Montclair State University
College of Science and Mathematics Department of
Computer Science
Master's Project Report
Spring 2024

Detection of Android Malware Using Machine Learning and Siamese Shot Learning Technique for Security

Professor
Dr. Daeyoung Kim
Department of Computer Science
Montclair State University Montclair, New Jersey

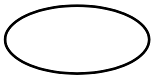
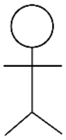
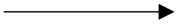
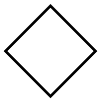
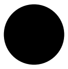
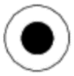
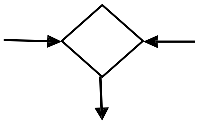
Table of Contents

ABSTRACT

This research proposes a novel framework for Android malware detection that combines permission analysis with machine learning. The framework utilizes multiple linear regression techniques to analyze the permissions requested by Android applications during a static analysis process. This analysis extracts critical security insights that serve as the foundation for malware identification. Data for the model is meticulously preprocessed from established repositories to remove noise and ensure compatibility with machine learning algorithms. The data is then split for training and testing purposes, allowing for robust model evaluation. Furthermore, feature selection techniques like Principal Component Analysis (PCA) are implemented to enhance the model's efficiency and interpretability. The core of the framework lies in the classification phase, where both linear regression and Multilayer Perceptron (MLP) algorithms are tested and compared across diverse datasets. Interestingly, the research finds that linear regression models achieve superior performance without the need for more complex algorithms. Finally, the study presents conclusive results evaluated through various performance metrics, with accuracy being a key measure of the models' ability to accurately predict malicious apps.

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Name
	Use Case
	Actor
	Control Flow
	Decision
	Start
	Stop
	Merge

1. INTRODUCTION

In the ever-growing world of mobile technology, Android devices have become a prime target for malicious software (malware). These programs, ranging from viruses and spyware to Trojans and ransomware, can infiltrate devices, steal sensitive data, disrupt functionalities, and cause significant financial harm. Traditional signature-based detection methods, which rely on identifying known malware patterns, struggle to keep pace with the evolving tactics of malware creators who employ techniques like encryption and obfuscation to evade detection. As a result, the field of cybersecurity is increasingly exploring intelligent detection techniques based on machine learning to combat this ever-present threat.

This research investigates a novel approach to Android malware detection using machine learning. The proposed framework, LinRegDroid, leverages the power of permission analysis and multiple linear regression models. App permissions, which dictate the resources an app can access, serve as a critical indicator of potential malicious behavior. LinRegDroid employs static analysis to examine the permissions requested by Android applications and extracts crucial security insights. These insights form the foundation for a machine learning model built upon multiple linear regression techniques.

The data used to train and test the model is meticulously preprocessed to ensure compatibility with machine learning algorithms. Feature selection techniques like Principal Component Analysis (PCA) are further employed to enhance the model's efficiency and interpretability. The core of LinRegDroid lies in the classification phase, where the efficacy of two distinct algorithms – linear regression and Multilayer Perceptron (MLP) – is evaluated.

The research compares the performance of these models across diverse datasets, with a focus on highlighting the potential of linear regression for achieving remarkable performance without the need for more complex classification algorithms.

Finally, the study presents conclusive results evaluated through various performance metrics, with accuracy being a key measure of the models' ability to accurately predict malicious apps. Overall, LinRegDroid offers a promising new direction for bolstering Android security by leveraging app permissions as a reliable indicator of malware presence. This framework demonstrates the potential of applying a relatively simple yet powerful machine learning technique like linear regression to address the growing challenge of Android malware detection.

2. LITERATURE SURVEY

2.1. AMalNet: A deep learning framework based on graph convolutional networks for malware detection.

The increasing popularity of Android apps attracted widespread attention from malware authors. Traditional malware detection systems suffer from some shortcomings; computationally expensive, insufficient performance or not robust enough. To address this challenge, we build a novel and highly reliable deep learning framework, named AMalNet, to learn multiple embedding representations for Android malware detection and family attribution, introduce a version of Graph Convolutional Networks for modelling high-level graphical semantics, which automatically identifies and learns the semantic and sequential patterns, use an Independently Recurrent Neural Network to decode the deep semantic information, making full use of remote dependent information between nodes to independently extract features. The experimental results on multiple benchmark datasets indicated that the AMalNet framework outperforms other state-of-the-art techniques significantly.

Advantages:

- It compared to its predecessors is that it automatically detects the important features without any human supervision.

Disadvantages:

- Complexity is high.

2.2. DL4MD: A Deep Learning Framework for Intelligent Malware Detection

In this paper, based on the Windows Application Programming Interface (API) calls extracted from the Portable Executable (PE) files, we study how a deep learning architecture using the stacked Auto Encoders (SAEs) model can be designed for intelligent malware detection. The SAEs model performs as a greedy layer wise training operation for unsupervised feature learning, followed by supervised parameter fine-tuning (e.g., weights and offset vectors). To the best of our knowledge, this is the first work that deep learning using the SAEs model based on Windows API calls is investigated in malware detection for real industrial application.

Advantages:

- Deep learning is feasible to learn higher level concepts based on the local feature representations.

Disadvantages:

- It occurred low accuracy.
- Prediction of malware is improper.

2.3. A multi-level deep learning system for malware detection.

To defend against an increasing number of sophisticated malware attacks, deep learning-based Malware Detection Systems (MDSs) have become a vital component of our economic and national security. Traditionally, researchers build the single deep learning model using the entire dataset. However, the single deep learning model may not handle the increasingly complex malware data distributions effectively since different sample subspaces representing a group of similar malwares may have unique data distribution. Experimental results show that our proposed system performs better than the traditional approach.

Advantages:

- Deep learning performs better than shallow learning models for malware detection.
- A single deep learning model has difficulty to capture complex malware distribution.
- Multi-level deep learning can capture complex malware data distribution effectively.

Disadvantages:

- Time consumption is low.

2.4. Towards Efficient Malware Detection and Classification using Multi-layered Random Forest Ensemble Technique

This paper proposes a hybrid stacked multi-layered ensembling approach which is robust and efficient than deep learning models. The proposed model outperforms the machine learning and deep learning models with an accuracy of 98.91%. The proposed system works well for small-scale and large-scale data since its adaptive nature of setting parameters (number of sequential levels) automatically. It is computationally efficient in terms of resources and time. The method uses very fewer hyper-parameters compared to deep neural network.

Advantages:

- The proposed ensembling method shows a higher accuracy of 98.91%.

Disadvantages:

- Encoder to predict the malware is low.

2.5. Existing System

Evaluates the classical MLAs and deep learning architectures for malware detection, classification, and categorization using different public and private datasets. And major contribution is in proposing a novel image processing technique with optimal parameters for MLAs and deep learning architectures to arrive at an effective zero-day malware detection model. Overall, this paper paves way for an effective visual detection of malware using a scalable and hybrid deep learning framework for real-time deployments.

Disadvantages

- Lower learning rate was found to be good in identifying the executable as either benign or malware.
- The performance is considerably very low.
- It doesn't efficient for large volume of data's
- Theoretical limits

2.6. Proposed System

In this system, malware dataset as input was taken from dataset repository like UCI repository. Then, we have to implement the feature selection for selecting the best features from the spitted data. Then, we have to implement the classification algorithm (i.e.) machine learning such as MLP and Linear Regression. Finally, the experimental results shows that the performance metrics such as accuracy, precision, and recall.

Advantages:

- The prediction results are efficient.
- To classify the result effectively.
- Time consumption is low.
- It can handle packed malware and can work on various malwares irrespective of the operating system.

3. SOFTWARE DESCRIPTION

3.1. Python and its Specification

Python is one of those rare languages which can claim to be both simple and powerful. You will find yourself pleasantly surprised to see how easy it is to concentrate on the solution to the problem rather than the syntax and structure of the language you are programming in. The official introduction to Python is Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. I will discuss most of these features in more detail in the next section.

Features of Python

- **Simple**

Python is a simple and minimalistic language. Reading a good Python program feels almost like reading English, although very strict English! This pseudo-code nature of Python is one of its greatest strengths. It allows you to concentrate on the solution to the problem rather than the language itself.

- **Easy to Learn**

As you will see, Python is extremely easy to get started with. Python has an extraordinarily simple syntax, as already mentioned.

- **Free and Open Source**

Python is an example of a FLOSS (Free/Libre and Open-Source Software). In simple terms, you can freely distribute copies of this software, read its source code, make changes to it, and use pieces of it in new free programs. FLOSS is based on the concept of a community which shares knowledge. This is one of the reasons why Python is so good - it has been created and is constantly improved by a community who just want to see a better Python.

- **High-level Language**

When you write programs in Python, you never need to bother about the low-level details such as managing the memory used by your program, etc.

- **Portable**

Due to its open-source nature, Python has been ported to (i.e. changed to make it work on) many platforms. All your Python programs can work on any of these platforms without requiring any changes at all if you are careful enough to avoid any system-dependent features.

You can use Python on GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE and PocketPC!

You can even use a platform like Kivy to create games for your computer and for iPhone, iPad, and Android.

- **Interpreted**

This requires a bit of explanation.

A program written in a compiled language like C or C++ is converted from the source language i.e. C or C++ into a language that is spoken by your computer (binary code i.e. 0s and 1s) using a compiler with various flags and options. When you run the program, the linker/loader software copies the program from hard disk to memory and starts running it.

Python, on the other hand, does not need compilation to binary. You just run the program directly from the source code. Internally, Python converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it. All this, actually, makes using Python much easier since you don't have to worry about compiling the program, making sure that the proper libraries are linked and loaded, etc. This also makes your Python programs much more portable, since you can just copy your Python program onto another computer, and it just works!

- **Object Oriented**

Python supports procedure-oriented programming as well as object-oriented programming. In procedure-oriented languages, the program is built around procedures or functions which are nothing but reusable pieces of programs. In object-oriented languages, the program is built around objects which combine data and functionality. Python has a very powerful but simplistic way of doing OOP, especially when compared to big languages like C++ or Java.

- **Extensible**

If you need a critical piece of code to run very fast or want to have some piece of algorithm not to be open, you can code that part of your program in C or C++ and then use it from your Python program.

- **Embeddable**

You can embed Python within your C/C++ programs to give scripting capabilities for your program's users.

- **Extensive Libraries**

The Python Standard Library is huge indeed. It can help you do various things involving regular expressions, documentation generation, unit testing, threading, databases, web browsers, CGI, FTP, email, XML, XML-RPC, HTML, WAV files, cryptography, GUI (graphical user interfaces), and other system-dependent stuff. Remember, all this is always available wherever Python is installed. This is called the Batteries Included philosophy of Python.

Besides the standard library, there are various other high-quality libraries which you can find at the Python Package Index.

3.2. Python Libraries

- **Streamlit**: A Python library used for creating interactive web applications for data science and machine learning projects. Streamlit simplifies the process of building web interfaces by allowing developers to write Python scripts that are automatically converted into web apps.
- **Base64**: A standard library in Python used for encoding and decoding binary data into ASCII characters. Base64 encoding is commonly used when transmitting binary data over text-based channels, such as email or URLs.
- **Matplotlib**: A widely used Python library for creating static, interactive, and animated visualizations. Matplotlib provides a comprehensive set of plotting tools for generating charts, graphs, histograms, and other types of plots from data.
- **Pandas**: A powerful data manipulation and analysis library for Python. Pandas provides data structures and functions for efficiently working with structured data, such as tabular data and time series. It is commonly used for tasks like data cleaning, transformation, and exploration.

- **CSV**: A module in Python's standard library used for reading and writing CSV (Comma Separated Values) files. CSV files are commonly used for storing and exchanging tabular data in plain text format.
- **Flask**: A lightweight and flexible web framework for Python used for building web applications. Flask provides tools and libraries for handling HTTP requests, rendering templates, and managing application routes. It is often used for creating RESTful APIs and small to medium-sized web applications.
- **Scikit-learn (sklearn)**: A popular machine learning library for Python that provides simple and efficient tools for data mining and data analysis. Scikit-learn includes a wide range of algorithms for classification, regression, clustering, dimensionality reduction, and model selection.
- **Seaborn**: A statistical data visualization library based on Matplotlib. Seaborn provides a high-level interface for creating informative and visually appealing statistical graphics. It is built on top of Matplotlib and integrates well with Panda's data structures.
- **Numpy**: A fundamental package for scientific computing with Python. Numpy provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. It is widely used in numerical computing and data analysis.
- **Pickle**: A module in Python's standard library used for serializing and deserializing Python objects. Pickle allows objects to be converted into a byte stream that can be stored in a file or transmitted over a network. It is commonly used for saving and loading trained machine learning models.
- **OS**: A module in Python's standard library used for interacting with the operating system. The OS module provides functions for performing

various tasks related to file operations, directory manipulation, process management, and environment variables.

- **From Faker import Faker**: A Python library used for generating fake data for testing and development purposes. Faker provides a wide range of fake data types, including names, addresses, phone numbers, email addresses, and IP addresses. It is useful for creating realistic-looking datasets for testing algorithms and applications.

4. SYSTEM DESIGN

4.1. System Architecture

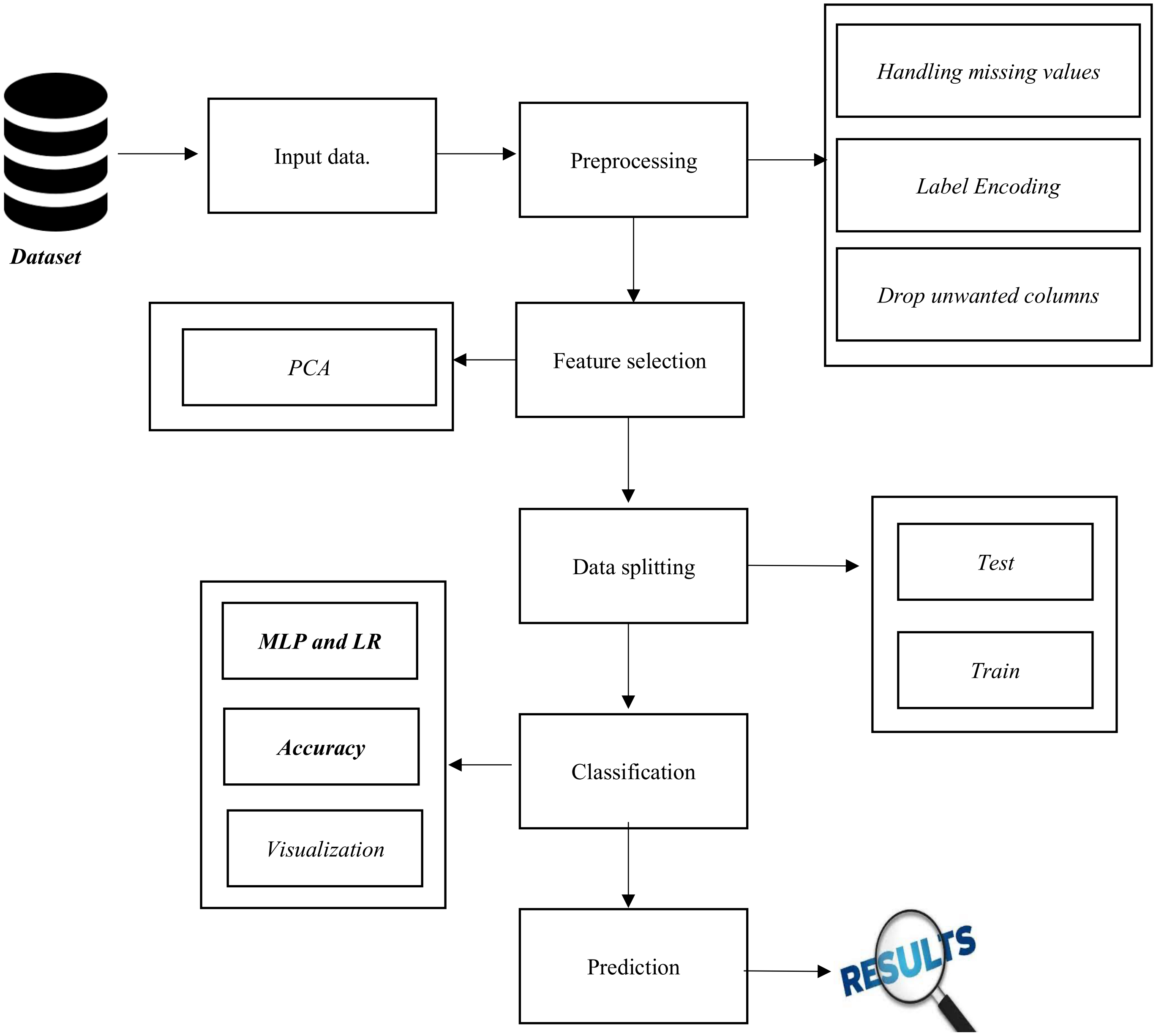


FIGURE 4.1: SYSTEM ARCHITECTURE

4.2. Use Case Diagram

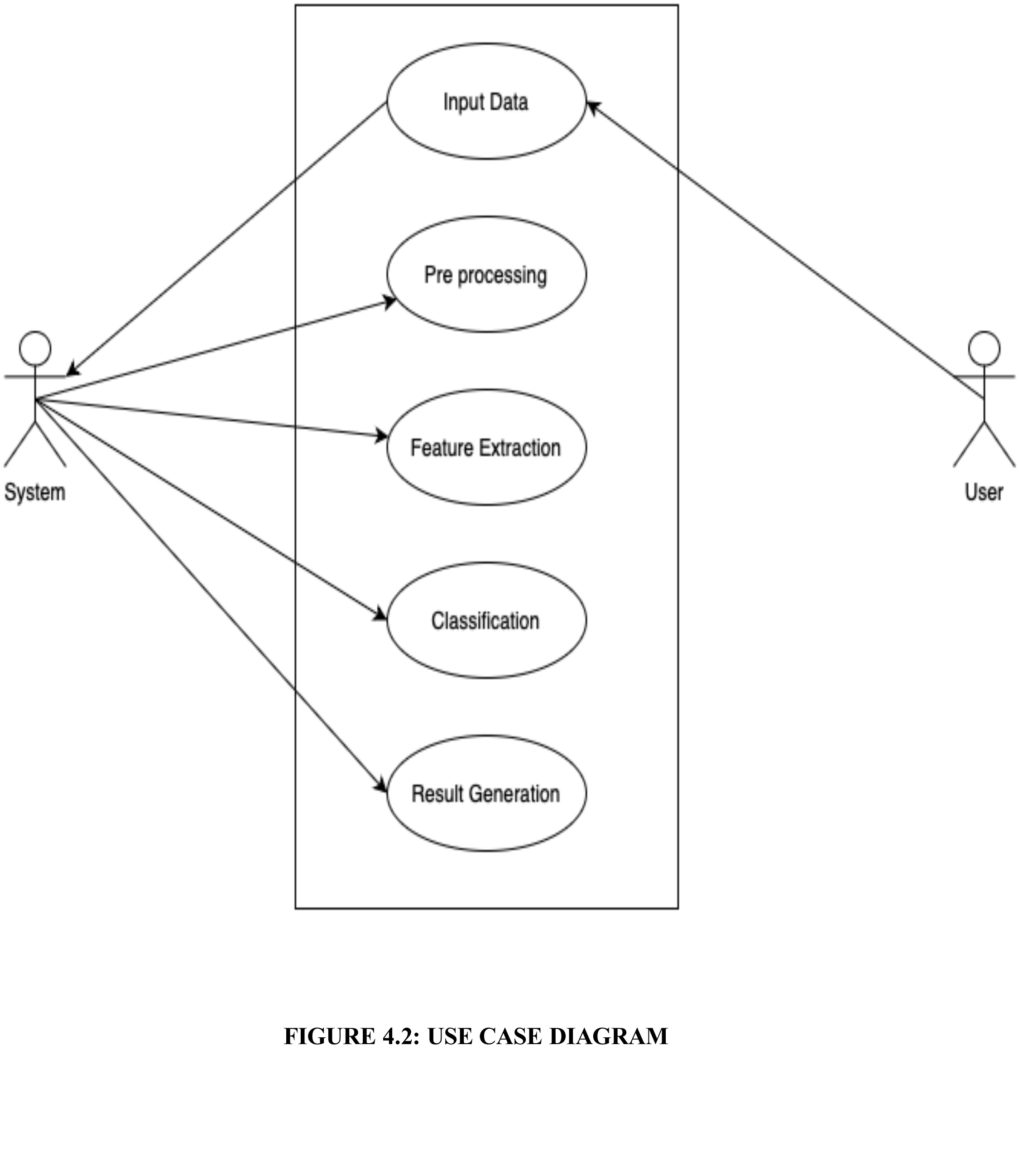


FIGURE 4.2: USE CASE DIAGRAM

4.3. Activity Diagram

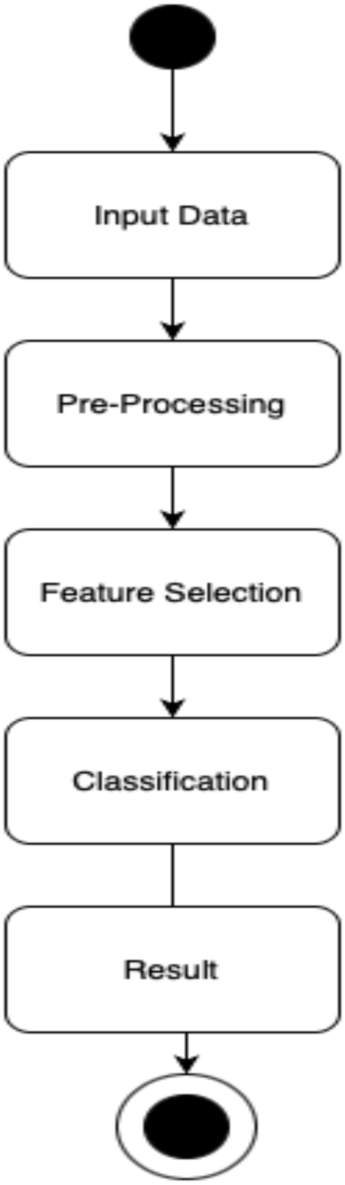


FIGURE 4.3: ACTIVITY DIAGRAM

5. SYSTEM IMPLEMENTATION

5.1. Modules

- Data selection
- Data preprocessing
- Feature selection
- Data Splitting
- Classification
- Result Generation

5.2. Module Description

5.2.1. Input Data

- The input data was collected from dataset repository.
- The data selection is the process of selecting the data for detecting the malware.
- In this project, we have to use the malware detection dataset.
- The dataset which contains the information about the classification (malware and benign), host etc.,
- In python, we have to read the dataset by using the pandas' packages.
- Our dataset is in the form of '.csv' file extension.

5.2.2. Preprocessing

- Data pre-processing is the process of removing the unwanted data from the dataset.
- Pre-processing data transformation operations are used to transform the dataset into a structure suitable for machine learning.
- Missing data removal: In this process, the null values such as missing

values and Nan values are replaced by 0.

- Encoding Categorical data: That categorical data is defined as variables with a finite set of labels values.

5.2.3. Feature selection

- In our process, we have to implement the feature selection such as Principle component analysis (PCA).
- Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning.
- It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation.

5.2.4. Data Splitting

- In addition to the data required for training, test data are needed to evaluate the performance of the algorithm in order to see how well it works.
- In our process, we considered 70% of the dataset to be the training data and the remaining 30% to be the testing data.
- Data splitting is the act of partitioning available data into two portions, usually for cross-validator purposes.
- One Portion of the data is used to develop a predictive model and the other to evaluate the model's performance.

5.2.5. Classification

- In our process, we have to implement the two machine learning algorithms such as linear regression and MLP.

5.2.6. Result Generation

- The Final Result will get generated based on the overall classification and prediction. The performance of this proposed approach is evaluated using some measures like,

5.2.7. Accuracy

Accuracy of classifier refers to the ability of classifier. It predicts the class label correctly and the accuracy of the predictor refers to how well a given predictor can guess the value of predicted attribute for a new data.

ACCURACY = (TP+TN)/ (TP+TN+FP+FN)

6. SAMPLE CODE

```
#============================== IMPORT LIBRARIES ========

import pandas as pd
from sklearn import preprocessing
from tkinter.filedialog import askopenfilename


#============================== DATA SELECTION ==========

filename = askopenfilename()
dataframe=pd.read_csv(filename)
print("========================== Input Data
=============================")
print()
print(dataframe.head(20))

#============================== PREPROCESSING ===========

#==== checking missing values ====

print("===================== Checking Missing Values
====================")
print()
print(dataframe.isnull().sum())


#==== label encoding ====

print("===================== Before Label Encoding
====================")
print()
print(dataframe['classification'].head(15))
print()
label_encoder=preprocessing.LabelEncoder()

print("===================== After Label Encoding
====================")
print()
```

```
dataframe['classification']=label_encoder.fit_transform(dataframe['classification
'])
print(dataframe['classification'].head(15))
print()

#============================== PCA ===========

X =
dataframe.drop(["hash","classification",'vm_truncate_count','shared_vm','exec_
vm','nvcsw','maj_flt','utime'],axis=1)
Y = dataframe["classification"]

from sklearn.decomposition import PCA

pca = PCA(n_components = 20)

X_train = pca.fit_transform(X)


print("--------------------------------------------------------")
print("====== Principle component Analysis ==============")
print("--------------------------------------------------------")
print()
print(" The original features is :", X.shape[1])
print()
print(" The reduced feature is :",X_train.shape[1 ] )
print()

#======================== DATA SPLITTING
=================================

from matplotlib import pyplot as plt
from sklearn import metrics

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=1
)

print()
print("============ Data Splitting ===================")
print()
print()
print("Total no.of data's",X.shape[0])
print()
print("Total No.of training data",x_train.shape[0])
print()
print("Total No.of testing data",x_test.shape[0])
print()
```

7. SYSTEM TESTING

System testing is the stage of implementation, which aimed at ensuring that system works accurately and efficiently before the live operation commence. Testing is the process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an error. A successful test is one that answers a yet undiscovered error.

Testing is vital to the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. A series of tests are performed before the system is ready for the user acceptance testing. Any engineered product can be tested in one of the following ways. Knowing the specified function that a product has been designed to from, test can be conducted to demonstrate each function is fully operational. Knowing the internal working of a product, tests can be conducted to ensure that “al gears mesh”, that is the internal operation of the product performs according to the specification and all internal components have been adequately exercised.

7.1. Unit testing

Unit testing is the testing of each module, and the integration of the overall system is done. Unit testing becomes verification efforts on the smallest unit of software design in the module. This is also known as ‘module testing’. The modules of the system are tested separately. This testing is carried out during the programming itself. In this testing step, each model is found to be working satisfactorily as regard to the expected output from the module. There are some validation checks for the fields. For example, the validation check is done for verifying the data given by the user where both format and validity of the data entered is included. It is very easy to find error and debug the system.

7.2. Integration Testing

Data can be lost across an interface, one module can have an adverse effect on the other sub function, when combined, may not produce the desired major function. Integrated testing is systematic testing that can be done with sample data. The need for the integrated test is to find the overall system performance. There are two types of integration testing.

They are:

i. Top-down integration testing.
ii. Bottom-up integration testing.

7.3. Testing Technique/Strategies

7.3.1. White Box Testing

Derived test cases that guarantee that all independent paths within a module have been exercised at least once.

7.3.2. Black Box Testing

- Black box testing is done to find incorrect or missing function.
- Interface error
- Errors in external database access
- Performance errors.
- Initialization and termination errors

In ‘functional testing’, is performed to validate an application conforms to its specifications of correctly performs all its required functions. So, this testing is also called ‘black box testing’. It tests the external behavior of the system. Here the engineered product can be tested knowing the specified function that a product has been designed to perform, tests can be conducted to demonstrate that each function is fully operational.

7.4. Software Testing Strategies

7.4.1. Validation Testing

After the culmination of black box testing, software is completed assembly as a package, interfacing errors have been uncovered and corrected and final series of software tests begin validation testing can be defined as many, but a single definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the customer.

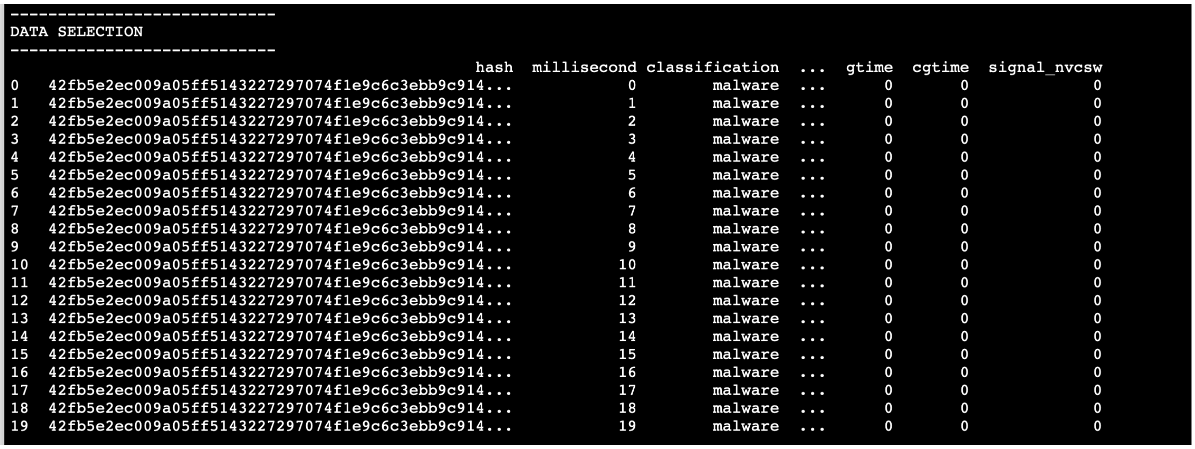
7.4.2. User Acceptance Testing

User acceptance of the system is the key factor for the success of the system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system at the time of developing changes whenever required.

7.5. Output Testing

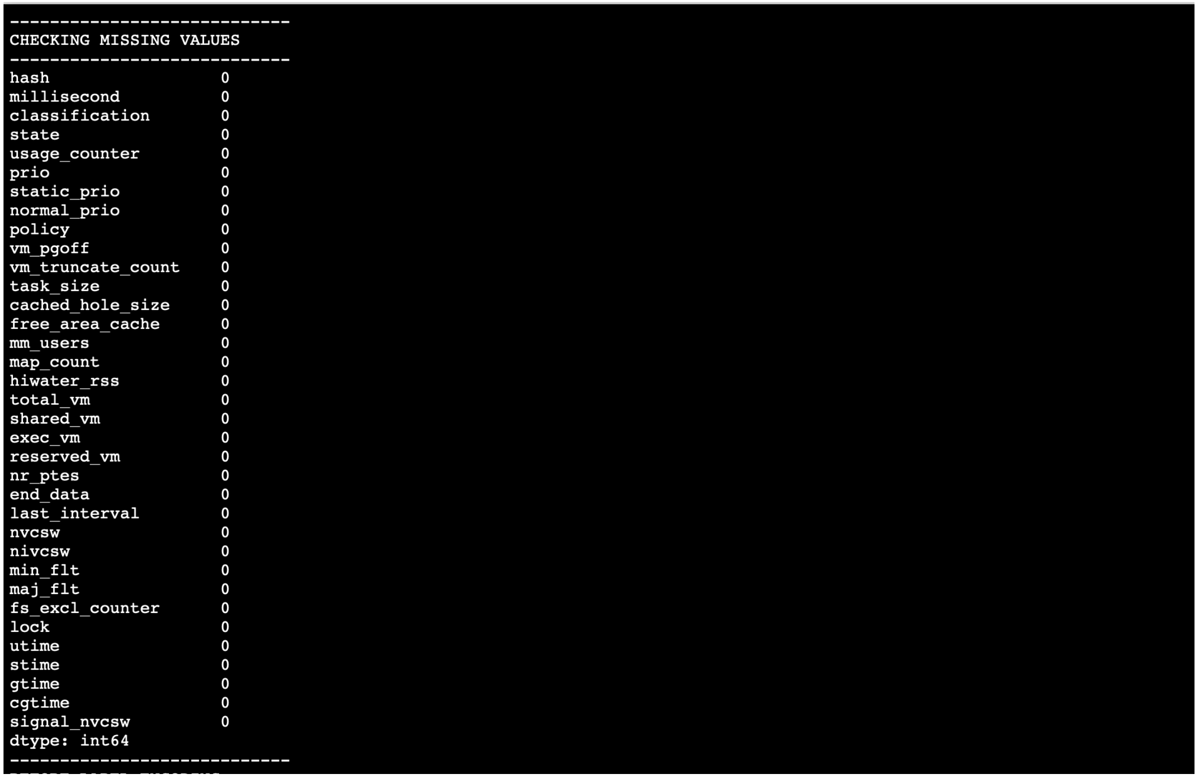
After performing the validation testing, the next step is output asking the user about the format required testing of the proposed system, since no system could be useful if it does not produce the required output in the specific format. The output displayed or generated by the system under consideration. Here the output format is considered in two ways. One is screen and the other is printed format. The output format on the screen is found to be correct as the format was designed in the system phase according to the user needs. For the hard copy also output comes out as the specified requirements by the user. Hence the output testing does not result in any connection in the system.

8. SCREENSHOT



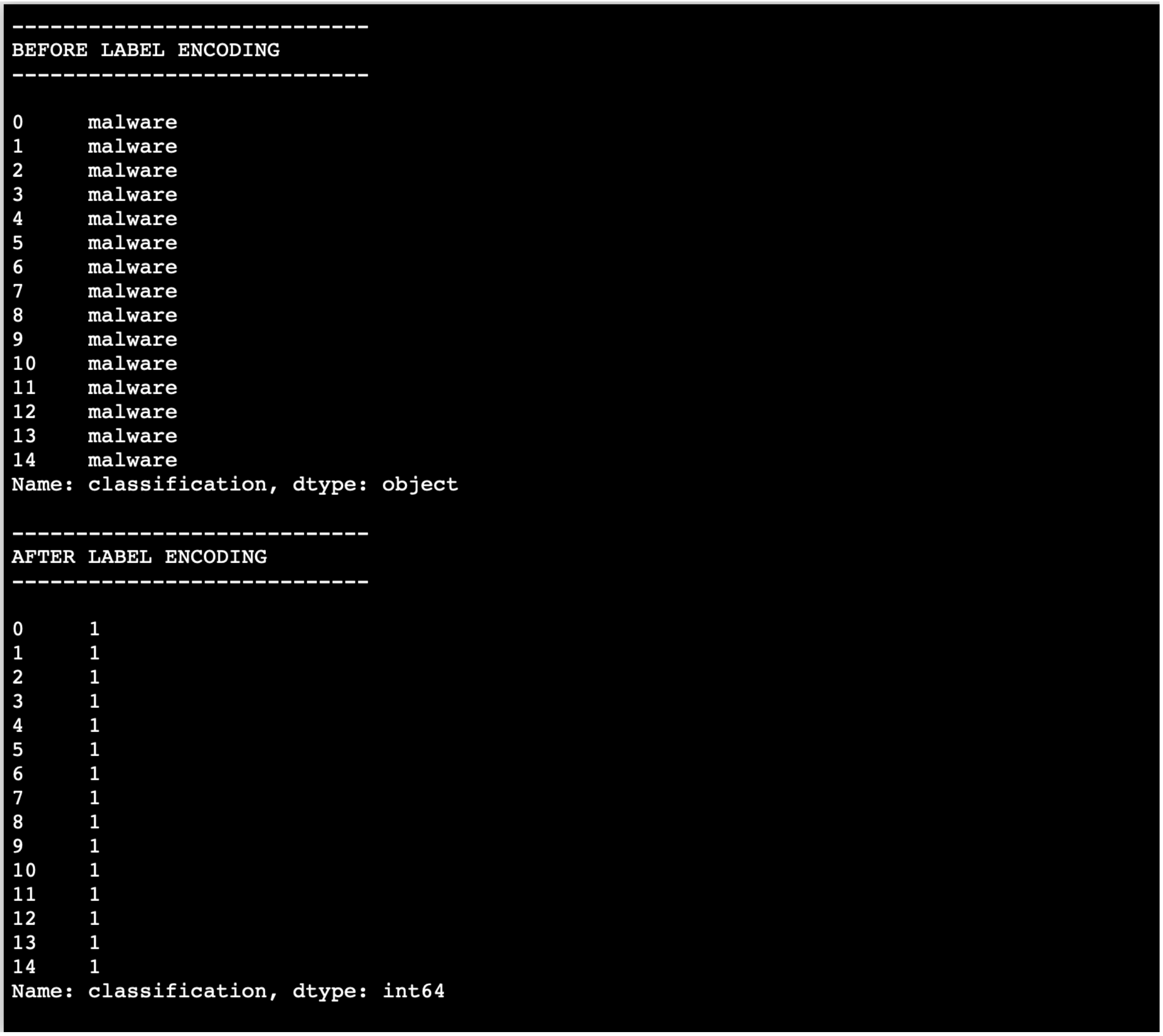
```
---------------------------
DATA SELECTION
---------------------------
                                                hash  millisecond classification  ...  gtime  cgtime  signal_nvcsw
0   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            0        malware  ...      0       0             0
1   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            1        malware  ...      0       0             0
2   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            2        malware  ...      0       0             0
3   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            3        malware  ...      0       0             0
4   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            4        malware  ...      0       0             0
5   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            5        malware  ...      0       0             0
6   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            6        malware  ...      0       0             0
7   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            7        malware  ...      0       0             0
8   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            8        malware  ...      0       0             0
9   42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...            9        malware  ...      0       0             0
10  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           10        malware  ...      0       0             0
11  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           11        malware  ...      0       0             0
12  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           12        malware  ...      0       0             0
13  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           13        malware  ...      0       0             0
14  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           14        malware  ...      0       0             0
15  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           15        malware  ...      0       0             0
16  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           16        malware  ...      0       0             0
17  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           17        malware  ...      0       0             0
18  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           18        malware  ...      0       0             0
19  42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...           19        malware  ...      0       0             0
```

FIGURE 8.1 DATA SELECTION



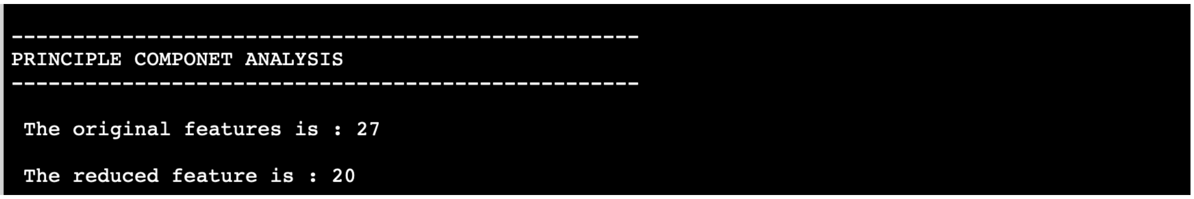
```
----------------------------
CHECKING MISSING VALUES
----------------------------
hash                 0
millisecond          0
classification       0
state                0
usage_counter        0
prio                 0
static_prio          0
normal_prio          0
policy               0
vm_pgoff             0
vm_truncate_count    0
task_size            0
cached_hole_size     0
free_area_cache      0
mm_users             0
map_count            0
hiwater_rss          0
total_vm             0
shared_vm            0
exec_vm              0
reserved_vm          0
nr_ptes              0
end_data             0
last_interval        0
nvcsw                0
nivcsw               0
min_flt              0
maj_flt              0
fs_excl_counter      0
lock                 0
utime                0
stime                0
gtime                0
cgtime               0
signal_nvcsw         0
dtype: int64
----------------------------
```

FIGURE 8.2.1 PRE-PROCESSING



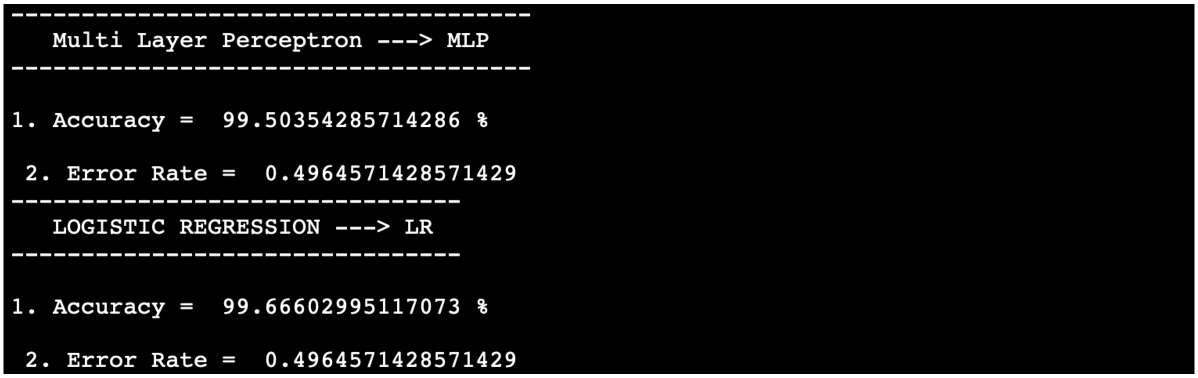
```
----------------------------
BEFORE LABEL ENCODING
----------------------------

0     malware
1     malware
2     malware
3     malware
4     malware
5     malware
6     malware
7     malware
8     malware
9     malware
10    malware
11    malware
12    malware
13    malware
14    malware
Name: classification, dtype: object

----------------------------
AFTER LABEL ENCODING
----------------------------

0     1
1     1
2     1
3     1
4     1
5     1
6     1
7     1
8     1
9     1
10    1
11    1
12    1
13    1
14    1
Name: classification, dtype: int64
```

FIGURE 8.2.2 PRE-PROCESSING (Label Encoding)



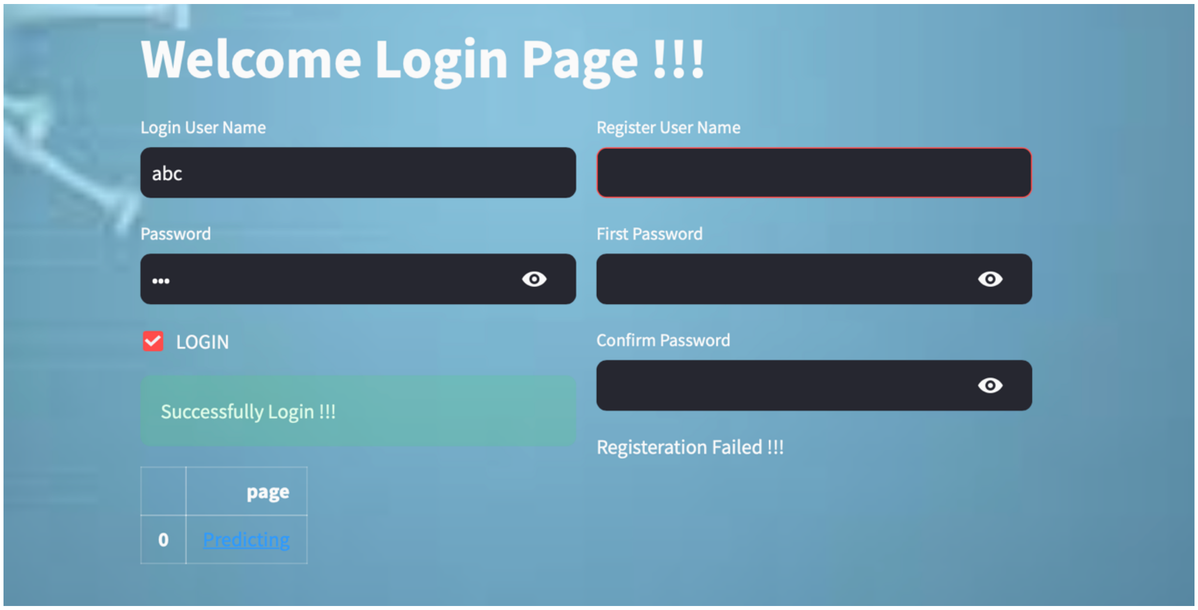
```
---------------------------------------------------
PRINCIPLE COMPONET ANALYSIS
---------------------------------------------------

 The original features is : 27

 The reduced feature is : 20
```

FIGURE 8.3 PRINCIPLE COMPONENT ANALYSIS



```
-------------------------------------
   Multi Layer Perceptron ---> MLP
-------------------------------------

1. Accuracy =  99.50354285714286 %

 2. Error Rate =  0.4964571428571429
--------------------------------
   LOGISTIC REGRESSION ---> LR
--------------------------------

1. Accuracy =  99.66602995117073 %

 2. Error Rate =  0.4964571428571429
```

FIGURE 8.4 CLASSIFICATION MODEL



Welcome Login Page !!!

Login User Name: abc

Password: •••

☑ LOGIN

Successfully Login !!!

	page
0	Predicting

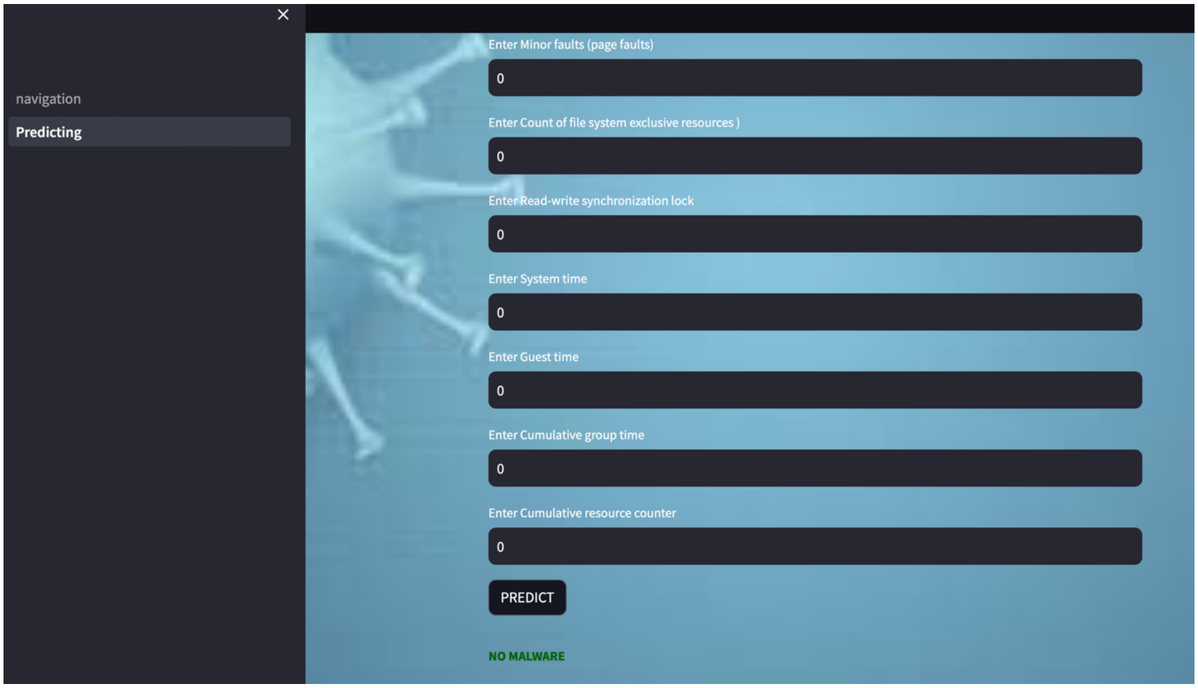
Register User Name:

First Password:

Confirm Password:

Registeration Failed !!!

FIGURE 8.5 APPLICATION LANDING PAGE



Enter Minor faults (page faults)

0

Enter Count of file system exclusive resources)

0

Enter Read-write synchronization lock

0

Enter System time

0

Enter Guest time

0

Enter Cumulative group time

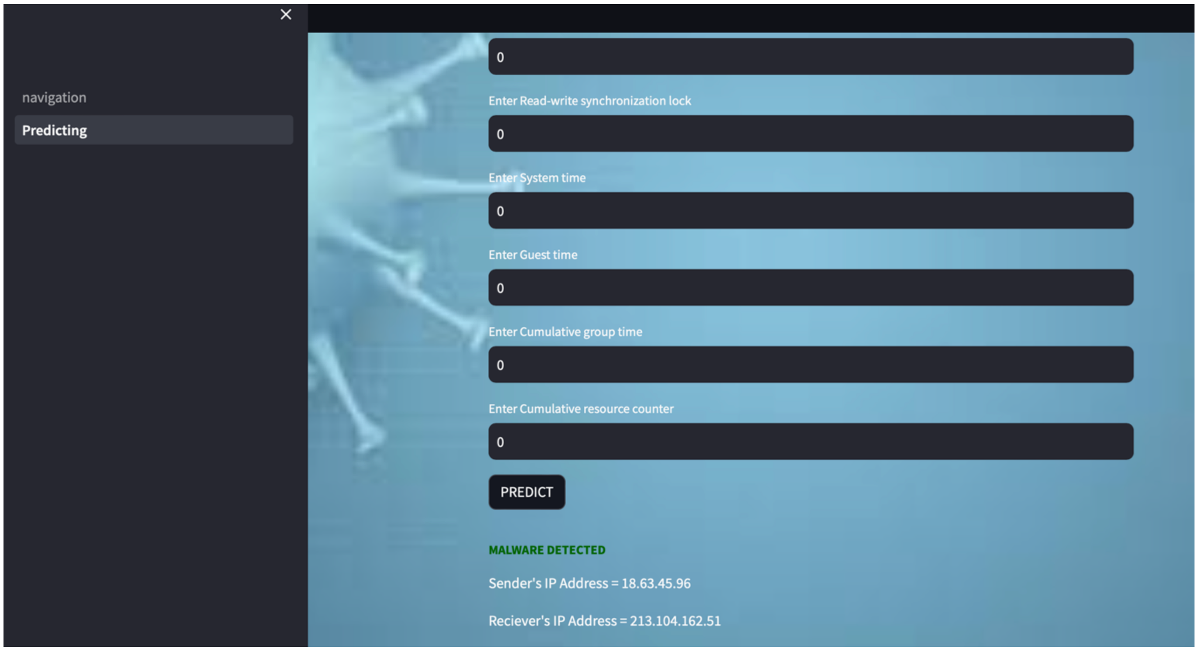
0

Enter Cumulative resource counter

0

NO MALWARE

FIGURE 8.6.1 RESULT- NO MALWARE DETECTED



0

Enter Read-write synchronization lock

0

Enter System time

0

Enter Guest time

0

Enter Cumulative group time

0

Enter Cumulative resource counter

0

MALWARE DETECTED

Sender's IP Address = 18.63.45.96

Reciever's IP Address = 213.104.162.51

FIGURE 8.6.2 RESULT- MALWARE DETECTED

9. CONCLUSION

In conclusion, this study presents a machine-learning based method for detecting malware attacks in software, with a focus on Android applications. By leveraging permission analysis and multiple linear regression models, the framework offers robust performance in identifying potential threats. Furthermore, the comparative analysis demonstrates the effectiveness of machine learning algorithms such as random forest and logistic regression in achieving notable detection accuracy.

Moving forward, this research contributes to the ongoing efforts to bolster cybersecurity in the dynamic landscape of mobile technology. By providing a reliable framework for malware detection, it aids in safeguarding users against evolving threats and ensures the integrity and security of Android ecosystems.

10. FUTURE ENHANCEMENT

In addition to the contributions made by this study, there are several avenues for future enhancement and exploration. Firstly, new features could be incorporated into the existing data to further enhance the detection capabilities of the framework. By exploring variations with additional features, future research endeavors aim to improve the accuracy and robustness of malware detection algorithms.

Furthermore, the application of machine learning techniques extends beyond traditional approaches such as random forest and logistic regression. Future research could explore the integration of advanced algorithms and ensemble methods to enhance detection performance and adaptability to emerging threats.

Moreover, considering the critical nature of malware detection in safety-critical environments, future work will prioritize the development of scalable and efficient solutions. By addressing the unique challenges posed by such environments, researchers can ensure the resilience and reliability of malware detection systems in safeguarding critical systems and infrastructure.

In summary, the future enhancement of malware detection frameworks will continue to evolve in tandem with the evolving threat landscape, driven by advancements in machine learning, data analytics, and cybersecurity research.

11. REFERENCES

- M. Alazab, S. Venkatraman, P. Watters, M. Alazab, and A. Alazab, ‘‘Cybercrime: The case of obfuscated malware,’’ in Global Security, Safety and Sustainability & e-Democracy (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 99, C. K. Georgiadis, H. Jahankhani, E. Pimenidis, R. Bashroush, and A. Al-Nemrat, Eds. Berlin, Germany: Springer, 2012.
- M. Alazab, ‘‘Profiling and classifying the behavior of malicious codes,’’ J. Syst. Softw., vol. 100, pp. 91–102, Feb. 2015.
- S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, ‘‘Hybrids of support vector machine wrapper and filter based framework for malware detection,’’ Future Gener. Comput. Syst., vol. 55, pp. 376–390, Feb. 2016.
- E. Raff, J. Sylvester, and C. Nicholas, ‘‘Learning the PE header, malware detection with minimal domain knowledge,’’ in Proc. 10th ACM Workshop Artif. Intell. Secur. New York, NY, USA: ACM, Nov. 2017, pp. 121–132.
- C. Rossow, et al., ‘‘Prudent practices for designing malware experiments: Status quo and outlook,’’ in Proc. IEEE Symp. Secur. Privacy (SP), Mar. 2012, pp. 65–79. [11] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas. (2017). ‘‘Malware detection by eating a whole exe.’’
- L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, ‘‘Malware images: Visualization and automatic classification,’’ in Proc. 8th Int. Symp. Vis. Cyber Secur. New York, NY, USA: ACM, Jul. 2011, p. 4

- L. Nataraj and B. S. Manjunath. (2016). “SPAM: Signal processing to analyze malware.”
- D. Kirat, L. Nataraj, G. Vigna, and B. S. Manjunath “SigMal: A static signal processing based malware triage,” in Proc. 29th Annu. Comput. Secur. Appl. Conf. New York, NY, USA: ACM, Dec. 2013, pp. 89–98.
- X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in Proc. 14th Int. Conf. Artif. Intell. Statist., Jun. 2011, pp. 315–323
- E. Raff et al., “an investigation of byte n-gram features for malware classification,” J. Comput. Virology Hacking Techn., vol. 14, no. 1, pp. 1–20, 2018.