malware.
2) Machine learning identify the malware and re-train the model.

3) The new type of malware will be predicted precisely.

4.1.1. Modified Feature Reduction Using Classifying Algorithm

The classification algorithm classifies the malware application for each cluster. KNN and c-mean is a very popular algorithm for classifying. Also, the feature extraction is an essential factor for high-dimensional data. To address the problem of irrelevant features, we enhance the classifying algorithm for feature reduction that takes less time for the large dataset and low-cost computation in the training process. The proposed scheme reduces the high dimensional data and removes unnecessary features. The method calculates the feature importance scores of each feature set and iterative reduced the unnecessary features that effectively distinguish malware and benign applications even though malware has many properties

similar to benign applications. By using a feature importance score learning process to reduce the features, it can handle multidimensional data. This characteristic makes our approach a strong contender for a variety of classifying challenges. The following are objective of our proposed scheme:

- Calculating the feature importance scores for each feature set.
- Developing a parametric study for optimization.
- Iterative reduction of unnecessary features having small feature importance scores.

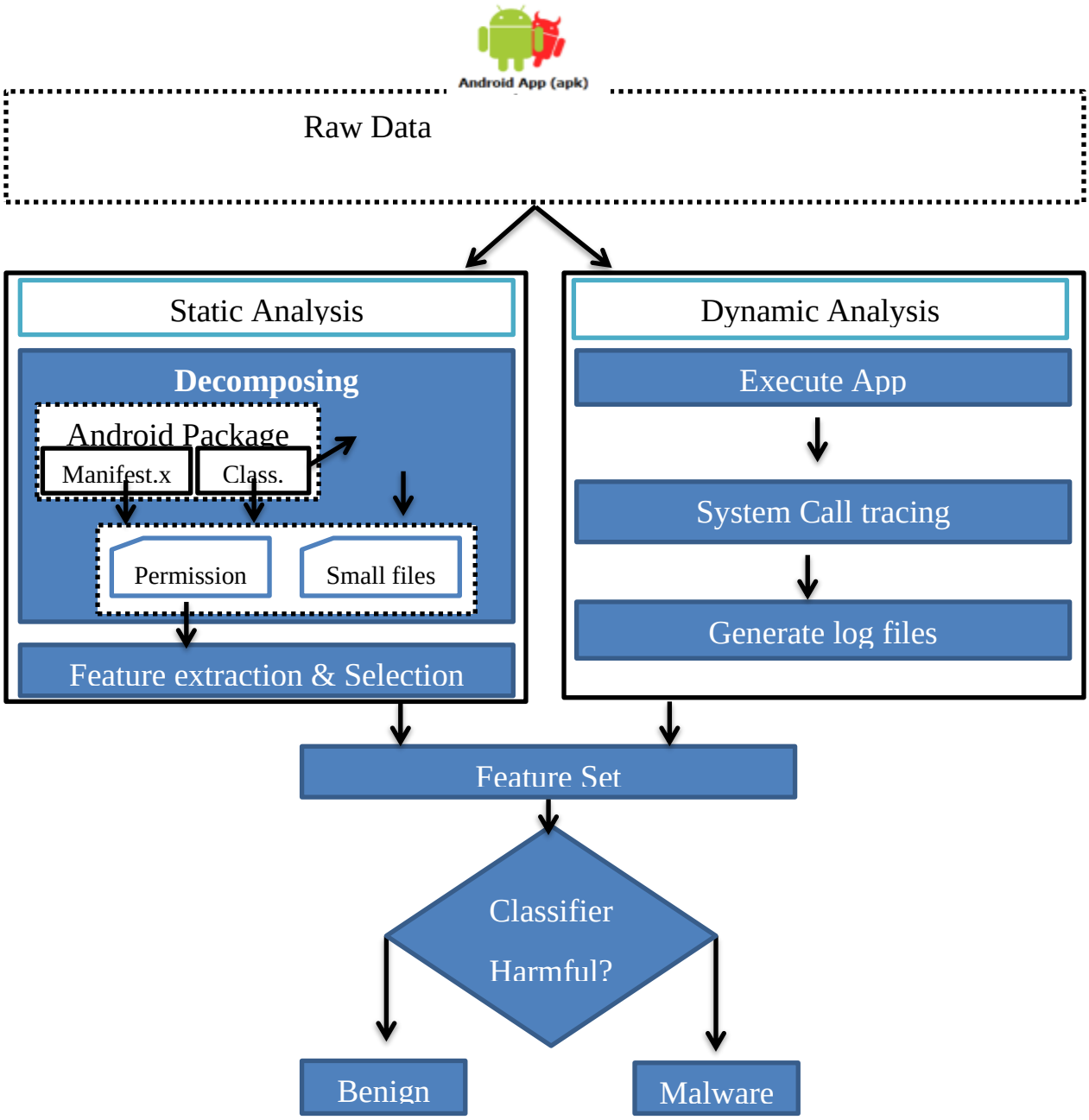


Figure 4.1 Proposed Analysis Model

4.2. Proposed Analysis Model

As was said in the previous section, when analyzing malicious Android applications, both static and dynamic methods of analysis are applied. The first technique, known as dynamic detection, uses an isolated environment to run the Android application while monitoring its behavior to identify behavior profiles that correspond to known malware applications. Additionally, the Android platform was created through extensive teamwork and financial investment from numerous businesses. This platform's development is being spearheaded by various companies, notably Google. The Android system's design, division into layers, and systemic construction are all referred to as the Android architecture. Android is a complicated system that requires careful building and structuring to guarantee that all the parts function together as a whole. Numerous internal parts of the Android application are organized into four layers:

1. Application
2. Application Framework
3. Libraries and Android Runtime
4. Linux Kernel

4.2.1. Static Analysis

Recently, static machine learning elements have been used to identify Android malware, including API calls, permissions, Intents, and internet access. The proposed paradigm is built on permissions and API calls. The features are taken from androidMainfest and the program executable file apk. Furthermore, when compared to the previous model, this one achieves high TPR (%) and FPR (%). The datasets used, which specifically explain the static properties, are made up of 1149 benign and 1850 malicious samples. Below are some static approaches for detecting malware on Android. Since most of them require a lot of processing time and memory, we perform better when utilizing decision tree-based reduction of irrelevant features from the dataset. The table below provides a description of the feature set this framework uses.

Table 3.1 Static Analysis Feature set

Features Set		
AndroidManifest	S1	Hardware components
	S2	Requested components
	S3	Application components
	S4	Filtered intent
Dexcode	S5	Requested API calls
	S6	Permissions
	S7	Suspicious API calls
	S8	Network addresses

- **Android Package**

To extract the feature sets we should have explore the Android package hierarchy tree from AndroidManifest and Dexcode file. An AndroidManifest.xml file (exactly that name) must be present in the root of the project source set for every app project. The Androidmanifest file provides crucial information about your app to Google Play, the Android operating system, and the Android build tools. An executable file saved in a format called DEX contains compiled code created for Google's Android mobile phone platform, which runs Linux. It can be interpreted by the Dalvik virtual computer and is formally known as a "Dalvik Executable." As demonstrated in the screenshot in Appendex-A, all components were located with a specific path for each file:

According to this work there are two different static analysis methods :

1) **Permission-Based Analysis**

Many Android security models are built on the permissions provided in the manifest file, which is where the features of the permissions in Android are extracted. Every Android application need the privilege to use certain functionalities. During the thorough installation of the application, the user should have properly provided the permissions requested by the Android platform. Malicious software can take advantage of certain apps. For instance, a program could be used to access the SD card and the Internet and filter out

sensitive data. 48 permissions are risky permissions, which may engage in dangerous behavior, out of the total 145 Android permission sets. And to create this model, 22 extremely risky permissions are segregated from the rest.

Risky Permissions	
ACCESS_WIFI_STATE	SEND_SMS
READ_LOGS	READ_CALL_LOG
CAMERA	DISABLE_KEYGUARD
CHANGE_NETWORK_STATE	RESTART_PACKAGES
WRITE_APN_SETTINGS	SET_WALLPAPER
CHANGE_WIFI_STATE	INSTALL_PACKAGES
READ_CONTACTS	WRITE_CONTACTS
WRITE_SETTINGS	GET_TASKS
RECEIVE_MMS	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	SYSTEM_ALERT_WINDOW
READ_HISTORY_BOOKMARKS	RECEIVE_BOOT_COMPLETED
ACCESS_NETWORK_STATE	CALL_PHONE
READ_EXTERNAL_STORAGE	ACCESS_FINE_LOCATION
EXPAND_STATUS_BAR	ADD_SYSTEM_SERVICE
PERSISTENT_ACTIVITY	INTERNET
GET_ACCOUNTS	WRITE_SMS
PROCESS_OUTGOING_CALLS	CHANGE_CONFIGURATION
READ_HISTORY_BOOKMARKS	GET_PACKAGE_SIZE
WAKE_LOG	ACCESS_MOCK_LOCATION
WRITE_CALL_LOG	WRITE_HISTORY_BOOKMARKS
READ_PHONE_STATE	RECEIVE_WAP_PUSH
SET_ALARAM	WRITE_SMS
RECEIVE_SMS	READ_SMS

Figure 4.3 Permissions used mostly in malware

ACCESS_WIFI_STATE	SEND_SMS
READ_LOGS	READ_CALL_LOG
RESTART_PACKAGES	DISABLE_KEYGUARD
READ_EXTERNAL_STORAGE	CHANGE_NETWORK_STATE
WRITE_APN_SETTINGS	SET_WALLPAPER
CHANGE_WIFI_STATE	INSTALL_PACKAGES
READ_CONTACTS	WRITE_CONTACTS
CAMERA	GET_TASKS
READ_HISTORY_BOOKMARKS	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	SYSTEM_ALERT_WINDOW
WRITE_SETTINGS	RECEIVE_BOOT_COMPLETED

Figure 4.2 Top 22 risky permissions

2. SUSPICIOUS API CALLS

This is the second technique for analyzing the source code of an application. Malicious scripts frequently combine API calls, services, and methods that are uncommon for applications that are intended to be benign. The machine learning algorithms should be able to learn those combinations of components that are primarily employed by malware programs in order to distinguish between malware and benign applications.

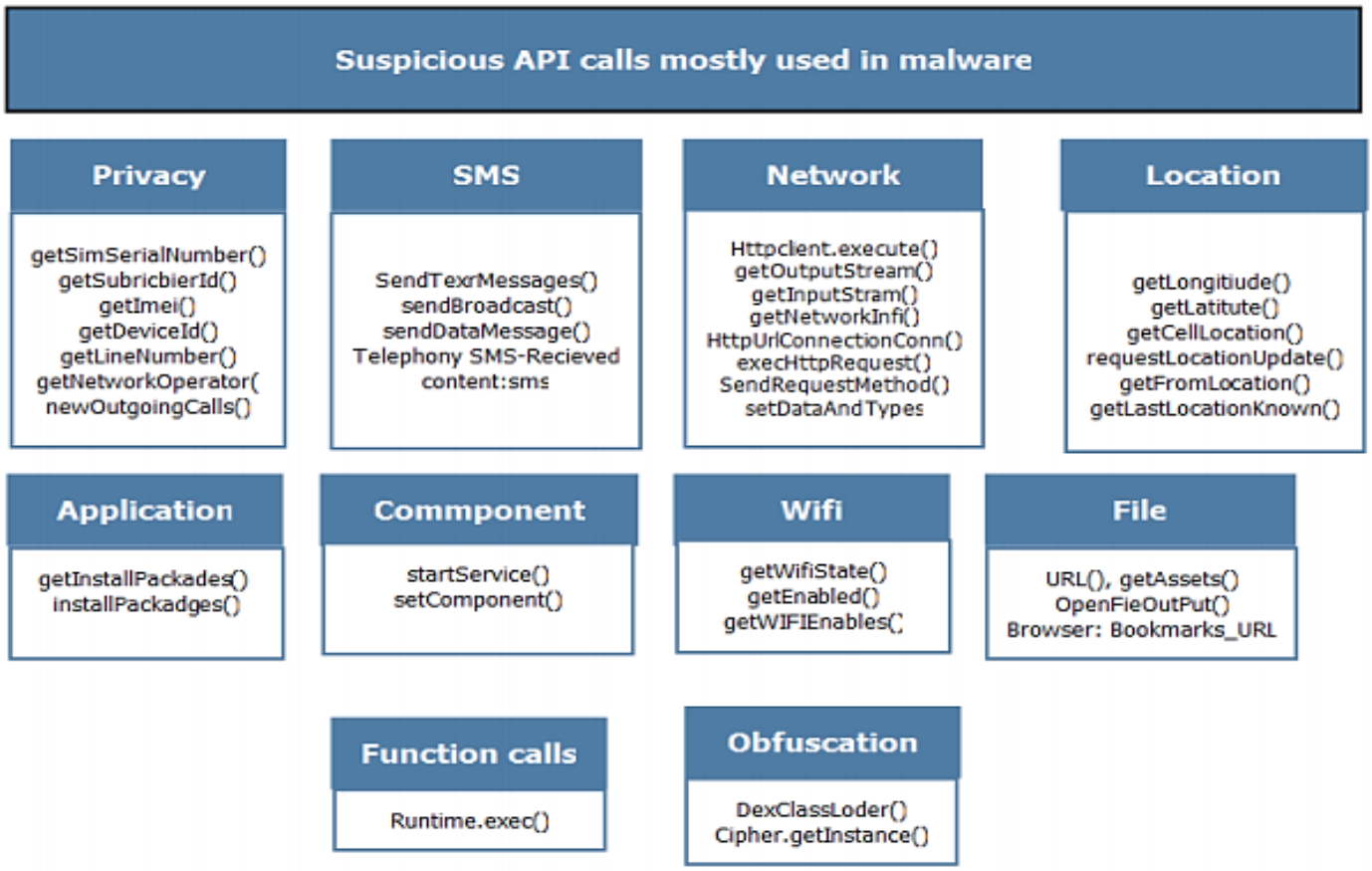


Figure 4.4 Suspicious API calls

4.2.2. Dynamic Analysis

Machine learning techniques like Genymotion and AntiMalDroid use feature vectors based on behavioral sequences to extract characteristics that are produced dynamically. In this paper we used Genymotion analysis tool for dynamic analysis. Many machine learning methods, including SVM, KNN, K-means classifying, Logistic Regression, Bayesian Network(BN), and Naive Bayes, involve dynamic analysis.

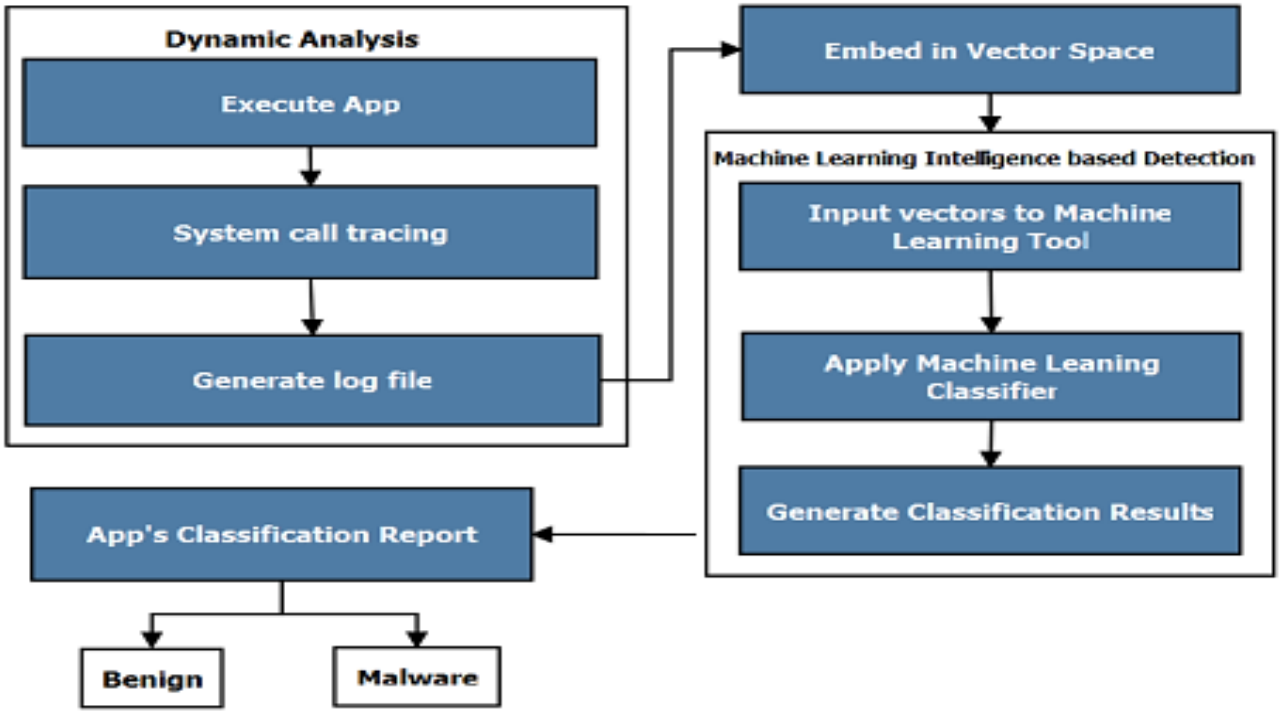


Figure 4.5 Dynamic feature extraction

4.2.3. Hybrid Analysis

Hybrid analysis combines static and dynamic analysis to enhance the performance of algorithms using various feature extraction and feature selection techniques. The key benefit of hybrid analysis over static analysis and dynamic is that it can execute with the maximum degree of accuracy. However, there are restrictions:

- highest complexity,
- framework requirement to combine the static and dynamic features,
- more resources utilization, and
- time-consumption

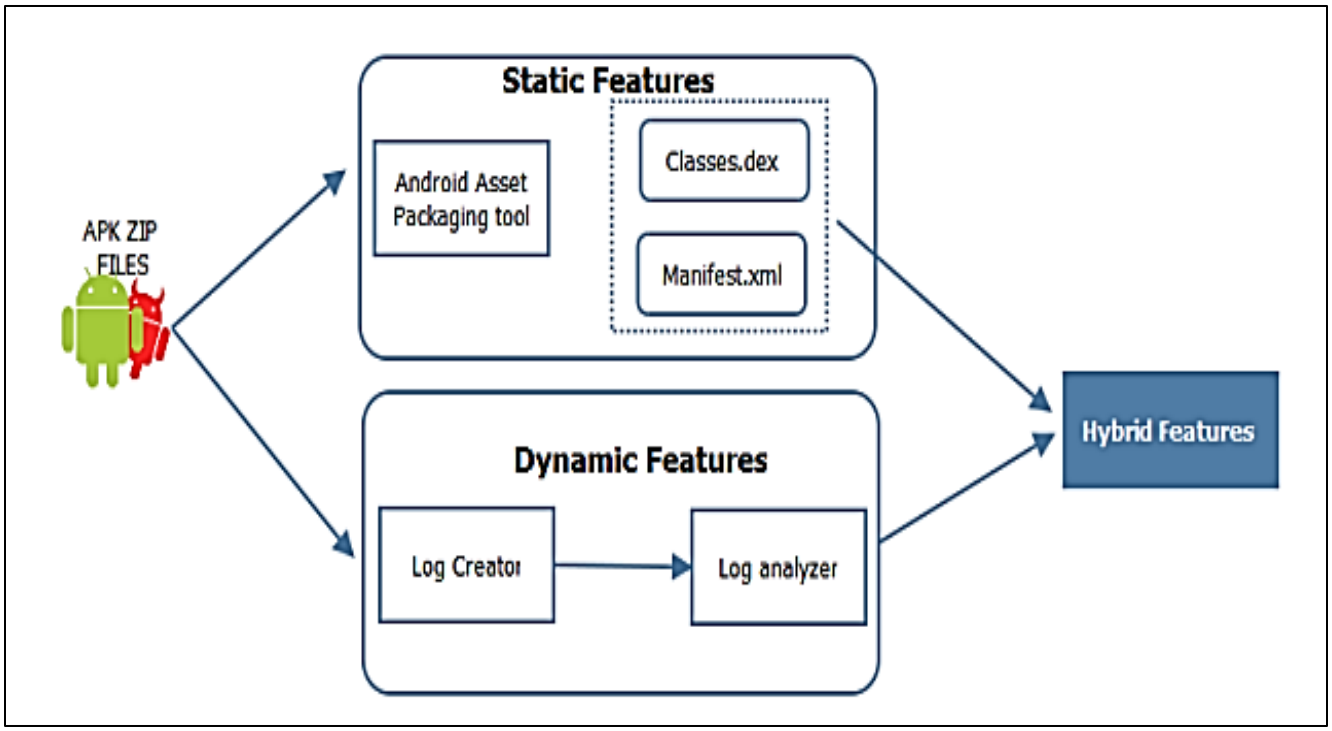


Figure 4.6 Hybrid feature extraction

4.2. Machine Learning Classifier

The feature selection are evaluating through different classification models. In this study, Decision tree classifier is used for high important feature selection, after that all those three classifier used for classifying the Android as Benign or Malware in order to choose the one with best performance. In the implementation, dataset are divided into training and testing datasets: 70%-30% and ten-fold cross-validation. In order to improve the accuracy of the model in this study, supervised learning algorithms used. There are different approaches for the classification; however, all of them require known classes for the training, testing, and also validation stages.

4.2.1 Naïve Bayes Classifier

The Nave Bayes algorithm is a supervised learning method for classification issues that is based on the Bayes theorem. This probabilistic classifier makes predictions based on the likelihood that an object will appear.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (3.5)$$

Where,

P(A|B) : is Posterior probability, Probability of the hypothesis A on the observed event B.

P(B|A) : is Likelihood probability, Probability of evidence given that the probability of a hypothesis is true.

P(A) : is Prior Probability, Probability of hypothesis before observing the evidence.

P(B) : is Marginal Probability, Probability of Evidence.

There are three Types of Naïve Bayes classifiers in machine learning those are :

A. Gaussian Naive Bayes

Gaussian Naive Bayes, which assumes that features are distributed according to a Gaussian distribution, performs well with features that have continuous values. known as the normal distribution. On a graph, it displays a bell-shaped curve with a symmetric form around the mean of the feature values.

B. Multinomial Naive Bayes

Natural Language Processing uses the Multinomial Naive Bayes algorithm as a probabilistic learning technique most frequently (NLP). The method, which guesses the tag of a text such as an email or newspaper article, is based on the Bayes theorem.

C. Bernoulli Naive Bayes

Features in a Bernoulli model are independent binary variables or Boolean expressions that describe inputs. Instead of focusing on the frequency of the word in the documents, this model is frequently used for binary term occurrence document classification jobs to determine if the word occurs in the document or not.

Input:

Training dataset T,

F= (f_1, f_2, f_3,.., f_n) // value of the predictor variable in testing dataset.

Output:

A class of testing dataset.

Step:

1. Read the training dataset T;
2. Calculate the mean and standard deviation of the predictor variables in each class;
3. Repeat

   Calculate the probability of f_i using the gauss density equation in each class;

   Until the probability of all predictor variables (f_1, f_2, f_3,.., f_n) has been calculated.
4. Calculate the likelihood for each class;
5. Get the greatest likelihood;

Algorithm 4.1 Algorithm for Naïve Bayes Classifier

4.2.2. Decision Tree Classifier

A supervised learning method called a decision tree can be used to solve classification and regression problems, but it is typically favored for doing so. It is a tree-structured classifier, where internal nodes stand in for a dataset's features, branches for the decision-making process, and each leaf node for the classification result. The Decision Node and Leaf Node are the two nodes of a decision tree. While Leaf nodes are the results of decisions and do not have any more branches, Decision nodes are used to create decisions and have numerous branches. The given dataset's features are used to execute the test or make the decisions.

It is a graphical depiction for obtaining all feasible answers to a choice or problem based on predetermined conditions. It is known as a decision tree because, like a tree, it begins with the root node and grows on subsequent branches to form a structure resembling a tree. The CART algorithm, which stands for Classification and Regression Tree algorithm, is used to construct a tree.

GenDecTree(Sample S, Features F)
Steps:

1. ***If****stopping_condition(S, F) = true* ***then***
   a. *Leaf = createNode()*
   b. *leafLabel = classify(s)*
   c. ***return*** *leaf*
2. *root = createNode()*
3. *root.test_condition =findBestSpilt(S,F)*
4. *V = {v | v a possible outcomecfroot.test_condition}*
5. ***For each*** *value v ∈ V:*
   a. S_v *= {s | root.test_condition(s) = v and s ∈ S };*
   b. *Child = TreeGrowth (*S_v*,F);*
   c. *Add child as descent of root and label the edge {root → child} as v*
6. ***return*** *root*

Algorithm 4.2 Algorithm for Decision tree classifier

4.2.3. Support Vector Machine Classifier

One of the most well-liked supervised learning algorithms, Support Vector Machine, or SVM, is used to solve Classification and Regression problems. However, it is largely employed in Machine Learning Classification issues. The SVM algorithm's objective is to establish the best line or decision boundary that can divide n-dimensional space into classes, allowing us to quickly classify fresh data points in the future. A hyperplane is the name given to this optimal decision boundary.

Algorithm 1 Training an SVM

Require: $\boldsymbol{X}$ and $\boldsymbol{y}$ loaded with training labeled data, $\boldsymbol{\alpha} \Leftarrow 0$ or $\boldsymbol{\alpha} \Leftarrow$ partially trained SVM

1: $C \Leftarrow$ some value (10 for example)
2: **repeat**
3: **for all** $\{x_i, y_i\}, \{x_j, y_j\}$ **do**
4: Optimize α_i and α_j
5: **end for**
6: **until** no changes in $\boldsymbol{\alpha}$ or other resource constraint criteria met

Ensure: Retain only the support vectors ($\boldsymbol{\alpha_i} > 0$)

Algorithm 4.3 Support Vector Machine classifier

4.3. Feature Extraction

To predict the class of Android application feature extraction performed for obtaining the feature data. The permissions used in APK file analysis are extracted from AndroidManifest.xml and this feature extraction is performed before analysis while preparing the dataset. The process of feature extraction is when the existing data are formed as a brand new. Simply it is called a dimensionality reduction or attribute reduction process. Currently, the dimensionality increase of data raises severe challenges on existing feature selection and extraction techniques concerning efficiency and effectiveness.

4.3.1. Feature Extraction with PCA

One of the most well-liked methods for feature reduction is principal component analysis. By identifying a more condensed set of features that best sum up the original feature, it manages to reduce the total number of features. PCA basically uses metrics such as variance and reconstruction error when performing this transformation. It attempted to maximize variance and minimize reconstruction error. PCA projects the initial dataset to a set of orthogonal axes. Data points that are highly correlated are clustered together in the new axes. These axes are then ranked based on their order of importance.

Since PCA is categorized as an unsupervised learning approach, the data labels are not necessary. In actuality, PCA can be used to reduce the dimension of the dataset to 1—where the categorization has already been completed. However, keep in mind that as it uses an unsupervised learning model, there may occasionally be misclassification.

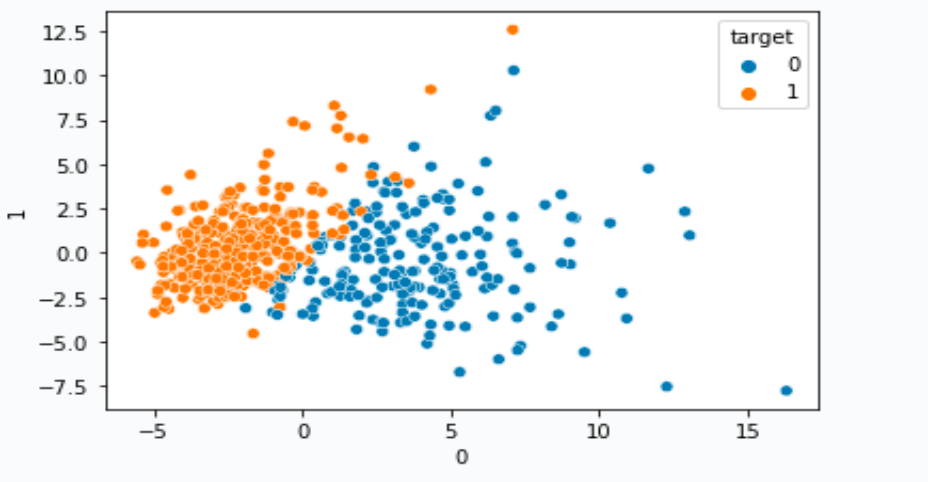


Figure 4.7 PCA feature extraction

4.3.2. Feature Extraction with LDA

LDA minimizes the separability between each feature to reduce dimension. It functions by projecting all the other features onto a new feature, which you may think of as a line. LDA aims to maximize the distance between each feature's mean during transformation. Additionally, the dispersion between the data points must be kept to a minimum. There would be very little data point overlap on the new axis if they were done. Less overlap ensures that misclassification is kept to a minimum and, thus, improves classification outcomes. Given that LDA is a supervised learning classifier, both the features and the labels are necessary (or X and Y).

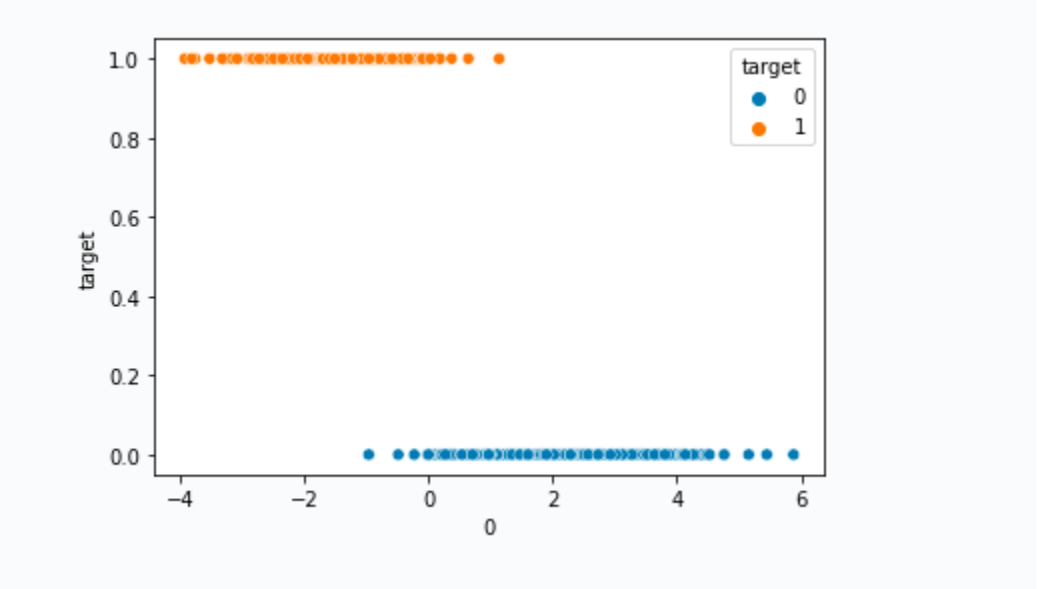


Figure 4.8 LDA feature extraction

4.3.3. Feature Extraction with Isomap

By determining the relationship between features after projecting them on an orthogonal plane, the PCA and LDA algorithms discussed above reduce the number of features. It is considerably distinct from linear relationships in that LLE (explained in the part below that follows) allows for non-linear relationships in the features as well. Manifold learning is a sort of learning that is used by Isomap.

The data is condensed through multimodal learning to a smaller set of features. However, this time, the generalization is constructed in a way that makes it sensitive to any kind of non-linear structure in the

dataset. The geodesic (or mean) distance between each point in the dataset is preserved while isomap constructs a lower-dimensional embedding of the data.

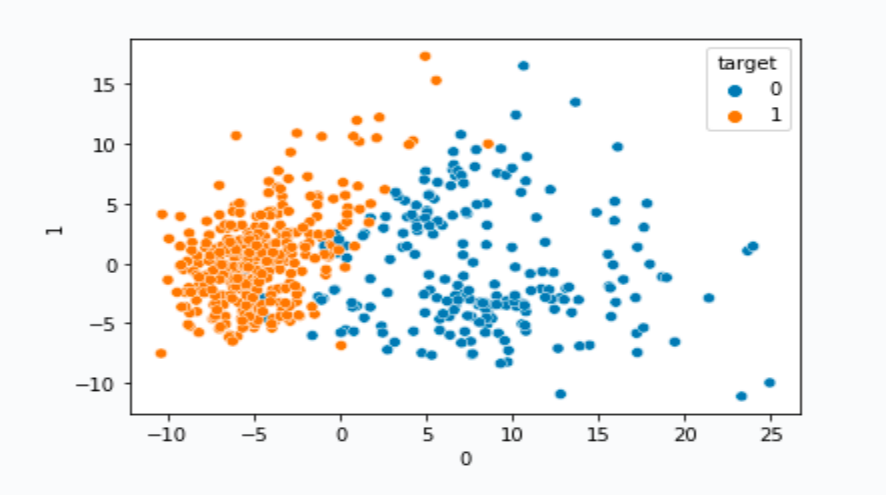


Figure 4.9 isomap feature extraction

4.3.4. Feature Extraction with LLE

Another non-linear feature reduction method that makes use of multidimensional learning is locally linear embedding. According to sci-kit learn's official documentation, locally linear embedding looks for a lower-dimensional projection while maintaining the separation between nearby neighborhoods. On the dataset, we may test the LLE method to see how it does.

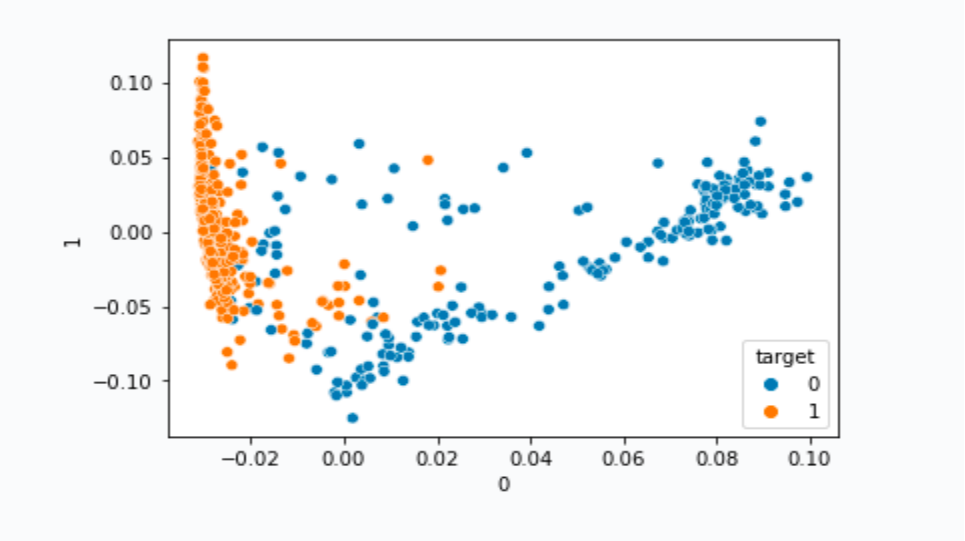


Figure 4.10 LLE feature extraction

4.4. Feature Selection

To train a model , we collect a huge quantities of data to help the machine learn better. In this thesis case, consider that there is a data of different application with many features of whether necessary or not. In our dataset there are features those minimize the performance of our model so they need to be dropped. The model may run slowly if there is too much extra data. The model can become erroneous as a result of learning from this unimportant data. By removing noise and using only pertinent data, feature selection reduces the input variable to your model. Using feature selection, we can optimize our model in several ways :

1. Preventing learning from noise (Overfitting)
2. Improved accuracy
3. Reduce train time

There are two main machine learning feature selection methods with sub-division inside them, these are unsupervised and supervised feature selection methods. Unsupervised feature selection methods are those where the output label class is not required for feature selection. While the term "supervised learning feature selection" refers to a technique that chooses features based on the output label class. The flow chart below show the feature selection methods.

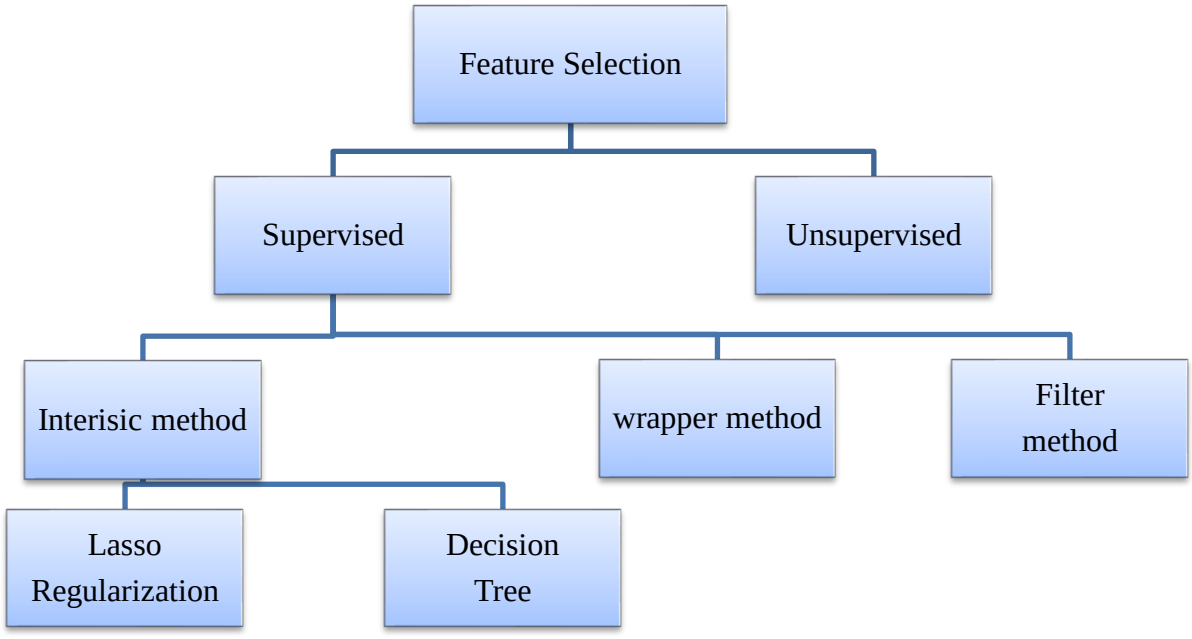


Figure 4.11 ML feature selection methods

CHAPTER FIVE

5. RESULTS AND DISCUSSIONS

5.1. Results of the Study

This section presents an overview of this study, the evaluation of models and results of the research. First, it provides the result of this research in the form of a discussion. Then, the research questions raised in section 1.3 are discussed based on the result of the study. The analyses of results are presented in tabular form and in figures or pictorial presentation. The results are interpreted in detail. The original work or contributions by the research are provided in this chapter. Moreover, the communicative accuracy from the analysis result is also presented. . Finally, it reflects the evaluation process, which emerged during this study.

5.1.2. Feature Selection

In this study, features selected by using the feature importance approach, which is a bagged decision tree, can be used to determine the features' importance approximately. The notion "feature importance" describes methods that rate input features according to how well they are able to predict a given target variable. Examples of the many different kinds and sources of feature significance scores include statistical correlation scores, coefficients generated as part of linear models, decision trees, and permutation importance scores. In a predictive modeling project, feature importance scores are crucial because they offer insight into the data, insight into the model, and the establishment for dimensionality reduction and feature selection, which can increase a predictive model's efficiency and effectiveness in solving the problem. To select more important feature from the whole feature given by the column the Decision tree feature importance selection approach is used, Importance scores of the features are offered by Decision tree algorithms like CART based on the reduction in the criterion used to select split points, like Gini or entropy. The amount of features chosen for feature engineering is based on Figure 5.1. The chosen features are shown together with their feature importance method of the classes determined importance score. These parameters make up the evaluation criteria since they have the greatest influence on training models.

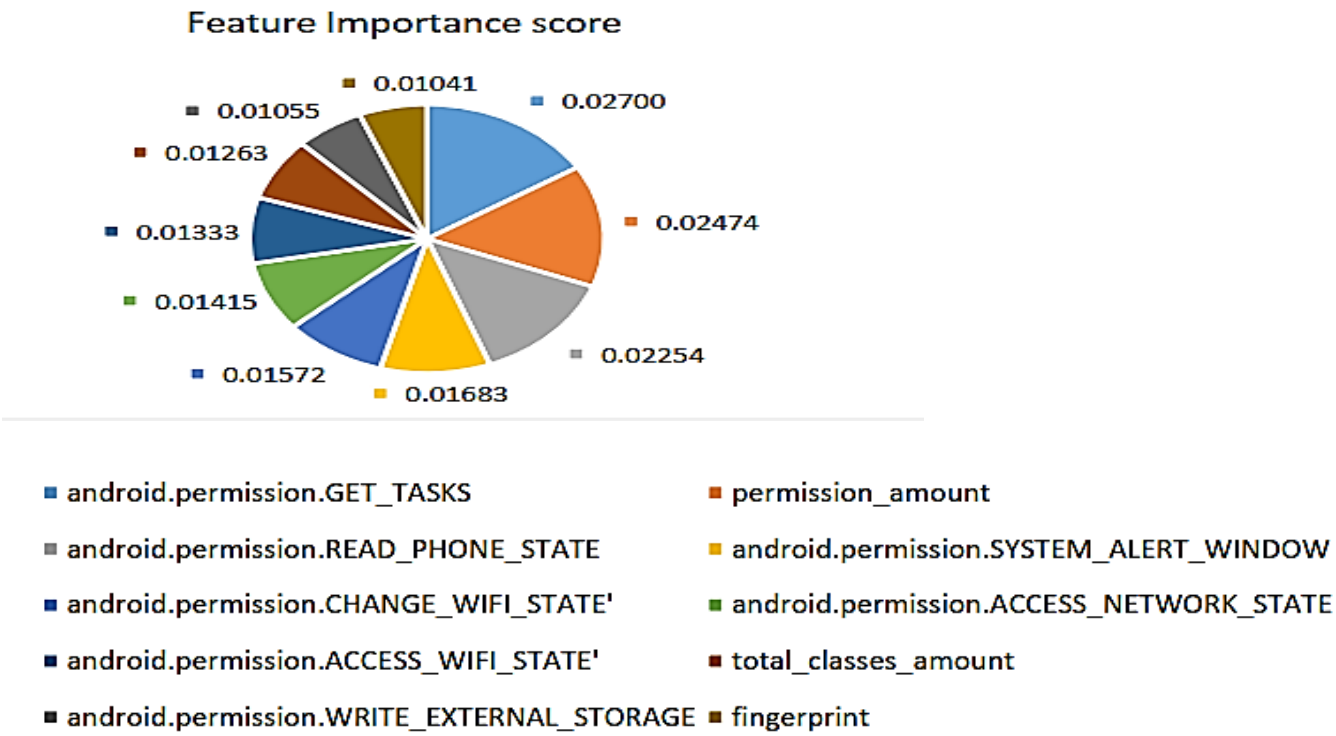


Figure 5.1 Importance of features in apk file

```
Reading the CSV file...
      SEND_DOWNLOAD_COMPLETED_INTENTS  READ_OWNER_DATA  ...  WRITE_LOGS   FAMILY
0                                   0                0  ...           0  Malware
1                                   0                0  ...           0  Malware
2                                   0                0  ...           0  Malware
3                                   0                0  ...           0  Malware
4                                   0                0  ...           0  Malware
...                               ...              ...  ...         ...      ...
2992                                0                0  ...           0   Benign
2993                                0                0  ...           0   Benign
2994                                0                0  ...           0   Benign
2995                                0                0  ...           0   Benign
2996                                0                0  ...           0   Benign

[2997 rows x 266 columns]
```

Figure 5.2 Total datasets rows by columns

	Feature Names	Importances
60	WRITE_CONTACTS	0.409212
19	WRITE_HISTORY_BOOKMARKS	0.209340
128	ACCESS_FINE_LOCATION	0.184408
133	USE_SOCIALSERVICE	0.121079
147	PERSISTENT_ACTIVITY	0.016611
...	...	...
93	MDM_SECURITY	0.000000
94	SCROBBLET_PRIVATE_SERVICE	0.000000
95	ACCESS_LAUNCHER_DATA	0.000000
96	ACCESS_SUPERUSER	0.000000
258	ACCESS_WEATHERCLOCK_PROVIDER	0.000000

Figure 5.3 Ranking of features_importance to select

Index	OWNER	TE CALL	OCIAL DA	TE SETTII	E DEVICE	FAMILY
0	0	0	0	0	0	Malware
1	0	0	0	1	0	Malware
2	0	0	0	0	0	Malware
3	0	0	0	0	0	Malware
4	0	0	0	0	0	Malware
5	0	0	0	1	1	Malware
6	0	0	0	1	1	Malware
7	0	0	0	0	0	Malware
8	0	0	0	1	0	Malware

Index	OWNER	TE CALL	OCIAL DA	TE SETTII	E DEVICE	FAMILY
1847	0	0	0	0	0	Malware
1848	0	0	0	0	0	Malware
1849	0	0	0	1	0	Benign
1850	0	0	0	1	0	Benign
1851	0	0	0	0	0	Benign
1852	0	0	0	0	0	Benign
1853	0	0	0	0	0	Benign
1854	0	0	0	0	0	Benign
1855	0	0	0	0	0	Benign

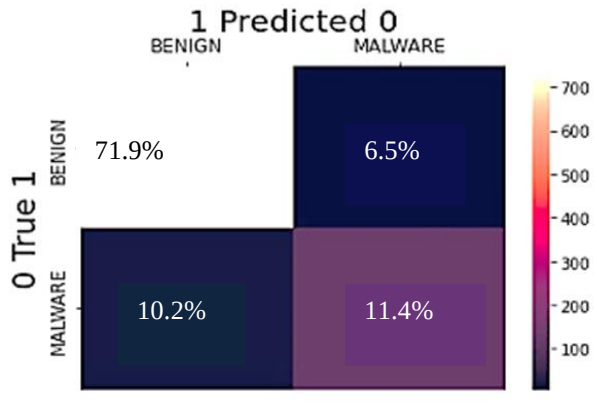
Figure 5.4 DataFrame of the selected Features

During implementation, the numbers of feature are reduced from the given dataset and the new set of features with small size than the given dataset is created. While the number of feature are reduced based on

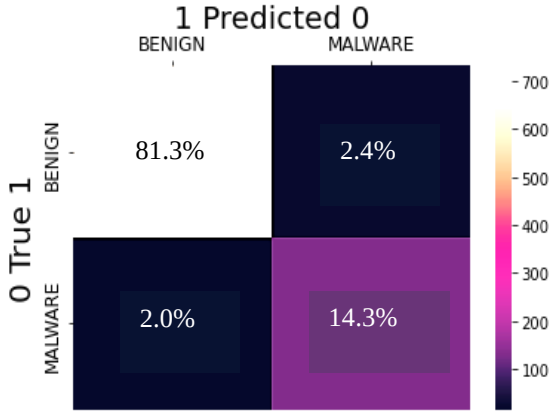
the feature_importance algorithm in python, the score is given to all the features and the feature with high importance kept in the dataset and the feature with low importance is reduced from the dataset. What's important in this case is that the new features should properly summarize the previous characteristics without losing much information. This is a crucial stage, particularly when there are numerous features.

5.2. Comparison and Validation of Models

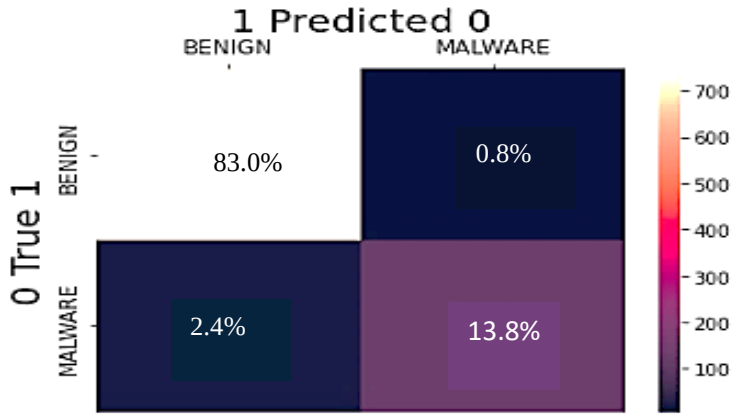
A train and test split is the most basic form of validation technique. A validation technique's goal is to see how your machine learning model responds to new data. All validation techniques will vary slightly but are based on the train and test split. K-fold cross-validation is typically used to assess the effectiveness of machine learning classifiers. The F1 score, which combines precision and recall, is the appropriate statistic for this issue. The dataset is divided ten times into ten separate sets in order to apply k-fold cross-validation with k = 10 for each classifier that has been employed. In this method, we use 30% of the data for testing each time, and 70% of the data for training. Use of stacked cross-validation is a better strategy. On various training, validation, and test folds, parameter optimization is carried out. Compared to the earlier evaluation method, this will lead to increased accuracy. It results from the usage of cross-validation rather than a train/test split. Figure 6 illustrates confusion matrices, which show the outcomes of layered cross-validation. Matrix a), b), and c), which depicts a component method using a Naive Bayes , Decision Tree and a SVM classifier, respectively. In the following matrix 1 represents true and 0 represents false. The four components of confusion matrix true_positive, true_negative, false_positive and false_negative are described in percent (%) as the result of demonstration.



(a) Naïve Bayes



(b) Decision Tree



(c) Support Vector Machine

Figure 5.5 Normalized confusion matrix according to train_test_split

Naïve Bayes, Decision Trees and SVM machine learning algorithms are used for classification. These machine learning algorithms have different detection performance. The algorithm with high classification accuracy is selected to provide a better Android malware detection system. The true positive (TP) value, false positive (FP) value, accuracy, precision, and recall have been calculated. The parameters from *Table 5.1* are used to predict the labels for the apps in the test set X_test and compare the results with the true labels y_test. The results are presented as confusion matrices shown in the depiction of *Figure 5. 1*. Matrix a), b), and c) represents a normalized confusion matrix of the classifiers which holds TP, TN, FP, FN values. The number of apps per category is stated in the cells.

Table 4.1 Performance evaluation of Naïve Bayes classifier

Class	Performance Measures			
	Precision (%)	Recall (%)	F1-score (%)	Support
Benign class	1.00	0.78	0.88	1049
Malware class	0.91	1.00	0.96	1754

Table 5.1 shows the performances of Naïve Bayes classifier is used for the classification of Malware and Benign. As shown above table, the malware class is classified successfully with better performance according to Recall and F1-score metrics of 100% and 96% respectively where the Benign class is 78% and 88% respectively. We recognized that the Benign class (100%) is more precisely classified than Malware (91%).

Table 5.2 Performance evaluation of Decision Tree classifier

Class	Performance Measures			
	Precision (%)	Recall (%)	F1-score (%)	Support
Benign class	0.95	0.92	0.97	1754
Malware class	0.97	0.80	0.96	1049

From the above table (Table 5.2) the precision and F1-score approximately the same by class-wise but in the measurement of Recall, Benign class (92%) performs better than Malware class (80%). As whole this Decision tree classifier performance is do well in Benign class with the metrics of Precision (95%) and F1-score (97%).

Table 5.3 Performance evaluation of SVM classifier

Class	Performance Measures			
	Precision (%)	Recall (%)	F1-score (%)	Support
Benign class	0.97	0.99	0.98	1754
Malware class	0.95	0.85	0.90	1049

From the above table we have seen that how the Support Vector Machine (SVM) performance looks like in the Malware and Benign classification. As shown above Benign class has been classified with high performance than Malware with the measures of precision of 97%, Recall of 99% and F1-score of 98%. In all measurement of SVM classifier, Malware classification is less accurate than Benign.

Table 5.4 Summarized performance of classifiers with selected features

Excution Time	Numbers of apk file		Number of features		Accuracy (%)		
	Benign	Malware	Total	Selected	NB	DT	SVM
First	1789	709	145	12	87%	82.8%	89%
Second	1602	1300	145	12	89.7%	86%	93%
Third	2047	916	145	12	92%	91%	95%
Fourth	1200	1603	145	12	93%	95%	97%

According to Table 5.4 the machine modeled four times with three algorithms and it perform better in the 4th round as it trained many times with different apk file. Accuracy of classifying Benign and Malware is

improved as we took the previous research as a benchmark that achieve accuracy of 93%. As seen in the table above the SVM achieve the prediction accuracy up to 97% and this work contribute to reduce the prediction deviation of Android malware detection technique.

5.2.3. Analysis of Module Evaluations

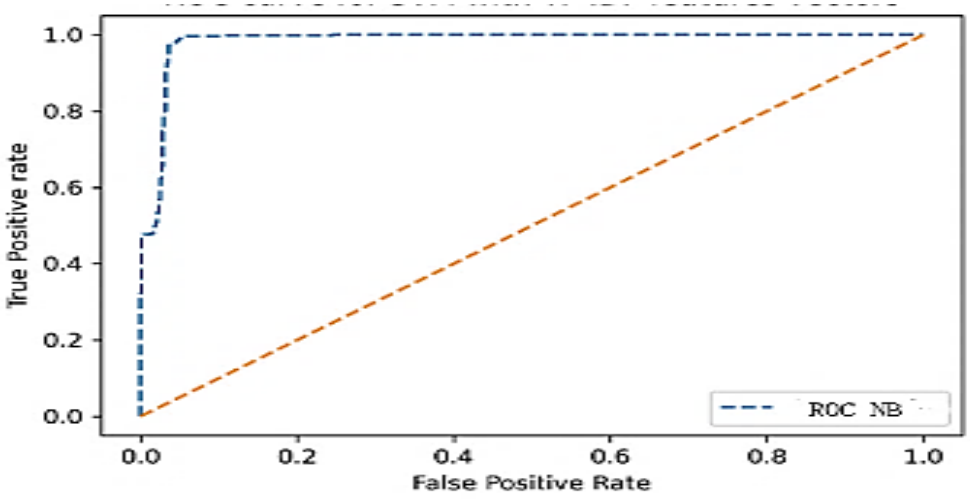


Figure 5.6 ROC curve for Naïve Bayes

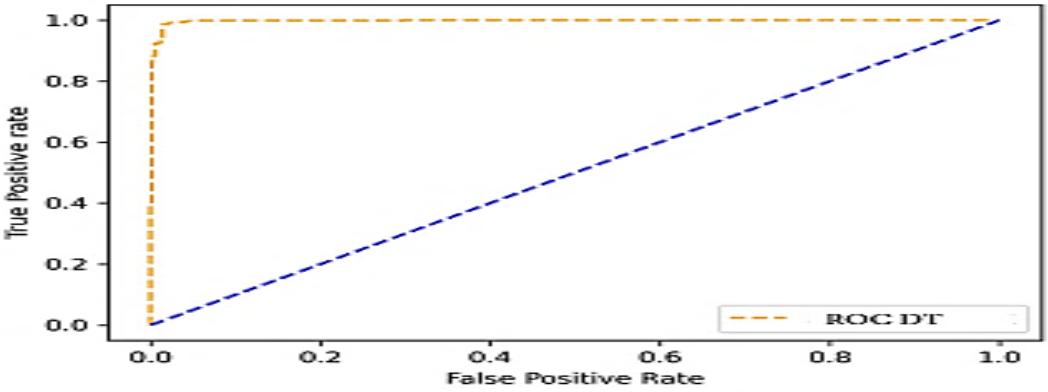


Figure 5.7 ROC curve for Decision tree

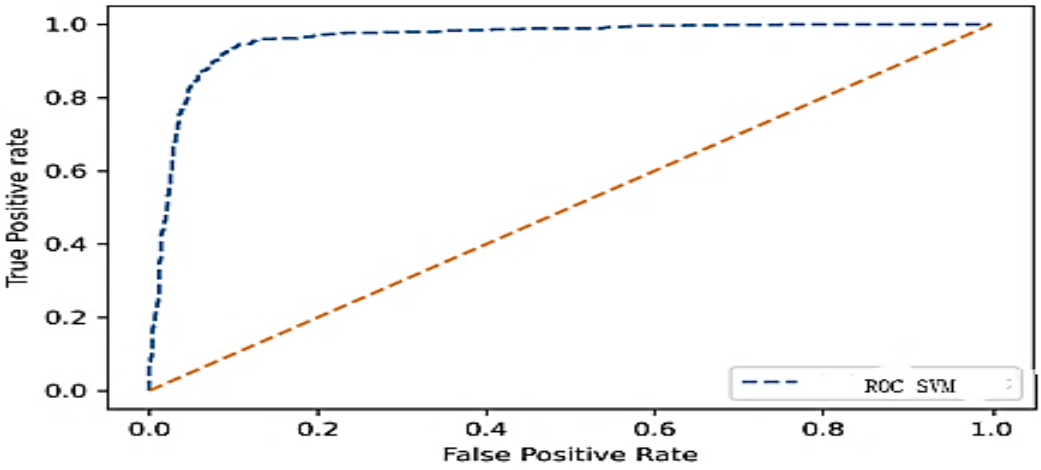


Figure 5.8 ROC curve for SVM

5.3. Discussions

The research results and the result's interpretation will be detailed in this part. The limits of the study and their implications for future research are discussed, which also helps to highlight the importance of this work. A respectable interpretation and discussion should come after the explanation of the implementation. Confusion matrices are produced by the first evaluation method, and accuracy values by the second. The data are divided into 70% training data and 30% test data using the train/test split. Training sets X train and y train for labels are created from the training data. Test sets X test and y test make up the test data. The scikit-learn method test train split from sklearn.model selection is utilized for the execution. There is no need to evaluate a specific Android application by another node once it has already been examined by that node. Using the train/test split, the data are split into 70% training data and 30% test data. The training data is used to build the training sets X train and Y train for labels. The test data are made up of the test sets X test and y test. The execution uses the scikit-learn method test train split from sklearn.model selection.

To answer the RQ1, the machine learning classification algorithm is implemented with various features of Android malware using enhanced Decision tree classifier and implemented in three ML algorithms. Moreover, Decision Tree, Naïve Bayes and SVM algorithms are compared to find best performance and use hybrid analysis methods for extracting more important features to provide classification for achieving high accuracy and robustness. The algorithms are implemented on the selected features with different running time and different numbers of apk and features, and the SVM classifier performed with better accuracy than others that is up to 97%. As discussed in **Table 5.4** to get more improved accuracy we have run the model

in different execution times with different numbers of apk file and numbers of features that helped us to find better performed algorithm. The SVM algorithm worked better in android malware detection of the selected features as shown in the **Table 5.4** above and this solve the RQ2. The feature reduction technique supported us well for the enhancement of the performance as shown in feature selection (**Figure 5.3**). In our framework, the features are not reduced randomly it is based the importance score that it have and that was the gap in many researches, this answers RQ3.

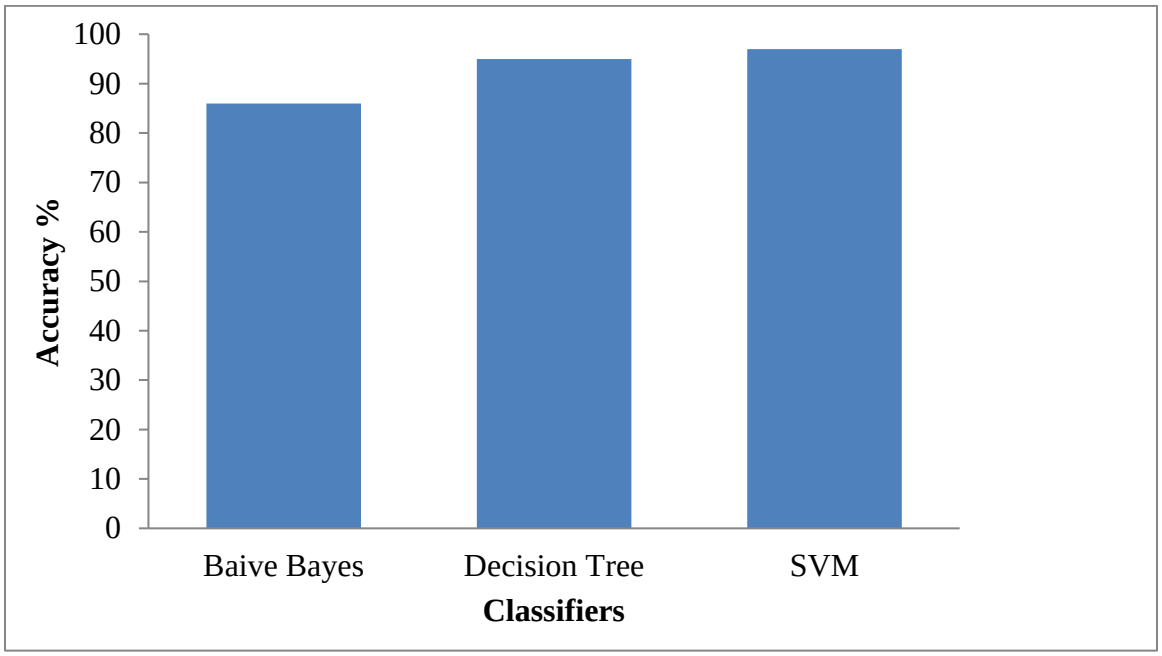


Figure 5.9 Machine Learning classifiers accuracy

CHAPTER SIX

6. CONCLUSIONS AND RECOMMENDATIONS

6.1. Conclusions

Understanding how the android package (APK) compiled, decompiled and developed in IDE environment are the critical steps to due Android-based findings like this research. Android application is user-friendly and flexible package that emerges with new technology and getting people attention by simplifying their make daily life easy. According to secure list of Kaspersky statistics, New version of detected every day to attacks the Android digital world (**by Kaspersky statistics**). This makes working on Android malware detection crucial. There are a frameworks done like, **Kumar *et al.* (2019)** and **Li *et al.* (2017)** to detect and extract risky permissions using different techniques from a common Android components. In this study top risky permissions, API call and Intents are identified from common Android components and used as input to feature selection (**Figure 4.3 & Figure 4.4**). Indeed this feature reduction technique supported us well for the enhancement of the performance as shown in feature selection (**Figure 5.3**). In our framework, the features are not reduced randomly it is based the importance score that it have and that was the gap in many researches.

A framework to detect and classify android malware based on the features feature importance score or importance score is constructed. The features with low feature importance score are reduced from the datasets and this improved the performance of classifications, the related work by **Z. Wang et al. (2018**) supported us well. The results are discussed with meaningful comparison of the results as explained in the **Table 5.4** Three classifiers are discussed in detail and the model that performs better is chosen. According to this work, three machine learning classifiers are analyzed through hybrid analysis to model the classification. From those the one with high accuracy is applied for Android malware detection.

In the scenario, without further modification, the hybrid analysis should be preferred since it has a better performance and is less error-prone compared to the individual analysis approach. This, in turn, could enhance the prediction accuracy using this approach. In order to answer the central research questions of this study, one can denote that the malware detection for detecting recent Android malware lies approximately 97% detection accuracy. As discussed in **Table 5.4** to get more improved accuracy we have run the model in different execution times with different numbers of apk file and numbers of features that

helped us to find better performed algorithm. The SVM algorithm worked better in android malware detection of the selected features as shown in the **Table 5.4** above. However, sometimes various approaches achieve outstanding precision even that are impossible to imagine like close to 100% accuracy. This might lie in the usage of outdated malware, which could be more comfortable to identify. May be cross-checking the ML model with different datasets and algorithms is necessary to find and fix. Metrics like accuracy, F1-score, precision and recall are conducted and the implementation results are cross-validated with different evaluation techniques and discussed broadly.

The contribution of this study is providing a better way of selecting apk features that reduces model execution time as well as improves the accuracy of detection. The problems stated in research questions handled and the hybrid analysis technique optimizes the algorithms. The new malware would be analyzed and will be detected based on the feature importance score. Therefore, Hybrid analysis prototype detects a malicious application with better performance and minimum computational resources and time.

6.2. Recommendations

In this study three different models are implemented to answer the research questions but still there are area and techniques that are not done due to time and specialization limitation. Firstly, logged internet traffic could be added as a feature, this may help to control the malware that spread over internet connections easily. This could be done through transforming the connection information like addresses and protocols into the feature for Android threat prevention and detection.

Secondly, this detection technique could be directly linked with Android development and app store company and ISP (Ethio-telecom) as well as security company like INSA. This enables us to prevent the malware in addition to detection. Making this linkage happen improves a lot of malware detection-prevention techniques and develops Android Mobile and Android IoT technology advancement.

REFERENCES

Kaspersky Lab (2021). R. Unuchek, “Mobile malware evolution 2021,”. [Online]. available: https://securelist.com/it-threat-evolution-q2-2021-mobile-statistics/103636/ (accessed on : 07 Aug 2021).

SIMON KEMP (2022) DIGITAL 2022: ETHIOPIA https://datareportal.com/reports/digital-2022-ethiopia

N. C. Lê, T. -M. Nguyen, T. Truong et al. (2020) "A Machine Learning Approach for Real Time Android Malware Detection," 2020 RIVF International Conference on Computing and Communication Technologies (RIVF), 2020, pp. 1-6

Y. F. Lu, C. F. Kuo, H. Y. Chen, C. W. Chen, and S. C. Chou (2018), “A SVM-Based Malware Detection Mechanism for Android Devices,” 2018 Int. Conf. Syst. Sci. Eng. ICSSE 2018**,** pp. 1–6, 2018

H. S. Ham and M. J. Choi, “Analysis of Android malware detection performance using machine learning classifiers,” *Int. Conf. ICT Converg.*, pp. 490–495, 2018

Li, Wenjia & Wang, Zi & Cai, Juecong & Cheng, Sihua. (2018). An Android Malware Detection Approach Using Weight-Adjusted Deep Learning. 437-441. 10.1109/ICCNC.2018.8390391.

K. Shibija and R. V. Joseph (2018), “A Machine Learning Approach to the Detection and Analysis of Android Malicious Apps,” *2018 Int. Conf. Comput. Commun. Informatics, ICCCI 2018*, pp. 1–4.2018.

Number of Mobile Phone Users Worldwide from 2016 to 2023 (In Billions). Available online: https://www.statista.com/statistics/ 330695/number-of-smartphone-users-worldwide/ (accessed on: 07 Aug 2021).

Mobile Operating System Market Share Worldwide (2021). Available online: https://gs.statcounter.com/os-market-share/mobile/ worldwide/ (accessed on : 07 Aug 2021).

H. Chen *et al.* (2018), ‘‘Malware collusion attack against SVM: Issues and countermeasures,’’ *Appl. Sci.*, vol. 8, no. 10, p. 1718, Sep. 2018.

I. Dogru and K. Ömer (2018), ‘‘Web-based android malicious software detection and classification system,’’ *Appl. Sci.*, vol. 8, no. 9, p. 1622, Sep. 2018.

G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra (2018), ‘‘Risk analysis of Android applications: A user-centric solution,’’ *Future Gener. Comput. Syst.*, vol. 40, pp. 505–518, Mar. 2018.

W. Li, Z. Wang, J. Cai, and S. Cheng (2018), ‘‘An android malware detection approach using feature importance score-adjusted deep learning,’’ in *Proc. Int. Conf. Comput., Netw. Commun.*, Mar. 2018.

Z. Wang (2017), ‘‘Orientation invariant feature embedding and spatial temporal regularization for vehicle re-identification,’’ in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017.

M. V. Varsha, P. Vinod, and K. A. Dhanya (2017), ‘‘Identification of malicious android APP using manifest and opcode features,’’ *J. Comput. Virol. Hacking Techn.*, vol. 13, no. 2, pp. 125–138, May 2017.

M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu (2017), ‘‘DAPASA: Detecting Android piggybacked apps through sensitive subgraph analysis,’’ *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1772–1785, Aug. 2017

F. Abdurahman, “ADDIS ABABA UNIVERSITY SCHOOL OF GRADUATE STUDIES INSTITUTE OF TECHNOLOGY,” 2014.

A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli (2018), ‘‘MADAM: Effective and efficient behavior based android malware detection and prevention,’’ IEEE Trans. Dependable Secure Comput., vol. 15, no. 1, pp. 83–97, Jan. 2018

R. Kumar et al. (2019), ‘‘Research on data mining of permission-induced risk for android IoT devices,’’ Appl. Sci., vol. 9, no. 2, p. 277, Jan. 2019.

Jingjing Gu, Bnglin sun, Xiaojiang Du (2018), “Consortium Blockchain-Based Malware Detection in Mobile Devices” IEEE Trans. Dependable Secure Comput. Feb. 2018

L. Ricci, F. Maesa, A. Favenza (2021), “Blockchain for COVID-19 Contact Tracing and Vaccine Support” IEEE Trans. Dependable Secure Comput. Mar. 2021

Digital 2021: Ethiopia report from DataReportal and partners Hootsuite and We Are Social, Jan. 2021. [Online]. available: http://www.connectingafrica.com/author.asp?section_id=761&doc_id=768066 (accessed on : 08 Aug 2021).

Omar N. Elayan, Ahmad M. Mustafa (2021), "Android Malware Detection Using Deep Learning" Elsevier B.V, Mar. 2021

Burak Tahtaci, Beyzanur Canbay, "Android Malware Detection Using Machine Learning" IEEE Trans. Dependable Secure Comput Oct. 2020

Y. Li et al. (2017), ''Deep joint discriminative learning for vehicle re-identification and retrieval,'' in *Proc. Int. Conf. Image Process.*, pp. 395–399

B. TAHTACI and B. CANBAY (2020), "Android Malware Detection Using Machine Learning," 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), 2020, pp. 1-6, doi: 10.1109/ASYU50717.2020.9259834.

Y. Youn et al (2018), ''Smart contract-based review system for an IoT data marketplace,'' *Sensors*, vol. 18, no. 10, p. 3577, 2018.

R. Stojkoska and K. V. Trivodaliev (2017), ''A review of Internet of Things for smart home: Challenges and solutions,'' *J. Cleaner Prod.*, vol. 140, no. 3, pp. 1454–1464, 2017

A. Kumar, K. S. Kuppusamy, and G. Aghila, ''FAMOUS: Forensic analysis of MObile devices using scoring of application permissions,'' *Future Gener. Comput. Syst.*, vol. 83, pp. 158–172, Jun. 2018.

A. Ferrante, M. Malek, F. Martinelli, F. Mercaldo, and J. Milosevic, ''Extinguishing ransomware—A hybrid approach to android ransomware detection,'' in *Proc. Int. Symp. Found. Practice Secur.*, 2018

APPENDIXES

Appendix A: Sample Android and python source code for feature extraction

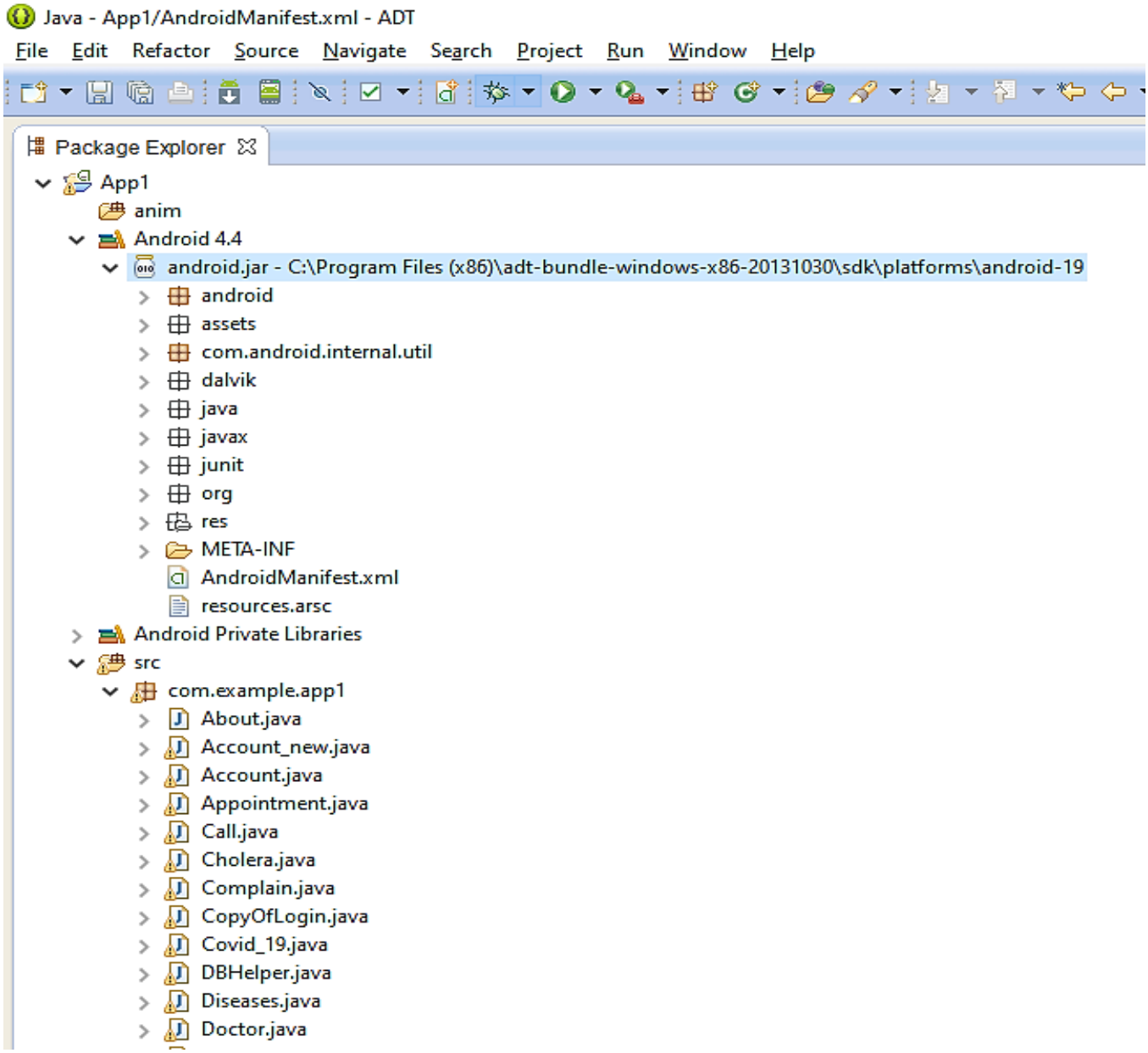


Figure A.1 Android Package explore for APK analysis

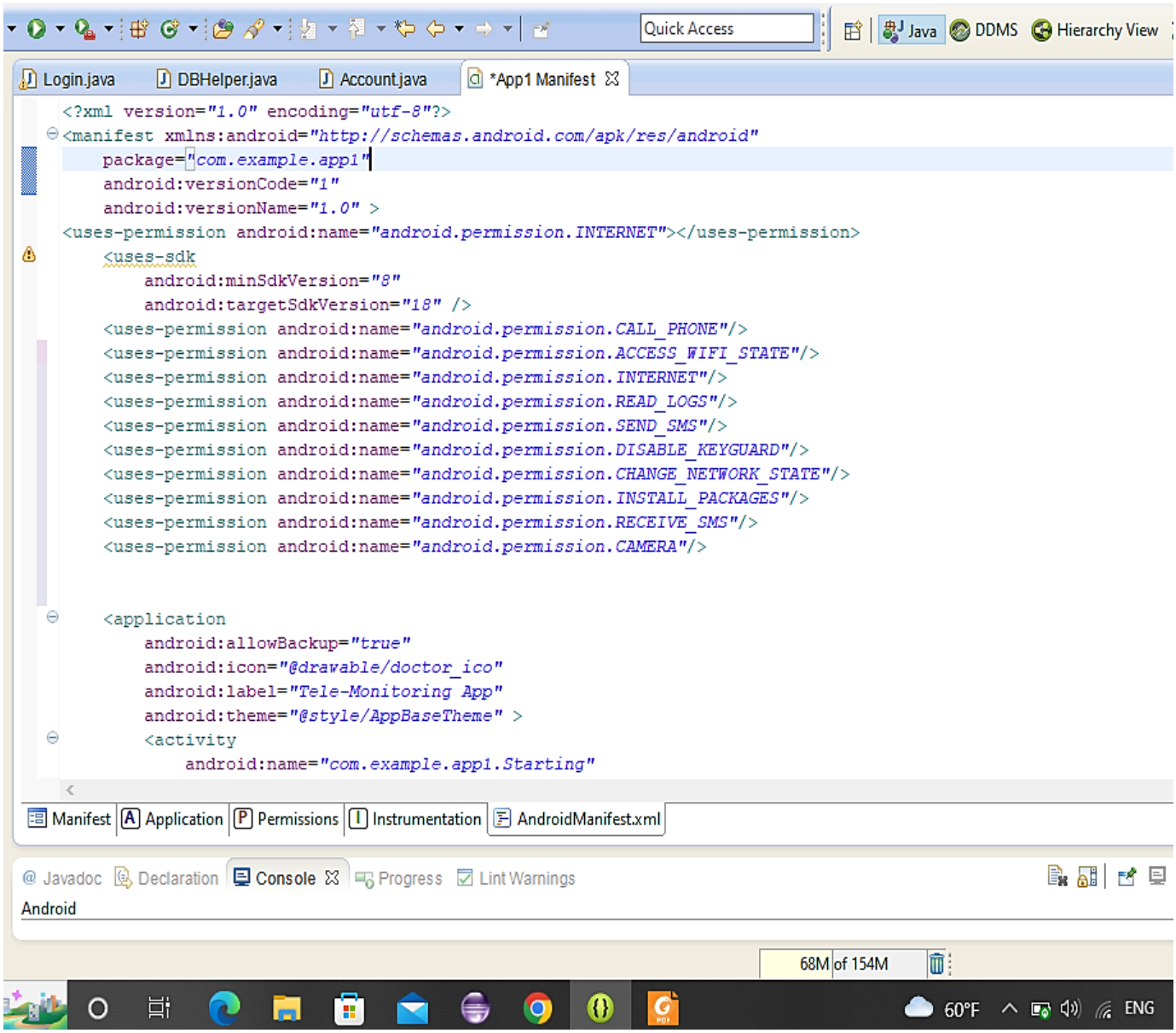


Figure A.2 Android permissions package for static analysis

```
private void processApkFile(File file) {
    List<Report> reports = new ArrayList<>();
    try {
        String apkHash = FileHasher.hashFile(file, FileHasher.SHA256);
        String checkForVirusTotalReport = apkHash;
        JSONObject response = db.receiveDocument(checkForVirusTotalReport, config.getElasticIndex()
        if (hasReport(response)) {
            LOG.info("The report with key {} is already exist.", apkHash);
            startAnalyzers(file, reports);
            updateReports(reports, apkHash);
        } else {
            startAnalyzers(file, reports);
            storeReports(reports, apkHash);
        }
    } catch (Exception e) {
        LOG.error("Could not parse APK: {} ", file.getName());
    }
}
```

Figure A.3 Sample Source code for static analysis

```
public abstract class DynamicAnalyzer {
    static final int INITIAL_LAUNCH_WAITING_TIME = 6000;

    Logger LOG;
    AdbAdapter aa;
    ClickRunner clickRunner;

    public abstract void analyze(App app, int reboots);

    public abstract Report getReport();

    void initializeAnalysis(String packageName) {
        LOG.info("Stop app {} and delete all logs.", packageName);
        aa.stopApp(packageName);
        //aa.deleteAllLogs();
        LOG.info("Check for unlocking of device.");
        aa.unlockDevice();
        LOG.info("Start initial launch of app.");
        aa.launchApp(packageName);
    }

    void fixEmptyApiLog() {
        aa.rebootDevice();
        if (aa.isDeviceReady()) {
            aa.unlockDevice();
        }
    }

    void checkAndReboot(App app, int reboots, String packageName) {
        if (app.getApiCalls().isEmpty() && reboots == 0) {
            LOG.info("API LOG is empty. Rebooting the device...");
            fixEmptyApiLog();
            reboots++;
            LOG.info("Device is rebooted. Try app {} again.", packageName);
            analyze(app, reboots);
        }
    }
```

Figure A.4 Sample source code dynamic analysis

```
import AnalaysisLayer.ElasticsearchDb.Database;
import AnalaysisLayer.ElasticsearchDb.ElasticAdapter;
import AnalaysisLayer.dynamic_analysis.DynamicAnalysis;
import AnalaysisLayer.static_analysis.StaticAnalysis;
import org.python.util.PythonInterpreter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

final public class AnalysisMain {
    private static final Logger LOG = LoggerFactory.getLogger(AnalysisMain.class);
    public static void main(String[] args) {
        AndroMLBlockchainConfig config = new AndroMLBlockchainConfig();
        Database db = new ElasticAdapter(config);
        initializeCoreIndex(config, db);
        StaticAnalysis sa = new StaticAnalysis(config, db);
        sa.checkForStartingAnalysis();
        DynamicAnalysis da = new DynamicAnalysis(config, db);
        da.checkForStartingAnalysis();
        //Execute Python code**************************
        PythonInterpreter interp = new PythonInterpreter();
        interp.execfile( filename: "src/main/MLClassificationModule/classify.py");
    }

    public static void initializeCoreIndex(AndroMLBlockchainConfig config, Database db) {
        if (!db.isDatabasePresent(config.getElasticIndex())) {
            db.createDatabase(config.getElasticIndex());
        }
    }
```

Figure A.5 Sample source code for hybrid analysis

Appendix B: Sample python source code for feature selection

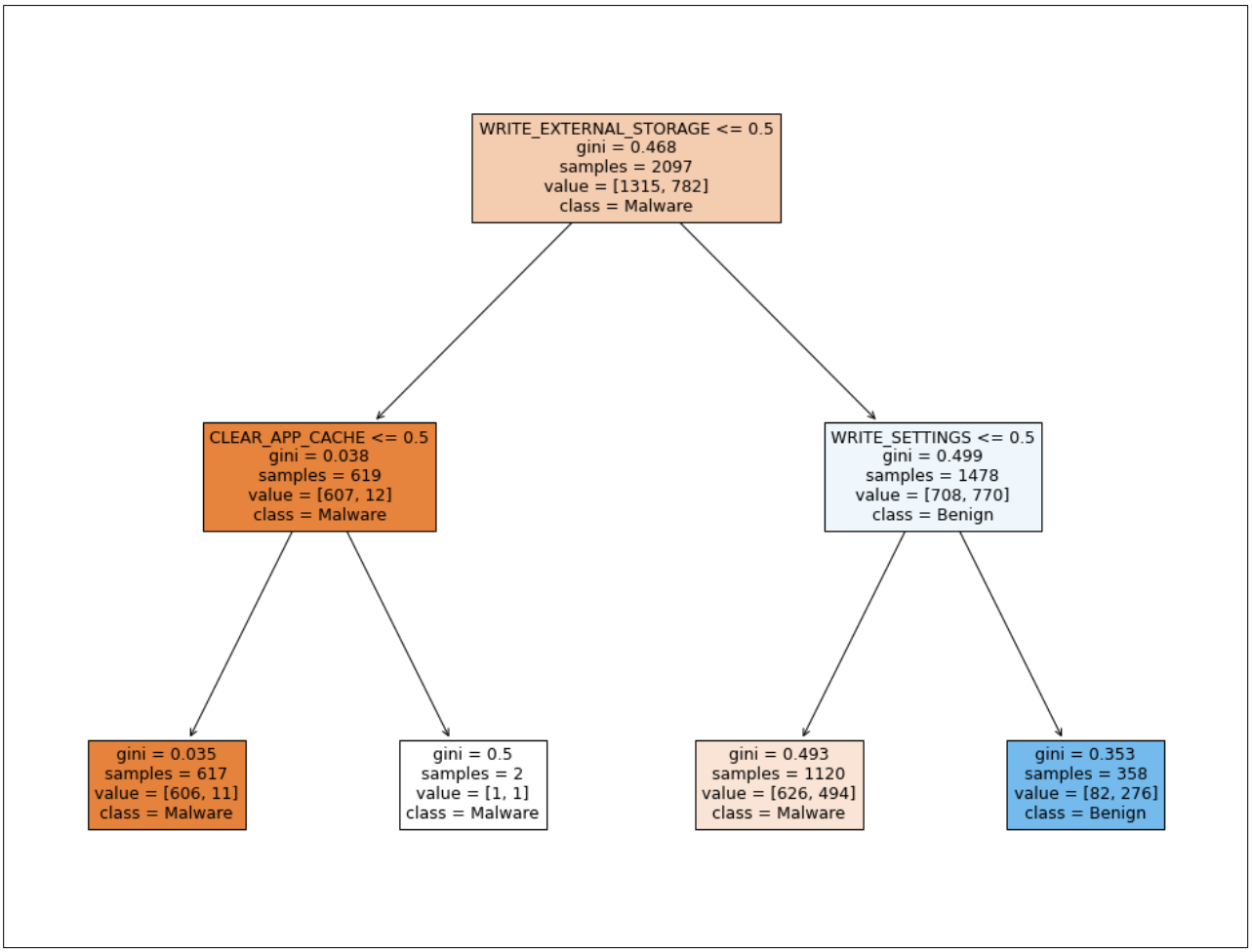


Figure B.1 Visualize decision tree for feature selection

```
In [30]: from sklearn.tree import export_graphviz
         from sklearn.tree import DecisionTreeClassifier
         clf=DecisionTreeClassifier(random_state=6, max_depth=2)
         clf.fit(x_train, y_train)

Out[30]: DecisionTreeClassifier(max_depth=2, random_state=6)

In [31]: data.columns

Out[31]: Index(['CLEAR_APP_CACHE', 'WRITE_SETTINGS', 'GET_TASKS', 'READ_PHONE_STATE',
                'READ_SMS', 'SEND_RESPOND_VIA_MESSAGE', 'WRITE_EXTERNAL_STORAGE',
                'READ_PHONE_STATE', 'INSTALL_PACKAGES'],
               dtype='object')

In [32]: fig=plt.figure(figsize=(10,12))
         _=tree.plot_tree(clf, feature_names=data.columns, filled=True, class_names=['Malware','Benign'])
         plt.show()
```

Figure B.2 Source code to visualize tree #Jupter

Out[290]:

	Feature Names	Importances
60	WRITE_CONTACTS	0.409212
19	WRITE_HISTORY_BOOKMARKS	0.209340
128	ACCESS_FINE_LOCATION	0.184408
133	USE_SOCIALSERVICE	0.121079
147	PERSISTENT_ACTIVITY	0.016611
...	...	...
93	MDM_SECURITY	0.000000
94	SCROBBLET_PRIVATE_SERVICE	0.000000
95	ACCESS_LAUNCHER_DATA	0.000000
96	ACCESS_SUPERUSER	0.000000
258	ACCESS_WEATHERCLOCK_PROVIDER	0.000000

259 rows × 2 columns

Figure B.3 Feature importance score

```
In [11]: import pandas as pd
         from sklearn.datasets import load_iris
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.tree import export_graphviz
         from sklearn.feature_selection import mutual_info_classif
         data=pd.read_csv("E:\Anaconda Python\BenignMalware.csv")
         data = data[['CLEAR_APP_CACHE', 'WRITE_SETTINGS','GET_TASKS','READ_PHONE_STATE','READ_SMS','SEND_RESPOND_VIA_MESSAGE','WRITE_EXTE

         # Rename columns
         #myData.columns = ['Feature-1', 'Feature-2','Feature-3','Feature-4','Feature-5','Feature-6','Feature-7','APP FAMILY']
         #Features and target value
         x=data.drop(columns='INSTALL_PACKAGES')
         y=data['INSTALL_PACKAGES']
         target_name=y
         #split_train_test
         from sklearn.model_selection import train_test_split
         x_train, x_test,y_train,y_test=train_test_split(x,y,test_size=0.3, random_state=20)

         clf=DecisionTreeClassifier(random_state=6, max_depth=2)
         clf.fit(x_train, y_train)
         clf.feature_importances_
```

Figure B.4 Source code for Feature importance score #Jupter

Appendix C: Sample python source code for model evaluation

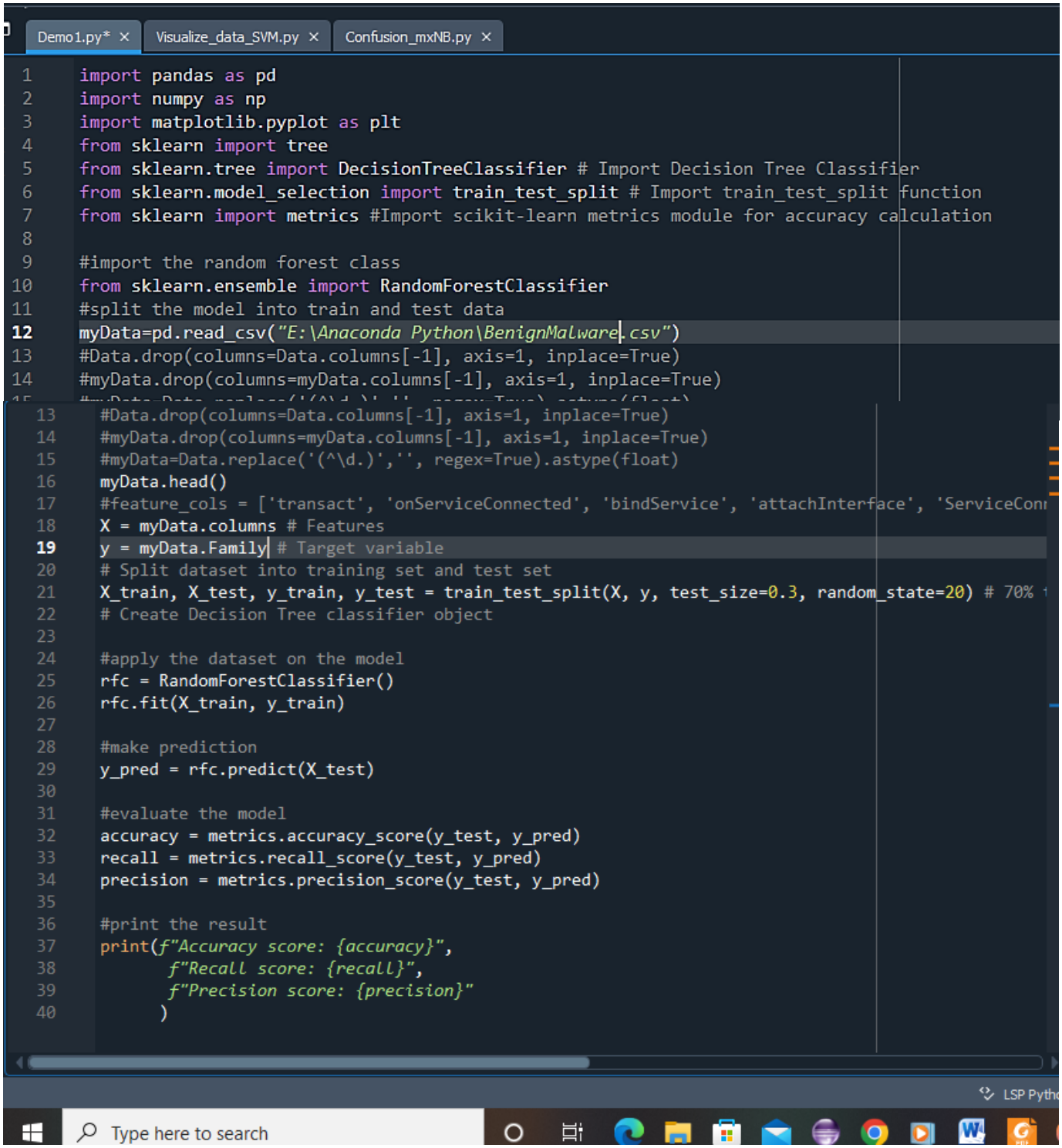


Figure C.1 Python source code for model evaluation using #Spider

```python
# Imports
import numpy as np
import pandas as pd
import os
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
import matplotlib.cbook
warnings.filterwarnings("ignore",category=matplotlib.cbook.mplDeprecation)


# Load Dataset
# Dataset can be found at: https://www.kaggle.com/uciml/sms-spam-collection-dataset

myData = pd.read_csv('E:\Anaconda Python\BenignMalware.csv', encoding = 'latin-1' )

# Keep only necessary columns
myData = myData[['WRITE_SETTINGS', 'UPDATE_DEVICE_STATS','GET_TASKS','READ_PHONE_STATE','READ_SE(

# Rename columns
myData.columns = ['Feature-1', 'Feature-2','Feature-3','Feature-4','Feature-5','Feature-6','Feat(

```

```python
X = myData # Features
y = myData['APP FAMILY'] # Target variable
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
mnb = MultinomialNB()


# Train the classifier/Fit the model
mnb.fit(X_train, y_train)

# Make predictions
y_pred = mnb.predict(X_test)

# Build confusion metrics
cm = confusion_matrix(y_true=y_test, y_pred=y_pred)
```

```
53    ax.set_ylabel('True', fontsize=20)
54    ax.yaxis.set_ticklabels(['BENIGN', 'MALWARE'], fontsize = 12)
55    plt.show()
56
57    from sklearn.metrics import classification_report
58
59    # printing the report
60    print(classification_report(y_test, y_pred))
```

Figure C.2 Python source code for confusion matrix

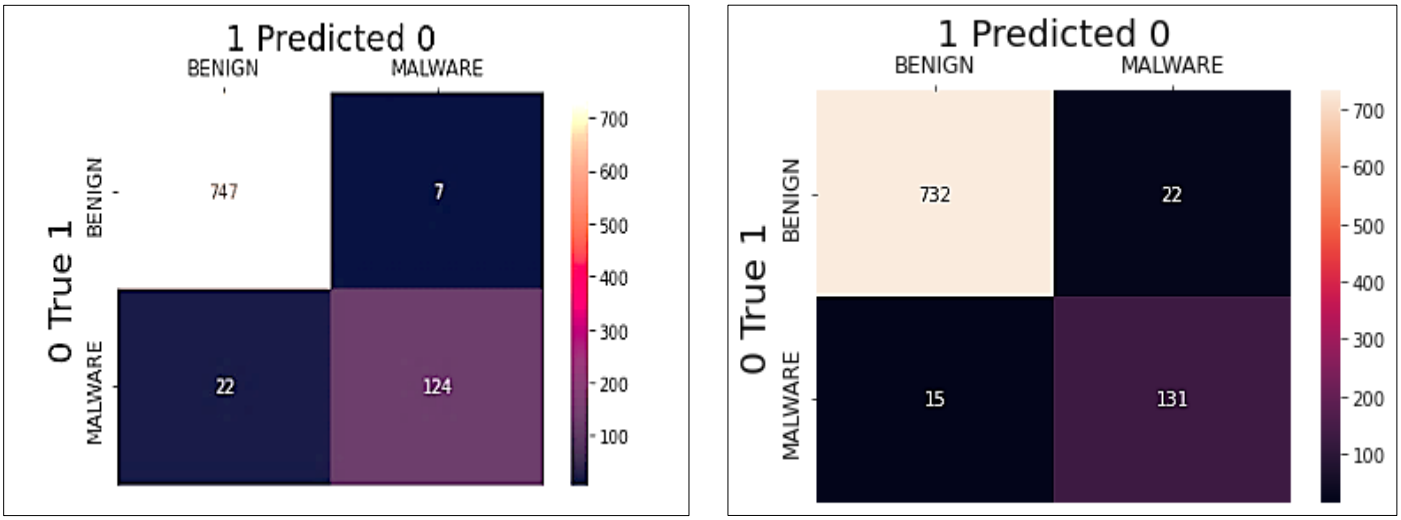


Figure C.3 Confusion matrix

Appendix D: Proposed Algorithms and UI

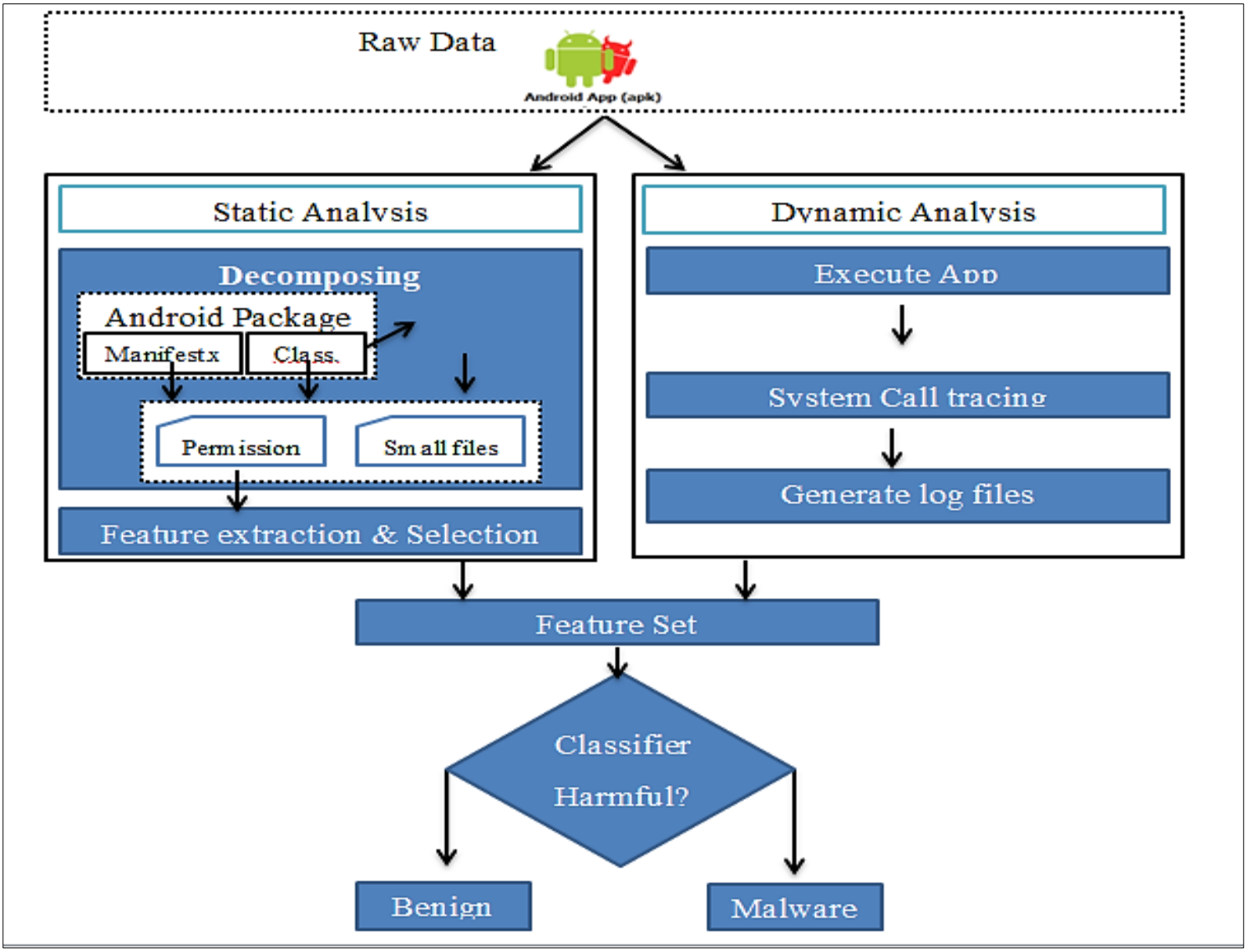


Figure D.1 Proposed hybrid analysis algorithm