AN ENSEMBLE BASED CLASSIFIER MODEL FOR THE DETECTION OF OBFUSCATED MALWARE IN PORTABLE EXECUTABLES.

BY

FADEN, David Nanven
MTech/SICT/2017/6842

DEPARTMENT OF CYBER SECURITY SCIENCE
FEDERAL UNIVERSITY OF TECHNOLOGY MINNA

SEPTEMBER, 2021

AN ENSEMBLE BASED CLASSIFIER MODEL FOR THE DETECTION OF OBFUSCATED MALWARE IN PORTABLE EXECUTABLES.

BY

FADEN, David Nanven
MTech/SICT/2017/6842

A THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGERIA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF TECHNOLOGY IN CYBER SECURITY SCIENCE

SEPTEMBER, 2021

ABSTRACT

This research work will focus on detection of malicious obfuscated malware by formulating an Ensemble based model for the detection of obfuscated malware in portable executable files with the ability to detect obfuscated malware with reasonable accuracy. A large dataset retrieved from the website https://github.com/chihebchebbi/Mastering-Machine-Learning-for-Penetration-Testing/blob/master/Chapter03/MalwareData.csv.gz was used. The training dataset comprises of 138,047 PE header file records samples which was divided into: 41,323 clean files containing exe and dll file samples and 96,724 malware file samples. The performance metrics evaluates the above mentioned machine learning algorithms in relation to their predictive capability. Based on the analysis of the tests and experimental results of the Ensemble Model, the Ensemble Model classifier predicted the obfuscated malware dataset with an Accuracy metrics of 98.8%, Precision metrics of 98.9%, Recall metrics of 98.9% and an F1-Score of 98.8%.

TABLE OF CONTENTS

Content

Page

Title page

i

Declaration

ii

Certification

iii

Dedication

iv

Acknowledgements

v

Abstract

vi

Table of Contents

vii

List of Tables

x

List of Figures

xi

CHAPTER ONE

1.0 INTRODUCTION

1

1.1 Background to the Study

1

1.2 Statement of the Research Problem

3

1.3 Aim and Objective of the Study

3

1.4 Scope of the Study

4

1.5 Significance of the Study

4

CHAPTER TWO

2.0 LITERATURE REVIEW

5

2.1 Overview of Malware Detection Techniques

5

2.2 Portable Executables Features

7

2.2.1 API and system calls

7

2.2.2 Byte sequence n Gram

9

2.2.3 OpCodes

12

2.2.4 File system

14

2.3 Machine Learning Classification Algorithm

15

2.3.1 Naïve bayes classifier

15

2.3.2 Support vector machine

17

2.3.3 Decision trees

19

2.3.4 Random forest

19

2.3.5 KNeigbour

20

2.3.6 Gradient boosting classifier

21

2.3.7 Stochastic gradient descent classifier

22

2.3.8 Linear discriminant analysis (LDA) classifier

22

2.4 Feature Selection Analysis

23

2.5 Related Empirical Studies

24

2.6 Summary of Literature Review

25

CHAPTER THREE

3.0 METHODOLOGY

27

3.1 Research Methodology

27

3.2 Overview

29

3.2.1 Data Sources

31

3.3 Model Training

32

3.3.1 Train, test and split

32

3.3.2 Obfuscated malware dataset

34

3.3.3 Feature selection/optimization

36

3.4 Model Testing

38

3.5 Machine Learning Evaluation Metrics

39

CHAPTER FOUR

4.0 RESULTS AND DISCUSSION

42

4.1 Classifier Performance Evaluation

42

4.2 Ensemble Model Performance Evaluation

44

4.3 Accuracy Performance Metrics

47

4.4 Precision Performance Metrics

49

4.5 Recall Performance Metrics

51

4.6 F-1 Score Performance Metrics

53

4.7 AUCROC Score Performance Metrics

55

CHAPTER FIVE

5.0 CONCLUSION AND RECOMMENDATIONS

56

5.1 Conclusion

56

5.2 Contributions to Knowledge

57

5.3 Recommendations

57

REFERENCES

58

LIST OF TABLES

Table Title

Page

3.1 Description of the four Training/test Variables

34

3.2 Obfuscated Malware Header File

35

3.3 36 Important Features after Removal of Features with Low Variance

36

3.4 Key Features Obtained after Optimization of High Variance Features

37

4.1 Classifier Training Result With Splitting Ratio (80 / 20)

43

4.2 Classifier Training Result With Splitting Ratio (70 / 30)

44

4.3 Performance Evaluation of Ensemble Model Compared to other Classifiers

45

4.4 Performance Evaluation of Ensemble Model Combined with another Classifier

46

4.5 Performance Evaluation of Ensemble Combined with two Other Classifiers

46

LIST OF FIGURES

Figure Title

Page

3.1 Methodology for an Ensemble Based Model for the Detection of Obfuscated Malware.

28

4.1 Accuracy Performance Metrics for the Ensemble Model Against other Classifiers

47

4.2 Accuracy Performance Metrics after Combining the Ensemble Model with the Decision Tree Classifier.

48

4.3 Accuracy Performance Metrics after Combining the Ensemble Model with the Decision Tree and K-Neigbour Classifiers.

48

4.4 Precision Performance Metrics for the Ensemble Model against other Classifiers

49

4.5 Precision Performance Metrics after Combining the Ensemble Model with the Decision Tree Classifier.

50

4.6 Precision Performance Metrics after Combining the Ensemble Model with the Decision Tree And Kneigbor Classifiers.

50

4.7 Recall Performance Metrics for the Ensemble Model against other

Classifiers

51

4.8 Recall Performance Metrics after Combining the Ensemble Model with the Decision Tree Classifier.

52

4.9 Recall Performance Metrics after Combining the Ensemble Model with the Decision Tree and Kneigbor Classifiers.

52

4.10 F-1 Score Performance Metrics for the Ensemble Model against other Classifiers

53

4.11 F-1 Score Performance Metrics after Combining the Ensemble Model with the Decision Tree Classifier.

54

4.12 F-1 Score Performance Metrics after Combining the Ensemble Model with the Decision Tree and Kneigbor Classifiers.

54

CHAPTER ONE 1.0

INTRODUCTION

1.1 Background to the Study

Malware is shorthand for malicious software. It was first introduced in the early 1970s when the creeper virus was introduced. Moreover different techniques have been explored to mitigate and detect malware chiefly the use of machine learning Ensemble algorithms on Portable Executables (PE) to predict whether a portable executable is a malware or benign file and to also determine the accuracy of prediction (Ucci *et al.,*
2018). Over the years we have seen multiple variants of malware running into well over 500 thousand malware variants which are all harmful to individuals and organizations that use the internet through electronic devices such as Portable executables and handheld devices (Olalere *et al.*, 2016).

Analysis of malware files are carried out in two ways either through static or dynamic techniques which are then classified into different malware families. Machine learning algorithms are used to predict and classify signatures based on features extracted from malware program code (Damodaran, 2015). The features extracted from static malware analysis may range from byte sequence n grams, operational code and syntactic library calls where function calls are checked to ascertain the libraries accessed by the functions (Pham *et al*., 2018). Malware authors began to develop new ways to stealth the payload of a malware through the introduction of polymorphism and metamorphism into malware behaviour; thus began the era of evasive/Obfuscated malware which cannot be classified using ordinary static analysis of malware code.

Major success has been recorded by anti-virus vendors in detecting malware since the advent of machine learning classifiers as tools to detect malware signatures and variants until the recent emergence of obfuscated malware ranging from viruses, worms, rootkits,

key loggers, spyware and ransomware (Agnihotri, 2018). Much of today's malware are created to stealth during infection and operation through obfuscation techniques to stall and prevent removal or behavioural analysis. Recent malware achieves such stealth manoeuvres using several obfuscation techniques to stealth detection such as dead code insertion, sub routine reordering in the operating system, code transportation, obscure filenames, or masking under the pretence of legitimate programs and services as a white listed program (You & Yim, 2010) .

This Obfuscated technique used by malware authors to bypass static analysis paved way for the introduction and development of dynamic analysis of malware. In dynamic analysis of a malicious code, the behaviour of the malware is monitored as it is executed in a sandbox which is a controlled environment; the natural behaviour of a malware can be observed without requiring the Portable Executable to be disassembled (Rieck *et al.*, 2011). This technique is more effective against evasive/Obfuscated malware because it reveals the malwares running pattern before and after payload exposing the Obfuscated behaviour naturally. The behavioural analysis concept is founded on the behavioural similarity between the unknown malware to the recorded behaviours of already stored discovered malware. In this way, the detection of unknown malware is possible. (Damodaran, 2015).

Machine learning is a scientific study of algorithms through the application of artificial intelligence that enables systems to be able to learn and improve from patterns otherwise known as experience without explicitly being programmed (Swamynathan, 2019). Machine Learning discovers patterns automatically through experience from predefined datasets in order to predict the outcome of unknown occurrences based on previously identified patterns (Baset, 2017). Ensemble learning is a branch of machine learning that

is used to improve the accuracy and performance of traditional machine learning classifiers through the combination of several base models with the aim to produce one optimal predictive model thereby improving the performance accuracy of the decision (Yan *et al.*, 2018).

1.2 Statement of the Research Problem

Due to the concept of obfuscation adopted by malware authors employing mutated hashes, sophisticated obfuscation mechanisms, self-propagating malware and intelligent malware; it is no longer sufficient to detect malware using the non-signature based approach therefore Ensemble learning offers the predictive ability that can provide a much needed advantage to detect the more ever evasive adversaries known as obfuscated malware (Kazanciyan & Hastings, 2014; Rubin et al., 2019 & Scott, 2017). Secondly multiple research work has been carried out in the detection of obfuscated malware using machine learning algorithms chiefly Ensemble classifiers, unfortunately even when successful, the research work is usually based on small malware datasets comprising of not more than 2000 benign and malware files hence the technique used cannot be simulated to accurately represent real life scenarios due to the small quantity of the datasets. This work therefore tends to enhance the predictive ability of detecting obfuscated malware in portable executable with accuracy by formulating an ensemble model with minimum dataset.

1.3 Aim and Objectives of the Study

The aim of this study is to enhance the predictive ability of detecting obfuscated malware in portable executable files with reasonable accuracy through formulating an ensemble with a minimum feature set. The objectives of this research work are to:

i.) Optimize large enough dataset through the application of feature selection to obtain the most significant and relevant features.

ii.) Enhance the detection of obfuscated malware through combining a Gradient Boosting and Random Forest classifier to form an ensemble based model. iii.) Evaluate the performance of the ensemble based model by using Accuracy, Precision, Recall, F1-Score and Area Under the Receiver Operating Characteristics (AUCROC).

1.4 Scope of the Study

This research work will focus on combining the Gradient Boosting and Random Forest classifier to form an ensemble based model. The malware dataset used are datasets for Portable Executables (PE) header files only. Evaluation and experimentation is based on machine learning simulation in Jupyter notebook.

1.5 Significance of the Study

The ensemble based classifier model offers predictive ability to detect obfuscated malware despite this, many researchers use little malware datasets which render the ensemble classifier technique unable to accurately predict malware. It is therefore important that this study be carried out with a large enough dataset so as to enhance the detection of obfuscated malware through combining a Gradient Boosting and Random Forest classifier to form an ensemble based model and also evaluate the performance of the ensemble base classifier model by using Accuracy, Precision, Recall, F1-Score and AUCROC. The ensemble based classifier model technique will enhance detecting of obfuscated malware more accurately when used with a large enough dataset by taking into account real life scenario of malware attacks and detection.

CHAPTER TWO

2.0 LITERATURE REVIEW

2.1 Overview of Malware Detection Techniques

Bazrafshan *et al.* (2013) gave exhaustive survey on comparing heuristic malware detection methods, describing three major malware detection methods commonly used namely; signature based detection, behavioural based detection and heuristic based detection. The research was also able to give advantages and disadvantages of each detection method and proffered reasons heuristic malware detection technique is the most proffered detection method adopted by researchers against metamorphic malware.

The researcher was able to identify three basic components of this method chiefly; the Data Collector; responsible for extracting static and dynamic components from Portable Executables (PE), The Interpreter; which converts the file features extracted from the data collector and The Matcher; which is used to compare behaviour signatures. Even though the behavioural approach builds on the weakness of the signature based approach two downsides of this approach is the high positive rate ratio. Another method explored is the heuristic method which involves data mining and machine learning algorithms most especially classifiers (which this research work is based) to predict and detect polymorphic and metamorphic malware variants with low false positive rate than the behavioural methods. This method leverages on classification of malware based on extracted features from PE files as input. This research work will focus on classification algorithms. Some of the features explored by Bazrafshan *et al*. (2013) include Application Package Interface/System calls, OpCode, N-Grams and control flow graphs by providing a comparison table of their advantages and disadvantages stating that combining two or more features gives better accuracy to the training models moreover the author was able

to show that combining Control Flow Graphs and Application Package Interface calls gives the most accurate results in the training models. Feature extraction shall be discussed later on in this literature. It is worthy of note that machine learning algorithms are trained in any of the three ways namely Supervised Learning, Unsupervised Learning and Semi supervised Learning.

According to Ucci *et al*. (2018), supervised learning is the process of using the concept of classification where a machine learning algorithm known as classifiers map input features of a malware dataset to output labels which are already known. When the process is to map input features to a continuous output label it is known as regression. Accurate output is usually achieved from the training data; the end product is to learn a function that accurately approximates the relationship between input and output malware features. This research work will adopt supervised learning to carry out a comparative analysis of the major classifiers used by previous research work based on this research work dataset. Some of the major classifiers used in detecting malware are Rule-Based Classifier, Bayes Classifier, Random Forest, Naïve Bayes, Artificial Neural Networks (ANN) and Support Vector Machine (SVM). Unsupervised learning is the process of learning relationship between data structures most times identifying the data structures themselves using unlabelled data that is data without output labels. Unsupervised learning works on the concept of learning directly from unlabelled data. It deals mostly with clustering and representation learning (Comar *et al.,* 2013). Unsupervised learning is beyond the scope of this research work.

The comparative analysis written by Vatamanu *et al*. (2015) was to find the best combination of machine learning algorithms to get the lowest false positive rate when classifying malware, the work was based on One Side Class (OSC) perceptron algorithm which can detect malware samples with a low false positive rate, they achieved this

through computing a set of malware features for every binary file in the training dataset which were trained with the OSC perceptrons algorithm using a dataset of clean and infected malware files furthermore cross validation was applied to the dataset to obtain the correct parameter values. Database of well over 2 million records were used, training features where extracted using both static and dynamic malware analysis techniques. The static features extracted includes file geometry, type of packer, type of compiler and executable flags while the dynamic features extracted during execution in a sandbox includes behaviour such as if the Portable Executable (PE) clones itself on the disk, if it seeks permission to connect to the internet, or if it uses the concept of stealth to include itself in some system processes. The result provided the best detection rate although low false positive rates was not achieved. It was concluding that the OSC perceptron algorithm is best used with a method of false positive filtering. The above approach might not be feasible in detecting metamorphic malware based on the extracted features used for training, how feature selection affects detection of metamorphic malware shall be discuss later on. This work will later on show comparison of several machine learning algorithms against the newly formulated Ensemble model in their predictive ability based on the lowest false positive rate using an obfuscated malware dataset.

2.2 Portable Executable Features

2.2.1 Application package interface (API) and system calls

Application Package Interface (API) is a group of commands which provides an interface between Portable Executables with the processor. It contains thousands of functions and, structures and constants that can be used to issue commands to the processor for execution whereas system calls are the only available interface to access the underlying operating system. Furthermore the system calls are the only available interface between a process

and the operating system. This combination makes API and System calls feature critical to detecting malware in that all the function calls and operating system request made by every process, program and malware as the case maybe can be seen.

Ki *et al.* (2015) used API call features to detect malware by the use of sequence alignment algorithms. This algorithm is used in natural language processing and biometrics to extract similar sub sequences efficiently. The study shows that most API call extraction is done for each class of malware and signatures are developed based on the extraction, simply developing signatures based on the frequency of called API is not effective in detecting polymorphic malware that modifies its behaviour and system calls evasively most especially when redundant API is introduced by malware authors therefore the need to use a more generic algorithm that can extract similar sub sequences in the API calls was used. Brahimi and Moussaoui (2015) gave one of the simplest uses of API calls to build a machine learning algorithm for detecting malware, incremental process of data mining was used which was able to efficiently use the number of training samples while reducing the cost of labelling samples basically using API calls as the feature for detection. N-grams where extracted from the API calls dynamically. The feature selection in this work used the following formula (**)** ∑ **(x c)** log (x c) (x c) ( ) x * , + v, b Where, X is variable indicates the existence of feature and C indicates the class (Cv:malware,Cb:benign) to reduce the size of the feature selection in API calls.

Salehi *et al*. (2012) also gave a good use of API calls in feature selection, innovative method that used API names and a combination of API names was introduced and input arguments as features this was because API calls alone are not able to describe the whole behavioural pattern of a malware file. Different classifiers were used to utilize the dataset along with 10-fold cross validation to achieve an accuracy of 98.4% with a false positive

rate less than two percent. System calls are best used when using clustering to classify malware into families based on their behavioural pattern.

Hlauschek *et al.* (2009) used ANUBIS which is an operating system emulator to generate patterns traces for system calls, control flow dependencies and network analysis results. Extraction of a system call behaviour pattern from the execution trace of system calls to identify and mine relevant aspects of a malware behaviour was done, thereby generating a behavioural pattern model. The key idea was to identify system calls that access operating system objects such as files, directories, and registers and form behavioural model based on this patterns. A locality sensitive hashing was used to compute approximations for clustering that require ON^2 distance computations then the Jaccard Index for measuring similarity between the generated behavioural patterns. One of the major downsides of this approach is the inability to tackle recent evasion techniques employed by malware authors like windows command line obfuscation technique.

2.2.2 Byte sequence n-gram

Byte sequence is basically the bytes contained in the machine language of a Portable Executable. These bytes represent the sequence of instructions and how they are executed and combined. When a byte sequence feature of a PE is extracted the common method is to use n-grams to discover the frequency of the set of co-occurring byte sequence to detect a pattern in the byte sequence known as a signature. These signatures are used to label files either as malware for benign. Most researchers use static analysis with no more than 3 n-gram sequences to extract byte sequence features in a dataset. A very practical example of how byte sequence was used to generate signatures that could accurately detect malware was done by Schultz *et al.* (2015). A framework was designed to detect new samples of malware files by using static analysis to extract the byte sequence on a

public dataset containing both malware and benign files then classifiers where trained over a subset of the data. The main goal was to detect new malware samples by separating their dataset into two sets; one set which was labelled the training set was classified using data mining algorithms to classify previously unseen binaries gotten from the byte sequence of the dataset as malicious or benign. The test set was a subset of the dataset that had no malware examples in it that were seen during the training of the data mining algorithms, this gave a good dataset to use in testing the performance of an algorithm over new malware examples. Cross validation techniques were use during the implementation of a traditional signature-based algorithm to compare with the data mining algorithms over new malware examples. A detection rate of 97.76%, was detected, this figure is double the detection rate of a signature based scanner over a set of new malware examples. Piyanuntcharatsr (2015) studied the comparative analysis on research methodology and performance of malware detection using data mining techniques. The main motivation for this research work was the difficulty of selecting which malware feature to extract from a malware dataset. Interest was more focus in comparing two approaches that use features which are based on statistical values and byte sequence instructions using 1,2,3 n grams. The data set used contains was given two labels; the reference data set and application data set. The reference set was used for creating the model and the application set was for testing the accuracy of the data mining algorithm, they were able to show the correctness by classes for the statistical approach. The method performs better when the n gram extracted block size is large but a better result was obtained when the n gram extracted block size equals to the file size.

As earlier stated obfuscation and polymorphism employed by malware authors to avoid detection at file levels has been the recent norm. The dynamic analysis of malware binaries during execution provides a technique for categorizing and mitigating the threat

of malicious software. In the research work by (Rieck *et al.*, 2011) dynamic analysis of PE binaries at runtime was the major method employed, the proposed framework allowed for the discovery of new malware classes with similar behaviour and then assign unknown malware variants to these new classes. An incremental approach was used for dynamic analysis which was able to process the behaviour of thousands of malware byte codes on a daily basis. The hypothesis for the incremental analysis was to reduce the run-time overhead of current analysis methods at the same time providing accurate discovery of malicious software variants. The framework first executes and monitors malware byte codes in a sandbox environment generating a sequential report of the monitored behaviour for each binary. These generated sequential reports are then placed into a high-dimensional vector space to enable the similarity of behaviour to be accessible geometrically, according to the researchers this allows for designing intelligent and powerful clustering and classification methods using machine learning algorithms which can identify unknown and known classes of malware based on the sequential reports. The next step employed was to alternate between clustering and classification processes providing the discovered behaviour of malware to be analysed incrementally on a daily basis. Furthermore reports with unidentified behaviour are clustered for discovery of unknown malware classes.

2.2.3 Opcodes

According to literature opcodes features are one of the most frequently used features. This is because opcodes reveal the machine language operations to be executed together with the data they will process and they are extracted by static analysis. Wong and Wong and Stamp (2006) analysed several metamorphic virus generators by using opcodes defining a similarity index to quantify the degree of metamorphism that each individual generator

produces. The model was based on Hidden Markov Models (HMM) which are used for statistical pattern analysis. The framework works by extracting the sequence of opcodes from two assembly programs to obtain opcode sequences of length n, and m, where n and m are the numbers of opcodes in the two assembly programs. Sequential numbers are given to each opcode in the sequence to allow comparison between n and m by considering all subsequence of three consecutive opcodes from each sequence, whenever any of the three opcodes are the same it is counted as a match and marked using coordinates x and y to obtain a graph. This approach provides a framework to effectively detect new malware variants if the variants are metamorphic and cannot be detected by signature based scanning. The downside of this method is the new malware variants that use command line obfuscation to hide a malware payload. A combination of extracted features will go a long way to enhance this method against command line obfuscation where opcode and system calls are used.

Observing the frequency of opcodes sequence occurrence is the one of the major ways of detecting novel malware as described by (Santos *et al.,* 2013); the model is based on the frequency of opcodes sequence in a PE. The opcode sequence feature was extracted for every file in the dataset for lengths n = 1 and n = 2, this was due to the downside of extracting large amount of features which renders the extraction very slow. After extraction, a k fold cross validation was applied to the features obtain by dividing the dataset into training datasets and test dataset with k = 10 which means for every classifier tested the datasets was split 10 times into 10 different sets. The four models used for learning were the Bayesian networks, Decision Trees, K- Nearest Neighbour and Support Vector Machines. These models were used for each validation step. The model was now tested to measure the processing overhead of the model by measuring the True Positive Ratio (TPR) of each classifier. The machine-learning classifiers achieved high

performance in classifying unknown malware but the downside is that the processing overhead of this framework is highly dependent of the length of the opcode sequences. Another research work which used the Hidden Markov model and opcodes sequence was done by Derhami *et al*. (2015) but the difference from Santos *et al.* (2013) was the method of opcode sequence feature extraction showing that extracting specific features in the opcodes sequence to train the HMM was more effective in detecting metamorphic malware variants. The extraction of these specific features was done by methods similar to sound processing. The specific features pin pointed by the authors are the various important opcode commands contained in malware files which were separated from each other by defining a label of less important opcodes which are identified as the ones that have more similarity to benign files this way the important opcode sequence are separated from the less important ones. The Hidden Markov Model is trained based on these separated commands of opcodes. The trained HMM is now used to classify files of the test set then the members of the same metamorphic malware variants are separated from non-members. The framework showed that the Hidden Markov Model when trained based on the important sequences of opcode was able to process with higher speed and was more than most HMM models that are trained with both the less important and important opcode sequence.

2.2.4 File system

File system changes are also used to monitor the behaviour novel malware. The type of executed file operations by malware variants is very key in getting behavioural patterns of these malwares. Operations such as, how many files were read, deleted or modified and in what directories are very important features to look into to detect novel malware. Nari and Ghorbani (2013) did a research work to address how ineffective antivirus products detect malware and categorize them wrongly by proposing a classification

technique that describes malware behaviour in terms of changes to a system's files rather than on system calls, opcode sequence or byte sequence. This is achieved dynamically by observing the malware behaviour in a controlled environment and classifying the behaviours accordingly. Network activity is one major way of monitoring the malware behaviour that cannot be otherwise categorized by antivirus products due to the metamorphic behaviour of recent malware. The main goal was to apply automated clustering to detect and understand malware behaviour by monitoring the state of file changes and network activity in a controlled environment to identify various malware families. Limitations of this research work is the common limitation associated with dynamic analysis where a metamorphic malware may not drop its payload if it suspects it is being monitored in a sandbox, such evasive techniques makes this process sometimes ineffective from intelligent malware.

The literature reviews above reveal that PE features are important in detecting malware behaviour most especially the new metamorphic malware variants. Some features are best suited to detect specific kind of malware for example opcodes and byte sequence n grams are best suited for detecting metamorphic malwares as they deal with the binaries and machine language operation sequence, API and system calls were used to classify more ancient malware family variants but they are not effective in detecting metamorphic malware variants due to fact that API/Systems calls only look at the function calls and operating system request made by every process.

What if there is a function calls masked inside an application such as the windows Power shell or windows Command line tool which are both white listed applications, the chances for detection becomes very slim not until the payload is delivered furthermore file system feature is another feature that deals with monitoring what files are written or deleted in a system but also not so effective in detecting file less metamorphic malware variants such

as the Powershell.exe which is a Trojan created to steal your data and disrupt normal activities of your system while hiding under the umbrella of a legitimate piece of software on your system. Therefore this research work will focus on a combination of features to use in comparing the different classifier algorithms so as to see the most efficient group of features to use in classifiers for detecting metamorphic malware variants, this work will use opcodes and byte sequence n-grams PE features.

2.3 Machine Learning Classification Algorithms (Ensembles)

As stated earlier the various classification algorithms will be discuss in detail and how it have been used in previous research papers.

2.3.1 Naïve bayes classifier

Bayes theorem is based on the probability of an event based on past knowledge of particular conditions that might be similar or related to that event; Naïve Bayes classifiers are based on applying Bayes theorem with the assumption that features of measurement are independent of each other. It is a family of algorithms where all of the algorithms have a common principle in which every pair of features being classified is independent of each other. It works by predicting family probabilities for each class of feature such as the probability that a given data point belongs to a particular class. The class with the highest probability is seen as the most likely class. It is known to work very well with natural language processing problems.

Schultz *et al. (*2015) conducted a research on a data mining framework which automatically detects malicious binaries. After feature selection was carried out on a data set consisting of 4,266 programs broken down into 3,265 malicious binaries and 1,001 clean files; every example in the set was labelled either malicious or benign by the commercial virus scanner. The researcher compared signature based methods and several

Classifier Algorithms such as RIPPER, Naive Bayes and Multi naïve Bayes on the extracted features to get the most accurate algorithm with least False Positives (FP) rate, for example the RIPPER algorithm was used on the dataset; which is a rule-based learner that builds a set of rules to identify the classes of either positive examples and negative examples while minimizing the amount of error. The Naïve Bayes algorithm was also applied by the researcher to express the probability that a program is in a given class given the program contains the set of features F by defining C to be a random variable over the set of classes; benign, and malicious executables, so as to compute P (C|F). The Naive Bayes algorithm using strings as features outperformed the other learning algorithm which was far better than the signature method in terms of false positive rate and accuracy. The researcher was able to get the most accurate result with 97.11% and within 1% of the highest detection rate which far exceeds other algorithms in every category.

Kolter and Maloof (2006) compared several major machine learning algorithms to detect and classify malicious executables in the wild, Naïve Bayes was one of the algorithms used to evaluate the training examples made up of n-grams of byte code although the results showed that boosted decision tree outperformed other classifiers with a true-positive rate of 0.98 and a false-positive rate of 0.05 which was desirable. There was a cost of misclassification error that was discovered by the researcher and to tackle this issue they used the receiver operating characteristic (ROC) analysis to get a graphical plot to illustrate the diagnostic ability of binary classifiers by plotting the true positive rate against false positive rate at various threshold settings and the boosted decision trees outclassed the Naïve Bayes and other classifiers with an area under the curve of 0.996. Firdausi *et al.* (2010) also did a comparison of machine learning classifiers using automated behaviour-based malware detection to carry out analysis of malware behaviour. The aim was to generate sparse vector models for classification using different

machine learning classifiers to get the overall best performance result of each classifier. The classifiers used in this research are the k-Nearest Neighbours
(kNN), Naïve Bayes, J48 Decision Tree, Support Vector Machine (SVM), and Multilayer Perceptron Neural Network (MLP). The researcher was able to prove that the overall best performance was achieved by J48 decision tree with a recall of 95.9%, a false positive rate of 2.4%, a precision of 97.3%, and an accuracy of 96.8%.

2.3.2 Support vector machine (SVM)

Support Vector Machine are learning models under supervised learning models that analyse data used for classification and regression analysis problems. SVM finds out the line in a hyper plane separating two defined classes. Support Vectors are simply the coordinates of individual observation.

Chen *et al.* (2012) carried out a research on how Support Vector Machines are used, in his research Support Vector Machines were used alongside decision trees to categorize malware. Support vector Machine was applied to the training dataset features to minimize the classification errors on a set of randomly selected samples to attain the best classification performance to detect malware evolution and zero-day attacks. The framework builds models with support vector machines (SVMs) and gradient boosting decision trees (GBDTs) to aid in visualizing malware categorizations.

Comar *et al.* (2013) also combined supervised and unsupervised learning for zero day malware detection using layer 3 and layer 4 network traffic features by harnessing on the advantage of accuracy offered by supervised learning classification in detecting known malware classes and families as well as the advantage of unsupervised learning to detect new and unknown classes of malware. The result was a framework that demonstrates high effectiveness in detecting zero day malware using real network data from large Internet

service providers. The researcher designed a tree-based kernel to use for one-class SVM supervised learning classifiers to remove the data imperfection issues that arise in the network flow data. The framework was able to detect existing and novel malwares with very high precision.

Santos *et al*. (2013) proposed a new method to detect unknown malware families based on the frequency of appearance of opcodes sequence, several classifiers were tested and SVM with polynomial kernel had the fastest of the tested algorithms for SVM, achieving a training time of 3.76 milliseconds and a testing time of 0.01 milliseconds, problems were encountered in feature selection due to the explosion of opcodes features which was tackled by Ranveer and Hiray (2015) using opcodes density histograms to reduce the explosion of features. Eigen vector subspace analysis was used to filter and lower the misclassification and interference of features. This paved way for a system that detects with high accuracy and low false.

2.3.3 Decision trees

Decision Trees are excellent for helping a researcher to choose between several courses of action. It provide options and describe the possible outcomes of choosing those options. Decision trees are mostly used in operations research to provide decision analysis to aid in discovering a strategy that will attain to a prescribed outcome. Decision Trees in Machine Learning are used as classifiers in classification and regression under supervised learning. The basic algorithm used in decision trees is known as the ID3 algorithm, the ID3 algorithm builds decision trees using a top-down greedy approach. Decision Tree shows the correlation between several

features and non-linearity between the features, a decision tree is easy to understand requires very little data cleaning and no constraints on the data type.

Anderson *et al.* (2011) came up with a framework for automated classification of malware samples based on malware network behaviour; the concept of the framework is to abstract network behaviour of malware samples to high level behavioural patterns that must contain all network activity communication together with dependencies between network activities. The patterns are modelled as graphs and the graph features were found to be effective in classifying malware samples. All classification of the graphs used Decision Trees to classify the malware samples.

2.3.4 Random forest

Random Forest algorithm can also be used for both classification and regression kind of problems. The Random Forest Algorithm works by creating a forest given a number of trees, the higher the number of trees in the forest the higher the accuracy of the results. When a Random Forest classifier is fed the training dataset with targets and features, the Random Forest classifier will come up with some **s**et of rules that are used to perform prediction on the test dataset. The underlying principle of Random Forest classifier is the principle that a group of weak learners can come together to form a strong learner thereby making the Random Forest classifier to be able to classify large amounts of dataset with high accuracy . Random Forests are do not over fit because of the law of large numbers by introducing the right kind of randomness it makes them accurate
classifiers.

Wang (2014) used Random Forest classifiers to get a detection rate of 95.6% on novel worms whose data was not used in the model building process. The author used opcode sequences as the underlying feature extraction method to form binary classification

problems and built tree based classifiers although Bagging and Decision Trees were also used to obtain optimal results.

Ahmadi *et al*. (2016) developed a framework that is effective in categorizing malware variants into actual family groups. The concept was to extract, and select a set of novel features for the effective representation of malware samples. These features were later grouped according to different characteristics of malware behaviour and the proposed method achieved high accuracy after using Random Forest classifier to categorize malware variants.

2.3.5 k-Nearest neighbour (k-NN)

K-NN is a type of instance-based learning, or lazy learning, where the function is approximated locally and all computation is deferred until classification. It is used for both classification and regression predictive problems. Its major advantage is the ease to interpret output, calculation time and prediction power. The goal is usually to find the k influence in the algorithm. The k-NN algorithm always assumes that similar things exist within the same area of focus and they are usually close to each other which means similar things are near to each other.

Kong (2013) used SVM and k-NN for classification; in this framework the classifiers are trained with respect to each attribute type after which the Adaboost algorithm is used to learn the confidence level associated with each classifier. Depending on the values provided for each attribute type together with the confidence level associated with each type of value obtained from the Adaboost algorithm, the classifiers were able to make a decision on which family the new malware sample belongs to by choosing the malware family that has the highest total confidence weight from all the individual classifiers. Kumar (2017) carried out an analysis of machine learning algorithms used in malware

classification in cloud computing environments. The classifiers used in this research are k-Nearest Neighbours (kNN) and J48 Decision Tree using n-grams byte sequence as features, although the overall best performance was achieved by J48 decision tree with a recall of 96.3%, the k-NN had a close recall to it.

2.3.6 Gradient boosting classifier

Boosting is a process of enhancing weak learning models into strong learning models therefore Gradient Boosting classifier is a machine learning classifier that joins several weak learning models together to create strong predictive models and it mostly uses Decision trees. The process of boosting involves fitting every new tree into a modified version of the original malware dataset. Agnihotri (2018) used Gradient boosting classifier to detect ransomware through a static analysis of the ransomware PE file. Extraction of the static attributes was first carried out to obtain numerical values for the attributes which were used as inputs to the gradient boosting classifier to predict if the given sample is malicious or not. The performance metrics used was the false positive rate to grade the performance of the classifier; 0.3 percent false positive rate was obtained. Furthermore (Pham *et al.,* 2018) did a research on Static PE Malware Detection Using Gradient Boosting Decision Trees Algorithm stating that the problem with gradient boosting is the training time and also the ability to predict using imbalanced data makes the performance metrics somewhat inaccurate. They were able to reduce the training time by selective feature extraction and obtained a detection rate of 99.394 percent and a false positive rate of 1 percent.

2.3.7 Stochastic gradient descent classifier (SGD)

SGD forms the basis of Neural Networks and a gradient means a slope or a slant surface therefore a gradient descent means to descend down a slope to achieve the lowest point

on that surface. Stochastic gradient descent is a simple and effective numerical optimization machine learning classifier which is used in solving large-scale machine learning problems particularly for ridge regression and regularized logistic regression clearly showing the superiority of stochastic gradient descent to other machine learning algorithms for large-scale machine learning problems.

2.3.8 Linear discriminant analysis classifier (LDA)

Linear Discriminant Analysis was developed as early as 1936 by Ronald A. Fisher. It is a statistical learning method mostly used for classifying observations to a class or category. LDA predicts a common covariance matrix that exists in all classes in a data set; a covariance matrix being a square matrix that contains the variance and covariance related with several variables. Kuriakose and Vinod (2014) did a research on metamorphic malware detection using LDA with an accuracy of 99%. The research work used non signature based approach using feature selection techniques to achieve their objective.

2.4 Feature Selection Analysis

Feature selection plays a vital role in training datasets with classifiers to categorize or detect novel malware variants most especially the new metamorphic malware variants. Jiang *et al*. (2011) stated that the type of feature selected affects the ability to detect accurately the metamorphic malware variants proving by his research work, unnecessary and redundant PE features when selected may decrease the detection rate of metamorphic malware variants. He proved that feature selection phase in malware detection plays a vital role in the whole detection process and can efficiently reduce the redundant and unnecessary features in the malware dataset, this in turn will reduce the false positive rate for a malware detection model using classifiers.

Obfuscated malware have proven to be more challenging to detect using random malware dataset feature selection for example (Derhami *et al.,* 2015) noted importance should be given to some part of the malware dataset with the goal to extract the significant sequences of malware opcodes in the dataset, they used the dissimilarity of these significant sequence of malware opcodes to the benign files to select the significant sequence because all parts of a malware dataset feature are not representative of the malicious nature of the malware. As seen above all the reviewed work used ngrams byte sequence, opcodes, API/System calls with feature selection for training the various classifiers used in malware detection which are effective in detecting known and unknown malware variants but not so effective in detecting metamorphic malware for example obfuscated malware in Portable executables that inject themselves inside a white listed software so as to disguise itself as a trusted system process.

2.5 Related Empirical Studies

A very good example of an obfuscated malware that injects itself inside a white listed software is the Powershell.exe virus that masks itself inside a Windows system Powershell tool and executes malicious Powershell commands traditional malware datasets feature selection may miss the obfuscation hidden in the Powershell during feature selection due to the fact that Powershell is white listed software under the windows operating system. Hendler *et al.* (2018) in their research on Detecting Malicious PowerShell Commands using Deep Neural Networks expanded on the wide gap between the lack of research on automatic detection of obfuscated malicious PowerShell commands and the high cases of PowerShell based malicious cyber

exploits.

This point was later developed upon by Bohannon, (2017) who showed that recent approach is effective in detecting metamorphic malware but not so effective in detecting obfuscated PowerShell attack which is fast becoming an ever increasing trend due to the fact that PowerShell attack is evasive and nearly impossible to detect because obfuscated command line arguments and PowerShell events are not logged and monitored and also Powershell commands are white listed commands therefore most malware samples do not capture command line arguments/Powershell commands. For example the following command: *Get-ChildItem -Force -R .*.txt | ForEach-Object {Get-Content $_ -TotalCount 4; Get-Content $_ -Tail 2} *>> o.log* will output the first and last four lines of every text file in a directory/folder, moving down child directories repeatedly and including any hidden or invisible files it finds. The following command: *Set-MpPreference -DisableRealTimeMonitoring $true* will disable the Microsoft Defender anti-virus engine. Obfuscation of such commands can make the CLA/powershell a powerful tool to use as a malware when masked by obfuscation. The above commands are all legitimate and white listed commands in windows CLA/Powershell.

2.6 Summary of Literature Review

From the exhaustive literature review it can be seen that machine learning has come a long way to aid the detection of malware chiefly the use of Ensembles. It is also worthy of note that some of the methods employed are not so effective in detecting obfuscated malwares depending on the mode of feature extraction and the Ensemble used. Furthermore, even while most research work and academia have carried out research on predicting malware engines with high end accuracy according to the performance metrics used it has been established by literature that Obfuscated malicious white listed malware are not detected easily and in cases that the research work achieves high accuracy, the dataset used is very small in nature to capture real life scenarios therefore a method of

identifying Obfuscated malware variants through a large enough dataset to accurately predict Obfuscated malware signatures in Portable Executable header files is needed to augment the process of malware detection. This research work intends to breach that gap through the development of an Ensemble based malware detection model using machine learning classifiers (ensembles) that can detect Obfuscated malware accurately.

In addition a baseline approach for feature selection/optimization comprising of a variance threshold and Pearson's Correlation to improve estimators, accuracy scores and to boost performance of the Ensemble model while training the model on the dataset. Feature selection is achieved by removing all features whose variance doesn't meet a specific threshold which will measure how strong a relationship is between the remaining features. This research work intends to draw techniques from (Swamynathan, 2019; Bohannon, 2017).

CHAPTER THREE

3.0 METHODOLOGY

3.1 Research Methodology

The description for the methods and steps adopted for the purpose of formulating an Ensemble based model for detecting obfuscated malware in a Portable Executable with reasonable accuracy is discussed in this chapter. A large dataset from https://github.com/chihebchebbi/Mastering-Machine-Learning-for-Penetration Testing/blob/master/Chapter03/MalwareData.csv.gz was used. All examples in the dataset are in the Windows Portable Executable header file format.

As stated in the literature review, this research will use a baseline approach based on SK_learn python library for feature selection/optimization comprising of a variance threshold and Pearson's Correlation to achieve optimization of the malware dataset features by removing all features whose variance does not meet some threshold and measure how strong a relationship is between the remaining features, this will improve estimators, accuracy scores and to boost performance of the predictive capability of the Ensemble based model on the dataset. This Research work shall use the Majority weighted voting method to combine the Gradient Boosting and the Random Forest algorithm which are both classification algorithms, the final prediction decision of the ensemble model relied on voting. There are two major optional strategies for voting which are majority voting and weighted voting. Some of the dependencies include, windows 10 operating system, python 3 server, **Anaconda**; which is a free and open source python programming language platform for scientific computing (data science, machine learning , data processing and predictive analytics) that aims to simplify package management and deployment, **PEfile**; which is an independent module to parse and work with PE files, **Pandas**; which is a software library for the python programming language for data

manipulation, data cleaning and analysis, **Jupyter Notebook**; an open source web application that allows you to create and share documents that contain live code, equations and visualizations used mostly for machine learning, modelling, simulation which will be used as the GUI and IDE for running tests on the malware datasets, DOSfuscator.exe; a software used to modify original malware dataset to get a new Obfuscated malware dataset version.

The methodology as shown in Figure 3.1 is implemented in four main steps which are data gathering, feature selection, training phase and testing phase.

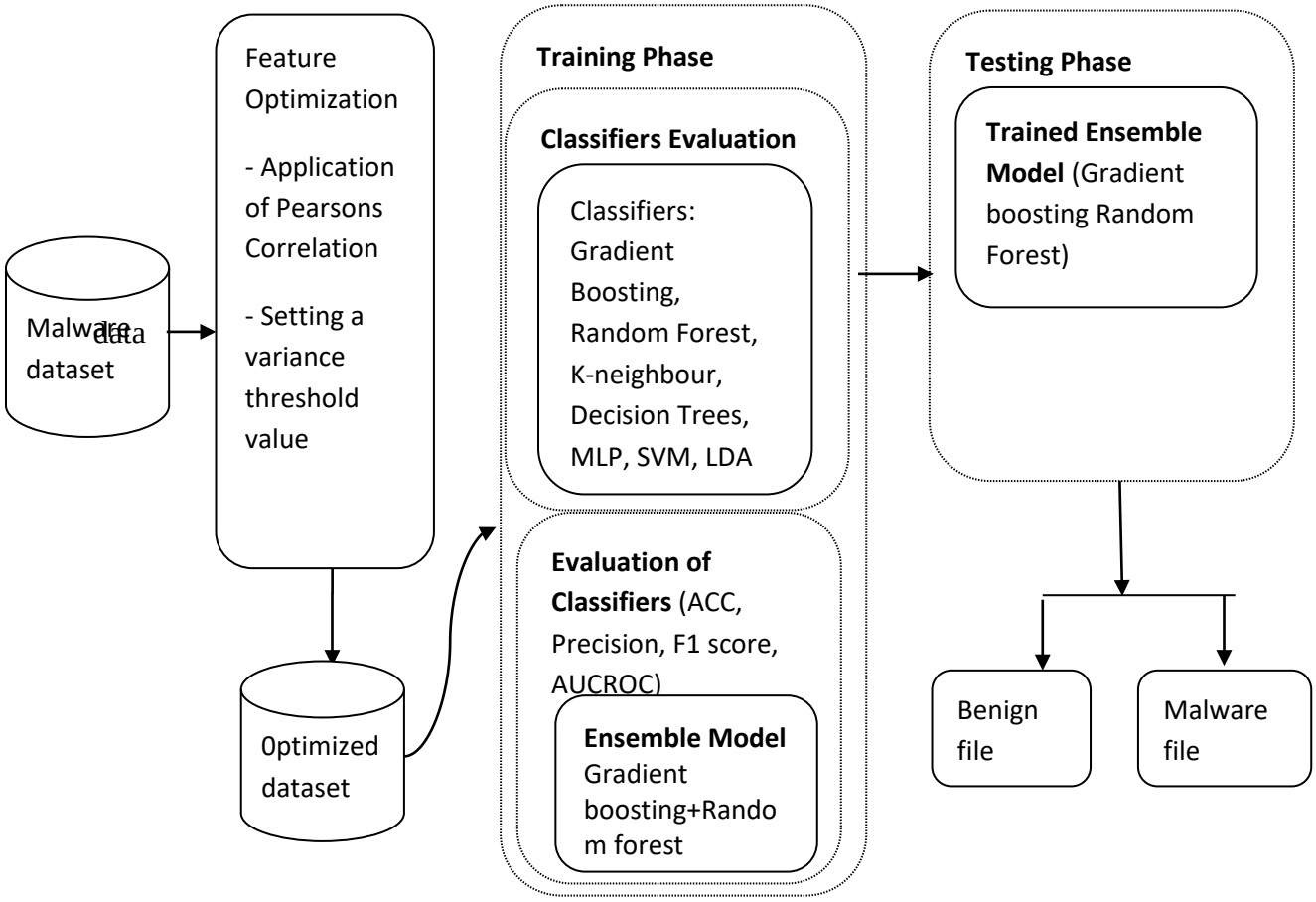

Figure 3.1: Methodology for an ensemble based model for the detection of Obfuscated Malware

3.2 Overview

The main goal of feature selection is to obtain features that would increase the efficiency of the Ensemble by selecting features that contribute strongly towards predictions. There was a total of 56 features that define the malware dataset. Reducing the dimensionality of the features will prevent the Ensemble model from overfitting and also reduce the computational time.

This research applied a baseline threshold variance. For analysis of baseline variance, baseline values need to be accurate in this case all the 96,724 malware files are accurate. Calculating the threshold will be based upon the features for the Portable executable header files for the each corresponding malware file by removing all low variance features. Features with a training set variance lower than the set threshold will be removed.

The idea is when a feature doesn't have a variation much within itself; it generally has very little predictive power (low variance) and it will be removed by the application of the threshold variance value. The application of sklearn. Ensemble library will remove features with a training set variance lower than the set threshold. The set Threshold value was 0.5 which means that any feature with a variance less than 0.5 will be removed. The choice of variance threshold was motivated by the literature review. The next step which is to improve the accuracy by using the Filter method through the application of Pearson Correlation. The Pearson correlation coefficient is symmetric: corr(X.Y) = corr(Y,X) Pearson correlation is the measure of the linear correlation between two variables in this case the two major variables derived from the malware dataset in section (3.5). It finds the mutual relationship or connection between the headers for the benign variable and the malware variable. The experiment selected features which have a correlation above 0.5 (factoring the absolute value) with the output variable. The correlation coefficient has

values between -1 to 1: (i.) A value closer to 0 implies weaker downhill correlation (exact 0 implying no correlation)

(ii) A value closer to 1 implies stronger uphill positive correlation

(iii.) A value closer to -1 implies stronger downhill negative correlation

The application of sklearn, feature selection library with a set correlation of 0.5 will further more reduce the dimensionality of the malware dataset features.

When training a model to handle a classification problem, a function is gained that takes an input and returns an output which is directly defined with respect to the training dataset. Owing to the theoretical variance of the training dataset, there exists variability to the fitted dataset therefore we want to fit several independent models and "average" their predictions in order to obtain an Ensemble model with a lower variance. However to fit fully independent models will require too much data therefore we depend on the good approximate properties of bootstrapping which assigns measures if accuracy like bias and variance to sample estimates to fit models that are almost independent. The initial step is to create multiple bootstrap samples in such a way as to make each new bootstrap sample to act as another independent dataset drawn from a true distribution. The Experiment can now further fit a weak learner for each of these samples and aggregate them such that an average for all their outputs is obtained therefore obtaining an ensemble model that has the characteristics with less variance than its elements. As the bootstrap samples are approximatively independent and identically distributed so are the learned base models so when we apply averaging on the weak learners outputs, the outputs do not change the expected results but hence reduce its variance similar to the case of averaging independent and identically distributed random variables retains the expected value but reduce variance.

So, assuming that there are dataset samples which is the approximations of independent datasets of size A and 7 machine learning classifiers denoted by

$\{E_{11},E_{21,\ldots,}E_{A1}\}, \{E_{12},E_{22,\ldots,}E_{A2}\}, \ldots, \{E_{17},E_{27,\ldots,}E_{A7}\}$ $E_{a7} \equiv a -$

$th\ observation\ of\ the\ a -\ th\ dataset\ sample$……. (i)

The experiment can now confidently fit 7 independent weak learners by fitting one on each dataset

$$dt_1, dt_2, \ldots, dt_7 \ldots \ldots \ldots \ldots \ldots \ldots \ldots \text{(ii)}$$

The experiment can now then aggregate results of the models into some kind of averaging process to get an ensemble model with a lower variance. Furthermore, we can define our strong model such that

$z_a = \arg\max[card(a|dt$

$$= k \quad (simple\ majority\ vote, for\ classification\ problems) \ldots\ldots \text{(iii)}$$

Fitting the model will now present several possible ways to aggregate the multiple models fitted in parallel for a classification problem but the experiment can check all the probabilities of each classes returned by all the models that can further average these probabilities and keep the class with the highest average probability which is known as soft voting. The averages or votes can either be simple or weighted if any relevant weights can be used weights being the proportional trust or performance of ensemble members on a dataset. **3.2.1 Data Sources**

As stated earlier the dataset was retrieved from https://github.com/chihebchebbi/Masteri ng-Machine-Learning-for-Penetration

Testing/blob/master/Chapter03/MalwareData.csv.gz containing 138,047 PE header file records samples which were divided into 41, 323 clean files containing exe and dll file samples and 96,724 malware file samples. The malware dataset comprises of a total of

features 56 features which is the information found in every sample that define the malware datasets as either malicious or benign.

3.3 Model Training

The training phase is the backbone of the Ensemble model. The purpose of the training phase is to evaluate the performance of different nominee classifiers. A total of ten different classifiers belonging to different learning models were used as nominee classifiers. The training phase as shown in Figure 3.1 was divided into two phases namely classifiers evaluation, and classifiers selection respectively. In classifiers evaluation, all nominee classifiers are trained using the same training dataset set after feature selection. The most predictive N classifiers are selected and combined into an ensemble model using the weighted voting framework. If N = 2 nominee classifiers.

Furthermore the choice of N is depending on the learning accuracy threshold value which means if a classifier has a predictive learning accuracy above 95 percent that nominee classifier is chosen during experimentation. In the classifiers selection phase, the formed ensemble model is also trained using the same training set as carried out in the classifiers evaluation phase. Performance of the ensemble model is also evaluated.

The output model of ensemble represents the final training model.

3.3.1 Train, Test and Split Variables

The tests and experiments was conducted using python 3 server and several dependencies explained in the previous sub section. In machine learning data separation is crucial so as to differentiate the benign files from the malware files in order to make training activities suitable to train the classifier to predict the malware files from the benign files. Data Separation shall be achieved by using the code snippet below which intends to create two variables separating the malware files from the benign files:

```
legitimate= = FUTmalData[0:41323].drop(["legitimate"], axis
```

```
1) FUTmalware = FUTmalData[41324::].drop(["legitimate"], axis
= 1)
```

Two major variables shall be obtained containing the legitimate files and the malware files.

In training the nominee classifiers, we prepare variables that will be used for training and testing the nominee classifiers in the Ensemble model. The first step in achieving this is to split the two major containing the Malware files and the benign files into smaller manageable bits. The intended output for this stage will be 4 variables namely benign_test, benign_train, malware_test, malware_train respectively. Table 3.1 gives a brief of the variables:

Table 3.1: Description of the four training/test variables

S/N	Variable Name	Description
1.	Benign_train	This variable will be employed to train the nominee classifiers to predict benign files, it is made up of 80 percent of the benign files.
2.	Benign_test	This variable will be employed to test the predictive performance of the nominee classifiers against data it hasn't seen. It is 20 percent of the benign files in the malware dataset
3.	Malware_train	This variable will be employed to train the nominee classifiers to predict malware files it is made up of 80 percent of the malware files.
4.	Malware_test	This variable will be employed to test the predictive performance of the nominee classifiers against data it hasn't seen. It is 20 percent of the malware files in the malware dataset.

The application of sklearn model selection library will achieve the splitting of the variables.

In addition cross validation will be carried out after training the nominee classifiers; this is done in section 3.5.

3.3.2 Obfuscated malware dataset

The next step in the research is to obfuscate the malware dataset using a DOSfuscator to obfuscate the original malware dataset creating an obfuscated malware dataset. The aim is to get an obfuscated version of the original malware dataset.

As earlier stated in the literature review, Invoke-DOSfuscator is a python script that can create and obfuscate PE header files in this case, the original malware dataset. This tool is key in this research work to obfuscate the original malware dataset to obtain an obfuscated version of the original malware dataset. The malware dataset will have 41,323 clean files containing exe and dll file samples and 96,724 obfuscated malware file sample after obfuscation. Table 3.1 shows an intended sample single PE header file information after obfuscating the original malware dataset attack vector created using invoke DOSfuscator

	Header File		**Description**
1	SizeOfOptionalHeader	332	This is required for executable files not object files.
2	MajorLinkerVersion	621568	This tells the OS the major DLL to use when running the exe.
3	MinorLinkerVersion	4.426324	This tells the OS the minor DLL to use when running the exe.

4	ResourcesMeanEntropy	2.846449	The memory location for the secrect resource needed to run the exe.
5	LoadConfigurationSize	270376	Memory location to load the configuration files to run the exe .

S/N	Feature

Table 3.2: Obfuscated Malware Header File

The next phase in the research work is the same procedure as outlined in (3.4) where the intended optimization of the new malware dataset is carried out to obtain key features to use in training the nominee classifiers.

6	LoaderFlag	('546869785432673206A75737420 7269646963756C6F75732E2E2E' split '(?<=\G.{2})',26|%{[char][int] "0x$_"})	This argument tells the operating system to split a hard disk sector. This is one of the obfuscated header by Dosfuscator.
7	SizeOfStackCommit	3110	Major Size of the exe to save on disk.

3.3.3 Feature Selection/Optimization

Section 3.1 discussed about feature selection and revealed the two aspects of feature selection which was carried out in this research to reduce the dimensionality of the obfuscated malware dataset and obtain only the most significant features. In the dataset, the original number of features is 56. The first step in feature reduction was to apply a variance threshold to the obfuscated malware dataset; Experimental results marked 36 features to be significant after the application of a variance threshold on the malware dataset. The idea is when a feature doesn't have a variation much within itself; it generally

has very little predictive power (low variance) and it will be removed by the application of the sklearn ensemble library. Table 3.3 shows 36 features that are classified as important by the application of the sklearn ensemble library.

Table 3.3: 36 Important features after removal of features with low variance

S/N 1	Feature Name	S/N	Feature Name	S/N	Feature Name
	Name	10	SizeOfUninitializedData	19	SizeOfStackReserve
2	md5	11	AddressOfEntryPoint	20	SectionsMeanEntropy
3	Machine	12	BaseOfCode	21	SectionsMinEntropy
4	SizeOfOptionalHeader	13	BaseOfData	22	SectionsMaxEntropy
5	Characteristics	14	ImageBase	23	SectionsMeanRawsize
6	MajorLinkerVersion	15	SectionAlignment	24	ResourcesMaxSize
7	MinorLinkerVersion	16	SizeOfStackCommit	25	ResourcesMinSize
8	SizeOfCode	17	SizeOfHeapReserve	26	SizeOfUninitializedData
9	SizeOfInitializedData	18	DllCharacteristics	27	SizeOfInitializedData
28	VersionInformationSize	31	SectionsMinRawsize	34	ResourcesMinEntropy
29	LoaderFlag	32	SectionMaxRawsize	35	ResourcesMaxEntropy
30	NumberOfRvaAndSizes	33	SectionsMinVirtualsize	36	MinorImageVersion

The features obtained after selecting features with high variance constitute about 66% of all 56 features. The reduction of features further decreases the complexity of the experiments using the ensemble model.

The second step in feature reduction is the application of the Filter method which will filter the 36 features with high variance. This research achieved this by applying Pearsons Correlation as stated in section 3.3.2. The idea is to further optimize the 36 features to obtain key features that will aid the ensemble model to achieve more accurate predictions

in detecting obfuscated malware. Table 3.4 shows the results obtained after the application of Pearsons Correlation.

Table 3.4: Key features obtained after optimization of high variance features

S/N	Feature Name	S/N	Feature Name
1	LoaderFlags	7	ResourcesMaxSize
2	NumberOfRvaAndSizes	8	SectionMaxRawsize
3	SizeOfHeapCommit	9	SectionsMeanRawsize
4	SizeOfUninitializedData	10	SizeOfImage
5	BaseOfCode	11	ResourcesMeanSize
6	SizeOfStackCommit	12	SizeOfCode

3.4 Model Testing

After the training phase of the model, 2 nominee classifiers were selected based on the set accuracy threshold value as earlier stated. The model testing stage is where the experiment is carried out with the 2 nominee classifier is now called the Ensemble model comprising of the Random Forest and Gradient Boosting classifiers. The experiment will now test the model using the obfuscated malware dataset with splitting ratio (80 / 20) which the Ensemble model has never seen. Testing is the last step of the Ensemble model. The main aim is to evaluate the Ensemble model performance in terms of measuring the prediction accuracy in classifying new benign and malware samples with splitting ratio (80 / 20) as stated above. For the purpose of testing, the experiment 8264 benign and 19344 malware

samples from the obfuscated malware dataset to be used as test samples which accounts for 20 percent each from the benign and malware files respectively.

As stated in section 3.2 voting and averaging are two methods employed to combine ensembles. Voting is normally used for classification algorithms and averaging is used for regression algorithms. In our experiment, we shall use the Majority weighted voting method to combine the Gradient Boosting and the Random Forest algorithm which are both classification algorithms, the final prediction decision of the ensemble model relied on voting. There are two major optional strategies for voting which are majority voting and weighted voting. In majority voting each classifier makes a prediction for each test instance and the final result for prediction depends on the classifier on the combination of the prediction accuracy for the two classifiers. However, in weighted voting, the average is taken for the prediction of the better models multiple times depending on the weight the researcher chooses. The experiment shall use the weighted voting strategy and set a weight depending on how large our dataset is and time taken to complete prediction.

3.4 Machine Learning Evaluation Metrics

To prove that the ensemble model accurately predicted an obfuscated malware, evaluation metrics shall be used to determine the prediction accuracy of a classifier. This research intends to use the following evaluation metrics explained below:

1.) **Accuracy:** provides general information about how many obfuscated malware samples are misclassified. Accuracy is calculated as the sum of correct predictions divided by the total number of predictions as show in equation iv

$$\boldsymbol{ACC} = \frac{\boldsymbol{TP+TN}}{\boldsymbol{TP+FP+FN+TN}} \quad \ldots\ldots\ldots \text{(iv)}$$

2.) **Precision Metrics:** This is used when the aim is to limit the number of false positives measured from highest to lowest which means the classifier with the highest precision score has the best precision and can be used in cases where false positive reduction is the aim. Precision can further be defined as the true positive class out of the predicted malware labels. This research intends to use the precision metrics to gauge the precision of the Ensemble Model to correctly predict obfuscated malware instances against a dataset that the classifier has not been trained on namely the malware_test variable dataset as show in equation v.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FP}} \ldots\ldots\ldots (v)$$

3.) **Recall Metrics:** This is the direct opposite of the Precision Metrics. It is mostly used when the aim of the research is to reduce the number of false negatives; it measures the predictive ability of the classifier to find all true samples/benign labels. It is also called sensitivity or true positive rate because it is the ratio of tp / (tp + fn) using the confusion matrix where tp is the number of true positives and fn is the number of false negatives. This research intends to use the recall metrics to measure the Ensemble Model's ability to predict obfuscated malware instances against the total number of true obfuscated malware instances as show in equation vi.

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

4.) **F1_SCORE:** This optimizes both precision and recall metrics. It is the numerical mean average between a set of positive variables; in this case it is the mean between precision and recall. It is used as a more verbose measure of a classifier's accuracy which goes beyond simply correctly classified obfuscated instances.

This research intends to use the F1_Score evaluation metrics to find the harmonic mean between the precision metrics and the recall metrics to understand how accurate the Ensemble Model is in predicting obfuscated malware as show in equation vii.

$$\boldsymbol{F1\ Score} = \frac{\boldsymbol{2*(Recall*Precision)}}{\boldsymbol{Recall+Precision}} \ldots\ldots\ldots \text{(vii)}$$

5.) **AUCROC:** Area Under the Receiver Operating Characteristics **(**AUCROC) is the possible thresholds that are considered during the Receiver Operating Characteristics (ROC) curve. The AUCROC is one of the most important evaluation metrics for checking the predictive performance of a classifier because the diagonal of an AUCROC graph can be interpreted by considering the classifiers performance according to its position below or above the diagonal in an ROC curve plot, for example if the Gradient Boosting classifiers performance falls into the top left corner of the AUCROC with a True Positive Rate of 1 and a False Positive rate of 0 means the classifier has a perfect predictive capability; the higher the AUC, the better the Ensemble Model is at predicting malware as malware and a benign file as a benign file. In the ROC curve plot, the True Positive Rate (TPR) against the False Positive Rate (FPR) is plotted with the TPR on the y-axis and the FPR on the x-axis. A perfect classifier has an AUCROC near to 1 meaning it has a good measure of classification and a poor classifier has an AUCROC near to zero. Whenever the AUCROC is 0.5 it means the Ensemble Model has no predictive capability for classification.

CHAPTER FOUR

4.0 RESULTS AND DISCUSSION

4.1 Classifier Performance Evaluation

In ensemble learning aspect of machine learning, it is necessary to build predictive models that have the ability to generalize effectively the extracted features. When training a model, a disturbed generalization is recognized by over fitting. A major workaround to avoid over fitting is to use an appropriate data splitting techniques. Classification is a process of supervised learning in which the computer program learns from the input data and uses the learning to classify new observations or behaviours. To avoid the poor generalization for the ensemble model experiments were carried out with different splitting ratios for training and testing, the goal is to train and test the nominee classifiers to learn behaviours and patterns accurately and to use that learning to make accurate

predictions. It is worthy of note that ensemble learning models could be ranked according to their efficiency; efficiency is the inherent capability of an ensemble learning model to generalize accurately to new data. To avoid poor generalization, the nominee classifiers are experimented using two splitting criteria with different ratios. The first experiment is done by splitting the optimized dataset into 80/20 for training and testing in the same way the second experiment is done by splitting the optimized dataset into 70/30 for training and testing respectively. Evaluation metrics such as accuracy, precision, recall, and F-measure are used to evaluate the nominee classifiers performance.

The results of the nominee classifiers performance for the 80/20 and 70/30 splitting criteria are summarized in Table 4.3 and Table 4.4 respectively. Results showed that the nominee classifiers performance revealed similar results with little differences. The splitting criteria didn't influence the classification decision of the nominee classifiers.

This research experimented with different ensemble classifiers like the Gradient Boosting, Random Forest, K-nearest Neighbour (KNN), Multi-layer perceptron (MLP) and Decision Trees (DT). In terms of accuracy according to the accuracy selection threshold value for our Ensemble Model during the selection phase, the Gradient Boosting Classifier and Random Forest classifier gave the best result among all other nominee classifiers. Other classifiers like MLP, Decision Trees and K-nearest neighbour also showed considerable high accurate results but not up to accuracy selection threshold value. Further experiments were carried out using a combination of non-ensemble classifiers with the nominee ensemble classifiers to measure whether it may affect the accuracy of the ensemble model. The ensemble model will be formed with the following two classifiers Gradient Boosting Classifier and the Random Forest classifier through the strategy of voting as stated in section 3.5.

Table 4.1: Classifier training result with splitting ratio (80 / 20)

Classifier	Accuracy	Precision	Recall	F1-Score	AUCROC
Gradient Boosting	96.8	96.7	96.4	96.6	0.7
Random Forest	96.5	96.4	96.0	96.4	0.68
Decision Tress	96.5	96.3	95.9	96.4	0.66
K-Neighbour	92.8	92.4	92.3	92.3	0.52
MLP	92.7	92.6	92.3	92.2	0.54

Table 4.2: Classifier training result with splitting ratio (70 / 30)

Classifier	Accuracy	Precision	Recall	F1-Score	AUCROC
Gradient Boosting	96.4	96.4	96.3	96.2	0.7
Random Forest	95.9	95.7	95.2	95.2	0.63
Decision Trees	93.3	93.2	93.4	93.2	0.56
K-Neighbour	92.8	92.6	92.4	92.5	0.53
MLP	92.6	92.6	92.3	92.0	0.54

4.2 Ensemble Model Performance Evaluation

This research carried out 3 different experiments to evaluate the performance of the ensemble model. The first experiment evaluates the performance of each of the ensemble model against individual ensemble classifiers and non-ensemble classifiers. The second experiment evaluates the ensemble model when integrating two ensemble models which are the ensemble model + Decision Trees. The third experiment evaluates the combination of three ensemble models which are the ensemble model + Decision

Trees + K-Neighbour. Evaluation of the ensemble model is showed in Table 4.3. Experimentation revealed that the performance of the ensemble model is slightly better

than the individual non-ensemble classifiers. Experimental results demonstrated that the ensemble model gave a similar accuracy to the Random forest and Gradient Boosting with a rate of 98.7%. It was observed that the performance of the ensemble model showed a minor accuracy difference to the Gradient Boosting classifier that returned accuracy rate of 96.4. However, one of the major drawbacks of the Gradient Boosting is over fitting which the ensemble model solves because ensembles classify more accurately when combined with reduced dimensionality in the features.

Table 4.3: Performance evaluation of Ensemble model compared to other classifiers

Classifier	Accuracy	Precision	Recall	F1-Score	AUCROC
Ensemble Model (Gradient Boosting+Random Forest)	98.7	98.4	98.8	98.4	0.8
Gradient Boosting	96.4	96.4	96.3	96.2	0.7
Random Forest	95.9	95.7	95.2	95.2	0.63
Decision Trees	93.3	93.2	93.4	93.2	0.56
K-Neighbour	92.8	92.6	92.4	92.5	0.53
MLP	92.6	92.6	92.3	92.0	0.54
SVM	92.3	92.7	92.2	92.1	0.53
LDA	91.8	90.9	91.3	91.6	0.52

Table 4.4 shows the result of combining the ensemble model with the Decision tree classifier. The combination achieved more accuracy results compared to the accuracy value obtained from the initial ensemble model. Table 4.5 shows the result of combining the ensemble model with two other classifiers namely the Decision Tree and the KNeigbour classifiers respectively.

Table 4.4: Performance evaluation of Ensemble Model combined with another classifier

Classifier	Accuracy	Precision	Recall	F1-Score	AUCROC
Ensemble Model (Gradient Boosting+Random Forest)+Decision Tree	98.8	98.6	98.9	98.6	0.8
Gradient Boosting	96.4	96.4	96.3	96.2	0.7
Random Forest	95.9	95.7	95.2	95.2	0.63
K-Neighbour	92.8	92.6	92.4	92.5	0.53
MLP	92.6	92.6	92.3	92.0	0.54
SVM	92.3	92.7	92.2	92.1	0.53
LDA	91.8	90.9	91.3	91.6	0.52

Table 4.5: Performance evaluation of Ensemble combined with two other classifiers

Classifier	Accuracy	Precision	Recall	F1-Score	AUCROC
Ensemble Model(Gradient Boosting+Random Forest)+Decision Tree + KNeighbor	98.7	98.9	98.9	98.8	0.8
Gradient Boosting	96.4	96.4	96.3	96.2	0.7
Random Forest	95.9	95.7	95.2	95.2	0.63
MLP	92.6	92.6	92.3	92.0	0.54
SVM	92.3	92.7	92.2	92.1	0.53
LDA	91.8	90.9	91.3	91.6	0.52

4.3 Accuracy Performance Metrics

As mentioned in section 3.6, accuracy performance metrics provides general information about how many samples are misclassified Figure 4.1 shows a bar chart for the accuracy performance metrics of the Ensemble model against other classifiers.

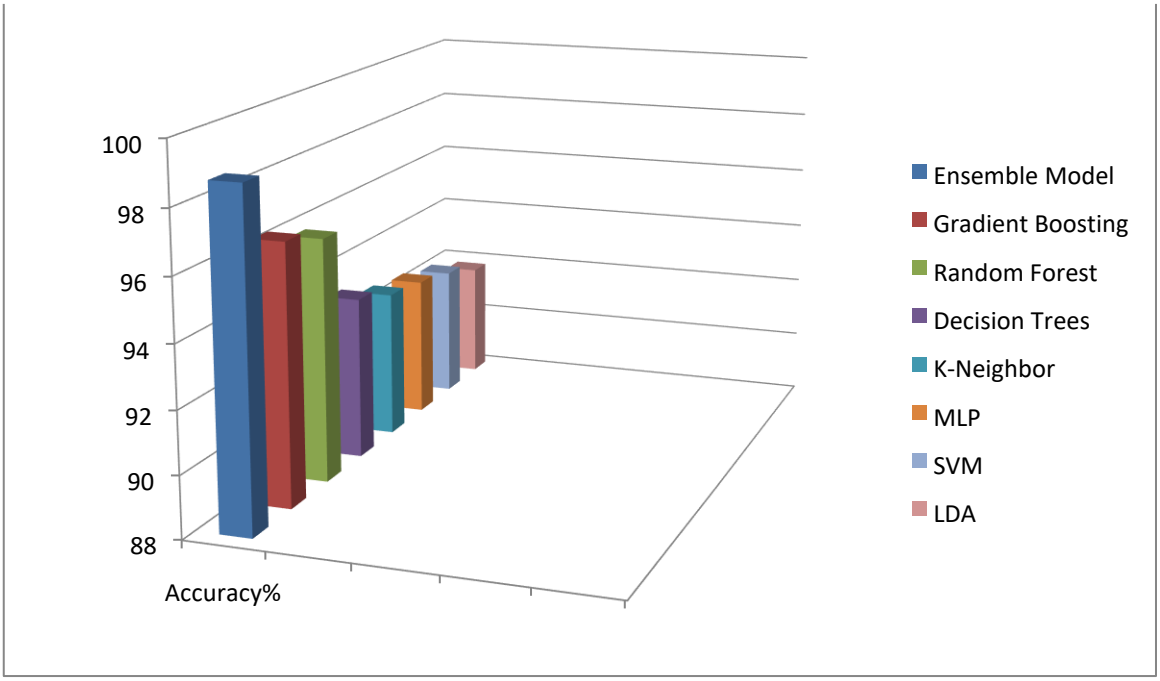

Figure 4.1: Accuracy performance metrics for the ensemble model against other classifiers

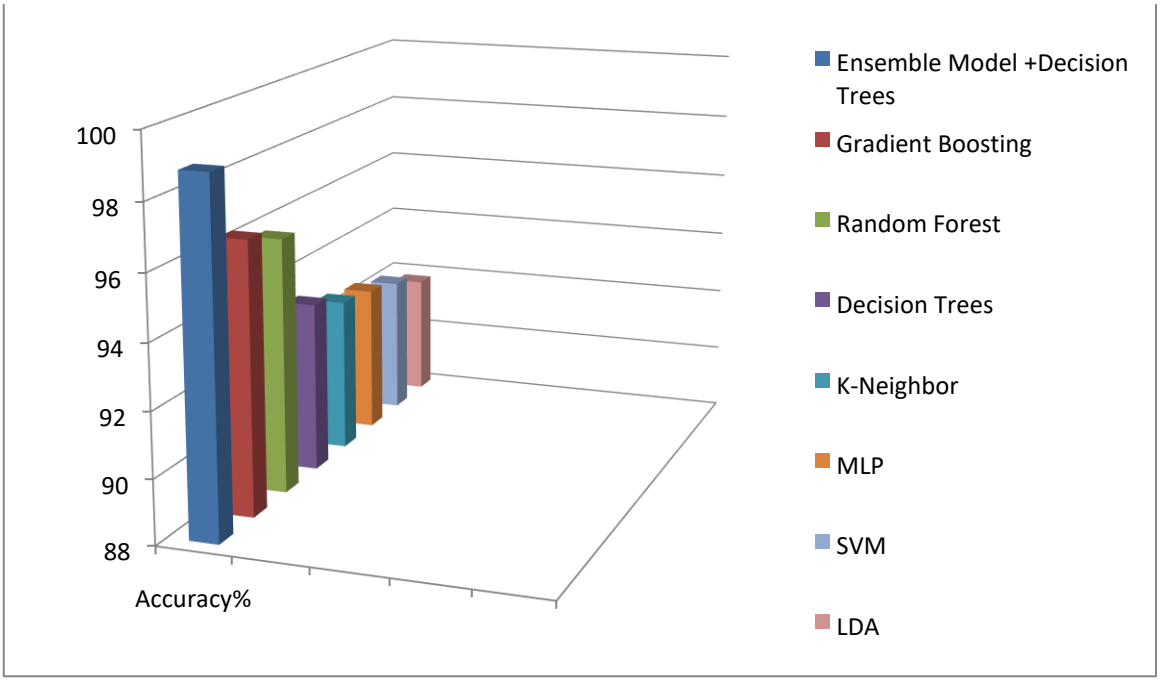

Figure 4.2: Accuracy performance metrics after combining the ensemble model with the Decision Tree classifier

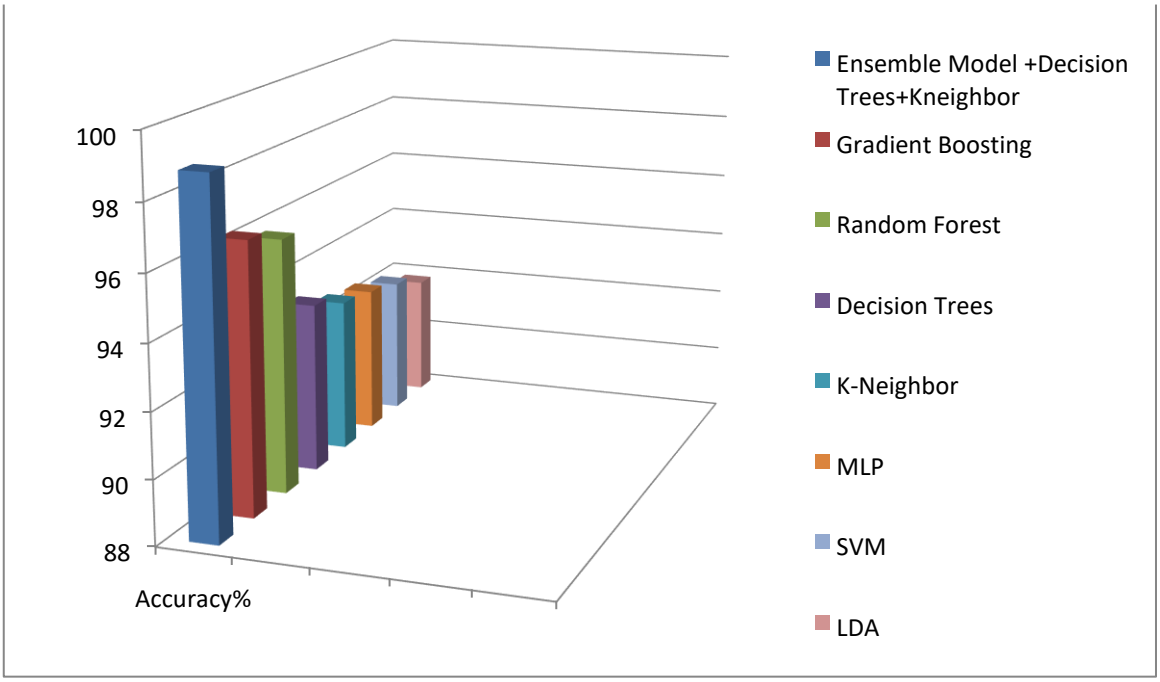

Figure 4.3: Accuracy performance metrics after combining the ensemble model with the Decision Tree and the KNeighbor classifier

Results revealed that the accuracy value returned from combining the ensemble model with two other classifiers is similar to the accuracy obtained from table 4.6 when the ensemble model was combined with one other classifier implying that out of the 96,724 obfuscated malware instances, the ensemble Model correctly predicted 95,563 samples. The experimental conclusion was that the combination of multiple ensemble models can enhance the accuracy of detecting obfuscated malware in general; however, the performance of combining two or more classifiers to an already existing Ensemble model may not increase the accuracy of prediction.

4.4 Precision Performance Metrics

Precision can be defined as the true positive class out of the predicted malware labels. Figure 4.4, 4.5 and 4.6 show bar charts for the precision performance metrics for the ensemble model.

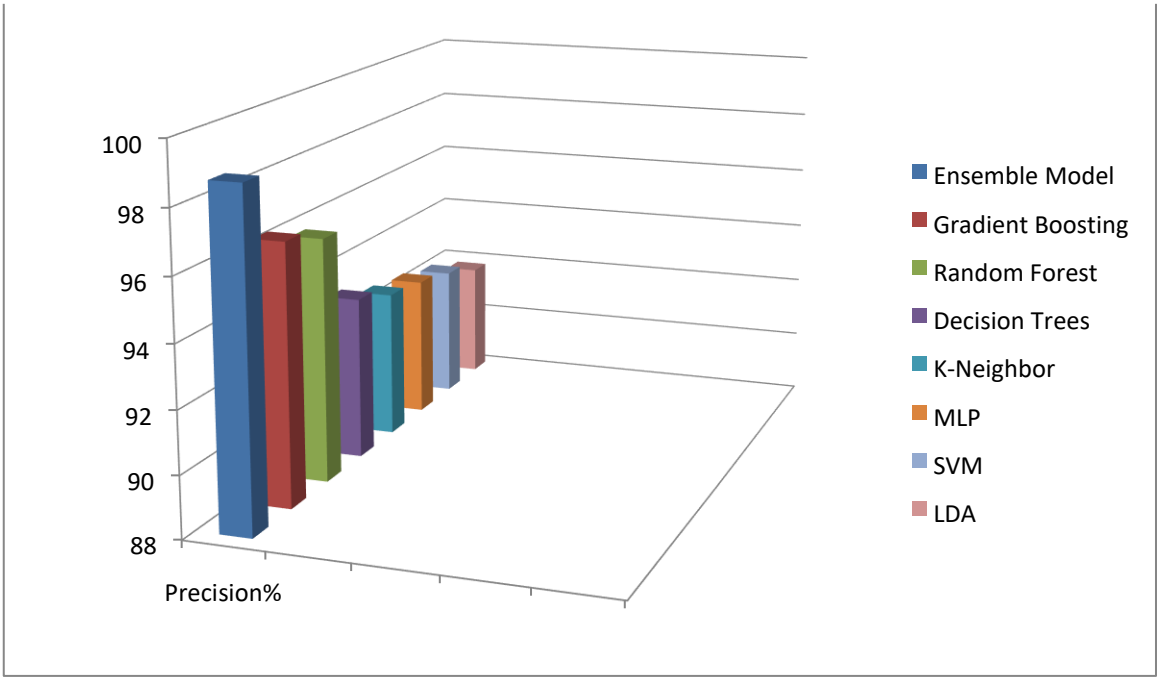

Figure 4.4: Precision performance metrics for the ensemble model against other classifiers

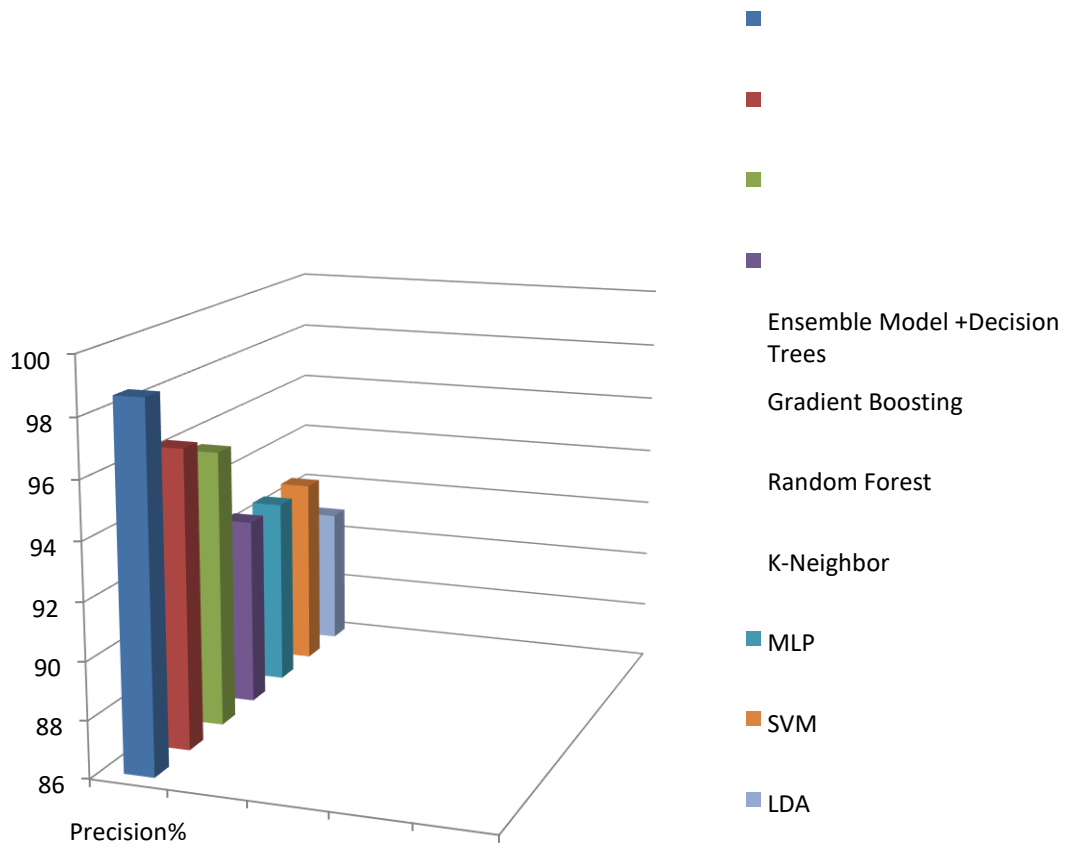

Figure 4.5: Precision performance metrics after combining the ensemble model with the Decision Tree classifier

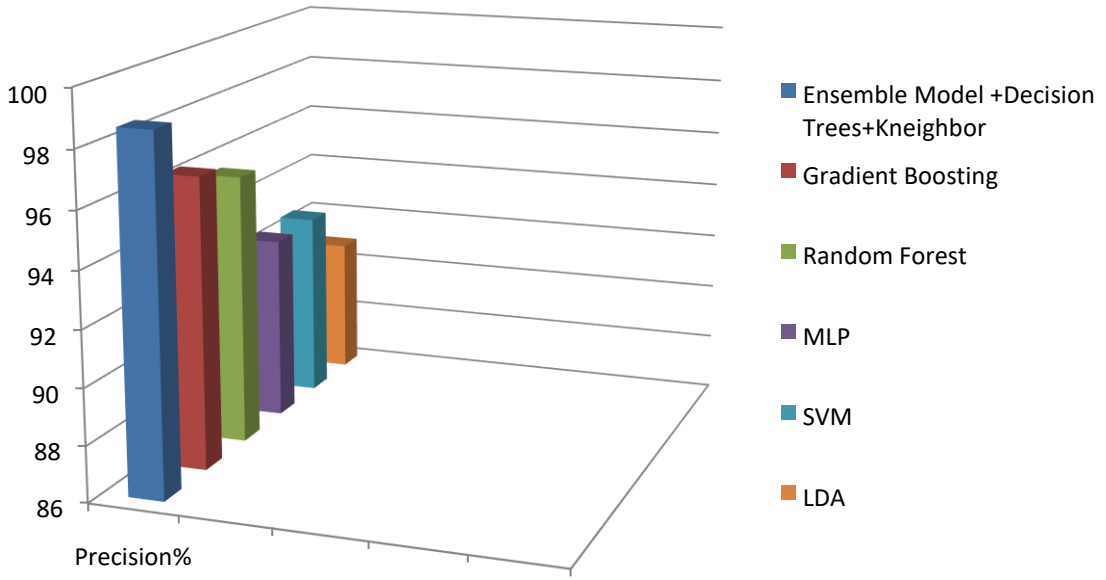

Figure 4.6: Precision performance metrics after combining the ensemble model with the Decision Tree and the KNeighbor classifiers

The ensemble model precision predictive output was at 98.6 percent this means that out of 96,724 malware instances the ensemble model predicted 1355 instances as false positives and 95,369 obfuscated malware instance as true positives which is relatively good.

4.5 Recall Performance Metrics

This is the direct opposite of the precision metrics. It is mostly used when the aim of the research is to reduce the number of false negatives; it measures the predictive ability of the classifier to find all true samples/benign labels. Figure 4.7, 4.8 and 4.9 show bar charts for the recall performance metrics for the ensemble model.

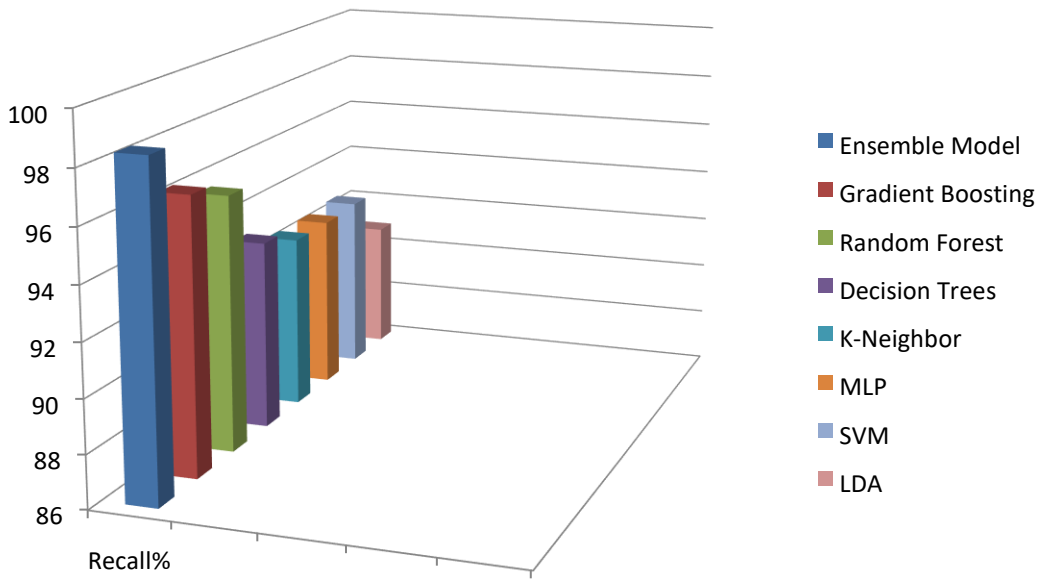

Figure 4.7: Recall performance metrics for the ensemble model against other classifiers

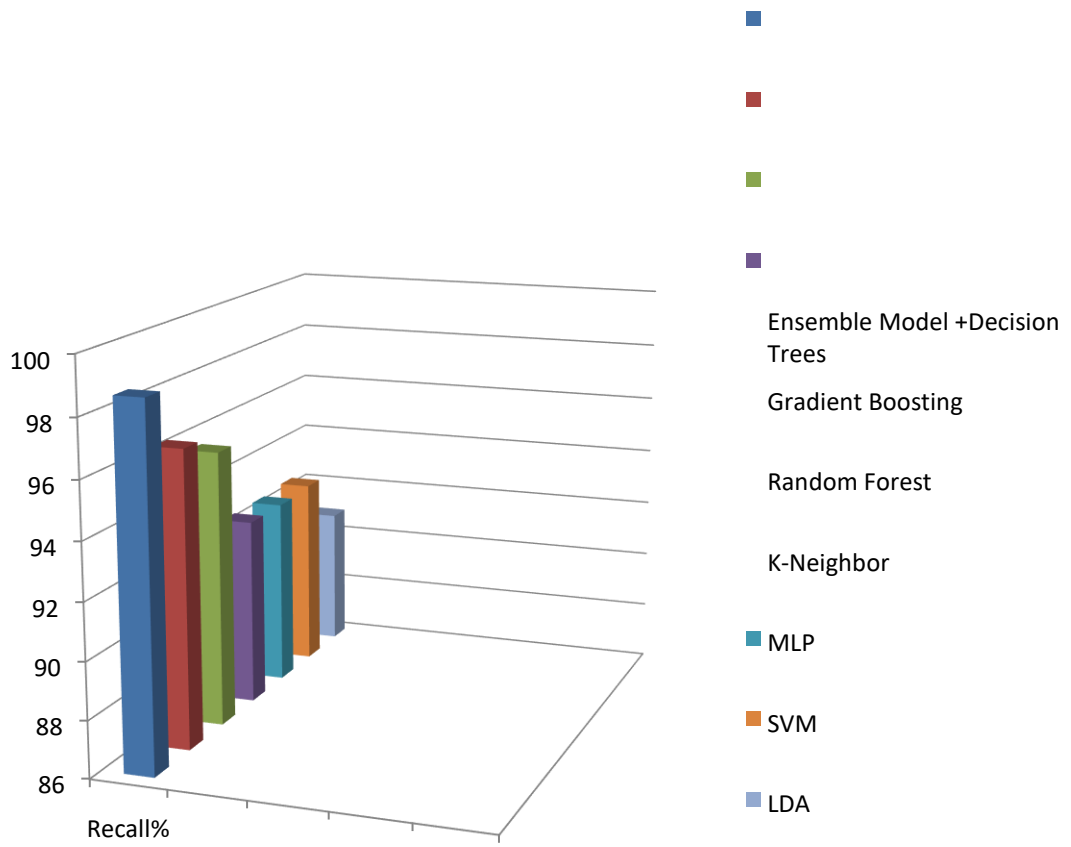

Figure 4.8: Recall performance metrics after combining the ensemble model with the Decision Tree classifier

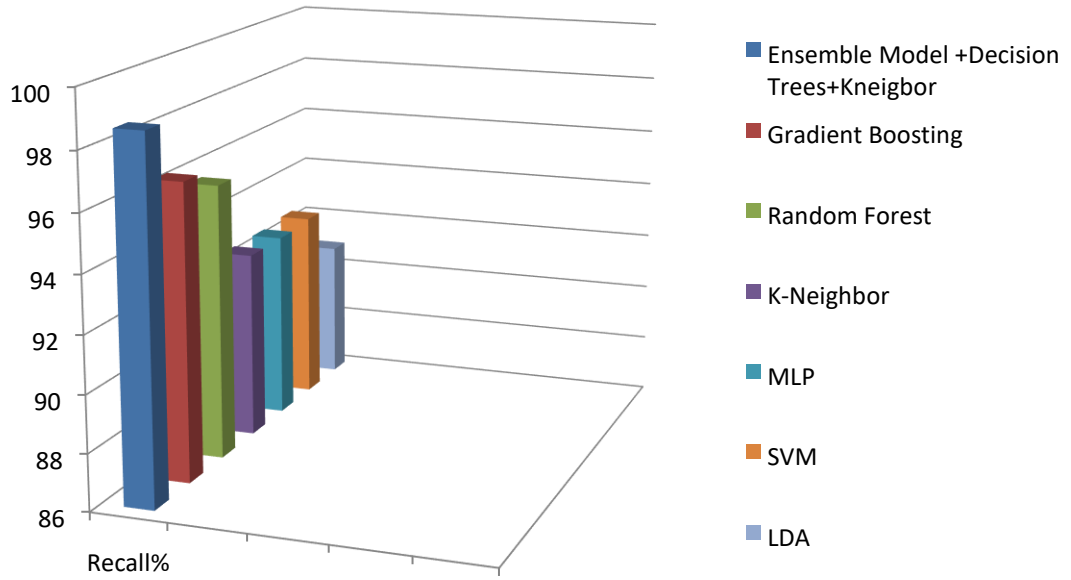

Figure 4.9: Recall performance metrics after combining the ensemble model with the Decision Tree and the KNeighbor classifiers

The ensemble model had a recall of 98.9% this implies that the ensemble model predicted 40165 to be benign/true samples out of the 41323 benign/true samples in the obfuscated malware dataset having the highest recall among the other classifiers

4.6 F-1 Score Performance Metrics

This optimizes both precision and recall metrics. It is the numerical mean average between a set of positive variables; in this case it is the mean between precision and recall. The F-1 Score reveals how accurate the ensemble model is in predicting obfuscated malware. Figure 4.10, 4.11 and 4.12 show bar charts for the F1 Score performance metrics for the ensemble model.

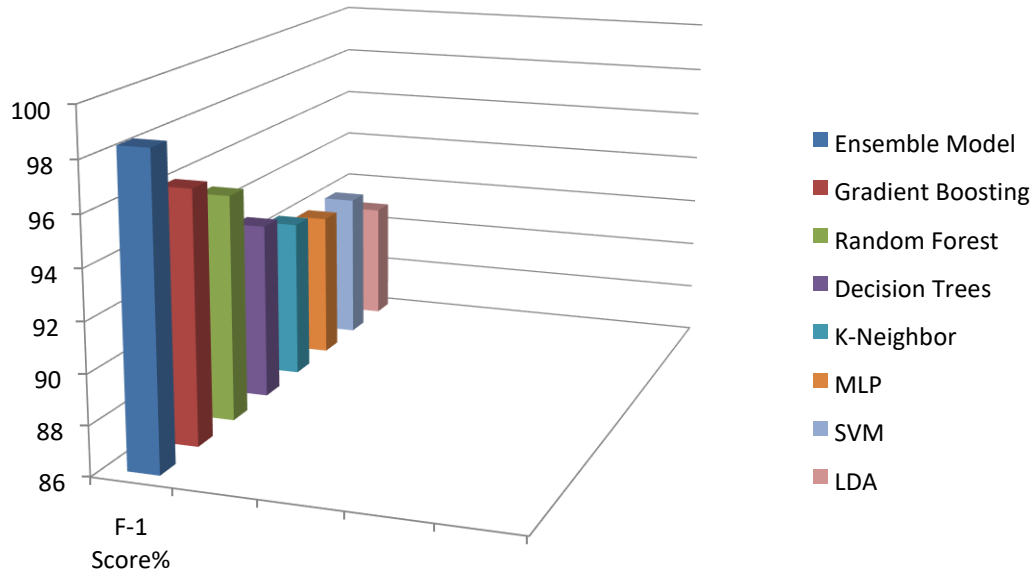

Figure 4.10: F-1 Score performance metrics for the ensemble model against other classifiers

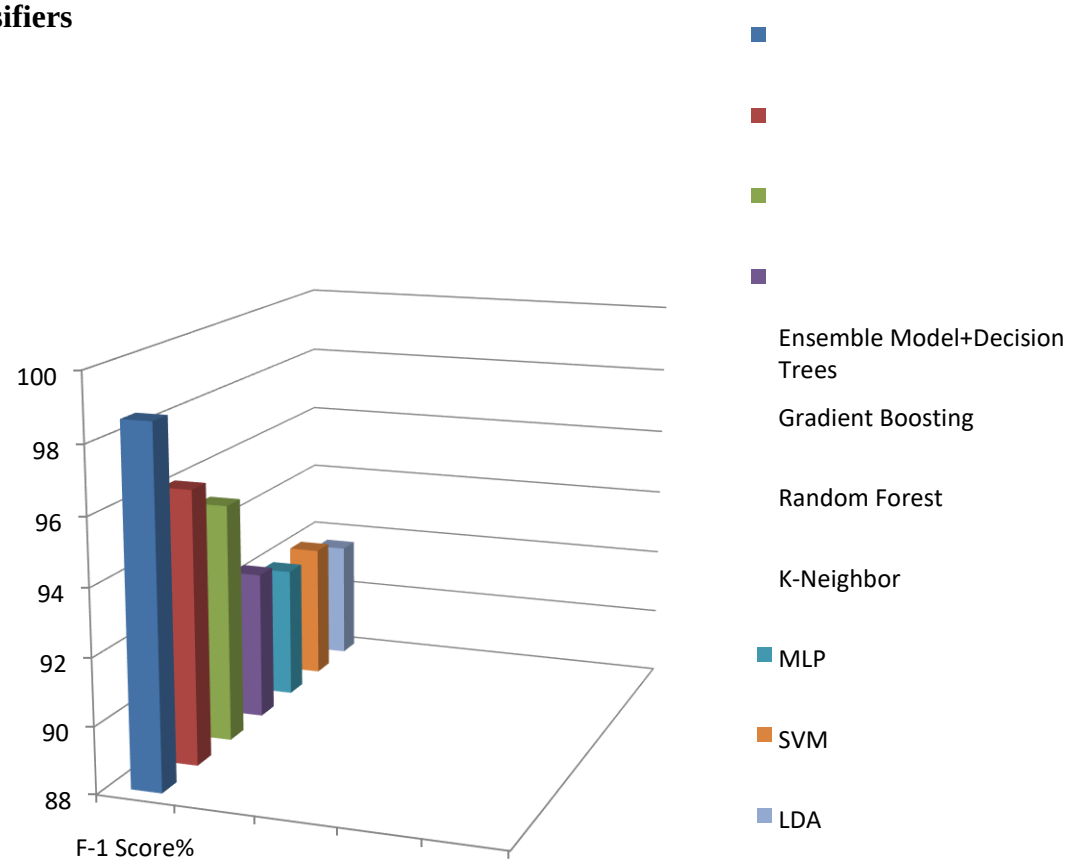

Figure 4.11: F-1 Score performance metrics after combining the ensemble model with the Decision Tree classifier

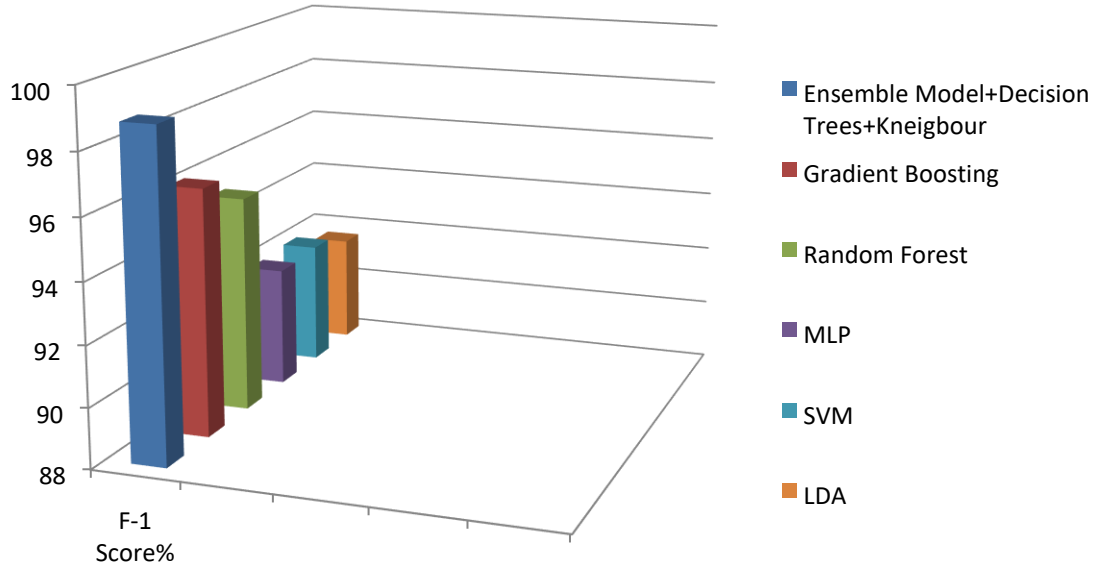

Figure 4.12: F-1 score performance metrics after combining the ensemble model with the Decision Tree and the KNeighbor classifiers

The ensemble model still had the highest F1_score of approximately 98.8% which is 0.98 making the harmonic mean very close to 1. This implies the ensemble model has very good predictive capability according to the harmonic mean.

4.7 AUCROC Score Performance Metrics

The AUC is one of the most important evaluation metrics for checking the predictive performance of a classifier because a perfect classifier has an AUC near to the number '1' meaning it has a good measure of classification and a poor classifier has an AUC near to the number '0'. The Ensemble Model came up with an AUCROC of 0.8 which implies the ensemble model has a very good predictive capability in detecting obfuscated malware according to the area under the curve of the receiver operating characteristics curve (AUCROC).

CHAPTER FIVE

5.0 CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Much of today's malware are created to stealth during infection and operation through obfuscation techniques to stall and prevent removal or behavioural analysis. Furthermore Non-signature based approaches to malware detection can be vulnerable to false positives or false negatives results after detection due to the concept of obfuscation adopted by malware authors. Hence, this research work adopted an ensemble machine learning approach to defeat the setbacks of recent commercial signature based approaches of obfuscated malware detection. By adopting the concept of reducing data feature set dimensionality, the most significant features that characterize obfuscated malware PE files was obtained. The features were experimented and evaluated by different classifiers to choose candidate classifiers to be used in the ensemble model based on an accuracy threshold of 95 percent.

The recorded experiments reveal that the classification accuracy was satisfactory for selecting candidate classifiers to use in the ensemble model by showing experimental comparisons between the efficiency of different classifiers. Furthermore, the recorded research combined experimentally two candidate classifiers to form an ensemble model based on the weighted voting method of combining classifiers.

Experimental results established that the ensemble model was more efficient than individual classifiers. The ensemble model obtained an accuracy rate in identifying unseen obfuscated malware samples with an accuracy ratio of 0.988, F1 Score of 0.988 and area under curve of receiver operating characteristic curve (AUCROC) of 0.8.

5.2 Recommendations

From the experimental findings of this research, the following recommendations were made;

i. More accuracy, tuning, adaptation and reliability can be achieved by the ensemble model with training larger malware samples.

ii. Malware dataset feature dimensionality reduction using other existing reduction methods can be explored to achieve more accurate predictive results from the ensemble model is still attainable.

5.3 Contributions to Knowledge

This research contributed the following to knowledge:

Combining two or more classifiers to form an ensemble model to enhance the predictive ability by classifiers in detecting obfuscated malware in Portable Executables against the baseline literature by Amer *et al.,* (2019); Yan *et al.*, (2018); Scott, (2017); Rubin *et al.*, (2019).

REFERENCES

Piyanuntcharatsr, S. S., Adulkasem, S., & Chantrapornchai, C. (2015). On the Comparison of Malware Detection Methods Using Data Mining with Two Feature Sets. *International Journal of Security and Its Applications, 9*(3), 293–318, doi:10.14257/ijseia.2015.9.3.23

Agnihotri, N. (2018). Ransomware Classifier using Extreme Gradient Boosting. *International Journal of Computer Science and Information Technologies 9*(2), 45–47.

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., & Giacinto, G. (2016). Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. Cornell University Library. arXiv:1511.04317

Amer, E. A., Zelinka, I., & Engineers, E. (2019). An ensemble-based malware detection model using minimum an ensemble-based malware detection model. *Mendel soft computing Journal*, 25, 2-3, doi.org/10.13164/mendel.2019.2.001.

Anderson, B., Quist, D., Neil, J., Storlie, C., & Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in Computer Virology.* 7, 247-258, doi:10.1007/s11416-011-0152-x.

Baset, M. (2017). Machine learning for malware. doi.org/10.13140/RG.2.2.18107.00801.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., & Hamzeh, A. (2013). "A survey on heuristic malware detection techniques," The 5th Conference on Information and Knowledge Technology, 113-120, doi:10.1109/IKT.2013.6620049.

Bohannon, D. (2017.). Techniques & How To (Try To) D " " e ` Tec ` T ’ Th ’ + ’ em ’ Invoke-Obfuscation. Retrievied from https://www.slideshare.net/DanielBohannon2/invokeobfuscation-derbycon-2016

Brahimi, M., & Moussaoui, A. (2015). Machine learning for malware detection using api calls. *Troisième conference internationale sur la vision artificielle*, 3. Retrieved from https://www.researchgate.net/publication/279854349_MACHINE_LEARNING_FOR_MALWARE_DETECTION_USING_API_CALLS

Chen, X., Yang, J., & Liang, J. A (2012). Flexible support vector machine for regression. Neural Computer & Application, 21, 2005–2013, doi:10.1007/s00521011-0623-5

Comar, P. M., Liu, L., Saha, S., Tan, P., & Nucci, A. (2013). Combining Supervised and Unsupervised Learning for Zero-Day Malware Detection. 2022–2030. Retrieved from http://www.cse.msu.edu/~ptan/papers/infocom2013.pdf

Damodaran, A. (2015). *Combining Dynamic and Static Analysis for Malware Detection*. Master's Projects. San Jose State University, California doi: 10.31979/etd.794g-7hf

Derhami, V., Hashemi, S., Mehdi, S., & Fard, H. (2015). Proposing an approach to detect metamorphic mal-ware based on Hidden Markov Model. *2015 4th Iranian Joint Congress on Fuzzy and Intelligent Systems*, 1-5, doi:

10.1109/CFIS.2015.7391648.

Kumar, A. (2017). Analysis of Machine Learning Techniques used in Malware Classification in Cloud Computing Environment. *International Journal of Computer Applications*, 133, 0975 – 8887, doi.org/10.5120/ijca2016908184

Firdausi, I., Lim, C., & Erwin, A. (2010). Analysis of machine learning techniques used in behavior-based malware detection. *Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 201-203

Hlauschek, C., Kirda, E., Krügel, C., & Lamarca, S. (2009). Slides Retrieved from http://cobweb.cs.uga.edu/~perdisci/CSCI6900-F10/SLaMarca_Presentation1.pdf.

Jiang, Q., Zhao, X., & Huang, K. (2011). A feature selection method for malware detection. doi:10.1109/ICINFA.2011.5949122.

Kazanciyan, R., & Hastings, M. (2014). Investigating PowerShell Attacks. *Mandiant*. Retrieved from https://www.blackhat.com/docs/us-14/materials/us-14-KazanciyanInvestigating-Powershell-Attacks-WP.pdf

Ki, Y., Kim, E., & Kim, H. K. (2015). A Novel Approach to Detect Malware Based on API Call Sequence Analysis. Sage Journal, 11, 6, doi.org/10.1155/2015/659101

Kolter, J. Z., & Maloof, M. A. (2006). Learning to Detect and Classify Malicious Executables in the Wild, *Journal of Machine Learning Research*, 7 (2006) 2721-2744. Retrieved from https://www.scirp.org/(S(351jmbntvnsjt1aadkposzje))/reference/ReferencesPapers.aspx?ReferenceID=2132764

Kong, D. (2013). Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification. 357–1365, doi:10.1145/2487575.2488219

Kuriakose, J., & Vinod, P. (2014). Ranked linear discriminant analysis features for metamorphic malware detection. *Souvenir of the 2014 IEEE International Advance Computing Conference*, 112–117, doi.org/10.1109/IAdCC.2014.6779304

Nari, S., & Ghorbani, A. A. (2013). Automated Malware Classification based on Network Behavior. *International Conference on Computing, Networking and Communication*, 642-647, doi: 10.1109/ICCNC.2013.6504162.

Olalere, M., Abdullah, M. T., Mahmod, R., & Abdullah, A. (2016). Identification and Evaluation of Discriminative Lexical Features of Malware URL for Real-Time Classification. *Proceedings - 6th International Conference on Computer and Communication Engineering: Innovative Technologies to Serve Humanity, 16*, 90–95. doi:10.1109/ICCCE.2016.31

Pham, H., Le, T. D., & Vu, T. N. (2018). Static PE Malware Detection Using Gradient Boosting Decision Trees Algorithm. In T. Dang, J. Küng, R. Wagner, N. Thoai, M. Takizawa (Eds) *Future Data and Security Engineering.* FDSE 2018. Lecture Notes in Computer Science, (vol 11251). Springer, Cham. https://doi.org/10.1007/978-3030-03192-3_17

Ranveer, S., & Hiray, S. (2015). SVM Based Effective Malware Detection System. *International Journal of Computer Science and Information Technologies*, 6(4), 3361-3365.

Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic Analysis of Malware Behavior using Machine Learning. *Journal of Computer Security*, 19(4)639-668, doi:10.3233/JCS-2010-0410

Rubin, A., Kels, S., & Hendler, D. (2019). AMSI-Based Detection of Malicious PowerShell Code Using Contextual Embeddings. Retrieved from http://arxiv.org/abs/1905.09538

Salehi, Z., Ghiasi, M., & Sami, A. (2012). A Miner for malware detection based on API function calls and their arguments. *The 16th CSI International Symposium on Artificial Intelligence and Signal Processing*, 563-568, doi:10.1109/AISP.2012.6313810

Santos, I., Brezo, F., Ugarte-pedrero, X., & Bringas, P. G. (2013). Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Information Sciences*, 231, 64–82, doi:10.1016/j.ins.2011.08.020

Schultz, M. G., Eskin, E., & Stolfo, S. J. (2015). Data Mining Methods for Detection of New Malicious Executables. Data mining methods for detection of new malicious executables. *Proceedings 2001 IEEE Symposium on Security and Privacy*, 38-49, doi:10.1109/SECPRI.2001.924286

Scott, J. (2017). Signature Based Malware Detection is Dead. *Cybersecurity Think Tank*, Retrieved from www.ICITForum.org

Swamynathan, M. (2019). *Mastering Machine Learning with Python in Six Steps. In Mastering Machine Learning with Python in Six Steps*. New york: Apress. doi:10.1007/978-1-4842-4947-5

Vatamanu, C., Cosovan, D., & Luchian, H. (2015). A Comparative Study of Malware Detection Techniques Using Machine Learning Methods. *International Journal of Computer and Information Engineering*, *9*(5), 1157–1164. doi.org/10.5281/zenodo.1100939

Wang, M. C. (2014). Detecting Internet Worms Using Data Mining Techniques. Retrieved from https://www.semanticscholar.org/paper/Detecting-Internet-Worms-Using-Data-Mining-Siddiqui-ang/b58621ddd3e79290112a3f7863dc58ca7e6963ab

Wong, W., & Stamp, M. (2006). Hunting for metamorphic engines. *Journal of Computer Virol*, 2, 211–229, doi:10.1007/s11416-006-0028-7.

Yan, J., Qi, Y., & Rao, Q. (2018). Detecting Malware with an Ensemble Method Based on Deep Neural Network. *Security and Communication Networks*, doi.org/10.1155/2018/7247095

You, I., & Yim, K. (2010). Malware Obfuscation Techniques: A Brief Survey. Proceedings - 2010 International Conference on Broadband, Wireless Computing Communication and Applications, 297-300, doi:10.1109/BWCCA.2010.85.5