WINDOWS MALWARE DETECTION TECHNIQUES USING STATIC AND BEHAVIOURAL-BASED FEATURES

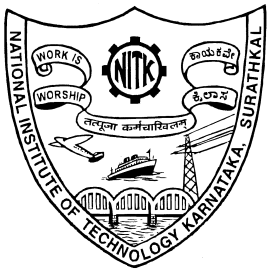
Thesis

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

Mr. SHIVA DARSHAN S. L.



DEPARTMENT OF INFORMATION TECHNOLOGY

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575 025

November, 2019

Declaration

I hereby *declare* that the Research Thesis entitled "Windows Malware Detection Techniques Using Static and Behavioural-based Features" which is being submitted to the National Institute of Technology Karnataka, Surathkal in partial fulfilment of the requirements for the award of the Degree of Doctor of Philosophy in Information Technology is a *bonafide report of the research work carried out by me*. The material contained in this thesis has not been submitted to any University or Institution for the award of any degree.

Mr. Shiva Darshan S. L.
Register No.: 155063IT15F02
Department of Information Technology

Place: NITK-Surathkal

Date:

Certificate

This is to *certify* that the Research Thesis entitled "Windows Malware Detection Techniques Using Static and Behavioural-based Features" submitted by Mr. Shiva Darshan S. L. (Register Number: 155063IT15F02) as the record of the research work carried out by him, is *accepted as the Research Thesis submission* in partial fulfilment of the requirements for the award of degree of Doctor of Philosophy.

Dr. Jaidhar CD
Research Guide
Associate Professor
Department of Information Technology
NITK-Surathkal - 575 025

Chairman - DRPC
(Signature with Date and Seal)

This thesis is dedicated to

My Parents and Sister

Acknowledgements

First of all, I would like to thank my supervisor for believing in me and for giving me the opportunity to carry out this research work. Without his technical assistance and suggestions throughout my research, it would not have been possible to complete this work. I also extend my gratitude to Head of the Department, Information Technology department, and Research Progress Assessment Committee members for their extensive support and encouragement.

I thank all non-teaching staffs of the Department of Information Technology NITK Surathkal for their cooperation.

Lastly, I would like to thank my parents, sister, family members, and friends for their moral support throughout my life.

(Mr. Shiva Darshan S. L.)

Abstract

The advancement in Internet-based communication technology has enabled malware to achieve its intent without the user's consent. It penetrates or harms a computer system's integrity, availability, and confidentiality. Forbye, a modern malware, is equipped with obfuscation techniques that maximize its capability to defeat anti-malware detection systems and evade detection. The conventional anti-malware detection techniques exhibit inherent delayed effectiveness due to their signature-based detection and are inadequate to ascertain advanced malware. Therefore, there is need for a proficient malware detection technique, which can precisely identify it.

The traditional Windows malware detection techniques can analyze malware without executing them. These techniques discern the malware by analyzing the static features of the Portable Executable (PE) files. However, they are incompetent against the emerging advanced malware attacks. To address this, behavioural-based malware detection technique emerges as an essential complement to defend against such sophisticated malware. The behavioural-based detection technique monitors and captures the activities of the malware during its runtime. It executes the input file (PE) in an isolated environment and records its behaviours during execution. However, in real-life scenario, it is tedious to examine all the recorded features. Hence, identifying significant features from the original features set is the primary challenging task in this technique. Several issues remain open in the development of an intricate malware detection system that can resist the attacks caused by the malware. Many examinations illustrate that the current malware detection systems are easily compromised by sophisticated malware. There are various solutions proposed in literature to uncover malware. However, each detection approach has its own limitation(s).

The present research work aims to propose a classic approach to detect and classify Windows malware by extracting static features or behavioural features or a combination of both (hybrid features) of the PE files. In this regard, initially, the Malware Detection System (MDS) was designed based on the information extracted related to Portable Executable Optional Header Fields (PEOHF) as static features. In addition, to identify the malicious activities of the malware, behaviour analysis of the PE files was also performed by considering Application Programming Interface (API) or API with their corresponding category (CAT-API) or System calls invoked by the input PE file during execution. Concurrently, for precise classification operation, preserving the informative features is highly necessary to detect and distinguish the unknown PE files as malware

or benign. With this in view, the performance of the Feature Selection Techniques (FSTs) in recommending the best features is crucial for classifiers in discriminating between benign and malware PEs was evaluated. Subsequently, a malware detection technique based on visualization images was proposed where the images were generated using behavioural features suggested by the FST. Moreover, the effectiveness of the hybrid features in the detection of malware was examined based on the significant features recommended by the FSTs. Several sets of experiments were carried out to evaluate and demonstrate the potency of the proposed approaches. The efficiency of all the proposed approaches was assessed using real-world malware samples with 10-fold cross-validation tests. Different evaluation metrics such as True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F-Measure, and Accuracy were used to evaluate the proposed approaches. Based on the obtained experimental results, it was observed that the proposed approaches are effective in the detection and classification of the Windows malware.

Keywords: Behavioural-based features; Cuckoo Sandbox; Feature Selection Technique; Portable Executable Files; Static features; Windows Malware Detection.

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
ARFF	Attribute-Relation File Format
AUC	Area Under the Curve
BD	Balanced Dataset
BNC	Benign N-gram Chunk
BNFs	Benign N-gram Files
CATs	Categories
CAT-API	API with their corresponding category
CNN	Convolutional Neural Network
CPD	Categorical Proportional Difference
CT	Contingency Table
ExeT	Executable Traces
DFS	Distinguishing Feature Selector
DIA	Darmstadt Indexing Approach
DLL	Dynamic Link Library
DOSH/DH	Disk Operating System Header
DTs	Decision Trees
FFS	Final Features Set
FFV	Final Feature Vector
FH	File Header
FLF	Function Length Frequency
FN	False Negative
FP	False Positive
FPR	False Positive Rate
FST	Feature Selection Technique
HFMDS	Hybrid Features-based Malware Detection System
IF	Import Function
IG	Information Gain
JSON	JavaScript Object Notation
LSVC	Linear Support Vector Classification
MDS	Malware Detection System
MI	Mutual Information
MIST	Malware Instruction Set
MNC	Malware N-gram Chunk

MNFs	Malware N-gram Files
Max-Rel	Max-Relevance
mRMR	Max-Relevance and Min-Redundancy
Opcode	Operational code
OH	Optional Header
OOBE	Out-of-Bag Error
PE	Portable Executable
PEHFs	Portable Executable Header Features
PEOHF	Portable Executable Optional Header Fields
PSI	Printable String Information
RF	Random Forest
ROC	Receiver Operating Characteristics
SMO	Sequential Minimal Optimization
SH	Section Header
TN	True Negative
TP	True Positive
TPR	True Positive Rate
UBD	Unbalanced Dataset
UBNF	Unique Benign N-gram File
UMNF	Unique Malware N-gram File

Chapter 1

Introduction

The growth in Internet technology has significantly and irreversibly influenced everyday lives in recent years. As the Internet turns out to be progressively pervasive, cyber threats have become gradually frequent and severe. The absence of adequate defensive mechanisms on computers allows cybercriminals to initiate security attacks.

In the near foreseeable future, almost all electronic devices will be connected to the Internet creating a vast potential of functionality. Devices will access remotely, and consequently, these devices will become prominent targets for cybercriminals. The motivations behind cybercrime are exemplary in that they support the attack and discard the defence. Cybercrime creates exceptional yields at minimal risk and minimal effort for programmers as the two most basic exploitation methods used by cybercriminals are social engineering and vulnerability. Cybercriminals deceive the user into granting access by employing social engineering methods. They use the vulnerability exploitation method and exploit the programming or execution failure to gain access. They mostly take advantage of the malware for their desired intent.

Malware is well-known as malicious software designed to disrupt the normal operations of computers or computer networks. It accomplishes this through illegitimate actions without the consent of the user. In early years, malware was created to highlight the vulnerabilities of the computers. Unfortunately, the growth of malware has expeditiously turned into a very profitable business in current days. So, malware authors started to use malicious code to compromise a large number of computers for monetary benefits. Malware can perform various types of attacks on modern computers and communication infrastructure by getting downloaded into the computer as a genuine application and emerging as a greater threat in the real world (Egele et al., 2012).

A few examples of recent malware samples affecting Windows machines are Astaroth, AZORult, Emotet, Petya, Trickbot, and Wannacry. Astaroth is a sophisticated malware that targets the Windows machines to fetch sensitive information like credentials, keystrokes, and other data, which it exfiltrates and sends to a remote attacker. AZORult is a trojan malware that compromises system security with backdoor capabilities and executes malicious commands to fetch the computer details such as a globally unique identifier, user name, and operating system version. Emotet is a banking trojan malware that obtains financial information by injecting malicious code into the Win-

dows machines. Petya ransomware targets Windows systems and encrypt a hard drive's file system table and prevents Windows from booting. WannaCry is a ransomware worm that can infect Windows machines by encrypting files on the hard drive and making them impossible for users to access. To confront such sophisticated malware, it is crucial to develop a sophisticated Malware Detection System (MDS) which adopts effective anti-malware defensive solutions.

Most of the existing malware detection approaches depend on automatic malware analysis tools (Rieck et al., 2008), and these are signature-based techniques. These techniques can be easily evaded since current malware are well- equipped with polymorphism techniques (Moser et al., 2007; Bailey et al., 2007). The complement approach is the behaviour-based malware detection approach, which checks the behaviour of the executable file (PE file) and gathers its information at runtime to identify whether the executable is malware or not (Ahmadi et al., 2013). More often, a malware analyst can notice that the unknown malware that emerges regularly exhibits slight change in its version compared with the earlier version. To tackle such malware, behaviour-based detection approach is a great choice as it can perform precise detection and classification.

1.1 MALWARE CHARACTERIZATION

Malware employs one or more anti-malware analysis techniques to evade these defensive solutions and makes it slow and tedious. However, malware can be characterized based on their concealment strategies. The common techniques adopted by malware to evade the detection include:

Obfuscation: Malware developers use the obfuscation technique to make malware tougher against reverse engineering (OKane et al., 2011). They use actions such as insertion of unnecessary jumps, addition of garbage instructions, and modify the actual program to a different form while retaining its functionality. Earlier techniques were developed to handle the infringement of the intellectual property of the software products, but currently, malware developers use them to prevent detection (You and Yim, 2010).

Polymorphism: The malware utilizes this technique to encrypt itself to a new variant by using the encryption algorithm (Sharma and Sahay, 2014). The polymorphic malware mutates its syntaxes each time while performing malicious events without change in semantics. Polymorphic malware can generate unlimited number of decryp-

tors by altering the instructions in the next variant of malware and evade detection.

Metamorphism: The malware uses this technique and transforms itself into a new instance of malware, rather than generating a new decryptor (Borello and Mé, 2008). This malware has no resemblance to the original one. This characteristic makes them sophisticated helps in evading the detection techniques.

Remote execution malware: The malware designed using this technique is capable of being triggered arbitrarily over the network and is referred to as remote execution malware (Saeed et al., 2013). Cybercriminals use these to achieve their intention using the Internet.

1.2 MALWARE TYPES

Malware are classified based on their behavioural characteristics. The taxonomy of these discriminative malware is important to perceive their threat level and to get fortified. The infamous malware are Worm, Virus, Trojan Horse, Spyware, Rootkit, etc.

Worm A worm is a self-replicating malicious program, which exploits the vulnerabilities of the target system.

Virus A virus is a program, that propagates by interpolating a copy of itself to another program and becomes a part of it.

Trojan A type of malicious program used to mislead legitimate users. It attracts the users to execute it on their systems and performs various types of malicious activities.

Spyware A type of malicious software which secretly monitors and collects various types of personal information of the user.

Rootkit A rootkit is a kind of malware, which allows an unauthorized entity to access and attain control of the host system without being detected.

1.3 MALWARE DETECTION APPROACH

Malware analysis is performed to identify the malicious elements in the input file(s) by either using the static approach or the dynamic approach or a combination of both. However, there has been great concern regarding the timely discovery of unknown malware in recent years and is still a critical issue. Existing techniques (Bazrafshan et al., 2013) related to malware detection are mainly classified into three types: (a) Static features-based approach, (b) Behavioural features-based approach, and (c) Hy-

brid features-based approach.

1.3.1 Static Features-based Approach

Static features-based malware detection approach in many ways is the backbone for most of today's existing anti-malware defensive solutions. This approach relies on the patterns (features) extracted from source files or signatures or database description. The static features of the PE files can be N-grams (byte sequences) (Kolter and Maloof, 2006; Ye et al., 2010), or Operational code (Opcode) (Santos et al., 2010), or Printable String Information (PSI) (Ye et al., 2009), or Dynamic Link Library (DLL) related information, or Optional Header (OH), or Disk Operating System Header (DOSH), or File Header (FH), or a combination of any of these extracted from the source PE files (Shabtai et al., 2009; Masud et al., 2008).

Signature-based technique is one of the static features-based techniques. It categorizes the known malware accurately by relying on the identification of a unique sequence of bytes in the binary code. To detect whether the input file is malware or not, static features-based anti-malware defensive solution computes signature (for example, Hash digest) for the input files and compares it with the signature database. If a match is found, then it declares it as malware, otherwise as benign or unknown malware. Later the unknown malware is analyzed to generate a new signature and update the database. While investigating the unknown malware and the creation of a unique signature to update the signature database, the unknown malware might attack modern computers and communication infrastructure that are vulnerable. Thus, static features-based malware detection approach is inefficient and incapable of identifying unknown malware that uses code obfuscation techniques. Thus, signature-based techniques achieve poor performance when attempting to detect unseen malware.

1.3.1.1 N-grams

N-gram represents a sequence of `'N'` consecutive bytes constructed from the byte sequence extracted from a source file. In other words, the extracted data is represented in overlapping substrings obtained based on the sliding window approach known as N-grams (Raff et al., 2018), and the N-grams concept is widely used in information retrieval. Although this type of feature does not provide meaningful information, it achieves high accuracy in identifying unknown malware.

1.3.1.2 Operational code

Opcode (Santos et al., 2013) is a part of the machine language instruction that specifies the operation to be performed. Opcodes are used for representing an PE file by streamlining it into a series of Opcodes. Moreover, it is used for statically analyzing the PE files and acts as a predictor for detecting malware with obfuscated characteristics.

1.3.1.3 Printable String Information

Printable String Information (Ye et al., 2009) is un-encoded strings present in the PE file, which can reveal information regarding the nature of the file as benign or malware. These strings can be a part of the file or information on system resources used. PSIs that are extracted from the PE files are used as static features in prediction and detection of unseen malware.

1.3.1.4 Dynamic Link Library

A Dynamic Link Library is executable code in the form of a computer library that contains a set of functions that are called by a PE file either at startup or whenever required. Specifically, DLLs are imported from the host operating system during execution. Identifying and understanding the DLLs invoked by the malware can define the nature of the malware and its purpose.

1.3.1.5 Portable Executable File Header Information

The PE file (Kumar et al., 2019) is represented as a common file format for all versions of the Windows operating system. It mainly constitutes of a MS-DOS Header, PE Header, Section Header, and Sections as depicted in Figure 1.1.

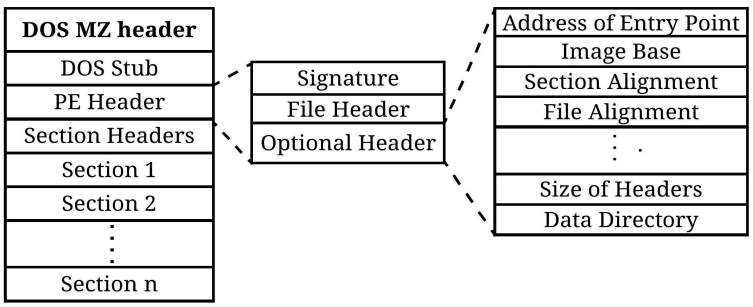


Figure 1.1: A general layout of the PE file depicting members of the PE Header (Belaoued and Mazouzi, 2015)

The PE file always starts with an MS-DOS Header. The purpose of the Disk Operating System (DOS) is to check whether the PE file is valid or not. If the file runs under

the DOS environment, then the MS-DOS stub is a built-in executable used to display the error message. The next field is the PE Header and its structure provides essential information used by the PE loader. After the PE Header, there exist several Sections which store the data in terms of blocks. In each Section, the data is organized based on common attributes. The PE Header format is a IMAGE_NT_HEADERS data structure, which consists of the PE-Signature, FH, and the OH. The OH is composed of several fields as shown in Figure 1.1.

The major disadvantage of the static features-based approach is that it fails to counter packed malware or unseen malware or new malware. Further, static features-based detection techniques often require a human analyst to analyze the code to understand the malware functionality. A large number of new malware keep emerging and the manual analysis of these samples is a tedious task. Hence, behaviour-based detection techniques are needed.

1.3.2 Behavioural Features-based Approach

In behavioural features-based malware detection approach, the malware is analyzed by observing its behaviour during runtime. The behaviour of the malware (PE file) is determined by recording the system calls and API calls invoked by the malware, generally in an isolated environment (sandbox). Further, the recorded behaviour of the malware is analyzed automatically using the sandbox technology, which understands and identifies the intention of the malware within the system and its harmful effects. Finally, it produces the execution time behaviour-based analysis report.

1.3.2.1 Sandbox Technology

The sandbox technology (Lindorfer et al., 2011; Guarnieri et al., 2012) is an effective measure to the challenge posed by various types of malware attacks. It provides a safe testing environment to perform dynamic malware analysis with minimal risk of infecting other machines. While analyzing the behaviour of a user mode process, every sandbox looks at the system calls and API calls. System calls are a routine that allow the operating system to interact with the user-level process to perform their desired task. These tasks include reading data from files, delivering packets across the network, and recording of entry from the registry. It provides an automated environment to closely observe the progress of the malware and prepare a database to categorize them into families.

1.3.2.2 Function Call Monitoring

A function is a code block designed to perform specific operations. Functions are useful whenever a particular task needs to be repeated and thereby, it gains the property of reusability. It provides a semantically oriented approach, instead of focusing more on the implementation details, and is easily adaptable for many platforms. To invoke a function, the name of the function needs to be called. Function calls are captured through the hooking process, which implements the analysis procedure and accomplishes tasks such as analyzing inputs and outputs, logging program execution, monitoring intermediate function calls, etc..

1.3.2.3 Application Programming Interface

Application Programming Interface is a collection of defined functions and methods required for interfacing with the operating system or service running on the system. It is generally used for importing functions from a DLL by establishing a reference to a library. APIs are explicitly called within the user's program or implicitly called by the compiler. Many anti-malware defensive solutions extract API calls for creating the behavioural pattern of the running PE files, and later analyze the captured data using machine learning concepts to classify the running PE file as malware or benign (Qiao et al., 2014).

1.3.2.4 System Calls

The programs within the operating system are executed in User mode or Kernel mode. The user level programs (browsing, word processing, image applications, etc.) are executed mainly in User mode. In Kernel mode, the operating system executes its own programs (for example, driver programs). The Kernel mode is also called as a system mode. The User mode process does not have direct access to the Kernel mode. In order to perform several tasks within the system, user level processes sometimes have to access kernel level processes, and this is accomplished through a special API called as system calls. Some kernel level malware such as Rootkits have the capability to gain privilege to the system and extract very sensitive information present in it (Rieck et al., 2011). However, many malware vitiate the User mode process and gain access to the Kernel mode. The dynamic malware analysis technique makes use of the sequence of system calls to observe the behaviour of the running PE file with the help of the API hooking.

1.3.2.5 Hooking

Hooking permits the malware analyst to track the specific runtime behaviour of the PE file at a specific point so that appropriate action can be taken to prevent infection by the malware. Hooking provides detailed behavioural analysis of the running PE file, and at the same time, identifies any suspicious activities caused by the running PE file (Sami et al., 2010). The resultant analysis report after the hooking process consists of rich information (for example, parameters accessed and processed, monitored function calls, etc.), which is referred to as Trace (Egele et al., 2012).

The major limitations of the behavioural features-based malware detection approach is that it allows only one PE file to execute in a controlled monitoring environment (Willems et al., 2007) to record all the activities performed by the PE file. The PE file must be executed for a certain amount of time to record its behaviour, which consumes time to produce the behavioural report. Some malware exhibit their infection actions only after certain conditions are met and for such type of malware, a controlled environment may not be suitable to acquire the execution time behaviour (Islam et al., 2013).

1.3.3 Hybrid Features-based Approach

Static analysis and dynamic analysis are complementary to one another (Bounouh et al., 2017). Dynamic analysis provides paramount insight of the malware, whereas static analysis is unable to provide any significant information required to analyze an obscure malware in real-time. MDS that merely relies either on static or dynamic analysis may not be efficient enough to detect sophisticated malware. Therefore, hybrid features-based malware detection techniques have emerged to perform more robust detection and classification while combining static features with behavioural features.

1.4 FEATURE SELECTION TECHNIQUE AND CATEGORIZATION

The FST is a process of identifying a set of relevant significant features from the original feature set that increases the effectiveness of the machine learning-based classifiers and is also able to recognize noisy features. The FST in machine learning is broadly classified as filter-based approach and wrapper-based approach. The filter-based approach provides a score to each feature independently without using any learning algorithm, whereas the wrapper-based approach uses a learning algorithm to score the features. The main reasons to incorporate the FST in the proposed approaches are because of its following advantages:

- The FST can reduce the curse of dimensionality and make the classifiers to train faster with less time;

- It can reduce the over-fitting problem by discarding the redundant features and lessen the chance to make a decision based on irrelevant features; and

- It can improve the prediction ability of the machine learning-based classifiers by recommending the best subset of features based on their highest computed score.

In the present work (proposed work), both the filter-based FST and the wrapper-based FST were used to examine their efficiency in recommending significant features that would enhance the predictive performance of the machine learning-based classifiers. The explanation regarding the FSTs utilized in the proposed work is provided in the respective chapters. Further, a set of experiments were conducted to evaluate the performance of the chosen FSTs.

1.5 MOTIVATION

People in modern times rely more on Internet-based services and hence, the Internet has become an essential part of the daily life. Most commonly, people who use the Internet for services like online banking, email, online shopping, and advertisement can accomplish their commercial/personal tasks easily as well as rapidly. However, in the real world, hackers with malevolent intent may utilize the Internet to exploit legitimate users' private information like bank transactions details, user login credentials, etc. Malignant users make use of malware to fulfil their desired goals.

Malware is destructive in today's digital world. Due to its proliferation, most of modern computers and communication infrastructure are getting compromised very easily. Figure 1.2, which is referred from a public source (Andreas, 2019), depicts the statistics of the rapid growth of malware over the last ten years. As per the information provided, most anti-malware defensive solutions register on average 350,000 new malicious programs every day (Andreas, 2019). Due to the evolution of new malware, many financial organizations such as banks, fund exchange related companies, crypto-currency exchange, etc. are subjected to attacks by cybercriminals. A report provided by Kaspersky reveals that attacks on corporate users had increased to nearly 25% in 2018 than on home users compared with 2017 (Eugene, 2019). The attackers use legitimate software to collect sensitive information from the attacked network.

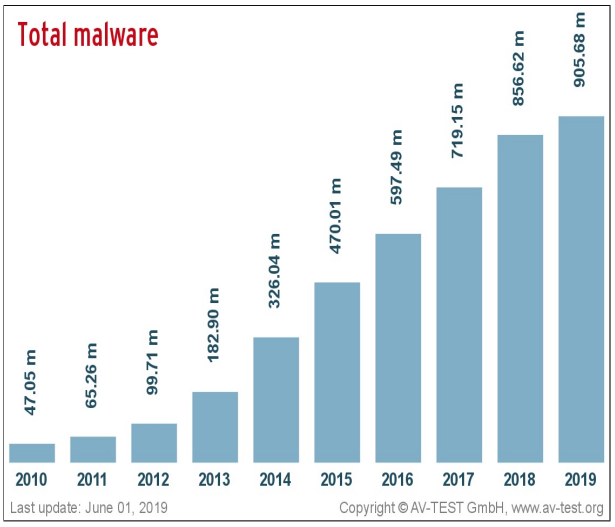


Figure 1.2: Statistics showing the growth of malware in millions (m) in last ten years (Andreas, 2019)

Based on the statistical study, the following reasons motivated the present research work:

- Static features-based malware detection approaches examine syntactic markers to discern malware. These markers are ineffective in thwarting the attacks caused by current malware equipped with various evasion techniques.
- The effectiveness of the attacks by the malware is creating havoc with the help of relatively simple programs.
- The existing anti-malware solutions rely mostly on signature databases that require frequent updating. The malware signatures need to be distributed to the user systems to spot the malware efficiently.
- Today's cybercriminals are well-mastered with almost all types of non-Windows platforms (Linux, Android, etc.) and expertise to create malicious programs for these platforms. However, the AVTest Institute reported that from 2017 to 2018 over 67% of all malware attacks were aimed at the Windows' systems (see Figure 1.3).

In order to safeguard legitimate users from these threats, security vendors have developed anti-malware defensive solutions to detect the malware and to quarantine them.

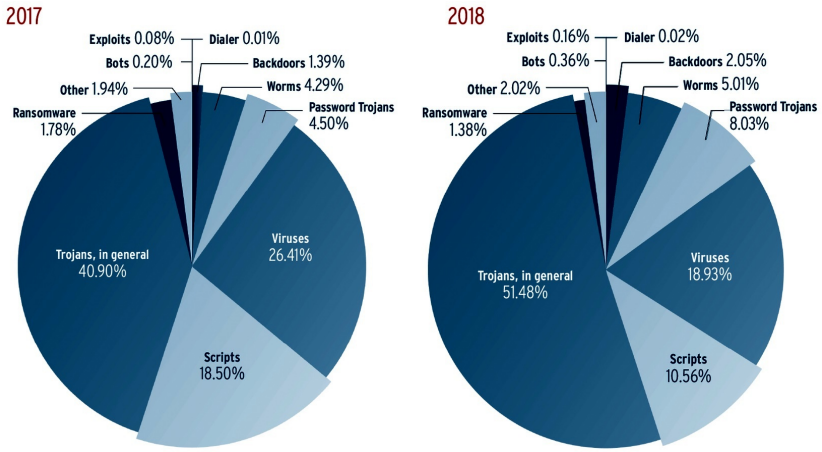


Figure 1.3: Statistics depicts the distribution of Windows malware during 2017-2018 (Andreas, 2018)

Many of these anti-malware defensive solutions identify the malware on signature-based technique. However, the signature-based technique is time-consuming and shows poor performance when attempting to classify unknown malware which use various code obfuscation techniques. This growing evasion capability of new malware needs to be countered by analyzing it dynamically in a sandbox environment, which provides an isolated environment. This has motivated the proposed research work so as to fulfil the defined research objectives.

1.6 MAJOR CONTRIBUTIONS OF THE THESIS

The major contributions of the thesis are as follows:

The static features-based MDS was proposed that employs PEOHF features with their correspondent values to detect Windows malware. A set of experiments were conducted to show the robustness of the proposed approach in precisely detecting and discriminating the malware. The predictive performance of the classifiers depends on the features used for classification. The Single-Stage-Feature-Selector was utilized to obtain the best PEOHF features, and a comparative analysis of the different FSTs was performed to identify the best FST with the potency to recommend the most significant PEOHF features. From the experimental observations, it was found that the best FSTs were Distinguishing Feature Selector (DFS) and Mutual Information (MI), since the features suggested by them resulted in obtaining better detection accuracy by the classifier.

The behavioural features-based malware detection technique based on the Cuckoo Sandbox generated report was proposed. The Cuckoo Sandbox was used to examine

the execution time behaviour of the PE files, and provide behavioural analysis report in JavaScript Object Notation (JSON) format. However, the JSON reports necessitate more storage space and more processing time. Hence, the reports were converted into Malware Instruction Set (MIST) format. The system calls sequence triggered by the PE files during their execution was extracted from the MIST report. The collected system calls were structured in the form of N-grams, and the generated N-grams were treated as features. The Information Gain (IG) FST suggested features were used to train a classifier. A comprehensive set of experiments were conducted to perceive the best fit classifier among the chosen six classifiers and simultaneously measured to find for which N-gram size the best detection rate could be achieved. The machine learning-based classifier Spegagos ensured better classification for both N-gram lengths of three and four bytes.

The some of the FSTs are computational resources demanded and takes more time to generate feature scores for larger datasets, if the processing is accomplished in the sequential mode. To address this issue, a multiprocessing model was proposed, which demonstrated IG score computation for N-gram features rapidly for larger N-gram datasets. The proposed model was designed, implemented, and compared with the sequential mode of the IG score computation. The obtained results manifested that the multiprocessing model was 80% faster than the sequential model.

The effectiveness of the behavioural features suggested by the Linear Support Vector Classification (LSVC) in identifying unknown malware was investigated. The two different types of behavioural features, namely, API calls and API calls along with their correspondent category (Category+API calls) were considered to know which type of features provided better malware detection rate. The highest detection accuracy was obtained with API calls as the behavioural feature.

A Convolutional Neural Network (CNN) based malware detector was proposed to show that the malware detection problem could be transformed into an image classification problem. It uses the runtime behavioural features (N-grams) generated by the sequence of the CAT-API of the PE files. However, in the present instance, the significant N-grams advised by the FST were used to create the images. The generated images were employed to examine the proficiency of the proposed approach. The experiments manifested that feature selection at the input level of the CNN is necessary to prevent the Neural Network from learning the association between the irrelevant features.

The hybrid features-based detection approach addresses the limitations of both the static and dynamic malware analysis by using a combination of both the features. However, its effectiveness depends on the types of features used to identify the malware. In this regard, the Hybrid Features-based Malware Detection System (HFMDS) was proposed. The LSVC was applied onto static features set as well as dynamic features set individually to obtain potential features that would increase the performance of the classifier. The LSVC recommended features were treated as final features to evaluate the effectiveness of the proposed HFMDS. The empirical results demonstrated that the proposed HFMDS is efficient in analyzing and detecting malware by achieving highest accuracy for a blend of hybrid features comprising of APICAT with other header features such as OH, DOSH, and FH.

An empirical study was performed to estimate the stability of the Random Forest classifier by considering the hybrid features recommended by the filter-based FST. The stability of the Random Forest classifier depends on the number of Decision Trees required to achieve consistent accuracy. To realize this, a set of experiments were conducted and the obtained empirical results demonstrated that from 160 Decision Trees (DTs) onwards, there was not much improvement in detection accuracy. Thus, there was no significance in increasing the number of DTs beyond 160.

1.7 ORGANIZATION OF THE THESIS

The remaining chapters of this thesis are organized as follows:

Chapter 2 presents the existing research works related to Windows malware ascertaining techniques. It highlights the literature review on static features-based, behavioural features-based, hybrid features-based, and visualization-based approaches in detection and discrimination of malware. Based on the literature outcomes, it defines the problem statement and research objectives.

Chapter 3 discusses the effectiveness of the proposed malware detection approach in recognition and classification of PE files as benign or malware based on the OH features along with their corresponding values as static features. Further, the obtained experimental results demonstrate the comparative analysis of the efficiency of the chosen filter-based FSTs. Finally, it shows that the significant PEOHF features recommended by the FSTs yield better detection accuracy by the classifier.

Chapter 4 discusses the proposed malware detection approaches based on behavioural

features. It describes the steps involved in generating the sequence of N-grams of specified length using the extracted system calls from the MIST reports. It also explains the steps followed to calculate a score for each N-gram by adopting the IG FST. Further, it demonstrates how for an N-gram length of specific bytes and specific topmost N-grams, the classifier achieved better accuracy. Additionally, the comparative analysis between the sequential model and the multiprocessing model in calculating the features (N-grams) score is discussed. Further, a MDS that was designed to explore the effectiveness of the LSVC is discussed. It considers two different types of dynamic features, namely, API calls and CAT-API, to know which type of features recommended by the LSVC provides better malware detection rate by the classifier. The proposed CNN-based Windows malware detection approach is discussed. It uses the runtime behaviour features suggested by the FST to create images, and the generated images are used in the precise detection and classification of malware as per their respective families.

Chapter 5 discusses the proposed hybrid features-based malware detection approach to identify unknown malware. It uses LSVC as a feature selector to choose prominent hybrid features. Further, its performance is verified to know which combination of static and dynamic features encourage the classifier to attain high accuracy. Additionally, an investigation in to the number of Decision Trees required by the Random Forest classifier to achieve consistent detection accuracy in the classification of PE files as malware or benign using hybrid features is discussed.

Finally, Chapter 6 provides conclusions to the entire thesis work and offers future directions.

1.8 SUMMARY

This chapter introduces malware detection techniques, which gives an insight into the characteristics, challenges, and motivation in the detection of malware. It provides an overview of the existing malware detection techniques. It gives a brief overview of the FSTs and their categorization. This chapter also presents an outline of the contribution and structure of the thesis.

Chapter 2

Literature Survey

This chapter reviews previous works related to Windows malware detection techniques used to identify malicious elements in the PE files. Some of them are static features-based approaches, while some are dynamic features-based (behavioural features-based) or a combination of both. However, there has been great concern regarding the timely discovery of unknown malware in recent years and remains a critical issue. Windows malware detection approaches such as static features-based techniques, behavioural features-based techniques, visualization-based techniques, and Deep Learning-based techniques are consecutively discussed in this chapter.

2.1 STATIC FEATURES-BASED MALWARE DETECTION TECHNIQUES

Most of the conventional malware detection techniques are equipped with static features-based detection approach to classify the source PE file. The static features-based malware detection approach examines the source files without executing them. The static features of the PE files can be consecutive byte sequence of size 'N' bytes (N-grams) (Reddy and Pujari, 2006; Raff et al., 2018), Opcodes (Santos et al., 2013; Ding et al., 2014), PSI (Ye et al., 2009; Lee et al., 2011), Import Function (IF) (DLL related information) (Narouei et al., 2015; Baldangombo et al., 2013), OH (Belaoued and Mazouzi, 2015), DOSH (Bai et al., 2014), FH (Kumar et al., 2019), and Section Header (SH) (Bai et al., 2014) or an amalgamation of any of these features.

The authors Schultz et al. (2001) pioneered the concept of data mining to detect unknown malware. They concentrated on mining consecutive byte sequence that represented a snippet of the program, while strings formed the PE files. Their approach employed machine learning-based classifiers to detect and classify unknown malware. Correspondingly, the obtained results demonstrated that all the machine learning-based classifiers were more accurate than traditional methods. Further, based on the comparative analysis performed, they claimed that the Naive Bayes classifier yielded better detection accuracy.

Kolter and Maloof (2006) proposed a detection method to classify malware, which appeared in the wild using machine learning-based classifiers and enhanced the results attained by (Schultz et al., 2001). They encoded all the PE files by employing the N-gram technique and used the IG FST to select supreme N-grams and to appraise the performance of the different classifiers. The experimental results proved that the

Boosted Decision Tree classifier achieved better detection rate.

Vinod et al. (2011) considered mnemonic N-grams and PE Headers as static features to distinguish between benign and malware PE files. The extracted features were preliminarily processed by utilizing a scatter criterion to eliminate the irrelevant features set, and thereby reduced the dimensionality of the original features set. Thus, their approach was successful in overcoming the computational complexity issues, which occur during the training and prediction phase. The authors approach accomplished detection accuracy of 96.80% with FPR of 0.138 ensuring that malware with evasion characteristics could be recognized.

Yan et al. (2013) proposed a framework to explore distinguishing features for automated malware classification. They extracted bits of characteristic, DLL, and system calls information from the PE files. The extracted information was represented as Boolean features to make a systematic study and gain insight as to which of these could highly discern the malware families.

Belaoued and Mazouzi (2015) designed and implemented a real-time PE malware detector that analyzed the PE optional header information to discern and discriminate unknown malware. They used two different FSTs such as the Chi-Square and Phi coefficient to eliminate noisy features that have Chi-Square score < 3.84. David et al. (2017) examined the importance of the different fields of the PE header and showed that the structural analysis of the header fields provided the required information to ascertain the malware.

Kumar et al. (2019) highlighted the importance of an integrated feature set to detect unseen malware. The authors used both the derived features and the raw static features and developed a discriminative model based on the machine learning approach. It achieved precision of 98.40% and the experimental results demonstrated enhancement in classification accuracy.

The main limitation of the static features-based malware detection techniques is that they are susceptible to inaccurate detection of sophisticated malware that adopts obfuscation methods such as garbage instruction insertion, code reordering, compression, encryption, etc. (Santos et al., 2013). Moser et al. (2007) substantiated that static features alone are insufficient to uncover unseen malware. Thus, to address these issues, they believed that the dynamic features-based malware detection approach as a complemented approach.

Table 2.1: Summary of static features-based malware detection approaches

Sl. No.	**Authors**	**Feature Type**	**Feature Extractor**	**Feature Selection Technique(s)**	**Remarks**
1	Reddy and Pujari (2006)	PE Features	□	. Class-wise document frequency . Information Gain	. Relevant N-grams and Class-wise document frequency were introduced. N-grams was utilized in the context of malware detection.
2	Moskovitch et al. (2008)	Opcode	IDA Pro Disassembler	. Document Frequency . Gain-Ratio . Fisher Score	. Text categorization concepts based malware detection method was presented to detect unknown malware.
3	Ye et al. (2008)	API calls	PE Parser	Max-Relevance	. Intelligent Malware Detection System (IMDS) was developed based on Objective-Oriented Association mining based classification.
4	Ye et al. (2009)	PSI	Feature Parser	Max-Relevance	. An Interpretable String-based Malware Detection System (SBMDS) was developed by considering both a Support Vector Machine ensemble and Bagging, to classify and precisely predict the PE files as malware or benign.
5	Ye et al. (2010)	API calls	PE Parser	Not Mentioned	. A proficient approach to detect and distinguish malware was proposed by adapting several post-processing methods of associative classification.

□: Information not explicitly mentioned

Table 2.1: Summary of static features-based malware detection approaches Contd.

Sl. No.	**Authors**	**Feature Type**	**Feature Extractor**	**Feature Selection Technique(s)**	**Remarks**
6	Sami et al. (2010)	API calls	PE analyzer	Fisher Score	. A framework was developed based on data mining technique for analyzing and classifying PE files.
7	Santos et al. (2010)	Opcode	NewBasic Assembler	Mutual Information	. A method to detect known malware variants was proposed based on the frequency of occurrence of Opcode sequence.
8	Bai et al. (2014)	PE Features	□	. CfsSubsetEval . WrapperSubsetEval	. Developed, a malware detection approach by mining header information of PE files.
9	Belaoued and Mazouzi (2015)	PE Optional Header Features	pefile	. Chi-square . Phi coefficient	. A real-time PE MDS has presented that analyzes PE files to discriminate them as benign or malware based on the information stored in the OH fields of the PE files.
10	David et al. (2017)	PE Header Features	□	Not Mentioned	. The structured analysis of many fields concerning PE header features of PE files are analyzed and stated these features could improve the detection accuracy.
11	Kumar et al. (2019)	PE Header Features	pefile	Not Mentioned	. An integrated feature-based malware identification approach was proposed blending the raw features of different fields of the PE file header with a set of derived features.

□: Information not explicitly mentioned

Table 2.1 shows the summary of the malware detection approaches that use the static features of the PE files.

In the present work, we have proposed the behavioural features-based malware detection techniques to uncover the malware using the sandbox technique. We have used Cuckoo Sandbox in our proposed approach to analyze the behaviour of the PE file dynamically. Further, the analyzed reports of the Cuckoo Sandbox are employed to acquire the logged behavioural features of the PE files during execution. The acquired behavioural features are then evaluated using FSTs to obtain the best features to train the classifier to perform prediction and classification of PE files as benign or malware.

2.2 BEHAVIOURAL FEATURES-BASED MALWARE DETECTION TECHNIQUES

Generally, dynamic features-based malware detection technique executes the source file in a sandbox to gather a runtime behavioural report, which is then analyzed to determine whether the source file is malware or benign. The reason is that the sandbox expedites quick inspection of the source file and ensures a clean environment after the completion of the analysis task of each input file without infecting the host system. Dynamic features can be API calls (Mohaisen et al., 2015), System calls (Lin et al., 2018), resource consumption, or any other function-based features captured while the source file is in the process of execution. Several dynamic malware detection techniques using the sandbox technology have been proposed (Bayer et al., 2006; Lengyel et al., 2014; Qiao et al., 2014; Sethi et al., 2018). An open source malware analysis tool called the CW-sandbox (Willems et al., 2007) captures System calls invoked by the source file while it is being executed and produces a precise runtime behaviour report of it. Yet, the CW-sandbox is deficient in discerning kernel mode rootkits. Later, Rieck et al. (2011) made an effort to create a proficient malware detection approach involving the System call sequence invoked by the source file, which provides insights and timely defence against malware.

Qiao et al. (2014) proposed a mechanized malware investigation system called the CBM. The behavioural report produced by it was utilized to transform the captured API call sequence into a byte-based consecutive information. The prime use of this approach was to diminish storage prerequisite and computational cost, and provide support for local deployment. It resulted in accomplishing high exactness for malware clustering.

David and Netanyahu (2015) proposed an approach, namely, the DeepSign. It employed the Deep Learning-based technique to generate the malware signature automatically. The authors used the Cuckoo Sandbox generated report to analyze the behaviour of the malware and its intent. Their approach yielded detection accuracy of 96.40%.

Fujino et al. (2015) presented a framework to extract identical malware samples by dynamically employing the concept of "API call topics". The authors employed the non-negative matrix factorization clustering approach to obtain "API call topics" from a vast collection of API calls to discern similar malware samples.

Salehi et al. (2017) presented an efficient and adaptable technique named MAAR to spot sophisticated malware. It examined the activities of the input file within the system during its execution and extracted features related to API calls and their corresponding arguments and returned values. Further, the most critical or distinguishing features prescribed by the FST were utilized to group benign and malware PE files and to achieve high detection accuracy with FPR <1. When the MAAR was analyzed on a bigger dataset involving new families of malware, it accomplished a precision of 96.30%. Pektaş and Acarman (2018) addressed the challenges of detecting and classifying malware by using behaviour-based artefacts. The Voting Expert algorithm was used to extract the malware's API patterns over the API calls.

Although behavioural features-based malware analysis technique ensured promising results, there were certain limitations such as: 1) It permitted only one source file to run at a time in a restraint examination environment to capture the execution time behaviour of the source file within the host system (Willems et al., 2007); 2) Each source file was required to be executed in a certain amount of time to gather the execution time behaviour. Hence, it expend time in providing a behavioural report; and 3) Some sophisticated malware disclose their purpose after particular constraints are met, and for these obscure malware, a controlled domain was not reasonable to record its execution time behaviour (Islam et al., 2013). An alternate approach is the hybrid features-based malware detection technique, which consolidates both the static features and dynamic features to examine the input file.

In the proposed hybrid features-based malware detection approach, we simultaneously analyze the PE files statically and dynamically to acquire both static and behavioural features of the PE files. The PE files that are unable to examine during the execution time due to their obfuscation techniques can be analyzed statically to under-

stand the intent. Thereby the proposed hybrid features-based malware detection approach overcomes the drawbacks of the behavioural features-based malware detection approach.

Table 2.2 provides a summary of behavioural features-based malware detection approaches.

2.3 HYBRID FEATURES-BASED MALWARE DETECTION TECHNIQUES

Hybrid features-based malware analysis performs more potent identification and characterization, while integrating the static and dynamic features of the source file.

Shijo and Salim (2015) proposed a methodology that used both the static and dynamic features of the PE files for automated detection and categorization of the malware. In this, the PSIs were used as static features, and the API calls called by the source PE files during runtime were considered as dynamic features. Further, the N-gram technique was employed to derive feature vectors. They used two different sizes of N-grams of 3 and 4 bytes. The experimental outcomes showed that the precision rate with static features was 95.88%, the dynamic feature was 97.16%, and the blend of both static and dynamic features was 98.70%.

Islam et al. (2013) exhibited a technique that used the advantages of both static as well as dynamic features. As static features, function length frequency and PSI were used. As dynamic features, API calls with their parameters were used. The investigations were conducted with 541 benign PE files and 2398 malware PE files. The output exhibited that the blend of both static and dynamic features enhanced the detection rate and accomplished detection rate of 97.05%.

Bounouh et al. (2017) proposed a malware classification approach using hybrid features. They mainly focused on distinguishing the respective families of the malware, rather than discriminating between malware and benign files. To this end, they used static features such as PSI and function length frequencies, and behavioural features as a set of actions on the system resources. Their approach was significant in achieving better precision of 0.994 and scalability by reducing the feature space for larger samples.

Table 2.2: Summary of behavioural features-based malware detection approaches

Sl. No.	Authors	Feature Type	Sandbox Technique	Feature Selection Technique	Remarks
1	Willems et al. (2007)	System calls	CW-sandbox	Not Mentioned	. A system call-based dynamic malware detection technique using CW-sandbox was proposed. Malware behaviours of the PE files are monitored and gathered through the execution of the PE files in a simulated environment to produce analysis reports dynamically.
2	Rieck et al. (2008)	Information related to execution trace of the PE file	CW-sandbox	Not Mentioned	. SVM-based malware detection and classification approach was proposed, which could cluster the execution behaviour of PE files to precisely classify them as malware.
3	Rieck et al. (2011)	System calls and their arguments	CW-sandbox	Not Mentioned	. MIST format was proposed to represent necessary information required to classify the PE files as malware or benign using the behaviour analysis reports produced by the CW-sandbox. MIST could able to cluster and categorize the malware to a predefined class.
4	Ahmadi et al. (2013)	API calls	□	. Fisher Score . CfsSubsetEval	. Malware detection technique was proposed to detect malware by considering API calls and their iterative patterns extracted from PE files.
5	Qiao et al. (2014)	API calls	Cuckoo Sandbox	Not Mentioned	. Designed Byte-based Behaviour Instruction Set to compress the behavioural report obtained by Cuckoo Sandbox and enhanced the malware clustering efficiency.
6	Salehi et al. (2014)	API calls with their arguments	□	Relief	. The runtime behaviours of the PE files are monitored using a controlled environment, to log the names of API calls invoked along with their corresponding arguments, and to generate the features sets which could discriminate the malware from non-malware PE files.

□: Information not explicitly mentioned

Table 2.2: Summary of behavioural features-based malware detection approaches Contd.

Sl. No.	Authors	Feature Type	Sandbox Technique	Feature Selection Technique	Remarks
7	David and Netanyahu (2015)	Logged behaviour of PE files	Cuckoo Sandbox	Not Mentioned	. Automatic MDS by name DeepSign was proposed. To accurately classify the unknown malware, their approach converted the behaviours of the PE files logged by the Cuckoo Sandbox into a binary vector and then provided as an input to the Neural Network for performing classification process.
8	Hansen et al. (2016)	API calls and their input argument	Cuckoo Sandbox	. Gain Ratio	. Dynamic analysis of the PE files was performed by monitoring their actions within the sandbox to detect and classify the malware into their respective families.
9	Salehi et al. (2017)	API calls, API calls with their arguments, API calls with their arguments and return values	Cuckoo Sandbox	. Fisher Score . Support Vector Machine based on Recursive Feature Elimination	. A dynamic malware feature selection method is proposed to produce a robust and scalable feature sets to perform dynamic malware behaviour analysis. Two phases of feature selection were utilized to reduce the number of features, to speed up the time of processing and increasing the detection rate.
10	Geden and Happa (2018)	API calls and System calls	Cuckoo Sandbox	Information Gain	. A malware family classifier was implemented using the runtime behaviours recorded by the Cuckoo Sandbox as features.
11	Kakisim et al. (2018)	API call sequence	Cuckoo Sandbox	Correlation based-feature selection	. The comparative analysis was performed among various conventional malware identification approaches by considering different behavioural features achieved by dynamic analysis.

Damodaran et al. (2017) compared malware detection techniques in view of static, dynamic, and hybrid features. They used the Hidden Markov Model to train both the static and dynamic features and validated the experiments using the 5-fold cross-validation tests and attained 0.98 Area Under the Curve (AUC) score.

Table 2.3 shows a summary of hybrid features-based malware detection approaches.

2.4 VISUALIZATION-BASED MALWARE DETECTION TECHNIQUES

Visualization is a predominantly used technique in computer security domain. It enables the human analyst to visually discern the features of the malware. Recently, several studies were proposed by employing the visualization technique for malware analysis (Han et al., 2015; Ni et al., 2018).

Yoo (2004) utilized self-organizing maps to visually recognize and discriminate malware. Quist and Liebrock (2009) proposed a visualization approach to comprehensively monitor the flow of program execution. Further, the authors applied the node-link visualization technique to determine the functional areas, which were obfuscated. Trinius et al. (2009) presented a parameterized technique to abstract the behavioural reports generated by the sandbox. Further, the behaviour of the malware was visualized using the Tree-Map and Thread Graph to detect and classify the malware by gathering information related to API calls and performed operations within the sandbox.

Goodall et al. (2010) described a prototype system that gathered the output of multiple malware detection tools into a visual environment to automatically detect the different vulnerabilities in the software code. Nataraj et al. (2011) proposed a framework to visually spot malware based on binary texture analysis. Initially, the malware binaries (executable files) were visualized as grayscale images. Later, the authors applied the K-Nearest Neighbours' approach with Euclidian distance method to classify the malware.

Kancherla and Mukkamala (2013) presented a visualization approach for malware detection by converting the executable files into a grayscale image called byteplot to extract intensity-based and texture-based features. Han et al. (2015) proposed an approach for malware family classification based on visualization images and entropy graphs. The obtained empirical results demonstrated that the authors' proposed approach could classify a malware family with low FPR.

Table 2.3: Summary of hybrid features-based malware detection approaches

Authors	Feature Type							Features Extractor(s)		Remarks
	API	FLF	ExeT	DLL	Opcode	PSI	PEHFs	Static Features	Behavioural Features	
Santos et al. (2013)	×	×	√	×	√	×	×	□	Sandbox using Qemu and Wine	. Implemented a machine learning-based malware detection approach by employing both static features and behavioural-based features.
Gandotra et al. (2014)	×	√	√	×	×	×	√	Python Script	Cuckoo Sandbox	. The effect of using only static features-based, only behavioural features-based, and combination of both static and behavioural features-based analysis, are discussed to show blend of both features could provide better accuracy.
Shijo and Salim (2015)	√	×	×	×	×	√	×	Strings Utility	Cuckoo Sandbox	. Elucidated the combined static and dynamic features will increase the detection accuracy when compared with standalone, static, and dynamic methods.
Awan and Saqib (2016)	√	×	×	×	×	√	√	IDA Pro Disassembler	Cuckoo Sandbox	. Presented a hybrid features based malware detection technique that exploits three different PE file features to classify the PE file as malware or benign.
Bounouh et al. (2017)	×	√	√	×	×	√	×	Hexdive tool and IDA Pro Disassembler	Anubis	. Malware detection using a hybrid features was proposed to demonstrate the classification of malware into respective families than merely discriminating the PE files as benign or malware.

□: Information not explicitly mentioned, FLF: Function Length Frequency, ExeT: Executable Traces, PEHFs: Portable Executable Header Features

Arefkhani and Soryani (2015) introduced local sensitive hashes as a new method based on image processing for malware clustering. They tested three different hashes such as AHash, DHash, and PHash, and from the obtained confusion matrices, they reasonably inferred that PHash was the least confusing of the three.

Zhang et al. (2016) showed that the malware detection problem could be transformed into an image classification problem. They first disassembled the executable files into Opcode sequence and then, converted them into images to recognize whether the source file was malware or benign by using CNN.

Ni et al. (2018) described a malware classification algorithm that performed its task by considering visualization images and Deep Learning-based techniques. First, it converted the disassembled executable files into grayscale images based on the SimHash algorithm and then, used the CNN to ascertain the malware family.

2.5 MALWARE DETECTION USING DEEP LEARNING

Deep Learning-based techniques have become more effective in solving complex classification problems by learning essential characteristics from the input files. Recently, Deep Learning was also used in malware detection and classification (Kan et al., 2018; Kumar et al., 2018).

Saxe and Berlin (2015) designed a Deep Feed-Forward Neural Network model to classify malware using the static features of the PE files. Kolosnjaji et al. (2016) constructed a Neural Network through a combination of Recurrent Neural Network and CNN to perform hierarchical feature extraction. Further, they utilized the N-gram technique for effective detection and classification of the malware.

Tobiyama et al. (2016) used two stages of the Deep Neural Network in their proposed malware detection method for infection detection. They generated an image via the extracted behavioural features from the trained Recurrent Neural Network. Later, the CNN was used to classify the feature images.

Yakura et al. (2018) proposed a method to derive more significant byte sequences in a malware. To achieve this, they used the CNN with attention mechanism for the images generated from the binaries and showed that higher detection accuracy can be achieved. Rhode et al. (2018) presented Opcode sequences-based malware detection method using the Long Short Term Memory and Recurrent Neural Network for early-stage malware prediction.

Table 2.4: Summary of the visualization-based and Deep Learning-based malware detection approaches

Authors	Static Features	Dynamic Features	Feature Selection Technique	Feature(s)	Model Type
Kancherla and Mukkamala (2013)	□	□	×	Wavelet-based, Gabor-based, and Intensity-based Features	Support Vector Machine
Saxe and Berlin (2015)	✓	×	×	Byte/entropy, PSI, PE Import, and PE Metadata Features	Feed Forward Neural Network
Zhang et al. (2016)	✓	×	×	Asm file features, Bytes file features, and Basic file property features	CNN
Kolosnjaji et al. (2016)	×	✓	×	System call Traces	CNN and Recurrent Neural Network
Tobiyama et al. (2016)	×	✓	×	API call Sequence	CNN and Recurrent Neural Network
Kolosnjaji et al. (2017)	✓	×	×	PE Metadata, PE Import, and Opcode Features	CNN and Feed Forward Neural Network
Rhode et al. (2018)	×	✓	×	Machine activity metrics	Long Short Term Memory and Recurrent Neural Network
Yakura et al. (2018)	✓	×	×	□	CNN
Cui et al. (2018)	□	□	×	□	CNN

□: Information not explicitly mentioned

Yan et al. (2018) proposed MalNet, a malware detection method that automatically learns essential features from the raw data. It uses a combination of CNN and Long Short Term Memory network to learn important features from the grayscale images generated from the Opcode sequences.

Cui et al. (2018) presented an approach to boost the automatic detection and discrimination of malware using Deep Learning. As part of the implementation, the authors initially converted the PE files into grayscale images. Further, the CNN was used to discern and classify these images. Additionally, they used the Bat algorithm to address the data imbalance problem.

HaddadPajouh et al. (2018) explored the significance of the Recurrent Neural Network in identifying the Internet of Things malware. They implemented three different Long Short Term Memory structures to spot the Internet of Things malware based on Opcode sequence and showed that their approach could achieve better detection accuracy.

Table 2.4 shows a summary of the visualization-based and Deep Learning-based malware detection approaches.

2.6 OUTCOME OF LITERATURE SURVEY

It was observed from the literature that many malware analysis techniques have been proposed, which employ either static features (Section 2.1) or behavioural features (Section 2.2) or hybrid features (Section 2.3) to ascertain malware and involves either machine learning-based or Deep Learning-based classifiers to precisely classify the input file. As discussed in Section 2.1, the previous works employed static features such as Opcode sequence, byte sequence, API call sequence, PSI, and PE header features in detection and discrimination of the PE files as malware or benign. Though several works in literature emphasize that the static features-based malware detection approaches are proficient to precisely detect malware, they are nevertheless said to be ineffective with regard to obfuscated malware. Moreover, promising results were presented in literature towards detection of malware when the features extracted from PE files were represented as N-grams. Further, some works highlighted the infeasibility of the malware analysis approaches due to the large dimensionality of the original features set.

Section 2.2 discusses the various behavioural-based malware detection approaches available in literature. Most of the behavioural-based malware detection approaches proposed the sandbox technique. However, it was noted that obfuscated malware could evade the sandbox technique. Thus, Section 2.3 highlights previous works regarding hybrid features-based malware detection approaches. In Section 2.3, it was found that blending both the static features and behavioural features could enhance the detection accuracy of the malware.

Based on the discussed literature in Section 2.4 and Section 2.5, it was noticed that the Deep Learning-based malware detection approach was gaining prominence. It was also noted from the previous research works discussed in Section 2.4 that most of the proposed approaches incorporated the visualization-based technique that converted the PE files to an images. Only a few research works mentioned the feature type used to construct an image in the context of detection of malware. It is also observed that the effect of feature selection at an input level of Deep Learning-based classifiers is not well-studied.

From the literature survey, it was evident that most of the proposed behavioural-based malware detection approaches used the Cuckoo Sandbox to gather the system-

level behaviour of the PE files. It was observed that the WEKA tool is popular among researchers to carry out the classification process using the different classifiers stated in them. Further, evaluation metrics such as Accuracy, TPR, FPR, Precision, Recall, and F-Measure were widely used in the literature.

$$\begin{aligned}
\text{TPR} &= \frac{TP}{(TP+FN)} \quad \text{FPR} = \frac{FP}{(FP+TN)} \\
\text{Precision} &= \frac{TP}{(TP+FP)} \quad \text{Recall} = \frac{TP}{(TP+FN)} \\
\text{F-Measure} &= 2 * \left(\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \right) \\
\text{Accuracy} &= \frac{(TP+TN)}{(TP+TN+FP+FN)}
\end{aligned} \tag{2.1}$$

Here, True Positive (TP) and True Negative (TN) represent malware samples precisely classified as malware, and benign samples accurately classified as benign. A False Positive (FP) and False Negative (FN) represent benign samples incorrectly classified as malware, and malware samples incorrectly classified as benign.

2.6.1 Problem Statement

In order to address these issues, the research problem is defined as, ”Design and develop an efficient (new or enhanced) Windows malware detection approach”.

2.6.2 Research Objectives

The research objectives are:

1. To design and develop an efficient static features-based Windows malware detection approach;

2. To design and develop an efficient dynamic (behavioural) features-based Windows malware detection approach; and

3. Verify the robustness of the proposed approach using publicly available Windows malware samples, and also to measure its efficiency (detection rate) in terms of Accuracy, Recall, and Precision.

2.7 EXPERIMENTAL SETUP

To achieve the research objectives, all experiments were conducted on a host system that had the Ubuntu 14.04.5 LTS operating system, Intel i7-3770 CPU 3.40 Ghz, and 8 GB RAM. Further, to appraise the detection and classification ability of the proposed Windows malware detection approaches as well as the performance of the machine learning-based and Deep Learning-based classifiers, different evaluation metrics were utilized and defined as per Eq. 2.1.

As discussed in Chapter 1, for the behavioural analysis of the PE files necessitates it to be executed in a controlled environment to monitor its runtime activities. In this regard, the Cuckoo Sandbox was employed to generate the behavioural report in the proposed works. Most of the previous research works utilized the same to perform the behaviour analysis of the PE file (see Table 2.2).

2.8 SUMMARY

In this chapter, the existing state-of-the-art approaches concerned with Windows malware detection based on static features, behavioural features, and hybrid features were discussed. Discussions about recent malware detection techniques existing in literature that used the visualization-based and Deep Learning-based approaches were also presented in this chapter. The problem statement and research objectives related to designing and implementation of malware detection techniques that rely on static features and behavioural-based features were highlighted. Finally, the experimental set-up employed in the present research works to accomplish the research objectives was described.

Chapter 3

Analysis of Static Features-based Malware Detection Approach

The PE file is represented as a native file format for all variants of the Windows operating system. Today, majority of the malware target PE files to acquire the necessary information. PE features are extracted from certain parts of the PE files, and these essential features indicate whether the PE file was created or infected to perform malicious activity.

Most of the existing anti-malware defensive solutions employ various malware detection techniques and determine the type of file before they parse to detect the embedded malicious data. They deliberately analyze static features such as binary sequence, header information, function calls, and any other information to determine whether the PE file is benign or malware. However, in a real scenario, it becomes tedious to examine all the gathered features to acquire the most predominant features for the classification operation. Under such circumstances, FST plays a vital role in minimizing the dimensionality of the original features set, which consequently boosts the predictive performance of the classifiers (Li et al., 2017).

3.1 PERFORMANCE EVALUATION OF FILTER-BASED FEATURE SELECTION TECHNIQUES IN CLASSIFYING PE FILES

In the present work, an MDS was designed, implemented, and tested to detect malware based on extracted information related to the PEOHF. Subsequently, the performance of the FSTs in selecting the most relevant PEOHF features crucial for discriminating between benign and malware PE files was also analyzed. The Single-Stage-Feature-Selector was employed to acquire significant features by adapting the filter-based FST. Different FSTs such as DFS (Uysal and Gunal, 2012), MI (Santos et al., 2010), Categorical Proportional Difference (CPD) (Simeon and Hilderman, 2008), and Darmstadt Indexing Approach (DIA) (Jin and Srihari, 2007) were used to select the most significant features. Two sets of experiments were conducted on the Balanced Dataset (BD) and the Unbalanced Dataset (UBD) to evaluate the performance of these four filter-based FSTs.

3.1.1 Methodology

An overview of the proposed MDS, which has two phases, namely, the Training phase and the Prediction phase is shown in Figure 3.1. The Training phase is used to build a training file needed to train the classifier. The Prediction phase measures the detection ability of the trained classifier. However, the main objective of the proposed approach is to measure the performance ability of the chosen four filter-based FSTs.

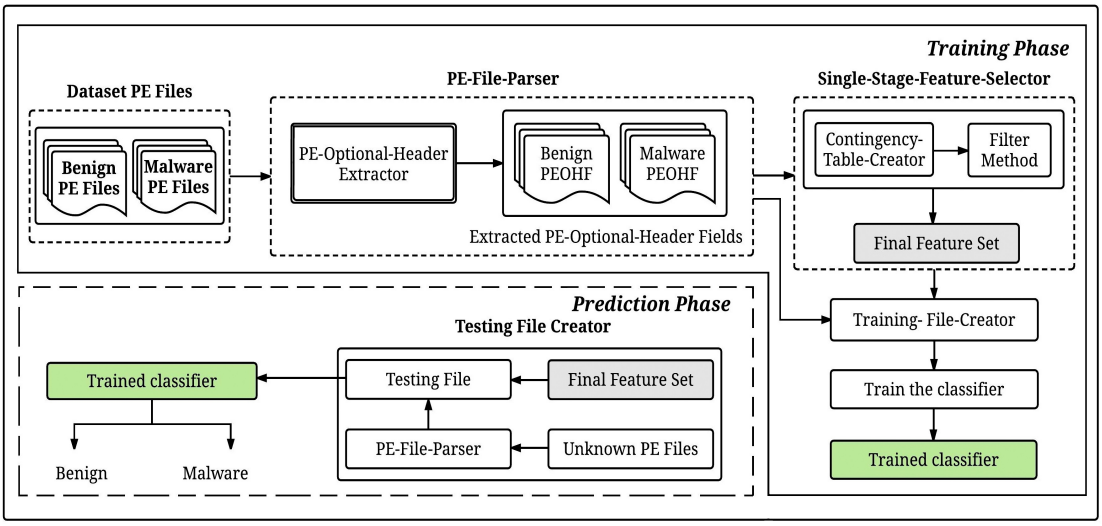


Figure 3.1: Proposed Malware Detection System architecture overview

To train and evaluate the performance of the proposed MDS, a dataset of PE files was collected separately from various sources (see Section 3.1.3) as BD and UBD for further processing. Both the datasets were used for the purpose of analysis of the chosen four filter-based FSTs. Each of these was individually supplied onto the PE-File-Parser to initiate the analysis task.

3.1.1.1 Training Phase

In the Training phase, the MDS is provided with a training set of benign and malware PE files. Each PE file is parsed to extract the features related to the OH fields. The representative features of the PE files are processed using the FST to obtain the most informative features to prepare a training file. To accomplish this task, the training phase utilizes the PE-File-Parser, Single-Stage-Feature-Selector, and Training-File-Creator as its essential components.

3.1.1.2 PE-File-Parser

The prime goal of the PE-File-Parser is to extract PEOHF names with their corresponding values as features from the dataset of PE files and to produce the output as benign and malware PEOHF files. This task is accomplished by employing its subcomponent PE-Optional-Header Extractor, which uses a python module called pefile (Ero, 2017) to acquire the OH features from the input PE files. However, the extracted set of features size is quite large due to the presence of noisy or irrelevant features. Therefore, the Single-Stage-Feature-Selector is utilized to identify the most crucial features required to prepare a training file and simultaneously reduce the dimensionality of the feature set.

3.1.1.3 Single-Stage-Feature-Selector

It plays a crucial role in preserving the informative features and to detect the unknown malware accurately from a number of other benign PE files. To perform this desired operation, it uses the Contingency-Table-Creator as one of its subcomponents.

In the Single-Stage-Feature-Selector, the task of the Contingency-Table-Creator is to create a Contingency Table (CT) that provides the frequency distribution of each feature such as the presence and absence in Benign PEOHF and Malware PEOHF files as an integer count. The FST utilizes the CT to generate a score for each feature, and based on the score, the features are selected as prominent features. In this regard, the four filter-based FSTs chosen in the present work are DFS, MI, CPD, and DIA. Further, to demonstrate their efficiency, the four Final Feature Sets (FFSs) are constructed by selecting the topmost features of different thresholds. However, each FFS consists of features suggested by individual FSTs such as DFS, MI, CPD, and DIA. Finally, a training file is built using the FFS with the PEOHF files corresponding to the training samples. Lastly, the classifier is trained using the constructed training file.

3.1.1.4 Final Feature Set

FFS is a set that consists of distinct benign and malware features of the benign and malware PE files. These features are obtained after carrying out all the processing steps with no further elimination of features. Therefore, it is named as FFS. The features available in the FFS are used to prepare a training file as well as testing files crucial in

measuring the efficiency of the classifiers.

3.1.1.5 Training-File-Creator

The Training-File-Creator creates a training file essential to train the classifiers. It parses the training dataset of the benign and malware PEOHF files with FFS features in order to create a training file.

3.1.1.6 Prediction Phase

In the Prediction phase, the Testing-File-Creator is used to create a testing file necessary to appraise the predictive performance of the trained classifiers. It makes use of the FFS features and the output of the PE-File-Parser to deliver a testing file. The generated testing file is sent to the trained classifier to ascertain whether the test input file is benign or malware.

3.1.2 Filter-based Feature Selection Techniques

This section provides a description of the four filter-based FSTs used in the present work for the purpose of performance analysis. The minimization of the immense dimensionality of the feature set is of the greatest concern in malware classification. The FST identifies the features that have high classification potential and filters features that are noisy or irrelevant. This results in massive reduction of computational cost. The following are the different filter-based FSTs utilized in the present work:

(i) Distinguishing Feature Selector

DFS (Uysal and Gunal, 2012) evaluates the contribution of the features in a representative vector to the class discrimination in a probabilistic approach and computes the score for each feature as per Eq. 3.1,

$$DFS(f) = \sum_{i=1}^{N} \frac{P(c_i|f)}{P(\bar{f}|c_i) + P(f|\bar{c_i}) + 1} \tag{3.1}$$

Where, N is the total number of categories, $P(c_i|f)$ denotes the conditional probability of category c_i when the feature f is present, $P(\bar{f}|c_i)$ signifies the conditional probability of the category c_i when the feature f is absent, and $P(f|\bar{c_i})$ is the conditional probability of feature f given the category other than c_i.

(ii) Mutual Information

MI (Singh et al., 2014) calculates the mutual dependence of any two random variables. It measures the decline in uncertainty about one random variable as a function of the other. If the MI score between two random variables is zero, then the variables are independent, and those with the highest score will have a large reduction in uncertainty. The MI score for a feature and the category pair is computed as per Eq. 3.2.

$$MI(f, c_k) = \sum_{v_f \in \{1,0\}} \sum_{v_{c_k} \in \{1,0\}} P(f = v_f, c_k = v_{c_k}) \ln \frac{P(f = v_f, c_k = v_{c_k})}{P(f = v_f)P(c_k = v_{c_k})} \quad (3.2)$$

Where, f indicates the feature that takes the value $v_f = \{1, 0\}$. If the value of $v_f = 0$, then the file does not contain the feature f, and if $v_f = 1$, it indicates that the file contains the feature f. c_k is the category that takes the value one, i.e., $v_{c_k} = 1$, when the file is present in a category c_k, otherwise, the value is zero, i.e., $v_{c_k} = 0$ indicates the absence of the file in the category c_k.

(iii) Categorical Proportional Difference

CPD (Simeon and Hilderman, 2008) calculates the degree to which a feature distinguishes a specific category from other categories. The attainable values for CPD are limited to the interval (-1, 1). The CPD score near to -1 denotes that the feature is present in most of the files in all the categories. If the score is equal to one, it represents that the feature is present in the file of only one category. The CPD score for the feature f in the category c_k is formulated as per Eqs. 3.3 and 3.4.

$$CPD(f, c_k) = \frac{N_{f,c_k} - N_{f,\overline{c_k}}}{N_f} \quad (3.3)$$

The CPD for the feature f is the ratio associated with the category c_k for which the value is the highest.

$$CPD(f) = max_k\{CPD(f, c_k)\} \quad (3.4)$$

(iv) Darmstadt Indexing Approach

DIA (Jin and Srihari, 2007) FST considers the properties of the features, categories, and pair-wise relationships as a dimension. The DIA score for the feature f is calculated as per Eq. 3.5.

$$DIA(f, c_k) = \frac{N_{f,c_k}}{N_f} \tag{3.5}$$

Where, N_{f,c_k} are the files containing feature f in the category c_k, and N_f denotes the number of files containing the feature f.

3.1.3 Experimental Results and Analysis

The experimental data consists of two datasets, BD and UBD. BD consists of an equivalent number of benign and malware PE files. In UBD, the malware PE files are twice the count of the benign PE files as shown in Table 3.1. The benign PE files include the Windows system files collected from a freshly installed Windows XP virtual machine. The malware PE files are downloaded from the public source VirusShare (VirusShare, 2011). To ensure that all the files in the dataset are correctly labelled, both the datasets are scanned with more than 40 anti-malware engines available on VirusTotal (VirusTotal, 2004a).

Table 3.1: Experiment Dataset details

Dataset Type	No. of Benign PE files	No. of Malware PE files
Balanced Dataset (**BD**)	200	200 (trojan(100) + backdoor(100))
Unbalanced Dataset (**UBD**)	200	400 (trojan(100) + backdoor(100) + rootkit(200))

As explained earlier (in Section 3.1.1), the PE-File-Parser receives both the benign and malware PE files to extract the information related to the OH Fields as features. Each extracted feature is indicated by its name and its corresponding value. The derived features are gathered to form an original feature set. Further, to attain the best features, the FSTs were applied separately onto the original feature set to get the score for each feature separately, and the topmost K number of features was chosen based on the highest score. Four different FSTs such as DFS, MI, CPD, and DIA were used with the intention of identifying the best one. Experiments were conducted for different values of K such as 25, 50, 75, and 100. These best features were processed to prepare a training file as well as testing files, which were supplied to the classifiers in order to

determine which classifier achieved the best malware detection rate with low FPR.

The main aim was to perform a comparative analysis of the four different FSTs and to identify the best FST with the potency to recommend the most significant features. The classifier predictive performance depends on the features used in the training file. From that perspective, the FFS generated consists of features recommended by the Single-Stage-Feature-Selector. The FFS features are employed as final features since there is no further feature elimination, and these features are used to prepare the training as well as the testing files desirable to measure the efficiency of the classifiers. Six different classifiers such as the Sequential Minimal Optimization (SMO), Simple Logistic, Logistic, J48, RF, and Random Tree available in WEKA were used to know which classifier outperformed for the derived FFS. The performance of each classifier was evaluated using evaluation metrics such as TPR, FPR, and Accuracy.

Two sets of experiments were carried out in the present work. In the first set of experiments, the BD was considered. The PE-File-Parser processed both the benign and malware PE files to extract the entire PEOHFs' information as features. The extracted data was then stored into an appropriate output file, and a separate output file was maintained for each individual PE file. Each extracted data was treated as an individual feature, which is a prerequisite task. Since 200 malware PEOHF and 200 benign PEOHF files produced by the PE-File-Parser were considered, these files were directly sent as input to the chosen FSTs separately. At first, the DFS FST was executed on 1323 distinct features to generate the DFS score using Eq. 3.1. Since all these 1323 features cannot be used to train the classifier, the predominant features were identified and selected as crucial features based on their highest score. To evaluate the performance of each FST, the topmost K number of features were selected in increments of 25, i.e., K=25, K=50, K=75, and K=100. Accordingly, the corresponding FFSs were prepared. Similarly, the other three FSTs were applied on the 1323 features separately to get the corresponding scores using Eq. 3.2, Eq. 3.4, and Eq. 3.5. Subsequently, the topmost K number of features was selected in increments of 25, i.e., 25, 50, 75, and 100 and the FFSs were constructed separately. The first 25 features that are recommended by each FST of BD and UBD, respectively, have tabulated in Appendix A and Appendix B. The features mentioned mainly enhance the prediction performance of the classifier during the classification of the PE as benign or malware.

Table 3.2: Accuracy of different classifiers on BD and UBD

Feature Selection Technique	No. of features	Accuracy (%)																	
		SMO			Simple Logistic			Logistic			J48			Random Forest			Random Tree		
		BD	UBD	Diff	BD	UBD	Diff	BD	UBD	Diff	BD	UBD	Diff	BD	UBD	Diff	BD	UBD	Diff
DFS	25	98.677	99.308	0.631	98.677	99.308	0.631	96.296	98.270	1.974	98.677	99.135	0.458	98.148	99.308	1.160	98.148	99.135	1.160
MI	25	98.677	99.308	0.631	98.677	99.308	0.631	97.090	97.232	0.142	98.677	99.135	0.458	98.677	99.135	0.458	98.677	98.962	0.285
CPD	25	53.968	67.474	13.506	56.085	67.474	11.389	56.614	67.474	10.860	52.910	67.474	14.564	56.614	67.474	10.860	56.614	67.474	10.86
DIA	25	51.852	67.301	15.449	51.852	67.474	15.622	51.323	67.301	15.978	50.265	67.474	17.209	51.323	67.474	16.151	52.116	67.301	15.185
DFS	50	98.677	99.135	0.458	98.677	99.308	0.631	98.148	98.616	0.468	98.677	99.135	0.458	98.677	99.308	0.631	98.413	98.270	0.143
MI	50	98.677	99.308	0.631	98.677	99.308	0.631	96.032	98.789	2.757	98.677	99.135	0.458	98.413	99.135	0.722	96.296	99.135	2.839
CPD	50	61.905	67.820	5.915	62.169	67.474	5.305	63.492	67.820	4.328	56.878	67.474	10.596	63.757	67.820	4.063	63.228	67.820	4.592
DIA	50	53.439	67.647	14.208	53.439	67.474	14.035	53.704	67.647	13.943	51.587	67.474	15.887	53.439	67.820	14.381	53.704	67.647	13.943
DFS	75	98.677	99.308	0.631	98.677	99.308	0.631	98.413	98.962	0.549	98.677	99.135	0.458	98.148	99.135	0.987	98.413	98.616	0.203
MI	75	98.677	99.308	0.631	98.677	99.308	0.631	98.148	98.097	0.051	98.677	99.135	0.458	98.413	99.308	0.895	97.619	98.270	0.651
CPD	75	63.757	68.512	4.755	61.905	68.166	6.261	64.286	68.512	4.226	56.878	68.166	11.288	64.286	68.512	4.226	63.492	68.512	5.020
DIA	75	57.672	67.647	9.975	57.407	67.474	10.067	57.672	67.647	9.975	55.027	67.474	12.447	57.672	67.820	10.148	57.672	67.647	9.975
DFS	100	98.677	99.308	0.631	98.677	99.135	0.458	98.148	98.616	0.468	98.677	99.135	0.458	98.413	99.308	0.895	97.355	98.443	1.088
MI	100	98.677	99.308	0.631	98.677	99.308	0.631	97.884	98.270	0.386	98.677	99.135	0.458	98.148	99.135	0.987	97.619	98.097	0.478
CPD	100	69.550	69.841	0.291	66.402	69.031	2.629	70.370	69.550	0.820	62.434	68.858	6.424	70.106	69.550	0.556	69.312	69.550	0.238
DIA	100	63.492	67.474	3.982	63.227	67.474	4.247	64.286	67.647	3.361	60.318	67.474	7.156	64.286	67.820	3.534	64.021	67.647	3.626
Diff: $\lvert$**BD** − **UBD**$\rvert$ where $\lvert\ \rvert$ indicates absolute value																			

From the experimental results depicted in Table 3.2, it was noticed that the SMO, Simple Logistic, and J48 classifiers outperformed by achieving maximum identical accuracy of 98.677% with 0.013 FPR (see Table 3.4) for all the topmost K number of features such as 25, 50, 75, and 100 recommended by the DFS and MI FSTs. The performance was not much appreciable when the same training file and testing files were supplied to other classifiers such as Logistic, RF, and Random Tree. The accuracy achieved by each of them is tabulated in Table 3.2. At the same time, it was noticed that the features recommended by the CPD and DIA FSTs for the topmost K number of features (K=25, K=50, K=75, and K=100) underperformed by achieving reduced accuracy.

The second sets of experiments were conducted with the UBD mentioned in Table 3.1. The operational steps performed in this set of experiments were similar to the first set of experiments, except for the number of files. In the second set of experiments, the PE-File-Parser was fed 200 benign and 400 malware PE files in order to extract all the PE OH Fields using the PE-Optional-Header-Extractor. The PE-File-Parser produced benign (200) and malware (400) PEOHF files and these files were supplied as input to the FSTs chosen in this experimental work. The DFS FST was executed on 1877 features to compute a score for each feature using Eq. 3.1.

Furthermore, the same original features set of 1877 features was supplied as input to the other FSTs consecutively to determine the score for the features using Eq. 3.2, Eq. 3.4, and Eq. 3.5. The uppermost K number of features was selected in increments of 25 up to 100 to prepare a separate FFS, and each of them was used to build a training file as well as testing files to measure the efficiency of the classifiers. The accuracy produced by the different classifiers is presented in Table 3.2. In the case of UBD, the SMO and Simple Logistic classifiers outperformed by accomplishing highest equivalent accuracy of 99.308% with 0.014 FPR (see Table 3.4) for all the topmost number of features (K=25, K=50, K=75, and K=100) recommended by the DFS and MI FSTs. The other classifiers such as J48, Logistic, RF, and Random Tree underachieved for the same topmost number of features (K=25, K=50, K=75, and K=100) and the accuracy accomplished by each of them is also tabulated in Table 3.2. FSTs such as the CPD and DIA were found to be inefficient and performed similar to the first set of experiments.

The two sets of experiments were computed substantially and analyzed thoroughly to decide the best FST based on the accuracy produced by the classifier. In this direction, the obtained and analyzed results proved that FST certainly provided the most significant features based on the computed score, but all the features may not contribute to the detection of the malware.

3.1.3.1 Evaluation on Balanced and Unbalanced Datasets

In order to measure the accuracy variation between BD and UBD, the difference was calculated. Table 3.2 demonstrates that the difference in classifier accuracy is not of much significance.

It was noticed that the SMO classifier produced an accuracy of 98.677% for BD and 99.308% for UBD with 25 features recommended by the DFS. The difference was 0.631% (i.e., $|98.677 - 99.308|$). Further, the SMO classifier accuracy difference between BD and UBD was 0.458%, 0.631%, and 0.631%, respectively, for the other three FFSs of sizes 50, 75, and 100 features suggested by the DFS. This indicates that the classifier accuracy did not decline for the UBD and justified that the accuracy variation was in the range of 0.458% to 0.631%. Similarly, when the same 25, 50, 75, and 100 features were applied to the Simple Logistic, Logistic, J48, RF, and Random Tree classifiers, their counselled range was 0.458% to 0.631%, 0.468% to 1.974%, 0% to 0.458%, 0.631% to 1.16%, and 0.143% to 1.16%, respectively. The accuracy difference in the range of 0% to 0.631% indicated better classification and specified that the efficiency of the classifiers did not minimize much and was good enough for both the BD and UBD. Other FST such as the MI was also efficient and guaranteed that the classifier performed well in the range of 0.458% to 0.631%.

On the other hand, the classifier performance was inefficient for the top features selected based on the highest score recommended by the CPD and DIA FSTs. The expected range to determine the better classifier accuracy was very high. Accordingly, as per the observations from Table 3.2, the SMO classifier gained an accuracy of 53.968% for BD and 67.474% for UBD with the highest difference of 13.506% for the top 25 features selected based on the CPD score. Subsequently, when the SMO classifier accuracy was checked with the foremost 100 features based on the CPD score, it attained 69.550% for BD and 69.841% for UBD with least difference of 0.291%. The other

Table 3.3: Comparison of TPR achieved by different classifiers on different feature lengths of BD and UBD

No. of Features	Classifiers	DFS		MI		CPD		DIA	
		BD	UBD	BD	UBD	BD	UBD	BD	UBD
25	**SMO**	0.987	0.993	0.987	0.993	0.540	0.675	0.519	0.673
	Simple Logistic	0.987	0.993	0.987	0.993	0.561	0.675	0.519	0.675
	Logistic	0.963	0.983	0.971	0.972	0.566	0.675	0.513	0.673
	J48	0.987	0.991	0.987	0.991	0.529	0.675	0.503	0.675
	Random Forest	0.981	0.993	0.987	0.991	0.566	0.675	0.513	0.675
	Random Tree	0.981	0.993	0.987	0.990	0.566	0.675	0.521	0.673
50	**SMO**	0.987	0.991	0.987	0.993	0.632	0.678	0.534	0.676
	Simple Logistic	0.987	0.993	0.987	0.993	0.627	0.675	0.534	0.675
	Logistic	0.979	0.986	0.968	0.988	0.632	0.678	0.537	0.676
	J48	0.984	0.991	0.984	0.991	0.503	0.675	0.503	0.675
	Random Forest	0.987	0.993	0.987	0.991	0.635	0.678	0.534	0.678
	Random Tree	0.976	0.983	0.974	0.991	0.635	0.678	0.537	0.676
75	**SMO**	0.987	0.993	0.987	0.993	0.638	0.685	0.577	0.676
	Simple Logistic	0.987	0.993	0.987	0.988	0.619	0.682	0.574	0.675
	Logistic	0.984	0.990	0.981	0.981	0.643	0.685	0.577	0.676
	J48	0.987	0.991	0.987	0.991	0.569	0.682	0.550	0.675
	Random Forest	0.981	0.991	0.976	0.993	0.643	0.685	0.577	0.678
	Random Tree	0.984	0.986	0.984	0.983	0.635	0.685	0.577	0.676
100	**SMO**	0.987	0.991	0.987	0.995	0.698	0.696	0.619	0.675
	Simple Logistic	0.987	0.991	0.987	0.990	0.680	0.690	0.616	0.675
	Logistic	0.987	0.986	0.979	0.983	0.701	0.696	0.632	0.676
	J48	0.984	0.991	0.984	0.991	0.545	0.689	0.545	0.675
	Random Forest	0.984	0.993	0.981	0.991	0.701	0.696	0.632	0.678
	Random Tree	0.976	0.984	0.979	0.981	0.696	0.696	0.630	0.676

classifiers' performance was not remarkable and achieved very less accuracy with maximum distinguishable range. The observed accuracy difference range for the Simple Logistic classifier was between 0.112% -15.622%, for the Logistic classifier it was between 0.051% - 15.978%, for the J48 classifier between 0.458% - 17.209%, for the RF classifier between 0.556% - 16.151%, and lastly, for the Random Tree classifier between 0.143% - 15.185%. The evaluation on BD and UBD showed that the accuracy difference range (i.e., $<1\%$) did not have much impact on the efficiency of the classifier.

Table 3.4: Comparison of FPR achieved by different classifiers on different feature lengths of BD and UBD

No. of Features	Classifiers	DFS		MI		CPD		DIA	
		BD	UBD	BD	UBD	BD	UBD	BD	UBD
25	**SMO**	0.013	0.014	0.013	0.014	0.455	0.675	0.477	0.676
	Simple Logistic	0.013	0.014	0.013	0.014	0.435	0.675	0.476	0.675
	Logistic	0.037	0.025	0.029	0.033	0.429	0.675	0.483	0.676
	J48	0.013	0.015	0.013	0.015	0.466	0.675	0.493	0.675
	Random Forest	0.018	0.012	0.013	0.012	0.429	0.675	0.488	0.675
	Random Tree	0.018	0.012	0.013	0.016	0.429	0.675	0.474	0.676
50	**SMO**	0.013	0.014	0.013	0.014	0.364	0.668	0.462	0.668
	Simple Logistic	0.013	0.014	0.013	0.014	0.369	0.675	0.461	0.675
	Logistic	0.021	0.018	0.032	0.022	0.364	0.668	0.458	0.668
	J48	0.013	0.015	0.013	0.015	0.503	0.675	0.503	0.675
	Random Forest	0.013	0.014	0.013	0.015	0.361	0.668	0.463	0.668
	Random Tree	0.024	0.019	0.026	0.012	0.361	0.668	0.458	0.668
75	**SMO**	0.013	0.014	0.013	0.014	0.359	0.653	0.419	0.668
	Simple Logistic	0.013	0.014	0.013	0.014	0.377	0.660	0.421	0.675
	Logistic	0.016	0.013	0.018	0.023	0.353	0.653	0.419	0.668
	J48	0.013	0.015	0.013	0.015	0.427	0.660	0.445	0.675
	Random Forest	0.018	0.015	0.024	0.014	0.353	0.653	0.419	0.668
	Random Tree	0.016	0.018	0.016	0.025	0.361	0.653	0.419	0.668
100	**SMO**	0.013	0.014	0.013	0.014	0.298	0.632	0.378	0.672
	Simple Logistic	0.013	0.014	0.013	0.014	0.317	0.642	0.380	0.675
	Logistic	0.016	0.015	0.021	0.017	0.296	0.632	0.364	0.668
	J48	0.013	0.015	0.013	0.015	0.451	0.646	0.451	0.675
	Random Forest	0.016	0.014	0.019	0.015	0.296	0.632	0.364	0.668
	Random Tree	0.024	0.021	0.021	0.020	0.301	0.632	0.366	0.668

3.1.3.2 Analysis of Evaluation Metrics

The performance of the proposed MDS was analyzed in terms of popular evaluation metrics such as the TPR and FPR. The TPR (FPR) indicated that the test input files were correctly (incorrectly) classified. For any MDS, it is desired that the TPR should be high and the FPR should be as low as possible. Table 3.3 and Table 3.4 summarize the TPR and FPR for different topmost K number of features such as 25, 50, 75, and 100 recommended by different FSTs such as the DFS, MI, CPD, and DIA.

From the statistics in Table 3.3, it can be easily inferred that MDS achieved high TPR of 0.987 for features of different thresholds in terms of 25, 50, 75, and 100 recommended by the DFS and MI on the BD with SMO and Simple Logistic classifiers. Similarly, maximum TPR of 0.993 was accomplished by the SMO and Simple Logistic classifiers for all the foremost features of various thresholds (25, 50, 75, and 100) suggested by the DFS and MI. On the other side, the features suggested by the CPD and DIA attained very low TPR.

Main reason for False Positive is, some of the malware do not alter the PEOHFs, rather, they alter other static features of the PE files such as DOS header, File Header, etc. Such malware samples get misclassified. This is one of the limitations of the static features-based detection. Misclassification also depends on Final Feature Set and classifier. However, Lower FPR value represents the best FST. However, the classifiers were evaluated with the features recommended by the DFS, MI, CPD, and DIA. Table 3.4 shows the FPR achieved by the classifiers. The SMO and Simple Logistic classifiers attained the lowest FPR of 0.013 for all the topmost numbers of features (25, 50, 75, and 100) recommended by the DFS and MI on the BD. Correspondingly, the same classifiers were successful in obtaining minimum FPR of 0.014 for the features suggested by the DFS and MI for the UBD. Moreover, the same classifiers achieved slightly high FPR for all the features of different thresholds (25, 50, 75, and 100) suggested by the CPD and DIA FSTs. In comparison, it is evident that FSTs such as the DFS and MI were able to influence the classifiers to attain highest TPR with lowest FPR.

3.2 SUMMARY

The proposed MDS is proficient in precisely distinguishing malware and benign PE files based on the PEOHF features recommended by the Single-Stage-Feature-Selector. The prime task of the present work was to investigate the effectiveness of the filter-based FSTs such as the DFS, MI, CPD, and DIA in classifying the PE files as benign or malware. The experiments carried out were evaluated using different classifiers available in the WEKA tool. From the experimental observation, it was found that the best FSTs were DFS and MI, since the features suggested by them resulted in obtaining better classifier accuracy. The classifiers performed well on both the BD and UBD for different feature lengths of 25, 50, 75, and 100. The accuracy difference calculation manifested

that the range specification of $<1\%$ did not affect the efficiency of the classifiers on BD and UBD.

Chapter 4

Analysis of Behavioural Features-based Malware Detection Approach

Static features-based detection techniques often require a human analyst to go through the code to understand malware functionality. Unfortunately, the anti-malware defensive solutions receive a large number of malware samples each day, thus making the manual analysis a tedious task. Consequently, there is necessity of behavioural features-based malware detection techniques.

This chapter describes the proposed behavioural-based Window malware detection approaches in which various feature engineering techniques have been incorporated to achieve better detection rate. The prime goal of the proposed approaches is to identify and classify the malware with the crucial features recommended by the FST. Further, to achieve malware detection, different experiments were conducted, and for all those experiments carried out, the experimental system was the same as described in Section 2.7.

4.1 WINDOWS MALWARE DETECTION BASED ON CUCKOO SANDBOX GENERATED REPORT USING MACHINE LEARNING ALGORITHM

Computerized malware examination frameworks (or sandboxes) are comprehensively used to detect malware based on behavioural traits. These frameworks allow the unknown malware to execute in an isolated environment and screen its runtime behaviour. The main benefit of this work is that it can recognize the unseen malware based on the observed activities gathered during the execution of the malware. Majority of the sandboxes observe at the system call interface. System calls are a routine that allows the operating system to interact with the user-level process to perform their desired task.

4.1.1 System Architecture

The proposed work distinguishes between malware and benign files on the basis of system calls' sequence structured using a heuristic method called N-grams analysis. It adopts the IG technique to compute the score for the each N-gram and extracts the top N-grams based on the highest IG score in order to prepare a Final Feature Vector (FFV)

needed for classification. Figure 4.1 depicts an overview architecture of the proposed work.

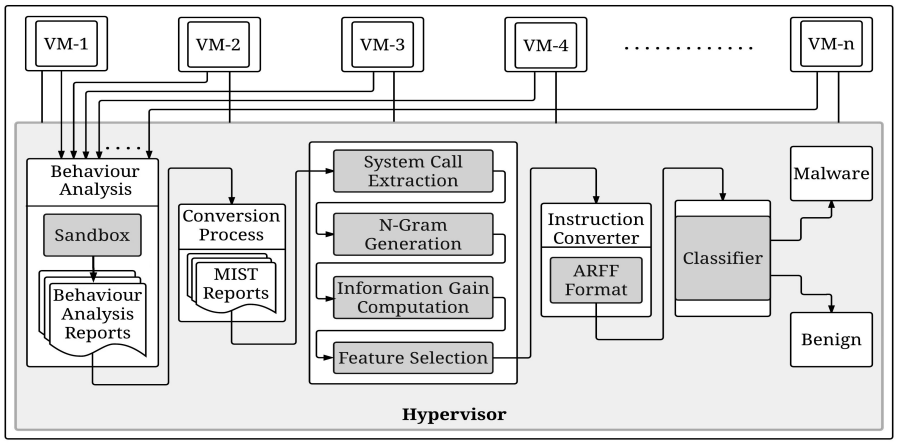


Figure 4.1: System Architecture of the proposed work

4.1.2 Behaviour analysis

Since the Cuckoo Sandbox functions at the hypervisor as a separate entity, it examines the behaviour of the malware running on the Virtual Machines to obtain the behavioural analysis report of the running PE files in the JSON file format.

4.1.3 Conversion process

The analysis reports obtained in the JSON file format are pre-processed to obtain MIST, since it is a preferred format that uses a smaller file size and reduces processing time. Since the proposed approach is specific to observation of monitored system calls, it is concerned with the operation field of the MIST files to generate N-grams (4 bytes) files as shown in Figure 4.2.

To generate the N-gram files, we follow the following steps:

- System calls extraction;
- N-gram generation;
- Sorting of N-grams; and
- Duplicate removal.

In *first step* system call extraction, only the operation field is selected, i.e., the system calls of all the benign MIST files (1, 2, ...,10, 11, ... n) and all the malware

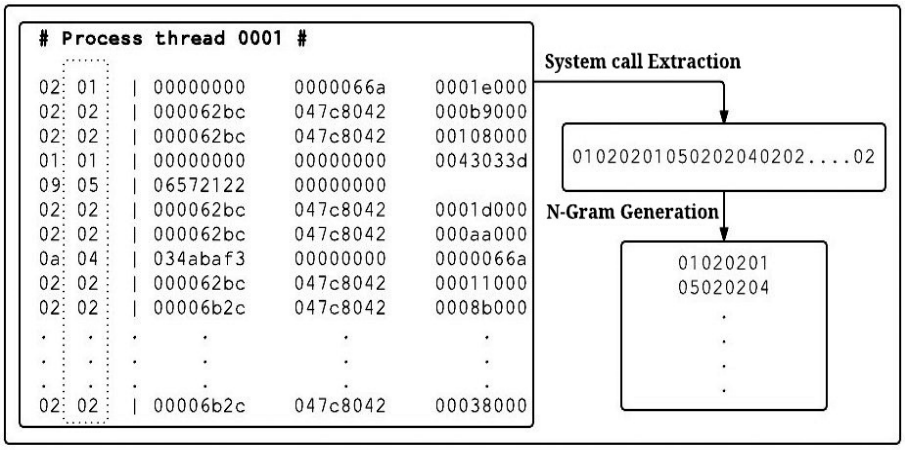
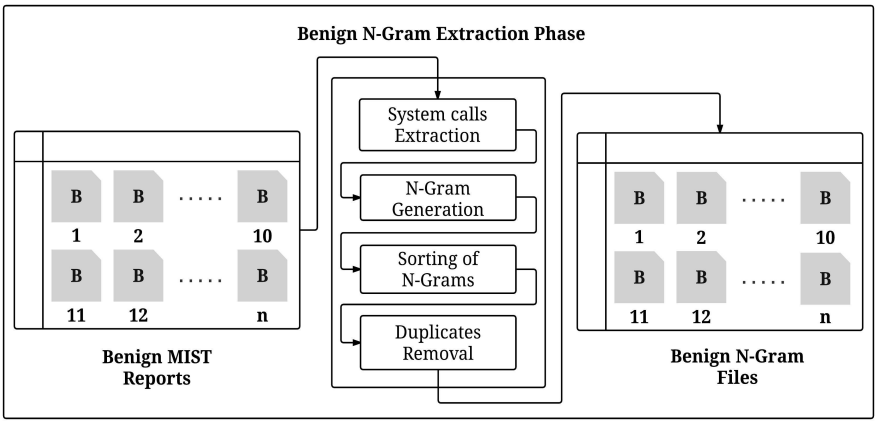


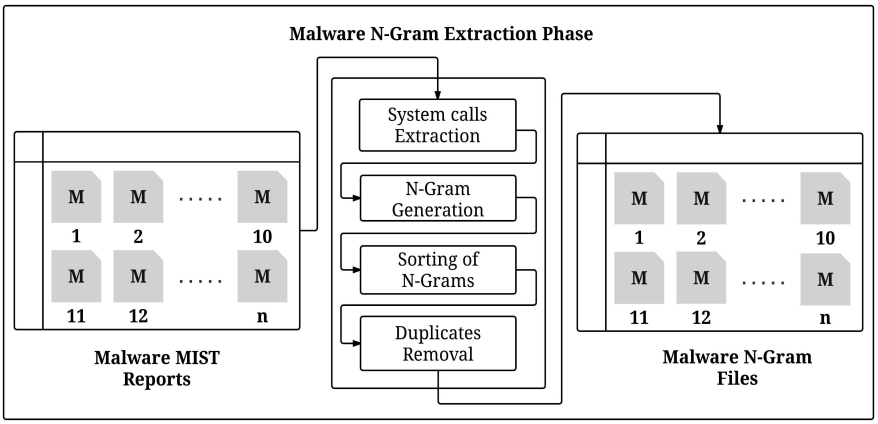
Figure 4.2: Snippet of N-gram extraction using MIST file.

MIST files (1, 2, ...,10, 11, ... n) as shown in Figure 4.3, since there is a record of all the system level behaviours. The extracted operation fields are stored in a text file and grouped in sequence to form N-grams of variable lengths, i.e., N=2, N=3, N=4, etc. The lengthier the N-grams size, the better the characteristics are represented. A snippet of extraction is shown in Figure 4.2. The N-grams have been grouped of length four bytes, while forming N-grams in the *second step* of the generation phase. In the *third step*, the formed N-grams are placed in descending order to get the highest order sequence of N-grams. Following this, any duplicates if observed are removed to get unique N-grams. These can be employed for better feature selection and also for better classification.

The above explanation is prerequisite for the feature selection approach, since it cannot be performed without the N-gram formation. The formed Benign N-gram files [B1, B2, B3,. . . ,Bn] and Malware N-gram files [M1, M2, M3, . . .,Mn] must undergo the union operation considering each benign N-gram files [B1 $\cup$ B2 $\cup$ B3 $\cup$ . . . $\cup$ Bn] and malware N-gram files [M1 $\cup$ M2 $\cup$ M3 $\cup$ . . . $\cup$ Mn]. After this, the benign union N-gram files and the malware union N-gram files must be placed in non-increasing order and the duplicates must be removed, if observed to achieve unique benign N-gram files and unique malware N-gram files. The occurrence of each unique benign N-gram in the benign N-gram files is observed and tabulated in the N-gram frequency table for the benign class, and in the same way, the occurrence of each unique malware N-gram in the malware N-gram files is observed and tabulated in the N-gram frequency table for the malware class.



(a)



(b)

Figure 4.3: System call extraction phase

The feature CT is then prepared based on the values accommodated in the N-gram frequency table for the benign category and malware category as depicted in Figure 4.4. The feature CT is used to calculate the IG (Reddy and Pujari, 2006) using the Eq. 4.1.

$$IG(N-gram) = \sum_{v_{N-gram} \in \{0,1\}} \sum_{C \in \{C_i\}} P(v_{N-gram}, C) \log \frac{P(v_{N-gram}, C)}{P(v_{N-gram}), P(C)} \tag{4.1}$$

Where, C is one of the two categories - benign or malware and v_{N-gram} is the value of the N-gram. v_{N-gram} = 1 indicates that the N-gram present either in the benign N-gram files or malware N-gram files and v_{N-gram} = 0, otherwise. P(v_{N-gram}, C) is the proportion of N-gram files in C in which the N-gram takes on the value v_{N-gram}. P(v_{N-gram}) is the proportion of benign N-gram files or malware N-gram files in the

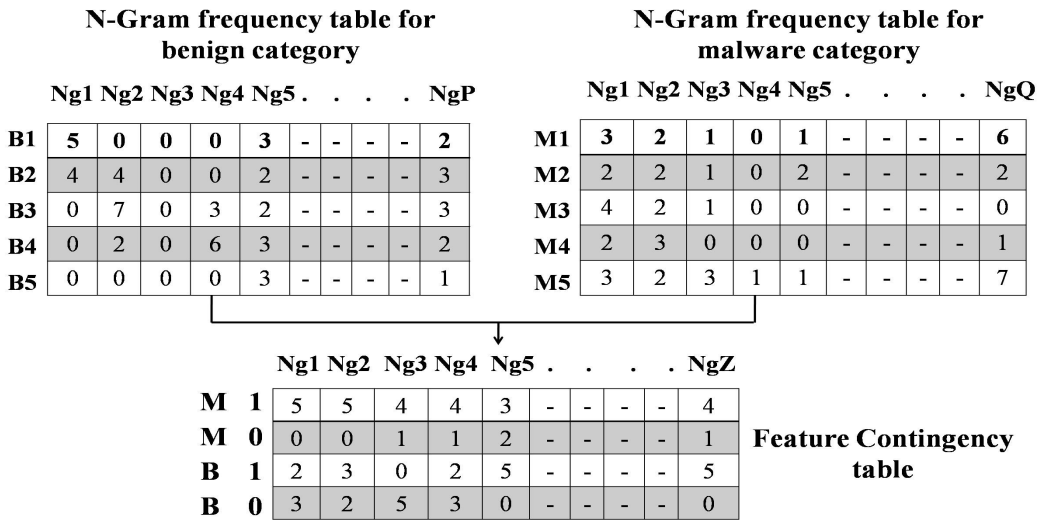


Figure 4.4: N-gram frequency table for benign class and malware class with feature CT

entire training set such that the N-gram takes the value v_{N-gram}. P(C) is the proportion of the dataset belonging to category C. The N-grams are organized in non-increasing order based on the IG score and the topmost L number of N-grams is extracted as the best features for the purpose of classification.

4.1.4 Instruction Converter

The instruction converter converts the extracted features into an Attribute-Relation File Format (ARFF) file. ARFF is an ASCII text file that describes a list of instances sharing a set of attributes. It is an important process because the classifiers of the WEKA tool used in the present approach works with the ARFF file.

4.1.5 Experiment Results

The experimental data in the current work consists of 3000 benign MIST files and 3100 malware MIST files. The malware MIST files consist of four different families such as Swizzor (1000), Basun (1000), AutoIt (1000), and Kelihos Trojan (100). Among the considered four different malware families, the first three were collected from the public source (Konrad, 2015) and the remaining 100 malware MIST files were obtained by implementing the MIST conversion process for all the runtime behavioural reports produced by the Cuckoo Sandbox by injecting the Kelihos Trojan. As explained earlier, N-grams of different sizes such as 2bytes, 3bytes, and 4bytes were extracted to measure which N-gram size achieves the best detection rate. A separate experiment was

conducted for each N-gram size. The N-grams were sorted in decreasing order based on the IG score and any duplicate N-grams, if found, were removed. The class-wise document frequency for each class was determined for each N-gram to prepare the CT. The IG method was used to calculate the score for each N-gram and the top K N-grams were determined based on the highest IG score. Experiments were conducted for different values of K such as 200, 400, and 600. Further, the best features were drawn at each K value for different N-gram lengths. The best features were pre-processed through the instruction converter to prepare ARFF files for the selected N-grams. The ARFF files were submitted to the WEKA tool for classification. A wide set of experiments were conducted to determine which classifier achieved best malware detection rate with low FPR. The performance of several classification algorithms stated in the WEKA tool was evaluated.

The objective was to know the best classification algorithm among the several stated in the WEKA tool. From that perspective, six classifiers were selected among the eight different categories mentioned in the WEKA tool. The six classifiers chosen were the Bayesian Logistic Regression, SPegasos, Ib1, Bagging, Part, and J48 classified under Bayes, functions, lazy, meta, rules, and trees of WEKA. For the purpose of evaluation, the values of TPR, FPR, Precision, Recall, F-measure, Receiver Operating Characteristics (ROC) Area and Accuracy for all the chosen six classifiers was measured and tabulated in Table 4.1 and Table 4.2.

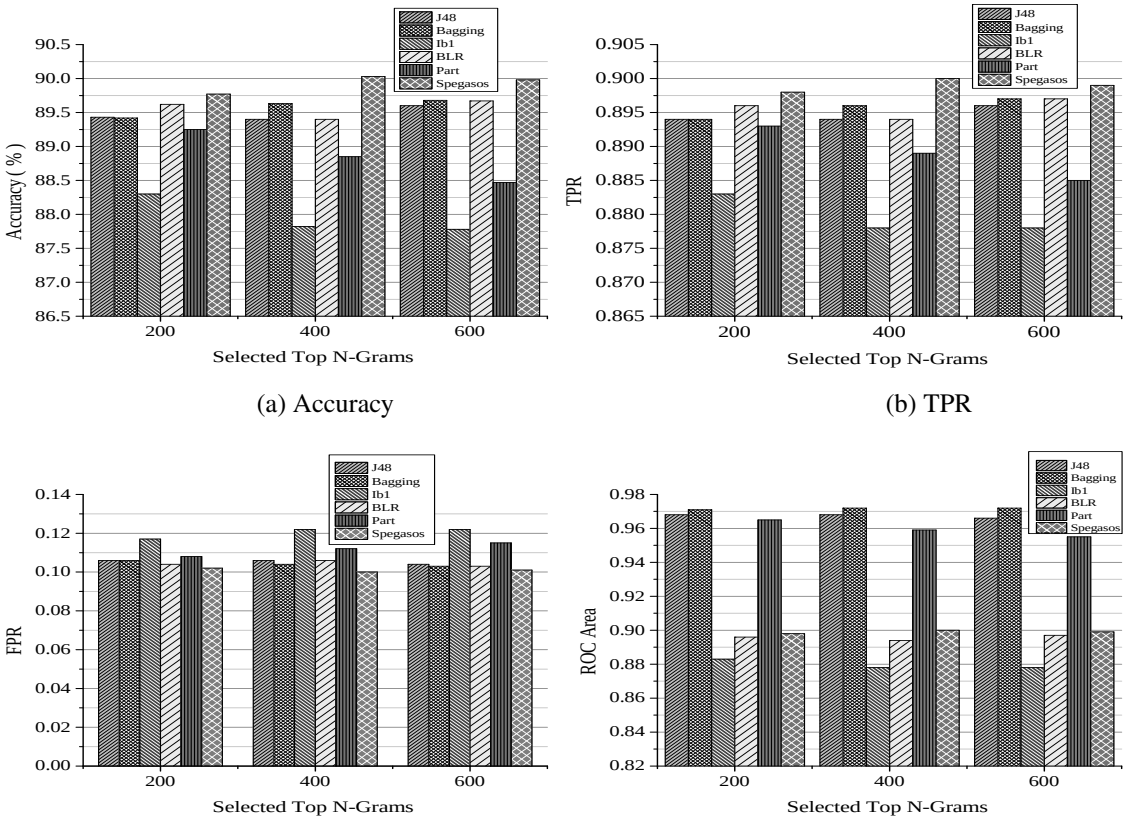
Two sets of experiments were carried out: In the first set of experiments, N-grams of three bytes were considered in order to select the top N-grams based on the highest score of IG. The top N-grams were selected in terms of 200, 400, and 600. From the experimental observation, as shown in Figure 4.5, the highest accuracy was 89.77% for 200 N-grams, 90.03% for 400 N-grams, and 89.88% for 600 N-grams as yielded by the SPegasos classifier (see Figure 4.5a). The highest TPR of 0.898 for 200 N-grams, 0.9 for 400 N-grams, and 0.899 for 600 N-grams was produced by the SPegasos classifier (see Figure 4.5b). The lowest FPR of 0.102 for 200 N-grams, 0.1 for 400 N-grams, and 0.101 for 600 N-grams was given by the SPegasos classifier (see Figure 4.5c). The ROC curves are mainly used to compare the classification capability of the different algorithms. Among the number of classifiers tested in the present work, it was observed that SPegasos classifier attained the best results.

Table 4.1: WEKA Classification results for N-gram Length **3** bytes

	N-gram Length= 3 Selected Top N-grams = 200						**N-gram Length= 3 Selected Top N-grams = 400**						**N-gram Length= 3 Selected Top N-grams = 600**						
Classifier	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	
TPR	0.894	0.902	0.881	0.912	0.899	0.896	0.882	0.894	0.874	0.903	0.877	0.886	0.882	0.904	0.874	0.908	0.888	0.874	***B***
	0.895	0.887	0.885	0.88	0.886	0.899	0.906	0.899	0.882	0.885	0.9	0.915	0.91	0.89	0.882	0.885	0.881	0.923	***M***
	0.894	0.894	0.883	0.896	0.893	0.898	0.894	0.896	0.878	0.894	0.889	0.9	0.896	0.897	0.878	0.897	0.885	0.899	**W**
FPR	0.105	0.113	0.115	0.12	0.114	0.101	0.094	0.101	0.118	0.115	0.1	0.085	0.09	0.11	0.118	0.115	0.119	0.077	***B***
	0.106	0.098	0.119	0.088	0.101	0.104	0.118	0.106	0.126	0.097	0.123	0.114	0.118	0.096	0.126	0.092	0.112	0.126	***M***
	0.106	0.106	0.117	0.104	0.108	0.102	0.106	0.104	0.122	0.106	0.112	0.1	0.104	0.103	0.122	0.103	0.115	0.101	**W**
Precision	0.895	0.888	0.885	0.884	0.888	0.899	0.903	0.898	0.881	0.887	0.897	0.912	0.908	0.891	0.881	0.888	0.882	0.919	***B***
	0.894	0.9	0.881	0.909	0.897	0.897	0.885	0.894	0.875	0.901	0.88	0.889	0.885	0.903	0.875	0.906	0.887	0.88	***M***
	0.894	0.894	0.883	0.897	0.893	0.898	0.894	0.896	0.878	0.894	0.889	0.901	0.896	0.897	0.878	0.897	0.885	0.9	**W**
Recall	0.894	0.902	0.881	0.912	0.899	0.896	0.882	0.894	0.874	0.903	0.877	0.886	0.882	0.904	0.874	0.908	0.888	0.874	***B***
	0.895	0.887	0.885	0.88	0.886	0.899	0.906	0.899	0.882	0.885	0.9	0.915	0.91	0.89	0.882	0.885	0.881	0.923	***M***
	0.894	0.894	0.883	0.896	0.893	0.898	0.894	0.896	0.878	0.894	0.889	0.9	0.896	0.897	0.878	0.897	0.885	0.899	**W**
F-measure	0.894	0.895	0.883	0.898	0.893	0.898	0.893	0.896	0.878	0.895	0.887	0.899	0.894	0.898	0.877	0.898	0.885	0.896	***B***
	0.894	0.893	0.883	0.894	0.892	0.898	0.895	0.897	0.879	0.893	0.89	0.902	0.897	0.896	0.878	0.895	0.884	0.901	***M***
	0.894	0.894	0.883	0.896	0.892	0.898	0.894	0.896	0.878	0.894	0.888	0.9	0.896	0.897	0.878	0.897	0.885	0.899	**W**
ROC Area	0.968	0.971	0.883	0.896	0.965	0.898	0.968	0.972	0.878	0.894	0.959	0.9	0.966	0.972	0.878	0.897	0.955	0.899	***B***
	0.968	0.971	0.883	0.896	0.965	0.898	0.968	0.972	0.878	0.894	0.959	0.9	0.966	0.972	0.878	0.897	0.955	0.899	***M***
	0.968	0.971	0.883	0.896	0.965	0.898	0.968	0.972	0.878	0.894	0.959	0.9	0.966	0.972	0.878	0.897	0.955	0.899	**W**
Accuracy (%)	89.43	89.42	88.30	89.62	89.25	***89.77***	89.40	89.63	87.82	89.40	88.85	***90.03***	89.60	89.68	87.78	89.67	88.47	***89.88***	
TPR: True Positive Rate, FPR: False Positive Rate, C1: J48, C2: Bagging, C3: Ib1, C4: Bayesian Logistic Regression, C5: Part, C6: SPegasos, B: Benign, M: Malware, W: Weighted Average																			

Table 4.2: WEKA Classification results for N-gram Length **4** bytes

	N-gram Length = 4 Selected Top N-grams = 200						**N-gram Length = 4 Selected Top N-grams = 400**						**N-gram Length = 4 Selected Top N-grams = 600**						
Classifier	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	
TPR	0.899	0.902	0.88	0.9	0.899	0.921	0.879	0.904	0.881	0.904	0.885	0.9	0.881	0.907	0.881	0.903	0.88	0.894	***B***
	0.885	0.885	0.878	0.878	0.887	0.88	0.907	0.887	0.873	0.878	0.9	0.891	0.904	0.887	0.882	0.877	0.887	0.905	***M***
	0.892	0.894	0.879	0.889	0.893	0.9	0.893	0.896	0.877	0.891	0.893	0.896	0.893	0.897	0.882	0.89	0.884	0.9	**W**
FPR	0.115	0.115	0.122	0.122	0.113	0.12	0.093	0.113	0.127	0.122	0.1	0.109	0.096	0.113	0.118	0.123	0.113	0.095	***B***
	0.101	0.098	0.12	0.1	0.101	0.079	0.121	0.096	0.119	0.096	0.115	0.1	0.119	0.093	0.119	0.097	0.12	0.106	***M***
	0.108	0.106	0.121	0.111	0.107	0.1	0.107	0.104	0.123	0.109	0.108	0.104	0.108	0.103	0.118	0.11	0.117	0.101	**W**
Precision	0.886	0.884	0.879	0.881	0.889	0.887	0.904	0.889	0.874	0.881	0.898	0.892	0.901	0.889	0.882	0.88	0.886	0.904	***B***
	0.898	0.918	0.88	0.898	0.898	0.9	0.882	0.902	0.88	0.902	0.887	0.899	0.884	0.905	0.881	0.9	0.881	0.895	***M***
	0.892	0.901	0.879	0.889	0.893	0.894	0.893	0.896	0.877	0.891	0.893	0.896	0.893	0.897	0.882	0.89	0.884	0.9	**W**
Recall	0.899	0.921	0.88	0.9	0.899	0.902	0.879	0.904	0.881	0.904	0.885	0.9	0.881	0.907	0.881	0.903	0.88	0.894	***B***
	0.885	0.88	0.878	0.878	0.887	0.885	0.907	0.887	0.873	0.878	0.9	0.891	0.904	0.887	0.882	0.877	0.887	0.905	***M***
	0.892	0.9	0.879	0.889	0.893	0.894	0.893	0.896	0.877	0.891	0.893	0.896	0.893	0.897	0.882	0.89	0.884	0.9	**W**
F-measure	0.893	0.902	0.879	0.89	0.894	0.895	0.891	0.897	0.877	0.892	0.892	0.896	0.891	0.898	0.882	0.891	0.883	0.899	***B***
	0.891	0.898	0.879	0.888	0.892	0.893	0.894	0.895	0.877	0.89	0.893	0.895	0.894	0.896	0.882	0.888	0.884	0.9	***M***
	0.892	0.9	0.879	0.889	0.893	0.894	0.893	0.896	0.877	0.891	0.892	0.896	0.892	0.897	0.882	0.89	0.883	0.899	**W**
ROC Area	0.964	0.97	0.879	0.889	0.964	0.894	0.964	0.972	0.877	0.891	0.963	0.896	0.965	0.972	0.882	0.89	0.956	0.9	***B***
	0.964	0.97	0.879	0.889	0.964	0.894	0.964	0.972	0.877	0.891	0.963	0.896	0.965	0.972	0.882	0.89	0.956	0.9	***M***
	0.964	0.97	0.879	0.889	0.964	0.894	0.964	0.972	0.877	0.891	0.963	0.896	0.965	0.972	0.882	0.89	0.956	0.9	**W**
Accuracy (%)	89.20	89.37	87.93	88.92	89.30	***90.03***	89.27	89.57	87.7	89.1	89.25	***89.57***	89.20	89.68	88.17	88.97	88.35	***89.95***	
TPR: True Positive Rate, FPR: False Positive Rate, C1: J48, C2: Bagging, C3: Ib1, C4: Bayesian Logistic Regression, C5: Part, C6: SPegasos, B: Benign, M: Malware, W: Weighted Average																			



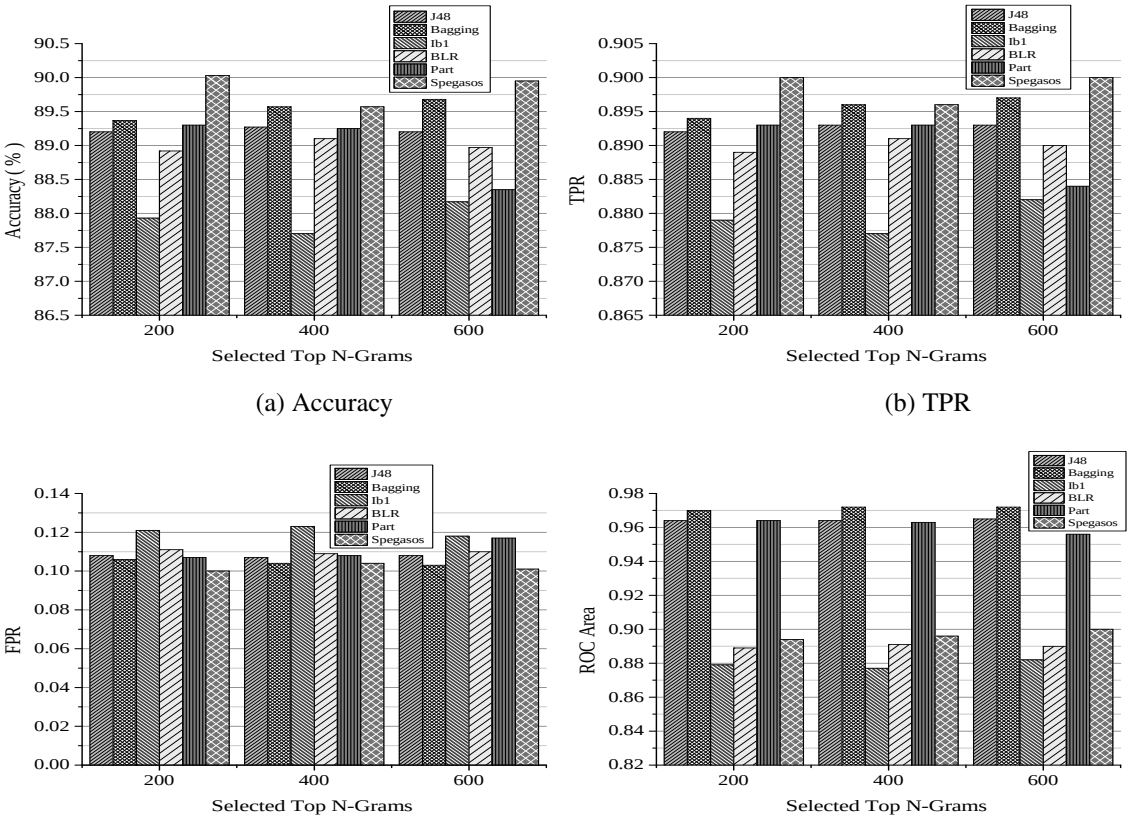
(a) Accuracy (b) TPR

(c) FPR (d) ROC Area

Figure 4.5: Graphical representation considering evaluation measures such as (a) Accuracy, (b) True Positive Rate, (c) False Positive Rate, and (d) ROC area. When N-gram length is three bytes

Similarly, in the second set of experiments, N-gram of length four bytes was analyzed, and the results for highest accuracy were 90.03% for 200 N-grams, 89.57% for 400 N-grams, and 89.95% for 600 N-grams with respect to the SPegasos classifier (see Figure 4.6a). The highest TPR was 0.9 for 200 N-grams, 0.896 for 400 N-grams, and 0.9 for 600 N-grams obtained by the SPegasos classifier (see Figure 4.6b). The lowest FPR was 0.1 for 200 N-grams, 0.104 for 400 N-grams, and 0.101 for 600 N-grams produced by the SPegasos classifier (see Figure 4.6c). From the visual inspection of Figure 4.5 and Figure 4.6, it can be concluded that the SPegasos classifier is the best and ensures better classification for N-gram lengths of three and four bytes.

In order to detect the malicious activities of the malware, the behaviour analysis of the PE file such as system calls invoked by the input PE file during execution have been employed. The gathered system calls' sequence was chunked into the N-gram, and each N-gram was treated as a feature. The IG feature selection method was used to choose



(a) Accuracy (b) TPR

(c) FPR (d) ROC Area

Figure 4.6: Graphical representation considering evaluation measures such as (a) Accuracy, (b) True Positive Rate, (c) False Positive Rate, and (d) ROC area. When N-gram length is four bytes

the best features based on the highest IG score, and the selected features were used to prepare the FFV needed by the classifier. The experiments were performed using different classifiers available in the WEKA tool. From the experimental observations, it was found that the machine learning-based classifier SPegagos ensured better classification for N-gram lengths of three and four bytes.

4.2 INFORMATION GAIN SCORE COMPUTATION FOR N-GRAMS USING MULTIPROCESSING MODEL

The FST is computational resources demanded and takes time to generate the scores for large feature (N-gram) datasets, if the processing is to be accomplished in the sequential mode. To address this issue, the present work presents a multiprocessing model that computes scores rapidly for large N-gram datasets. For the purpose of demonstration, the proposed multiprocessing model was implemented considering the IG FST. Further to ensure the processing efficiency of the proposed model, a comparative analysis was

conducted with the sequential mode of IG score computation.

4.2.1 Multiprocessing model to compute score

In the present work, the multiprocessing model to compute score for N-grams using IG FST was designed, implemented, and validated with a larger N-gram dataset. The Benign N-gram Files (BNFs) and Malware N-gram Files (MNFs) were both included in the N-gram dataset, where each N-gram file consisted of more number of N-grams. For each N-gram, the IG score was computed using the steps depicted in Figure 4.7. The IG score generation steps were grouped into four phases:

1. Pre-processing Phase;
2. Chunks Construction Phase;
3. Chunks to Process Allocation Phase; and
4. IG score Computation Phase.

4.2.1.1 Pre-processing Phase

In this phase, the N-grams are generated by extracting the binary instructions from both the benign and malware PE files available in the public source (Malwr, 2010). This step is referred to as the intermediate stage. In this stage, the extracted instructions are continuously placed in a row. Later, N-grams of different feature lengths are generated such as $NG_L = 2$, $NG_L = 3$, $NG_L = 4$, etc. based on the length specified by the user to create BNF and MNF. Subsequently, in order to get the highest order sequence and better selection of N-grams, the created BNF and MNF are arranged in non-increasing order by removing any duplicates, if found. The generation of BNF and MNF is illustrated in Figure 4.8.

The generation of BNF and MNF is a very important step since it is the start-up step or is defined as the input for the proposed multiprocessing model. The next step involves the union operation on the BNFs (B1, B2, B3 . . . Bn) and MNFs (M1, M2, M3 . . . Mn) individually on both the categories, benign and malware, to identify and remove the redundant N-grams. After the union operation, the Unique Benign N-gram File (UBNF) and Unique Malware N-gram File (UMNF) are obtained as shown in Figure 4.7.

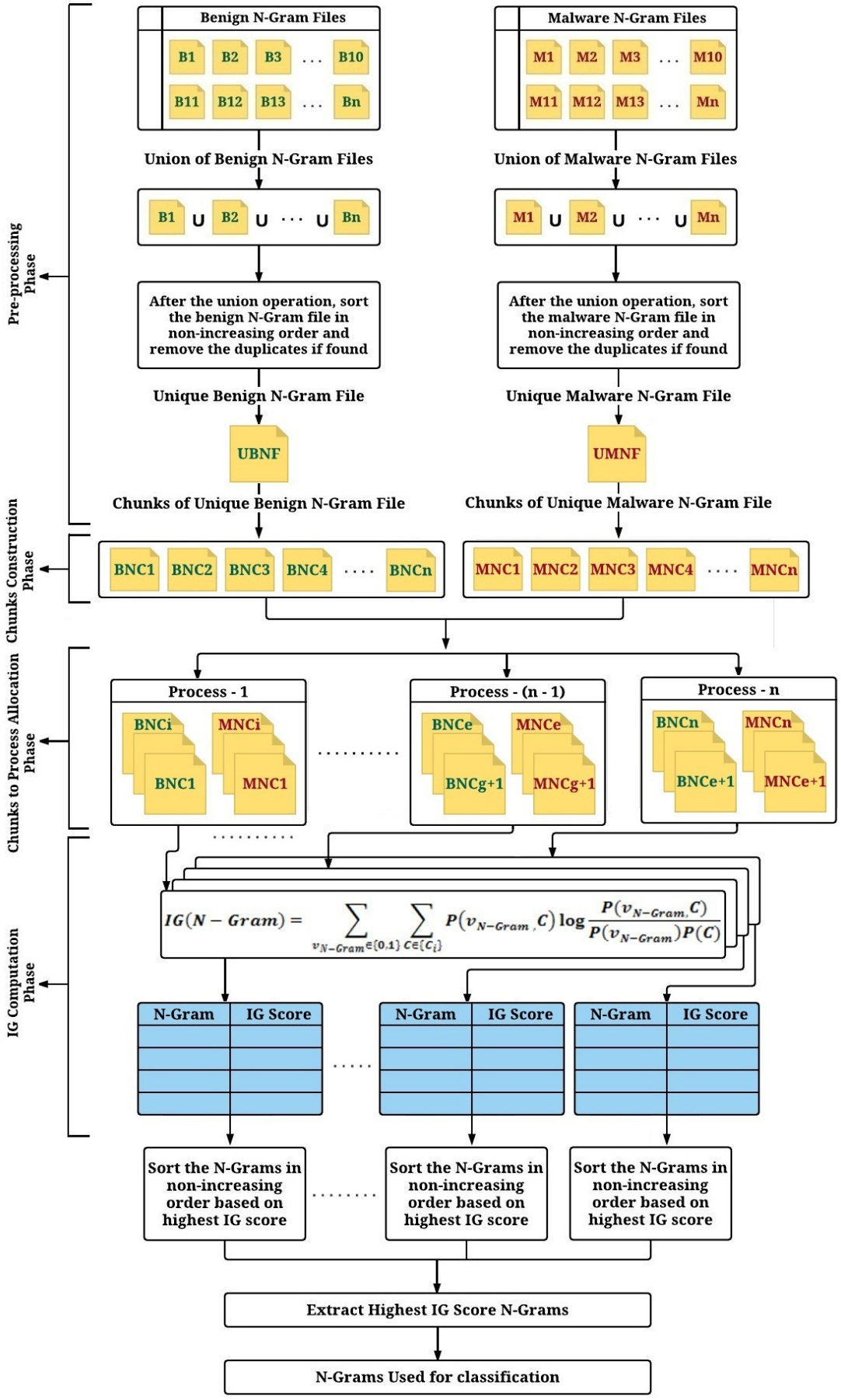


Figure 4.7: Multiprocessing model to compute Information Gain score

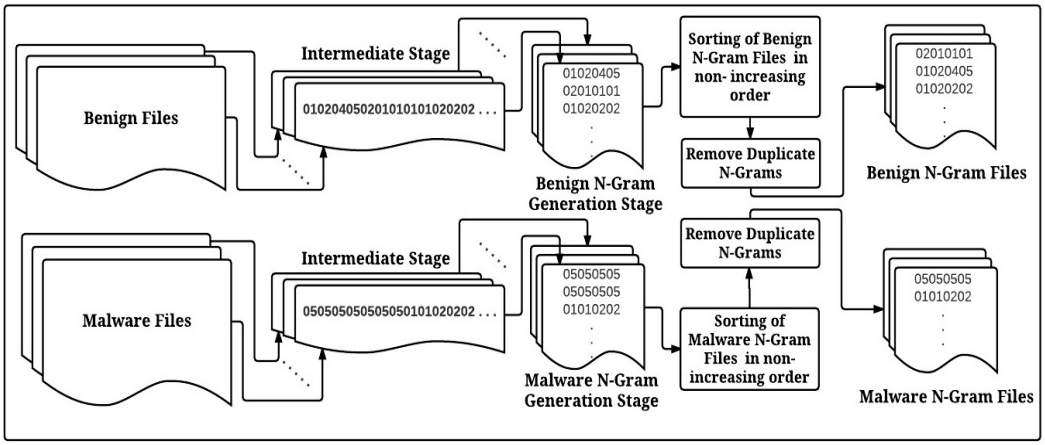


Figure 4.8: Generation of Benign N-gram Files and Malware N-gram Files

4.2.1.2 Chunks Construction Phase

The UBNF and UMNF consist of a large number of N-grams, and computing the IG score for each N-gram in the UBNF and UMNF sequentially is computational resource demanding. To reduce this, the multiprocessing model is proposed in the present work that computes the IG score for each N-gram. In this phase, the UBNF is divided into files of smaller benign N-gram files [Benign N-gram Chunk - (BNC)], namely, BNC1, BNC2, BNC3, BNC4 . . . BNCn, and similarly, the UMNF is divided into smaller malware N-gram files [Malware N-gram Chunk - (MNC)], namely, MNC1, MNC2, MNC3, MNC4. . . MNCn as shown in Figure 4.1. A pair of chunks (BNC1, MNC1), (BNC2, MNC2) . . . (BNCn, MNCn) is assigned to a created process. Each benign chunk N-gram file and malware chunk N-gram file consists of a fixed number of N-grams. The number of N-grams to be present in each BNC file and MNC file is a constant value specified by the user, and accordingly, the generation of chunk files takes place.

4.2.1.3 Chunks to Process Allocation Phase

In this phase, one chunk file from both the benign and malware categories is accessed sequentially and assigned to the process. The distribution of the number of chunk pairs to each process is calculated as per the equation given below:

$$\frac{\begin{array}{c}Distribution\ of\ chunk\ pair\ to\ each\ process = \\ (Highest\ number\ of\ chunks\ made\ either \\ from\ benign\ or\ malware\ category)\end{array}}{No.\ of\ process} \tag{4.2}$$

4.2.1.4 IG Computation Phase

To compute the IG score, each process follows the following steps:

- On each pair of benign and malware chunk N-gram file, the union operation is conducted to recognize the occurrence of the same N-gram;
- Similar occurrences of N-gram are removed, if observed after performing the union operation; and
- Finally, a unique N-gram file is prepared.

Each N-gram from the unique N-gram file is accessed to check its existence in all the BNF and MNF to compute the IG score using Eq. 4.1.

4.2.2 Experiment Results

The experimental datasets is of two types, benign dataset and malware dataset. The benign dataset is a collection of non-malicious executable files, whereas the malware dataset is a collection of malicious executable files obtained from a public source (Malwr, 2010). For the purpose of the experiments, ten benign files and ten malware files were taken in to consideration. The length of the N-grams was fixed for four bytes to generate the BNFs and MNFs. Three processes were created to demonstrate the multiprocessing model to compute the IG score.

Furthermore, as per the explanation in Section 4.2.1, the generated BNFs and MNFs underwent the union operation, and then, the N-grams were sorted in non-increasing order and all duplicate N-grams were removed, if observed, to obtain UBNF and UMNF. The generated UBNF and UMNF in this experiment consisted of 414606 N-grams of the benign category and 2475931 N-grams of the malware category.

The multiprocessing model was implemented using the Python programming language. It constructed the chunks separately from the UNBF and UNMF based on the threshold value set by the user. Then, the IG score for each N-gram was computed. In the present work, N-grams chunks with varying threshold values such as 10000, 15000, 20000, 25000, and 30000 were created to demonstrate the efficiency of the proposed model. For the threshold value 10000, 42 benign chunk files and 248 malware chunk files were obtained. For other threshold values like 15000, 20000, 25000, and 30000, benign chunks files of 28, 21, 17, and 14, and malware chunk files of 166, 124, 100, and

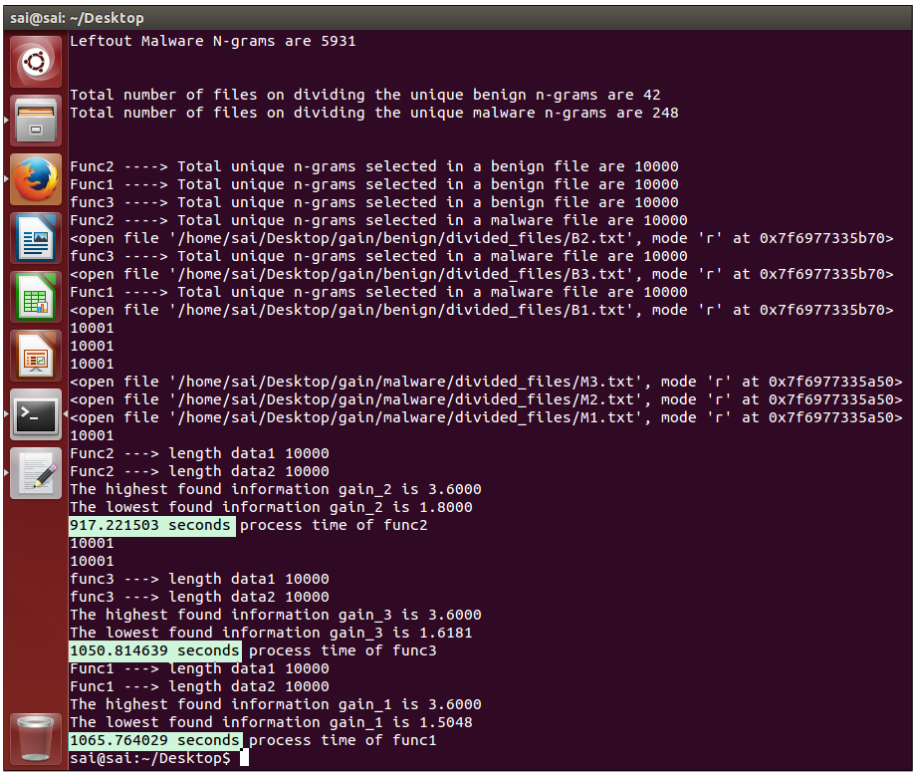
83, were constructed and are tabulated in Table 4.3. To demonstrate the efficiency of the multiprocessing model against the sequential model, the first three pairs of the chunk files from both the categories of benign and malware files were obtained for different threshold values such as 10000, 15000, 20000, 25000, and 30000.

When the threshold value was set to 10000, each chunk file from both the benign and malware categories consisted of a fixed number of 10000 N-grams. If three benign chunk files and three malware chunk files were considered, then the number of N-grams in the benign category was 30000, and similarly, the N-grams size of the malware category was also 30000. The time taken to compute the IG score of all the 60000 N-grams was 5244.857 seconds in the sequential model and 1065.764 seconds in the multiprocessing model. In the same way, when the threshold value was set to 15000, each chunk file from both the benign and malware categories consisted of a fixed number of 15000 N-grams. For the first three chunk files, the N-gram size of both the benign and malware was 45000 and the computational time observed to compute the IG score for all the 90000 N-grams by the sequential model was 7989.225 seconds, and for the multiprocessing model was 1513.592 seconds. Similarly, on increasing the threshold value to 20000, each benign chunk file and malware chunk file now consisted of a fixed number of 20000 N-grams. For three benign chunk files and three malware chunk files, the N-gram size increased to 120000 and the time taken to compute the IG score for all the 120000 N-grams was 11006.653 seconds by the sequential model and 2081.264 seconds by the multiprocessing model. Correspondingly, the computational time taken by the sequential model and the multiprocessing model to compute the IG score for 150000 N-grams and 180000 N-grams is tabulated in Table 4.3.

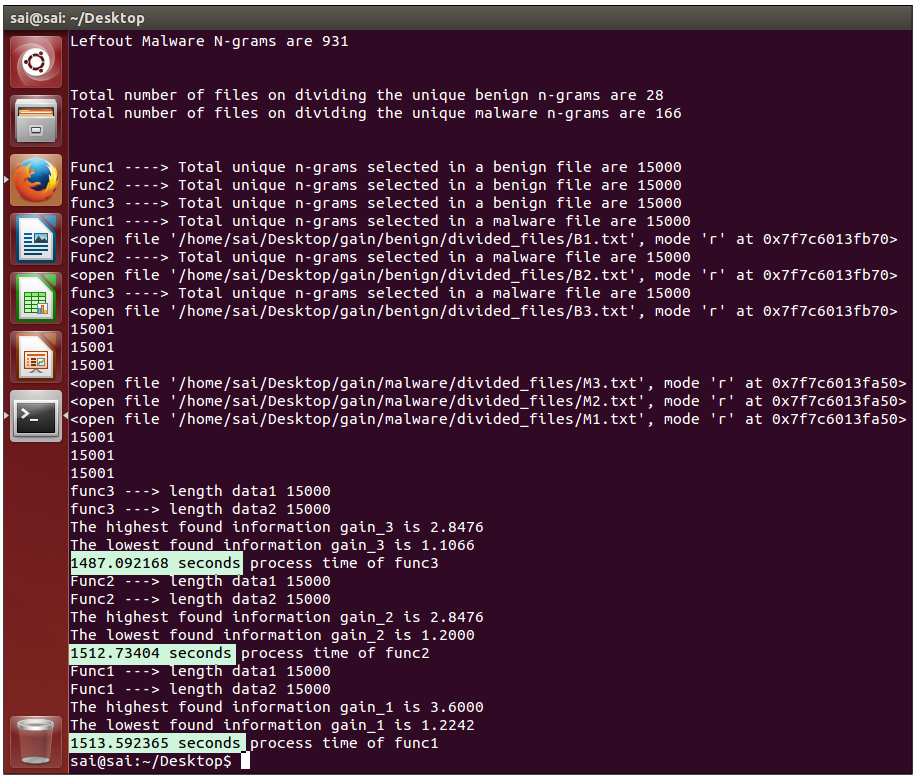
The computation time taken to compute the IG score by the multiprocessing model for chunks of 60000 N-grams and 90000 N-grams is shown in Figs. 4.9a and 4.9b, where the highlighted part in Figs. 4.9a and 4.9b denotes the computation time taken by the process (FUNC1, FUNC2, and FUNC3 denotes Process1, Process2, and Process3, respectively) to compute the IG score. Similarly, experiments were carried out for other chunks such as 120000 N-grams, 150000 N-grams, and 180000 N-grams. Figure 4.10a and Figure 4.10b represent the computational time taken to compute the IG score by the sequential model for chunks of 60000 N-grams and 90000 N-grams.

Table 4.3: Information Gain score computation time

Benign N-gram Files	**Malware N-gram Files**	**Unique N-grams File**		**Threshold Value for Chunk Construction**		**No. of Chunk Files Constructed**		**N-gram Size (No. of N-grams x No. of Chunks)**		**Time Taken for IG Score Computation in Sequential Mode1 (Sec)**	**Time Taken for IG Score Computation in Multiprocessing Mode1 (Sec)**
		Benign	**Malware**	**Benign**	**Malware**	**Benign**	**Malware**	**Benign**	**Malware**		
10	10	414606	2475931	10000	10000	42	248	10000 x 3	10000 x 3	5244.857	1065.764
10	10	414606	2475931	15000	15000	28	166	15000 x 3	15000 x 3	7989.225	1513.592
10	10	414606	2475931	20000	20000	21	124	20000 x 3	20000 x 3	11006.653	2081.264
10	10	414606	2475931	25000	25000	17	100	25000 x 3	25000 x 3	13722.352	2684.321
10	10	414606	2475931	30000	30000	14	83	30000 x 3	30000 x 3	16688.499	3364.588

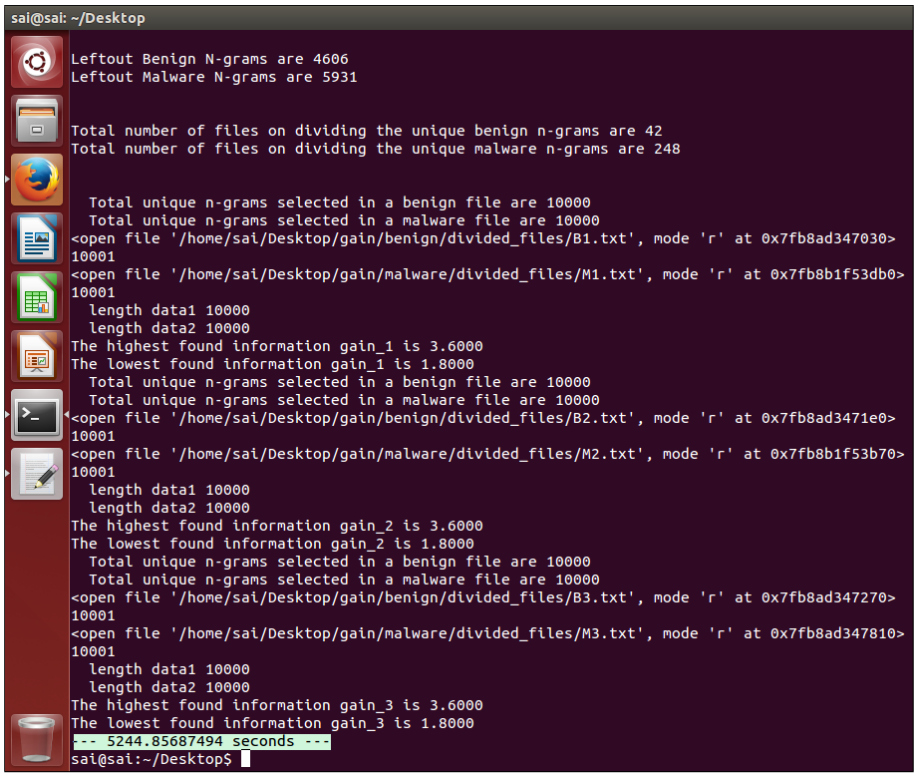


(a) Computation Time(sec) - Chunk of 60000 N-grams

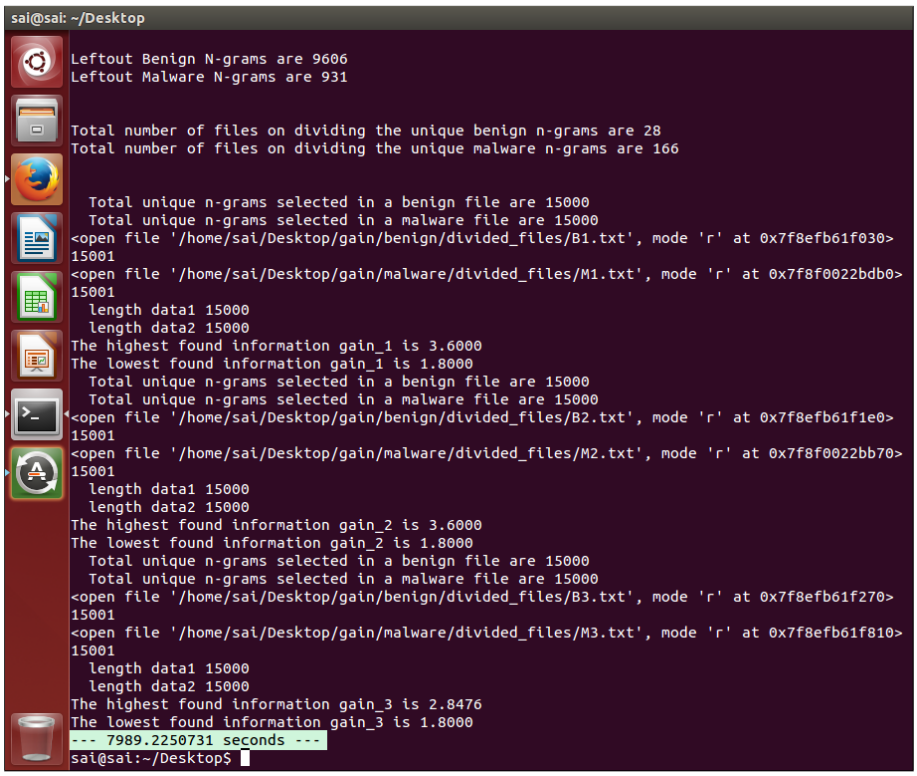


(b) Computation Time(sec) - Chunk of 90000 N-grams

Figure 4.9: Information Gain score computation time - Multiprocessing model



(a) Computation Time(sec) - Chunk of 60000 N-grams



(b) Computation Time(sec) - Chunk of 90000 N-grams

Figure 4.10: Information Gain score computation time - Sequential model.

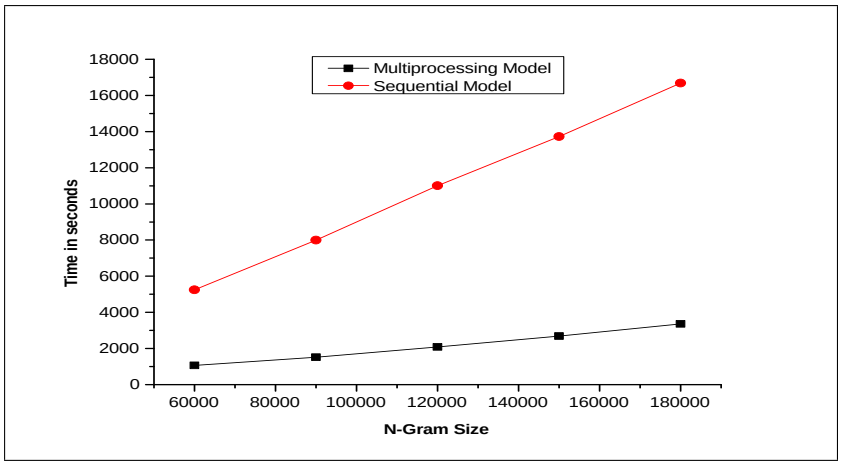


Figure 4.11: Comparison between multiprocessing model and sequential model to compute Information Gain score

The highlighted part signifies the overall computation time required by the sequential model to generate the IG score. A similar demonstration was performed for other chunks such as 120000 N-grams, 150000 N-grams, and 180000 N-grams in the sequential model. For each N-gram, the computation time needed to generate the IG score by the sequential model as well as by the multiprocessing model was observed and is depicted in Table 4.3.

Looking at Table 4.3, one can notice that there is radical improvement in the computation time of the IG score by the proposed multiprocessing model. For the chunk of 60000 N-grams, the computation time of the IG score got reduced by 79.68%, and for other chunks such as 90000, 120000, 150000, and 180000 N-grams, the computational time was reduced by 81.05%, 81.09%, 80.44%, and 79.84%, respectively, against the sequential model. On average, the proposed approach was 80% faster than the sequential model for the computation of the IG score. Analogizing the two parameters from Figure 4.11 manifests that the multiprocessing model is faster than the sequential model.

4.3 EMPIRICAL STUDY ON FEATURES RECOMMENDED BY LSVC IN CLASSIFYING UNKNOWN WINDOWS MALWARE

To build an efficient and robust machine learning-based MDS, identifying crucial features from the original feature set of a large size is essential. In order to identify and eradicate noisy features from the original feature set, the LSVC was employed as fea-

ture selector in the present work. In so as is known, the LSVC has not been widely explored as a feature selector to select the dynamic features given by the Cuckoo Sandbox report. Thus, the present work investigates the effectiveness of the behavioural features suggested by the LSVC in identifying unknown Windows malware. The major contributions of the present work are as follows:

- In the current work, the MDS has been designed, implemented, and evaluated using publicly available Windows malware samples. It was evaluated with two different types of dynamic features, namely, 1) API calls and 2) API calls with their category (Category+API calls), to know which type of dynamic features suggested by the LSVC can provide better malware detection rate.

- The 10-fold validation tests were conducted to measure the malware detection rate of the proposed MDS. It is capable of identifying malware and benign Windows PE files precisely using the dynamic features suggested by the LSVC. However, the obtained experimental results demonstrate that the proposed MDS is able to attain highest detection accuracy with API calls alone as dynamic features.

4.3.1 Architecture Overview

An overview of the proposed MDS is shown in Figure 4.12. It utilizes dynamic malware analysis technique to spot malware. It mainly consists of two phases, namely, the Training phase and the Prediction phase.

4.3.1.1 Training Phase

In the training phase, a set of benign and malware Windows PE files are executed one at a time on the Cuckoo Sandbox to gather the runtime behavioural report of the PE files. Each runtime behavioural report is pre-processed separately to obtain Category+API calls. The gathered sequence of the Category+API calls is processed to generate N-grams, and each N-gram is treated as an individual feature. In order to identify and remove the noisy features from a set of acquired features, the LSVC is employed as feature selector. The LSVC recommended features are used as the final features to prepare a training file, essential to train the classifier. The main components of the

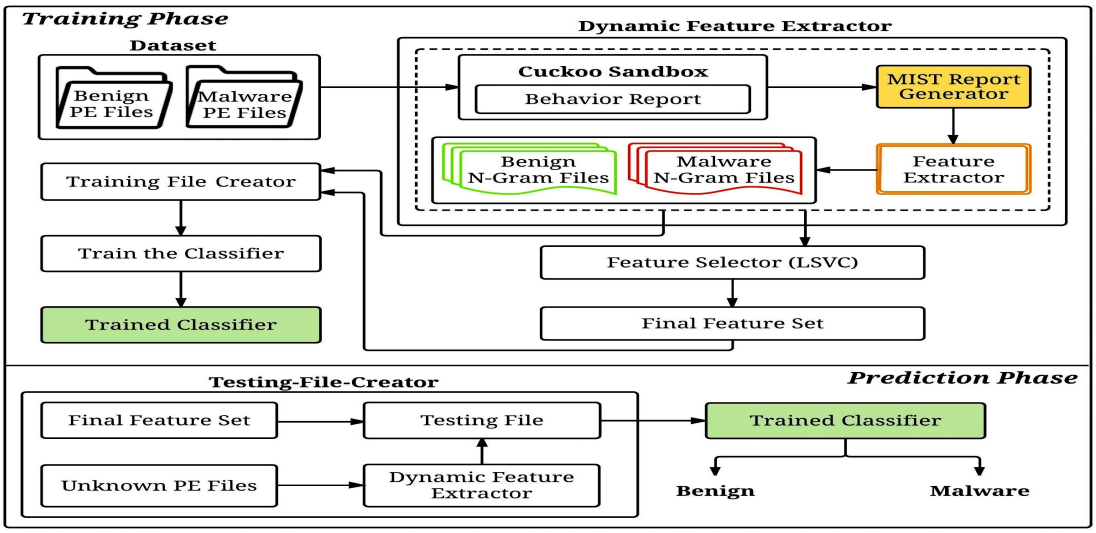


Figure 4.12: The architecture of the proposed MDS

training phase are the Dynamic Feature Extractor, Feature Selector, Final Feature Set, and Training File Creator.

Dynamic Feature Extractor

The prime task of the Dynamic Feature Extractor is to observe and record the execution time behaviour of the PE file, while the source file is being executed on a controlled monitoring environment. Its main sub-components are the Cuckoo Sandbox, MIST Report Generator, and Feature Extractor.

Cuckoo Sandbox

The Cuckoo Sandbox is a Windows PE file runtime behaviour acquiring tool (Guarnieri et al., 2012). It is used to obtain a runtime behavioural report of the PE files. It records the details onto the behavioural report in the JSON file format. The behaviour report is also called as the behaviour analysis report (Firdausi et al., 2010). The Cuckoo Sandbox captures the API calls and classifies them into one of the categories on the basis of the type of operations the API call performs, which includes `network`, `process`, `system`, `services`, `registry`, `misc`, `crypto`, `file`, `resources`, etc. (Miller et al., 2017). Figure 4.13 shows a snippet of the recorded API call `"LdrLoadDll"` classified under `system` category during the execution of `"01B43C0C8D620E8B88D846E4C9287CCD.bin"`. Moreover, the category of an API call in terms of the type of operation performed is a better indicator to understand the actions undertaken

```
{
      "category": "system",
      "status": 1,
      "stacktrace": [],
      "api": "LdrLoadDll",
      "return_value": 0,
      "arguments": {
            "basename": "comctl32",
            "module_address": "0x773d0000",
            "flags": 0,
            "module_name": "comctl32.dll"
      },
      "time": 1491435510.263125,
      "tid": 1124,
      "flags": {}
}
```


Figure 4.13: Snippet of the Cuckoo Sandbox generated report in JSON file format

by the executable file. So, the focus is to extract details such as category API call with corresponding arguments from the JSON file format, and then represent the acquired particulars in the MIST format.

MIST Report Generator

The behavioural analysis report obtained in the JSON file format requires high storage and more processing time. In order to address this issue, the MIST report generator has been used in the present work (Rieck et al., 2011). It extracts system level behaviour such as category API calls and their arguments from the JSON file format, and organizes them in different levels of blocks as shown in Figure 4.14. MIST is the preferred format because it uses a smaller file size and reduces the processing time.

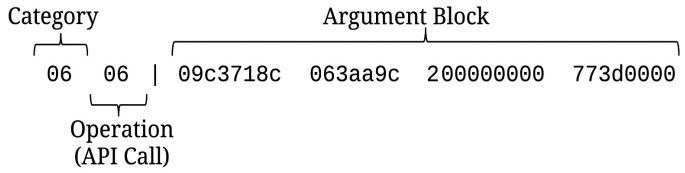


Figure 4.14: MIST representation of API call reported in JSON file format shown in Figure 4.13

Feature Extractor

The job of the Feature Extractor is to extract API calls (operation field) and their category from the MIST report (see Figure 4.14). In the present work, the Feature Extractor is implemented using the Python programming language. The extracted data is represented in overlapping substrings, which are obtained based on the sliding window approach known as N-grams (Raff et al., 2018) and are extensively used in information

retrieval. Its major drawback is that the set of all the gathered N-grams (features) from a set of byte strings of malware and benign PE files is enormous and applying them directly for classification techniques degrades the performance of the classifier. Therefore, the selection of the most relevant N-grams plays a crucial role. In order to choose prominent features, the feature selector is employed.

4.3.1.2 Feature Selector

The LSCV (Belaoued and Mazouzi, 2015; Kawaguchi and Omote, 2015) is a wrapper-based FST. It is used to identify the most significant features from a set of gathered features. All the LSVC recommended features are treated as final features because there can be no further elimination of the features. The final features are used to prepare a training file as well as testing files to measure the efficiency of the classifiers. Suppose p, q, d, and n is the input vector, class label, dimension, and number of samples, respectively. The training data (p_i, q_i) is separated by the hyperplane decision function D(p) based on the values of x and y. Where, $p \in P^m$, $q_i \in \{+1, -1\}$ and i = 1, 2, 3 . . . n. The decision function D(p) is given by Eq. 4.3.

$$\text{D(p)} = \{x^T p\} + y = \sum_{i=1}^{n} x_i p_i + y \tag{4.3}$$

Where, $x = [x_1, x_2, ..., x_n]^T$ is the weight vector of the hyperplane. The training data with value $q_i = +1$ gets classified under D(p)>0, and conversely, the training data with value $q_i = -1$ gets classified under D(p)<0.

For the LSVC, the recommended default parameters provided in sklearn.svm.LinearSVC python library were used (Pedregosa et al., 2011). Accordingly, the value of the penalty parameter 'C' was set to 1. The value of 'penalty' specifies the norm used in the penalization, i.e., 'l2' and the value of 'dual' is preferred to be false.

4.3.1.3 Training File Creator

The Training File Creator creates a training file essential to train the classifiers. It parses the benign and malware N-gram files corresponding to the training dataset with the final features in order to create a training file.

4.3.2 Prediction Phase

In the prediction phase, the main task of the Testing-File-Creator is to create a testing file necessary to appraise the predictive performance of the trained classifiers. It makes use of the final features and N-grams of the testing file to deliver a testing file. The generated testing file is sent to the trained classifier, which classifies whether the test input file is benign or malware.

4.3.3 Experimental Analysis

To evaluate the performance of the proposed MDS, all the experiments were conducted on a host system and its specifications are as mentioned in Section 2.7.

The experimental data consisted of 200 benign and 200 malware PE files. The benign PE files included the Windows system files, which were collected from a freshly installed Windows virtual machine. The Windows malware PE files used in the experimental work were downloaded from a public source VirusShare (VirusShare, 2011). Six different machine learning classifiers such as the SMO, Simple Logistic, Logistic, J48, RF, and Random Tree, which were available in the WEKA (Frank et al., 2009) tool were adopted. These classifiers were used throughout the experiments in 10-fold cross-validation with default parameters settings. Further, the effectiveness of the proposed MDS and the detection performance of the classifiers were estimated using five evaluation metrics such as the TPR, FPR, Precision, Recall, and Accuracy as defined in Eq. 2.1.

4.3.4 Results and Discussions

In the present work, the prime aim of the proposed MDS is to explore the accurate detection and categorization of the malicious PE files using the final features suggested by the LSVC. Two different types of behavioural features, namely, API calls and Category+API calls for different N-grams of sizes 3 bytes, 4 bytes, and 5 bytes were used to measure the accuracy. Two stages of experiments were conducted. The first stage comprised of API calls' sequence (N-grams) as features. The second stage consisted of Category+API calls together as a single element. The sequence of combination of Category and API call as a feature (N-gram) was taken into consideration. The features suggested by the LSVC in both the experiments were used to appraise the efficiency of

the classifier separately. Finally, a comparative analysis was made to know the impact of the features recommended by the LSVC.

During the training file creation phase, the Cuckoo Sandbox stores the monitored behaviour of the PE file in JSON file format, and the obtained data is pre-processed to be converted into the MIST format. From the MIST format file, the API call sequence is extracted, and then, the consecutive API calls are grouped to prepare N-grams of different sizes such as 3 bytes, 4 bytes, and 5 bytes. Subsequently, a combination of Category and API calls are extracted as a pair from the MIST format file and a separate N-gram file is prepared for each individual MIST file. The duplicate N-grams are removed and further LSVC is applied as feature selector to choose the distinct features and ignore the noisy features. The LSVC assigns a separate score to each individual feature, and then selects a set of features based on their score.

First, the experiments were conducted with concern to the monitored API calls present in the MIST files. Initially, the length of the N-gram as NL=3 was selected to prepare N-grams considering all the benign and malware N-gram files. Further, all N-grams of size 3 bytes belonging to benign class and malware class were consolidated individually. In total 4133 and 6555 distinct N-grams related to the benign and malware class, respectively, were obtained. All these benign and malware N-grams were combined and any duplicates were eliminated to obtain unique N-grams. Consequently, 7645 N-grams were attained as features. However, it was tedious to consider all these 7645 N-grams as features to prepare a training file. Therefore, the relevant features (90) were selected as final features by using the LSVC as feature selector. Finally, a training file was constructed to train the classifier using the final features with the N-gram files corresponding to the training files. Similarly, experiments were carried for N-grams of length 4 bytes and 5 bytes, where the LSVC suggested 186 (4byte N-grams) best features out of 26143 N-gram features and 124 (5byte N-grams) significant features out of 35226 N-gram features. The LSVC suggested N-gram features were treated as final features to train the classifier.

In another set of experiments, both the Category and API calls present in the MIST files were considered to know their effectiveness. With respect to this, consecutive three pairs of category and API call were grouped to form N-grams of size 3 bytes. After re-

moving duplicate N-grams, 16188 N-grams were attained from all the N-gram files, and the LSVC was applied to choose the distinct features. The LSVC recommended 134 N-grams as prominent features and these were used to train the classifier. Correspondingly, experiments were conducted for N-grams of size 4 bytes (4-pairs of Category and API call) and 5 bytes (5-pairs of Category and API call). In this case, the LSVC advised 156 best features among 29821 N-grams of size 4 bytes and 221 predominant features among 42742 N-grams of size 5 bytes.

4.3.5 Analysis of Proposed Malware Detection System based on Evaluation Metrics

From the experimental observations, the detection rate and FPR obtained for N-grams of size 3 bytes is depicted in Figure 4.15a and Table 4.4. In particular, the SMO classifier achieved maximum accuracy of 96.923% with 0.031 FPR for the combination of Category+API call features. Further, the Simple Logistic classifier also performed well by attaining accuracy of 93.846% with FPR 0.062. Relatively, the overall performance of the other classifiers such as the Logistic, J48, RF, and Random Tree reported lowest accuracy and their corresponding values were 88.717% with FPR 0.113, 87.692% with FPR 0.123, 91.794% with FPR 0.082, and 88.717% with FPR 0.113.

The accuracy and FPR achieved by the different classifiers for N-grams of size 3 bytes for API call features are shown in Figure 4.15a and Table 4.4, respectively. It can be seen from Figure 4.15a that the highest accuracy of 98.429% with 0.016 FPR (see Table 4.4) was yielded by the SMO classifier. However, the performance of the other classifiers was not remarkable. The second highest accuracy of 95.549% was accomplished by the Logistic classifier with 0.045 FPR (see Table 4.4). The Simple Logistic classifier recorded low accuracy of 94.764% with 0.052 FPR (see Table 4.4). Moreover, other classifiers such as the J48, RF, and Random Tree underperformed by achieving least accuracy of 87.958% with 0.120 FPR, 93.193% with 0.068 FPR, and 87.696% with 0.123 FPR, respectively.

A comparative analysis was made between the values accomplished by the evaluation metrics on two different N-gram features such as the API calls alone and Category+API calls of size 3 bytes. The highest TPR of 0.984, Precision of 0.985, and Recall of 0.984 was recorded by the SMO classifier when only API calls were considered as N-gram features. The other classifiers like the Simple Logistic, J48, RF, and

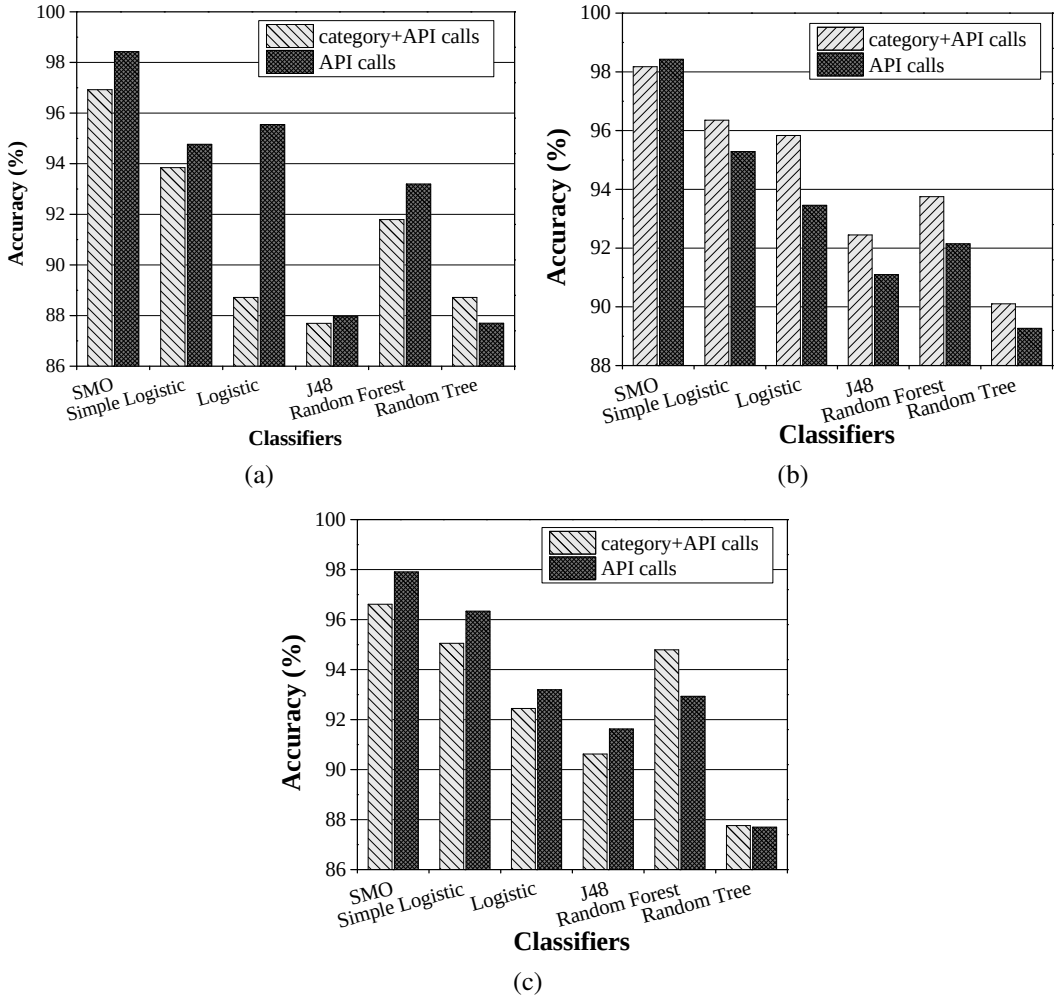


Figure 4.15: Comparative analysis in terms of accuracy obtained for API calls alone as N-gram features and combination of Category+API call as N-gram features of different sizes (a) 3bytes, (b) 4bytes, and (c) 5bytes

Random Tree performed poor and resulted in achieving less accuracy as demonstrated in Table 4.4.

Experiments were also performed by considering N-grams size of 4 bytes and 5 bytes. Accordingly, the final features were constructed for both types of behavioural feature, namely, API calls and Category+API calls. As illustrated in Figure 4.15b and Table 4.5, the maximum detection accuracy of 98.429% with 0.016 FPR was accomplished for the final features comprised of only API calls. However, a combination of final features consisting of Category+API calls also achieved nearly equivalent detection rate of 98.177% with FPR of 0.018 when compared with API calls as N-gram features. The second highest accuracy recorded was 96.335% with FPR of 0.036 by

Table 4.4: Comparison analysis of evaluation metrics obtained for N-grams of size 3bytes

Classifiers	TPR		FPR		Precision		Recall	
	C+API	API calls	C+API	API calls	C+API	API calls	C+API	API calls
SMO	0.969	0.984	0.031	0.016	0.969	0.985	0.969	0.984
Simple Logistic	0.938	0.948	0.062	0.052	0.939	0.948	0.938	0.948
Logistic	0.887	0.955	0.113	0.045	0.888	0.957	0.887	0.955
J48	0.877	0.880	0.123	0.120	0.877	0.880	0.877	0.880
Random Forest	0.918	0.932	0.082	0.068	0.922	0.932	0.918	0.932
Random Tree	0.887	0.877	0.113	0.123	0.888	0.878	0.887	0.877
C+API: Category+API call								

Table 4.5: Comparison analysis of evaluation metrics obtained for N-grams of size 4bytes

Classifiers	TPR		FPR		Precision		Recall	
	C+API	API calls	C+API	API calls	C+API	API calls	C+API	API calls
SMO	0.982	**0.984**	0.018	**0.016**	0.982	**0.984**	0.982	**0.984**
Simple Logisitic	0.964	0.953	0.036	0.047	0.964	0.953	0.964	0.953
Logistic	0.958	0.935	0.042	0.065	0.960	0.936	0.958	0.935
J48	0.924	0.911	0.076	0.089	0.925	0.914	0.924	0.911
Random Forest	0.938	0.921	0.063	0.079	0.939	0.924	0.938	0.921
Random Tree	0.901	0.893	0.099	0.107	0.902	0.894	0.901	0.893
C+API: Category+API call								

the Simple Logistic classifier for the final features consisting of Category+API calls as N-gram features, and correspondingly, the same classifier was successful in achieving an almost identical accuracy of 95.288% with 0.047 FPR with final features of API calls. The Logistic classifier attained less accuracy of 95.833% with 0.042 FPR for final features comprising of both Category+API calls, but it gained nearest accuracy of 93.455% with FPR 0.065 for final features built with only API calls. The performance of other classifiers such as the J48, RF, and Random Tree was not remarkable.

The highest values of TPR, Precision, and Recall signified better MDS. The experimental results tabulated in Table 4.5 provides details of the comparative analysis made for the final features (N-grams size of 4 bytes) consisting of only API calls and the combination of both Category+API calls. Relatively, the nearest value to one was achieved for final features made with API calls and as Proof-of-Concept, the SMO classifier achieved the highest TPR with 0.984, Precision with 0.984, and Recall with 0.984.

The last set of experiments were performed on N-grams of size 5 bytes and the performance of the classifiers for both types of features such as API calls alone and Category+API calls was observed. Figure 4.15c and Table 4.6 provides details of the detection accuracy and FPR achieved by the different classifiers. It was observed that

Table 4.6: Comparison analysis of evaluation metrics obtained for N-grams of size 5bytes

Classifiers	TPR		FPR		Precision		Recall	
	C+API	API calls	C+API	API calls	C+API	API calls	C+API	API calls
SMO	0.966	0.979	0.034	0.021	0.967	0.980	0.966	0.979
Simple Logistic	0.951	0.963	0.049	0.037	0.953	0.964	0.951	0.963
Logistic	0.924	0.932	0.076	0.068	0.925	0.933	0.924	0.932
J48	0.906	0.916	0.094	0.084	0.908	0.917	0.906	0.916
Random Forest	0.948	0.929	0.052	0.071	0.951	0.932	0.948	0.929
Random Tree	0.878	0.877	0.122	0.123	0.878	0.878	0.878	0.877
C+API: Category+API call								

the SMO classifier was successful in achieving highest accuracy of 97.905% with 0.021 FPR for final features comprised of API calls. Moreover, the same classifier performed well for the N-gram features of type Category+API calls and gained nearly equivalent accuracy of 96.614% with 0.034 FPR when compared with N-gram features of API calls type. Further, the Simple Logistic classifier attained second highest accuracy of 96.355% with 0.037 FPR for final features consisting of API calls, and for the final features of Category+API calls, it yielded an accuracy of 95.052% with 0.049 FPR. The least accuracy obtained was 93.193% with 0.068 FPR by the Logistic classifier for final features of API calls, whereas, for the final features of Category+API calls, it recorded an accuracy of 92.447% with 0.076 FPR. The other classifiers such as the J48, RF, and Random Tree underperformed by achieving less accuracy as shown in Table 4.6.

The detection performance of the classifiers was also analyzed for N-grams of size 5 bytes using other evaluation metrics as tabulated in Table 4.6. The highest TPR, Precision, and Recall was achieved by the SMO classifier with 0.979, 0.980, and 0.979, respectively, as witnessed for the final features consisting of API calls as N-gram features. However, the performance of the classifiers was not appreciable for the final features comprised of Category+API calls.

The experiments were conducted substantially and analyzed to decide which N-gram size encouraged the classifier to achieve better accuracy. In this direction, the obtained and analyzed results proved that N-gram features of type API calls alone exhibited promising results for N-grams of size 4 bytes. Further, it was crucial to have LSVC as feature selector to get the best compact set of features and better accuracy. Accordingly, the accuracy of 98.429% was achieved by the SMO classifier with few features

recommended by the LSVC for the final features consisting of API calls. Meanwhile, experiments were also conducted by combining Category+API calls, which resulted in less accuracy.

Some of the malware exhibits their malicious action only after certain condition met. Till certain conditions met, they behave like benign files. If the malware detection system analyses behaviour report of the PE file that generated before the actual event trigged, such behaviour report of the malware misclassified as benign.

4.4 WINDOWS MALWARE DETECTOR USING CONVOLUTIONAL NEURAL NETWORK BASED ON VISUALIZATION IMAGES

Despite the success of the machine learning-based techniques in the identification of obscure malware, it may sometimes be susceptible to wrong predictions, if the learning process is inadequate. To overcome this, the Neural Network-based methods (Nataraj et al., 2011) have emerged as a promising approach, and it is applied as a superior technique to detect malware.

In the present work, a Convolutional Neural Network (CNN)-based Windows malware detection approach was proposed that used the CAT-API as the behavioural features to detect and classify unknown Windows malware. The Cuckoo Sandbox was utilized to obtain the runtime behavioural reports. These reports were in the JSON file format and rendered relevant information related to the CAT-API that are triggered together while a source file is in the process of execution.

From the previous work mentioned in Section 4.3, it can be understood that the extraction of only API calls as behavioural features from the MIST report leads to ambiguity because two or more different API calls may have the same identifier in two or more different Categories (CATs). To overcome this issue, in the present work, an endeavour was made to define the combination of CAT-API as a behavioural feature to recognize an obscure malware. Such a blend gives knowledge about the intent of the malware and proper learning for the classifier to perform its task. Hence, CAT-API sequences were extracted and generated as N-grams. However, all the generated N-grams cannot be considered as a part of the classification process because 1) the redundant features impact on storage space as well as processing time, and 2) the noisy features

degrade the classifier performance by achieving high FPR. Thus, to select a set of the most crucial N-grams from the original N-gram features set, the filter-based FST was employed. The selected topmost N-grams were used to create an image. Subsequently, a set of experiments were conducted with four different FSTs to know which one performed the best. The key contributions of the present work are as follows:

- In this work, a CNN-based Windows malware detection approach was proposed, implemented, and evaluated. It used the runtime behaviour features (N-grams) of the PE files that are advised by the FST to create images. The generated images were employed to examine the proficiency of the proposed approach.
- A set of experiments were conducted to demonstrate the classification ability of the proposed approach and compared with the chosen six machine learning-based classifiers. The obtained empirical results manifested that the proposed approach was predominant for the N-gram features recommended by the Relief FST and achieved maximum detection accuracy than that of machine learning-based classifiers. Thus, the proposed approach is proficient in the detection of Windows malware.
- The effectiveness of the four different filter-based FSTs in suggesting the best N-grams was evaluated and compared to show which one gave the best performance.

4.4.1 Windows Malware Detector using Convolutional Neural Network

The proposed CNN-based Windows malware detector mainly comprises of the Training Phase and the Prediction Phase as shown in Figure 4.16. The training phase is used to train the malware detector so that it can detect and classify the unknown PE files. The essential modules of the training phase are the Behaviour-based Feature Extractor, Feature Selector, Final Features Set, Image Generator, and the CNN.

4.4.1.1 Behaviour-based Feature Extractor

The PE file has behavioural features that are invoked, while the PE file is being executed. Behaviour-based Feature Extractor observes and records the behavioural features of the PE file, which is under execution in a controlled monitoring environment. It acquires the execution time behavioural report of the PE file to extract the CAT-API features and then, the acquired features are processed to derive the N-grams. The main sub-components of the Behaviour-based Feature Extractor are shown in Figure 4.17.

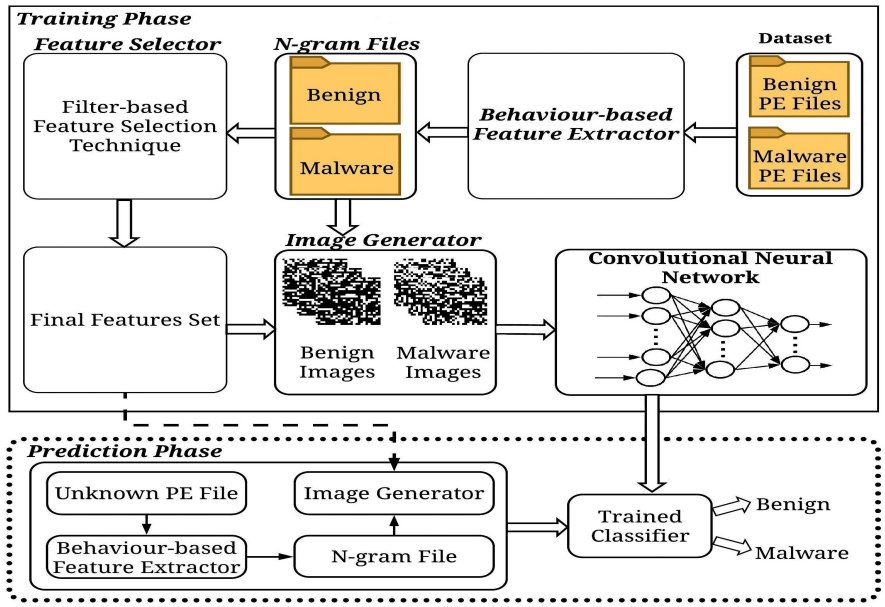


Figure 4.16: CNN-based Windows Malware Detector architecture

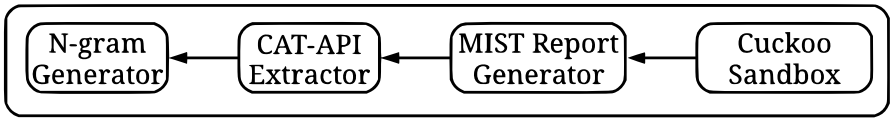


Figure 4.17: Behaviour-based Feature Extractor

Cuckoo Sandbox and MIST Report Generator

The Cuckoo Sandbox provides vast information about the behaviour of the input PE file, namely, registry keys, accessed files, network traffic triggered, mutexes, and Windows API calls, as mentioned in Section 4.3.1.1. Thus, the behavioural report generated by the Cuckoo Sandbox can be of great importance to determine obscure malware.

Besides its functionalities described in Section 4.3.1.1, the main benefits of adopting the Cuckoo Sandbox is that it can restore the infected machine to a clean state after performing the analysis so that new analysis operation can be carried out with a new source file. The maximum time that a PE file is allowed to run in the Cuckoo Sandbox is based on the default timeout option set within it. On timeout, the Cuckoo Sandbox terminates the analysis. This is because some sophisticated malware never end or loop for an extended period (Miller et al., 2017).

Thus in the present work, the Behaviour-based Feature Extractor utilizes the Cuckoo Sandbox to obtain the analysis report to derive the behavioural features. Further, the present work also involves the MIST Report Generator as described in Section 4.3.1.1

Table 4.7: Number of API calls identified under different categories

Sl. No.	Identified Categories	# API Calls	Sl. No	Identified Categories	# API Calls
01	_notification_	02	11	ole	07
02	certificate	02	12	process	41
03	crypto	10	13	registry	29
04	exception	02	14	resource	06
05	file	38	15	services	10
06	filesystem	05	16	synchronization	05
07	memory	01	17	system	27
08	misc	30	18	threading	02
09	netapi	01	19	ui	10
10	network	48	20	windows	03

to convert the JSON format-based behavioural reports to MIST format-based reports.

About 400 PE files were executed onto a Windows-7 virtual machine through the Cuckoo Sandbox to acquire the runtime behavioural reports, and the gathered reports were converted into MIST reports to derive a sequence of CAT-API that were invoked during the execution of the source files. The observed 20 unique categories with their corresponding number of unique API calls are depicted in Table 4.7.

4.4.1.2 CAT-API Extractor

The Windows API gathers user-mode library routines defined under a specific category. It comprises of Windows Native API, which is used for invoking the services of the operating system. The malware must call upon the local API whenever it needs to access one or more services such as the allocation or de-allocation of the virtual memory, interact with global resources, start a thread or process, etc. that exist in kernel-mode. The API calls invoked during the execution of the source PE file are classified under several CATs based on the type of operation performed including resources, system, misc, network, file, crypto, registry, services, process, etc. Different CATs, which are defined based on the operation being performed, are indicated numerically using a one-byte hexadecimal number in the MIST report. Similarly, the API calls that are invoked under each category are numerically denoted by one-byte hexadecimal number in the MIST report. Two byte combination of the CAT-API representation supports 255 CATs to get their unique number and each CAT assigns a unique number to the 255 API calls. Overall, this representation supports 65535 API calls uniquely. For instance, assume that the `process` CAT is denoted by hexadecimal number `'0a'` and during

the execution of the source PE file within the sandbox, the source file invokes 255 APIs corresponding to `process` CAT, then it can be represented as `0a01, 0a02, . . . , 0afe, 0aff` along with its corresponding arguments in the MIST format.

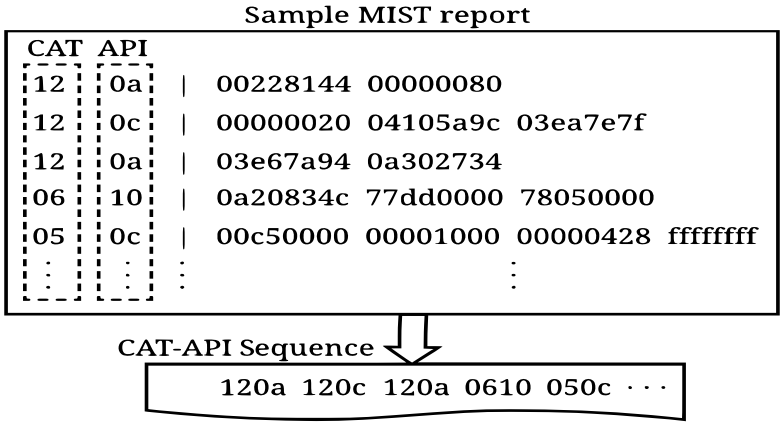


Figure 4.18: Example of CAT-API extraction from sample MIST report

The extraction of only API calls as behavioural features from the MIST report leads to ambiguity because two or more different API calls may have the same identifier in two or more different CATs. For instance, the extraction of API calls from the MIST instructions as shown in Figure 4.18 produces the sequence as `'0a 0c 0a 10 0c . . . '`. The second and fifth byte in the extracted sequence represents the same API call, but they are different in their respective CAT. To overcome this issue, in the present work, an endeavour was made to define the combination of CAT-API as a dynamic feature to recognize an obscure malware (Pektaş and Acarman, 2018). Such a blend gives knowledge about the intent of the malware and proper learning for the classifier to perform its classification task. The CAT-API extractor was implemented using the Python programing language with the objective of extracting CAT-API behaviour features from the MIST report.

4.4.1.3 N-gram Generator

The N-gram constitutes a sequence of 'N' consecutive bytes from a byte sequence (Reddy and Pujari, 2006), where 'N' indicates the number of predefined bytes. The foremost task of the N-gram Generator, in the proposed work, is to generate N-grams. The first and second columns of the MIST report represent the sequence of CAT and API, respectively (see Figure 4.18), and the pair of the first two bytes of each row of the MIST report denote CAT-API. For the extracted CAT-API sequences, a sliding window

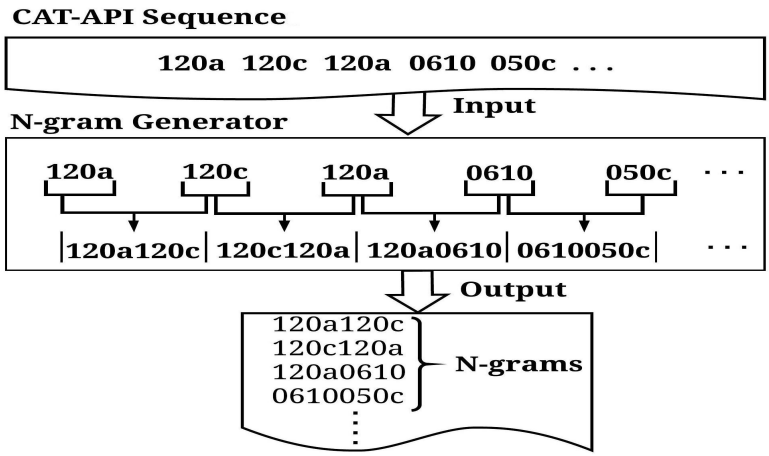


Figure 4.19: N-grams generation for the CAT-API sequence extracted from the sample MIST report shown in Figure 4.18

approach is applied to derive N-grams of size 4 bytes as shown in Figure 4.19. Each of the derived N-grams is viewed as a feature and the entire collection of the N-grams is considered as an original N-gram features' set. Past works illustrate that N-gram of size four bytes shows encouraging results (Kolter and Maloof, 2006; Masud et al., 2008). Accordingly, N-grams of size four bytes were adopted in the present work. The N-gram technique generates numerous N-grams that comprise of redundant N-grams. The repetitive N-grams affect space, processing time, and classifier performance. Thus, all the constructed N-grams cannot be used to train the classifier. To address this issue, duplicate N-grams were eliminated from the original N-gram features set and only distinct N-grams were considered as input to the Feature Selector to obtain prominent features.

For instance, to remove the redundant features, at first, the union operation was applied on the BNF $[BNF_1, BNF_2, ..., BNF_n]$ and MNF $[MNF_1, MNF_2, ..., MNF_m]$ that was represented as $[BNF_1 \cup BNF_2 \cup ... \cup BNF_n \cup MNF_1 \cup MNF_2 \cup ... \cup MNF_m]$, which then achieved a consolidated file as the unique N-gram features' set.

4.4.2 Feature Selector

The Feature Selector utilizes the filter-based FST to select significant N-grams from the unique N-gram features' set, thereby increasing the viability of the classifier in recognizing the malware. The prime aim of considering the FST in the proposed method was to regulate the elaborate learning process behind the CNN by, (i) recommending

significant features which are informative for effective classification by the classifier, and (ii) reducing significant redundancy in the CNN by pruning the neurons. Reducing inputs is essential because it avoids the network from learning of the associations between the noisy features, which result in negative and leads to degrading performance. Therefore, in the present approach, an attempt was made to investigate the effectiveness of the CNN in discriminating malware from benign PE files based on the images generated by using the features advised by the FST. In the present work, widely used FSTs such as Chi-Square (Belaoued and Mazouzi, 2015), MI (Yang and Pedersen, 1997), IG (Kolter and Maloof, 2006), and Relief (Kononenko, 1994) were employed individually to choose prime N-grams. Any classifier model ought to be trained with an adequate number of features so that it can quickly and easily distinguish malware from benign files. Thus, experiments were conducted by selecting the topmost 676 features (i.e., $\approx$ 25% of unique N-gram features set) recommended by the FST.

The explanation for two FSTs such as MI and IG is provided in Section 3.1.2 and Section 4.1.3, respectively. The description concerning other FSTs such as Chi-Square and Relief is described below.

Chi-Square ($\tilde{\chi}^2$)

The Chi-Square FST is employed to examine the dearth of independence of the two variables, i.e., feature 'f' and class 'c' (Yang and Pedersen, 1997) (Belaoued and Mazouzi, 2015; Ajay Kumara and Jaidhar, 2017). The Chi-Square score was computed as per Eq. 4.4 and the higher Chi-Square score signifies a close relation between feature and class.

$$\tilde{\chi}^2(f,c) = \frac{N[AD - BC]^2}{(A+C)(B+D)(A+B)(C+D)} \tag{4.4}$$

Where, 'N' represents the total files in the training dataset, 'A' represents the total files containing the feature 'f' in class 'c', 'C' indicates the total files present in class 'c' in which feature 'f' does not exist, 'B' denotes the total files containing the feature 'f' that does not exist in class 'c', and 'D' denotes files not present in class 'c' and does not have the feature 'f'.

Relief

Relief FST employs feature relevance criterion to rank the features. It is very efficient in estimating the features (Kononenko, 1994; Coronado-De-Alba et al., 2016). It quantifies features by how well their values discriminate among instances that are close to each other. For instance, it explores its two nearest neighbours: i) one from the same class defined as the nearest hit, and ii) the other from a different class termed as the nearest miss. The Relief estimates W[f] of feature 'f' as per Eq. 4.5.

$$W[f] = P(different\,value\,of\,f|nearest\,instance\,from\,different\,class) - P(different\,value\,of\,f|nearest\,instance\,from\,same\,class) \tag{4.5}$$

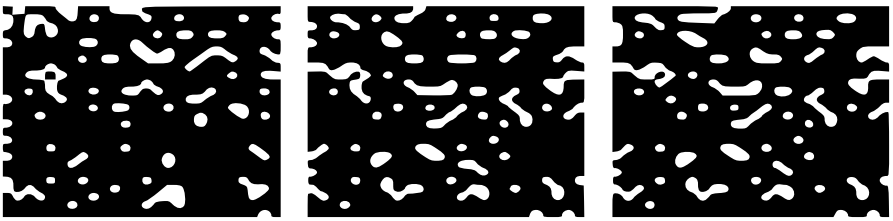
It is reasonably inferred that using Relief, good features can distinguish between instances of different classes and subsequently, instances of the same class should have the same value.

4.4.3 Final Features Set

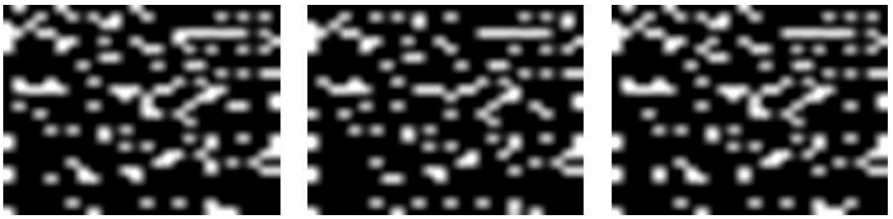
The Final Features Set comprises of the topmost unique N-grams advised by the FST. These N-grams are utilized as ultimate features, since there is no further reduction in the features' set. In the proposed work, the N-grams were highly crucial to generate images, as they are prerequisite for the chosen CNN to perform the classification task.

4.4.4 Image Generator

Image generation is the foremost task in the proposed work. The Image Generator takes the N-grams present in the Final Features' Set one at a time and checks its occurrence in the N-gram file to construct an image for the N-gram file. Based on the presence or absence of the N-gram in the N-gram file, the value 255 or 0 is written onto the image ('0' represents that the N-gram is absent and '255' indicates that the N-gram is present). Thereby, black and white images corresponding to the N-gram files are created, and the generated images are employed to measure the potency of the proposed approach. The dimension (height x width) of the image mainly relies on the topmost significant



(a) AutoRun Family



(b) Sality Family

(c) Zbot Family

Figure 4.20: Illustration of malware images of different families

features advised by the FST. As seen in Figure 4.20, the generated images that belong to the same malware family are visually similar, and they differ conspicuously from the images that belong to other families. This helps the CNN to detect and precisely classify the input PE file.

4.4.5 Convolutional Neural Network

CNN is a multilayer perceptron that has the proficiency to learn a representation of the raw data with multiple levels of abstraction (Albelwi and Mahmood, 2017) for identifying two-dimensional shapes. It consists of a set of learnable kernels, which are convolved across the width and height of the input features. The CNN has a standard structure composed of three types of layers: 1) convolutional layer, 2) sub-sampling layer (pooling layer), and 3) fully-connected layer, as depicted in Figure 4.21.

First is the Input Layer, which fetches the training images into the Neural Network. It is then followed by the convolution and sub-sampling layers. The convolution layer intensifies the signal characteristics to minimize the noise, whereas, the sub-sampling layers reduce the data processing while preserving useful information.

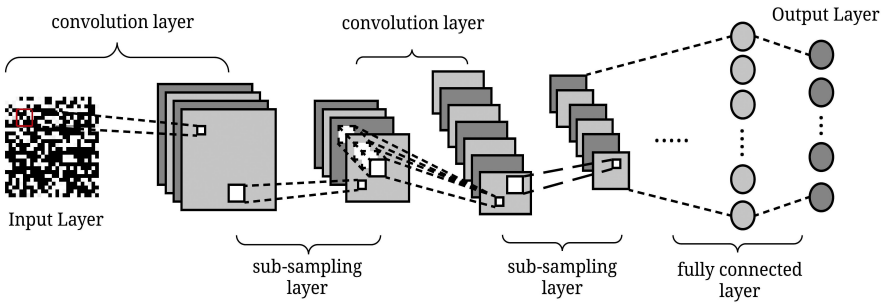


Figure 4.21: General overview of the Convolutional Neural Network to perform image classification

Next is the fully connected layer that converts the two-dimensional features into a one-dimensional feature to comply with the classification criteria. The CNN is trained by back-propagation as a means of Stochastic Gradient Descent to compute the weight and bias that minimizes loss function to map arbitrary inputs into the targeted outputs. To make the training faster, activation functions are employed because they train the Neural Network without significant penalty. In the proposed method, each input image proceeded through 2 convolution layers, 2 sub-sampling layers, and 2 fully connected layers. The process of convolution included 256 learnable filters of size 5×5. Similarly, during the sub-sampling process, it adopted a window size of 2×2, which minimized the training parameters. After the second sub-sampling layer, the output feature map was flattened and connected to 2 fully connected layers whose dimensions were 256, 128, and 10 (number of malware classes). The first fully connected layers adopted 'ReLu' as the activation function and the second used 'softmax' as its activation function.

4.4.6 Prediction Phase

In the prediction phase, the test image was created from the unknown PE files, which is essential to evaluate the discriminative performance of the trained CNN. The Final Features' Set and the output of the Behaviour-based Feature Extractor were used to deliver the test image, which was then sent to the trained CNN to measure the detection ability of the trained CNN.

4.4.7 Experimental Results and Discussion

4.4.7.1 Data Collection

Initially, a dataset comprising of 200 benign and 200 malware PE files was utilized to acquire the execution time behavioural reports from the Cuckoo Sandbox. However,

the 400 PE files were insufficient to train the malware detector as they did not mimic the real-world scenario. Thus, the Malheur dataset (Rieck et al., 2011) was used and it provided the behavioural reports of the PE files in the MIST format. To demonstrate the effectiveness of the proposed CNN-based Windows malware detector, 3282 benign and 4151 malware MIST files were used that included ten different types of malware such as Allaple (719), AutoIt (365), AutoRun (128), Basun (393), Lipler (119), NetMon (54), Sality (66), Swizzor (594), Texel (1662), and Zbot (51).

The CAT-API features were extracted from the MIST files to generate N-grams as explained in Section 4.4.1.3. The constructed unique N-gram features set consisted of 2779 N-grams, which was quite large. Therefore, the FST was used to obtain prominent N-grams. The topmost 676 N-grams advised by the chosen four FSTs were acquired independently and considered as the Final Features Set; accordingly, four Final Features Sets were obtained. The Final Features Set achieved by employing the Chi-Square FST and a dataset of 7433 N-gram Files were utilized to create 7433 images (i.e., 3282 of benign images and 4151 of malware images) of 26×26 dimension, and every image indicated one PE file. Similarly, three other sets of 7433 images were created separately using the three other Final Features' Sets as suggested by the FSTs such as Relief, MI, and IG. Finally, there were four image sets with each set consisting of 7433 images. These generated sets of images were used in the 10-fold cross-validation tests to appraise the ability of the proposed CNN-based approach.

Four evaluation metrics such as F-Measure, Accuracy, Precision, and Recall (Ajay Kumara and Jaidhar, 2017) were utilized in the present work to appraise the detection and classification ability of the proposed CNN-based Windows malware detector as well as the performance of the six machine learning-based classifiers (see Eq. 2.1). A malware detection method with a high detection rate and low misclassification rate is said to be proficient in identifying any unknown malware. An ideal malware detector is said to have high Precision and high Recall. However, in practice, it is difficult to achieve both.

4.4.8 Examination of the Proposed Approach with Machine Learning-based Classifiers

The proficiency of the proposed CNN-based Windows malware detection approach was demonstrated by conducting a series of experiments based on the prominent N-grams advised by the chosen four different filter-based FSTs. Based on the obtained experi-

Table 4.8: Comparison of performance achieved by machine learning-based classifiers and the proposed CNN-based approach for collected Malheur dataset

Feature Selection Technique	**Metrics**	**Classifier(s)**						
		SMO	**Simple Logistic**	**Logistic**	**J48**	**Random Forest**	**Random Tree**	**Proposed approach**
Chi-Square	Accuracy	96.542	96.650	94.093	96.381	95.681	93.354	97.928
	Precision	0.966	0.967	0.942	0.964	0.957	0.934	0.979
	Recall	0.965	0.967	0.941	0.964	0.957	0.934	0.979
	F-Measure	0.965	0.966	0.941	0.963	0.957	0.933	0.979
	Time (Sec)	0.343	2.703	8.170	0.291	0.324	0.016	3.093
Information Gain	Accuracy	96.515	96.515	94.080	96.367	95.910	93.730	97.713
	Precision	0.965	0.965	0.942	0.964	0.960	0.937	0.977
	Recall	0.965	0.965	0.941	0.964	0.959	0.937	0.977
	F-Measure	0.965	0.965	0.941	0.963	0.959	0.937	0.977
	Time (Sec)	0.429	1.646	8.218	0.301	0.354	0.011	3.125
Mutual Information	Accuracy	96.515	96.461	93.959	96.340	95.856	93.717	97.847
	Precision	0.965	0.965	0.941	0.964	0.959	0.937	0.978
	Recall	0.965	0.965	0.940	0.963	0.959	0.937	0.978
	F-Measure	0.965	0.964	0.940	0.963	0.958	0.937	0.978
	Time (Sec)	0.533	2.319	8.119	0.303	0.333	0.007	3.089
Relief	Accuracy	96.488	96.650	94.053	96.273	95.896	93.771	97.968
	Precision	0.966	0.967	0.942	0.963	0.960	0.938	0.979
	Recall	0.965	0.967	0.941	0.963	0.959	0.938	0.979
	F-Measure	0.965	0.966	0.941	0.962	0.959	0.937	0.979
	Time (Sec)	0.411	1.743	7.893	0.300	0.310	0.006	3.109

Table 4.9: API calls and their respective Categories exist in the Final Features Set recommended by Relief Feature Selection Technique

Category Name	**Category Identifier**	**Invoked API call #**	**API's**
hooking	01	2	SetWindowsHookExW, SetWindowsHookExA
network	02	3	getaddrinfo, HttpOpenRequestA, HttpSendRequestA
windows	03	6	FindWindowA, FindWindowExA, FindWindowExW, FindWindowW, CreateWindowExW, CreateWindowExA
threading	04	7	CreateRemoteThread, OpenThread, CreateThread, ExitThread, NtGetContextThread, NtSuspendThread, NtResumeThread
process	05	8	NtWriteVirtualMemory, TerminateProcess, VirtualFreeEx, NtReadVirtualMemory, OpenProcess, VirtualProtectEx, NtCreateProcessEx, CreateProcessInternalW
system	06	7	UnhookWindowsHookEx, LdrGetProcedureAddress, IsDebuggerPresent, LookupPrivilegeValueW, LdrLoadDll LdrGetDllHandle, NtDelayExecution
services	07	2	StartServiceW, OpenServiceA
synchronization	08	2	NtOpenMutant, NtCreateMutant
registry	09	8	RegCreateKeyExW, RegDeleteKeyA, RegEnumValueW, RegCloseKey, RegDeleteKeyW, RegQueryValueExA, RegEnumKeyExW, RegOpenKeyExW
filesystem	0a	7	NtOpenFile, NtReadFile, CreateDirectoryW, NtWriteFile, MoveFileWithProgressW, NtCreateFile, CreateDirectoryW
device	0b	1	DeviceIoControl
memory	0c	3	NtOpenSection, NtCreateSection, NtAllocateVirtualMemory
socket	0d	1	WSAStartup

Table 4.10: Comparison of performance achieved by machine learning-based classifiers and the proposed CNN-based approach for generated dataset

Feature Selection Technique	Metrics	Classifiers						
		SMO	Simple Logistic	Logistic	J48	Random Forest	Random Tree	Proposed approach
Chi-Square	Accuracy	79.250	82.250	70.250	78.250	75.500	73.500	97.499
	Precision	0.786	0.815	0.693	0.776	0.742	0.727	0.974
	Recall	0.793	0.823	0.703	0.783	0.755	0.735	0.973
	F-Measure	0.788	0.818	0.694	0.778	0.743	0.729	0.974
	Time (Sec)	0.104	0.672	6.027	0.072	0.229	0.007	3.086
Information Gain	Accuracy	80.000	82.250	68.250	77.250	75.250	74.000	95.749
	Precision	0.792	0.814	0.674	0.762	0.736	0.732	0.956
	Recall	0.800	0.823	0.683	0.773	0.753	0.740	0.957
	F-Measure	0.794	0.816	0.677	0.765	0.740	0.734	0.956
	Time (Sec)	0.077	0.662	6.546	0.056	0.191	0.006	3.164
Mutual Information	Accuracy	79.750	81.500	66.250	80.250	78.250	75.750	95.500
	Precision	0.790	0.808	0.656	0.795	0.773	0.750	0.955
	Recall	0.798	0.815	0.663	0.803	0.783	0.758	0.953
	F-Measure	0.792	0.811	0.658	0.797	0.774	0.752	0.955
	Time (Sec)	0.120	0.868	4.937	0.050	0.161	0.009	3.094
Relief	Accuracy	81.000	79.250	69.250	82.000	80.500	74.000	97.999
	Precision	0.804	0.786	0.686	0.820	0.798	0.739	0.979
	Recall	0.810	0.793	0.693	0.820	0.805	0.740	0.978
	F-Measure	0.806	0.789	0.688	0.819	0.796	0.739	0.979
	Time (Sec)	0.110	0.536	6.482	0.077	0.159	0.008	3.110

mental results, the performance of the proposed approach was compared with the performance of the other six machine learning-based classifiers available in the WEKA (Frank et al., 2009) tool such as RF, SMO, J48, Logistic, Random Tree, and Simple Logistic. Table 4.8 demonstrates the accuracy achieved by the proposed approach for 50 epochs and six machine learning-based classifiers.

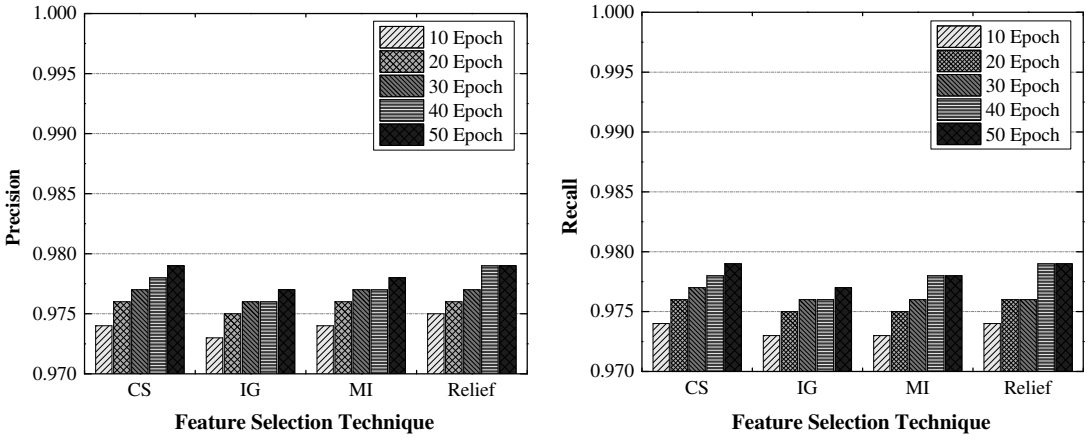
It can be noticed from Table 4.8 that the proposed CNN-based approach gained highest accuracy of 97.968% with 0.979, 0.979, and 0.979 of F-Measure, Precision, and Recall, respectively, with the images created by using the topmost 676 N-grams recommended by the Relief FST. Contrarily, the experimental results delineated in Table 4.8 demonstrates that the machine learning-based classifiers such as J48, Logistic, RF, SMO, and Random Tree underperformed by attaining less accuracy for the same topmost 676 N-grams. However, the Simple Logistic classifier reported better accuracy of 96.650% with 0.966, 0.967, and 0.967 of F-Measure, Precision, and Recall, respectively. Similar experiments were conducted by employing the other three FSTs and the obtained experimental outcomes are shown in Table 4.8.

The analysis of results obtained by the proposed CNN-based Windows malware detector with four different FSTs led to understand that employing FST at the input level

of CNN boosts the detection accuracy of the classifier as shown in Table 4.8. However, the relative analysis performed using the CNN-based approach with each of the chosen FST demonstrated that the accuracy accomplished by them was almost the same with an accuracy difference of less than 0.5%. It is evident that the performance of the CNN-based approach is appreciable in terms of achieving better accuracy when it is trained with the images generated using the most relevant features recommended by the FST. However, the accuracy achieved by the machine learning-based classifiers was comparatively less (i.e., $< 97\%$) as illustrated in Table 4.8. The main reason behind this is due to their shallow architecture (Deng, 2012; Hardy et al., 2016). Although these have shown effective performance in simple or well-constrained scenarios, their limited modelling and representation capabilities experience difficulties in dealing with colossal training data. The execution of machine learning models with shallow structures gets saturated for more substantial training data due to their constrained learning capacity. Thus in the present work, a CNN-based Windows malware detector was proposed and the obtained experimental results demonstrated that the proposed approach was significant when compared with machine learning-based classifiers due to their efficient learning ability. Most relevant CAT and API calls suggested by the Relief FST are represented in Table 4.9.

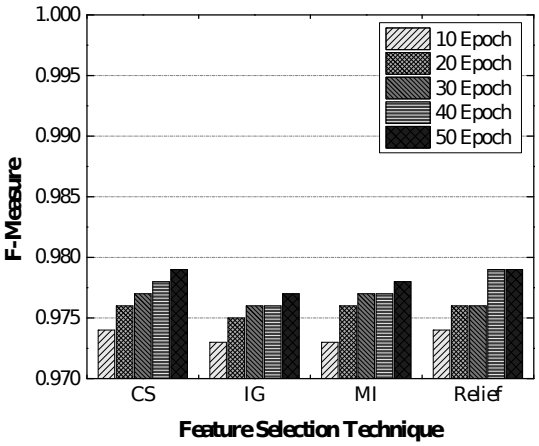
Table 4.10 shows the obtained experimental results for the generated dataset that consisted of 200 benign and 200 malware PE files. It can be noticed from Table 4.10 that the proposed CNN-based Windows malware detector performed better than the machine learning-based classifiers even for a smaller dataset. However, the machine learning-based classifiers' performance was not remarkable. The main reason is since the 10-fold cross-validation tests were performed on a smaller dataset, the machine learning-based classifiers was subjected to overfitting due to insufficient number of training files to perform the prediction. With these comparative results, it can be reasonably inferred that the proposed CNN-based approach is significant in detection and classification of unknown malware for both, smaller datasets (generated dataset) and larger datasets (collected Malheur dataset).

Another observation was made in terms of accuracy and time, when the proposed CNN-based approach for 50 epochs was compared with the machine learning-based classifiers, and the recorded values are shown in Table 4.8 and Table 4.10. The proposed



(a) Precision

(b) Recall



(c) F-Measure

Figure 4.22: Performance evaluation of the proposed CNN-based approach using different metrics based on best relevant N-grams recommended by each Feature Selection Technique (CS: Chi-Square, IG: Information Gain, MI: Mutual Information, Relief) by varying the number of epoch

CNN-based Windows malware detector showed advantages in accuracy, but performed slightly slower than the machine learning-based classifiers.

4.4.9 Analysis of the Proposed Approach

Another set of experiments were conducted to determine the efficiency of the proposed approach by varying the number of epochs, and the subsequently obtained F-Measure, Recall, and Precision values were recorded to measure its effectiveness. Exactness is measured by Precision and completeness is measured by Recall. However, the malware detection method is said to be ideal if these performance metrics attain a value closer to 1.

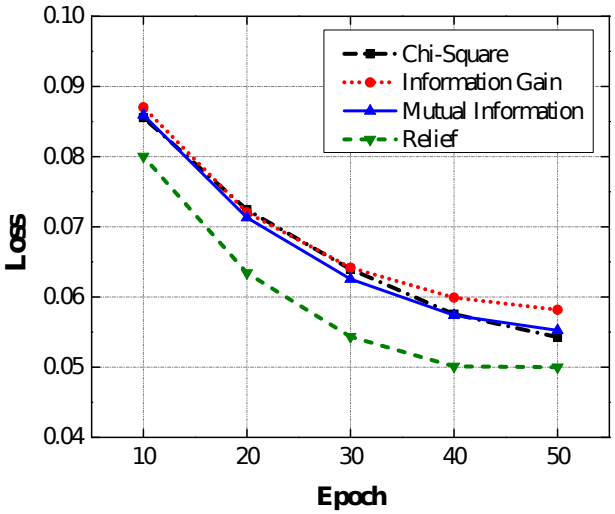


Figure 4.23: Evaluation of loss occurred by the proposed CNN-based Windows malware detector

Figure 4.22 demonstrates the results accomplished by the proposed CNN-based Windows malware detector by employing the best 676 N-grams recommended by the four different FSTs such as Chi-Square, IG, MI, and Relief, by varying the number of epochs from 10 to 50. It can be seen that when the CNN is trained with the images generated by the N-grams recommended by the Relief FST, the proposed approach achieves F-measure equal to 0.979, Precision of 0.979 and high Recall of 0.979, when the number of epochs is set at 50 (See Figure 4.22a, Figure 4.22b, and Figure 4.22c). Similarly, the proposed approach showed competitive performance for images that were generated based on the topmost 676 N-grams suggested by the Chi-Square FST. In this case, the CNN yielded high Precision of 0.979, F-Measure of 0.979, and Recall of 0.979 for 50 epochs as shown in Figure 4.22a, Figure 4.22b, and Figure 4.22c. It can be observed from Figure 4.22, that the performance of the proposed approach is poor when it is trained with the images generated based on the N-grams recommended by the IG and MI FSTs (see Figure 4.22a, Figure 4.22b, and Figure 4.22c).

On thorough analysis, it is understood that increased number of epochs can improve the accuracy of the proposed CNN-based approach. Further, to prevent the network from learning the associations between the irrelevant features, feature selection at an input level of CNN is necessary to enhance the predictive performance of the classifier. As evidence, the proposed CNN-based Windows malware detector attained better detection accuracy and outperformed for Relief FST based images when compared with other individual FSTs.

Table 4.11: Time to pre-process and predict three different test files of different size

Process Steps	**Time (Sec) taken for 50 Epoch**		
	Min. File 584KB	**Avg. File 1.8MB**	**Max. File 2.2 MB**
Conversion of MIST File (Behavioural Report) to Image File	0.260	0.418	0.504
Prediction	3.044	3.076	3.085
Total	3.304	3.494	3.589

The loss function or optimization score function was also used to measure the performance of the proposed classification model. From Figure 4.23 it can be understood that the proposed CNN-based approach performed well by gaining sequentially less loss value at each epoch when it was trained with images generated based on the topmost 676 N-grams recommended by the Relief FST.

Table 4.11 provides the time taken by the proposed CNN-based Windows malware detector to predict the given test files (MIST) of three different sizes such as Minimum (Min.), Average (Avg.), and Maximum (Max.). As the size of the test image file (26×26) is the same for all the three different test files, the time taken for prediction is almost equal. To investigate the feasibility of using the proposed approach in real-time situations, it was implemented on the host system to do the prediction.

4.5 SUMMARY

In Section 4.1, the performance of the system calls-based Windows MDS in achieving a better detection rate was investigated. The Cuckoo Sandbox was employed to gather system-level behaviour of the PE files. The system calls sequence, triggered by the PE files, was extracted from the MIST reports obtained by pre-processing the JSON reports. The IG FST suggested prominent N-gram features were employed to construct the Final Feature Set for different N-gram lengths individually. A separate experiment was conducted for the Final Feature Set of each N-gram size. Through thorough experimental analysis, the machine learning-based classifier Spegagos ensured better classification for both N-gram lengths of three and four bytes.

In Section 4.2, a multiprocessing model was proposed to minimize the time required by the FST to compute the score for the N-grams (features). The computation time taken by the multiprocessing model and the sequential mode1 was measured to

compare the processing efficiency. Experiments were conducted with N-grams datasets of different sizes, and in each experiment, the performance of the proposed multiprocessing model in computing the IG score was high compared with the sequential model. On average, the proposed approach was 80% faster than the sequential model of the IG score computation.

In Section 4.3, the effectiveness of the behavioural features suggested by the LSVC in identifying unknown malware was investigated. Two different types of behavioural features, namely, API calls and Category+API calls were considered to know which type of features provided better malware detection rate. The highest detection accuracy was obtained with API calls as the behavioural feature.

Section 4.4 proposed CNN-based Windows malware detector showed that the malware detection problem could be transformed into an image classification problem. Further, to prevent the Neural Network from learning the associations between the irrelevant features, experiments conducted to demonstrate feature selection at the input level of CNN is necessary. A set of experiments were conducted to demonstrate the classification ability of the proposed approach and compared with the chosen six machine learning-based classifiers. The obtained empirical results reasonably presented that the proposed approach is superior to the machine learning-based classifiers.

The main limitation of the proposed behavioural features-based malware detection approach is that it is sometimes likely getting susceptible and fail to detect the PE file as benign or malware accurately. The reason is that sophisticated malware can unnecessarily enter sleep mode and remain idle to monitor the anti-defensive approach applied to identify them. In such cases, statically analyzing the PE file becomes most essential, and that provides information about the intent of the malware behaviour of the PE file. To address this issue, the hybrid features-based malware detection approach was proposed and discussed in Chapter 5. It mainly uses the combination of both the static and behavioural features to discern the PE file as benign or malware.

Chapter 5

Hybrid Features-based Malware Detection System

Static analysis and dynamic analysis are complementary to one another (Bounouh et al., 2017). Dynamic analysis provides paramount insight to the malware, whereas static analysis is unable to provide significant information required to analyze the malware in real-time. In the present work, a Hybrid Features-based Malware Detection System (HFMDS) has been proposed that detects Windows malware by extracting and analyzing static features and dynamic features of the PE files. The prime objective is to measure the effectiveness of the MDS that uses both static and dynamic features.

5.1 WINDOWS MALWARE DETECTION SYSTEM BASED ON LSVC RECOMMENDED HYBRID FEATURES

The proposed HFMDS uses header information of the PE files such as OH, DOSH, and FH as static features together with dynamic features of the PE files to ascertain the Windows malware. Two types of dynamic features were used, namely, API calls alone and API calls with their correspondent Category (APICAT), and these features were extracted from the Cuckoo Sandbox generated behavioural report. The following are the key highlights of the present work:

- The HFMDS was designed, implemented, and evaluated using publicly available real-world malware samples. To validate its effectiveness, different set of experiments were conducted by considering different combinations of the hybrid features (API+OH+FH+DOSH, API+OH, API+FH, API+ DOSH, and OH+DOSH+ FH) and individually with only API calls as features. Further, another set of experiments were carried out by considering APICAT as dynamic features with the same static features. Finally, a comparative analysis was made to know which combination of the features provided better detection rate.

- To measure the detection rate of the HFMDS, 10-fold cross-validation tests were conducted. The obtained experimental results demonstrated that the HFMDS was effective in precisely identifying malware and benign PE files using hybrid

features. The HFMDS was able to attain highest detection accuracy for API-CAT+OH+DOSH+FH hybrid features.

5.2 Architecture Overview of HFMDS

An overview of the proposed HFMDS is shown in Figure 5.1. It mainly consists of two phases: 1) the Training Phase and 2) the Prediction Phase. The HFMDS utilizes the merits of both the static and dynamic malware analysis techniques.

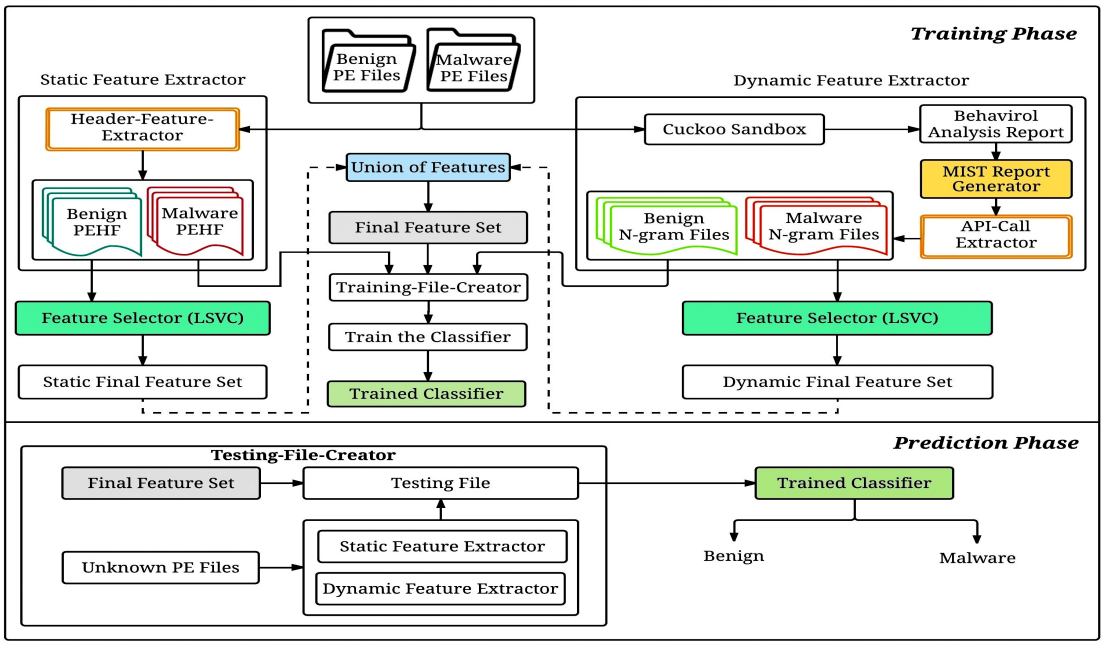


Figure 5.1: Architecture of the proposed HFMDS

5.2.1 Training Phase

In the training phase, a set of benign and malware PE files are provided to derive static and dynamic/behavioural features. The extracted features are processed further to eliminate noisy features using a specific approach elucidated in the HFMDS, and then a training file is prepared, which is essential to train the HFMDS. The main components of the training phase are Static Feature Extractor, Dynamic Feature Extractor, Feature Selector, Final Feature Set, and Training-File-Creator.

5.2.1.1 Static Feature Extractor

The prerequisite task of the Static Feature Extractor is to identify and extract static features such as OH, DOSH, and FH from the PE files. Each extracted data is considered

as an individual feature. Its sub-component is the Header-Feature-Extractor.

5.2.1.2 Header-Feature-Extractor

It uses the Python module called pefile (Ero, 2017) to extract all the OH, DOSH, and FH related data, and then stores the extracted data into an appropriate output file. For each input file, the Header-Feature-Extractor maintains three separate output files; one file to store OH related information and the other two files to store DOSH and FH related information. Since the HFMDS treats each extracted data as an individual feature, all the OH related features extracted from all the training files are gathered to form an OH original feature set. Similarly, all DOSH and FH related features are gathered to form a DOSH original feature set and an FH original feature set, respectively. However, all the extracted features in the original feature set may not contribute to the detection of unknown malware because the extracted features may comprise of redundant and noisy/irrelevant features. Since noisy features reduce the predictive performance of the classifier, identifying and eradicating these features plays a crucial role. In order to address this problem, the Feature Selector is utilized.

5.2.1.3 Feature Selector

It is used to identify and select significant features from the original feature set that increase the effectiveness of the classifier in detecting unknown malware. It employs the LSVC (see Chapter 4, Section 4.3.1.2) as a feature selector to detect relevant and irrelevant features from the original feature set by assigning a separate score to each individual feature. The features with the highest score are considered as predominant features and are selected as topmost features for the classification task.

In the present work, the LSVC is used to select the most significant static features (OH, DOSH, and FH) and dynamic features (API calls and APICAT) from their respective original feature set. Subsequently, the topmost static features and dynamic features are used to prepare a Static Final Feature Set and Dynamic Final Feature Set, respectively. Finally, all these selected features play an important role in precisely classifying the input files.

5.2.1.4 Static Final Feature Set

The Static Final Feature Set is a set of static features derived from the PE files. Later, these features are provided as input to the Union of Features to amalgamate with the Dynamic Final Feature Set, and finally, a Final Feature Set is prepared to build training as well as a testing file essential to evaluate the efficiency of the classifier(s).

5.2.2 Dynamic Feature Extractor

The prime task of the Dynamic Feature Extractor is to observe and record the behaviour of the PE files, while one is being executed in a controlled monitoring environment. Its main sub-components are: 1) Cuckoo Sandbox, 2) Behavioural Analysis Report, 3) MIST Report Generator, and 4) API-Call Extractor.

5.2.2.1 Cuckoo Sandbox

It is used to obtain a behavioural report of the PE file during the runtime of the one being executed (Guarnieri et al., 2012) in the JSON file format, as explained in Section 4.3.1.1.

5.2.2.2 Behavioural Analysis Report and MIST Report Generator

Behaviour analysis is also called as a dynamic analysis report (Firdausi et al., 2010). The Cuckoo Sandbox captures the API calls triggered by the PE file and classifies them into several categories based on the type of operation performed (Miller et al., 2017). Moreover, as per Chapter 4, it is understood that the category of an API call in terms of the type of operation performed helps to determine the actions taken by the PE file. So, the focus is to extract details such as category and API call with the corresponding arguments from the JSON file, and then, represent the acquired particulars in the MIST format.

The MIST format (Rieck et al., 2011) is used in the present work to record all system level behaviour in which the category, API call, and arguments are organized in different levels or blocks, as shown in Figure 4.14. MIST reports are created by implementing the MIST conversion process for all the runtime behaviour reports (JSON) produced by the Cuckoo Sandbox.

5.2.2.3 API-Call Extractor

The task of the API-Call Extractor is to extract API calls and APICAT from the MIST report and its implementation is programmed using the Python programming language. As the name indicates, the approach is specific to observation of the monitored API calls and APICAT. So, the category and operational field of the MIST report are of concern. The values in the operational field and the values of both the categories with the operation field are extracted independently to prepare N-grams (Reddy and Pujari, 2006; Masud et al., 2008; Pektaş et al., 2011; Das et al., 2016) of size four bytes.

5.2.2.4 Dynamic Final Feature Set

The Dynamic Final Feature Set is a set of features that is derived from the behavioural analysis report produced by the Cuckoo Sandbox for all the PE files of the dataset.

5.2.3 Final Feature Set and Training-File-Creator

The Final Feature Set is a set of hybrid features that consists of the most significant static and dynamic features. These features are employed as final features since there is no further features elimination.

The Training-File-Creator creates a training file essential to train the classifiers. It parses the training dataset of the benign and malware files with the Final Feature Set features to create a training file as shown in Figure 5.1.

5.2.4 Prediction Phase

In the prediction phase, the Testing-File-Creator is used to create a testing file, which is necessary to appraise the predictive performance of the trained classifiers. It makes use of the Final Feature Set and the output of the Static Feature Extractor and the Dynamic Feature Extractor to deliver a testing file. The generated testing file is sent to the trained classifier, which classifies whether the test input file is benign or malware.

5.3 Results

5.3.1 Experimental Setup

To evaluate the performance of the proposed HFMDS, all the experiments were conducted on the host system, and its specifications are as mentioned in Section 2.7. Six

different classifiers such as SMO, Simple Logistic, Logistic, J48, RF, and Random Tree available in the WEKA (Frank et al., 2009) tool was adopted to evaluate the performance of the proposed HFMDS with default parameters settings.

5.3.2 Data Collection

The experimental data consisted of 3856 benign and 3856 malware PE files to demonstrate the effectiveness of the proposed HFMDS in detecting unknown malware. The collected files in the dataset were in the Windows PE file format and all were unpacked, and further, no duplicate PE files were found in the dataset. The benign Windows PE files were gathered from freshly installed Windows virtual machines that included the Windows-XP, Windows-7, and Windows-8 operating system files and other program files of 'x64' and 'x86' architecture. Some benign PE files were downloaded from free online software archives (CNET, 1996). The Windows malware PE files used in this experimental work were downloaded from a public source (VirusShare, 2011), and the collected files in the dataset included seven different types of malware such as Trojan, Worm, Exploit, Backdoor, Virus, Rootkits, and Flooder. The same dataset was utilized to extract the static and dynamic features required to validate the HFMDS. To have a reliable feature set for the training and prediction phases, it is necessary to ensure that all the PE files in the dataset are correctly labelled. Therefore, the dataset was scanned using (VirusTotal, 2004a).

5.3.3 Evaluation Metrics

An HFMDS with the highest detection rate and minimum FPR is proficient in identifying unknown malware. The effectiveness of the HFMDS and the detection performance of the classifiers were estimated using six evaluation metrics such as TPR, FPR, Precision, Recall, F-Measure, and Accuracy as shown in Eq. 2.1.

5.3.4 Results and Discussions

In the present work, the prime aim of the HFMDS was to explore the accurate detection and categorization of the malicious PE files using the Final Feature Set that consisted of hybrid features.

During the training file creation phase, the Static Feature Extractor produced 14985

OH features as an original OH feature set. Similarly, it produced 532 DOSH features and 4600 FH features independently. The LSVC was applied onto each original feature set separately to get the score for each of the features present in the original feature set. Accordingly, the LSVC recommended 1334 significant features from the original feature set of the OH. Further, 98 DOSH and 1492 FH features were advised as predominant features by the LSVC considering their respective original feature set. These features were then used to prepare three independent Static Final Feature Sets for each of the OH, DOSH, and FH.

One of the components of the Dynamic Feature Extractor, namely, the Cuckoo Sandbox was employed to monitor and record the runtime behaviour of the PE file. It stored the monitored behaviour of the PE file in the JSON file format. This was then pre-processed to convert the data into MIST format so that only relevant data such as the APICAT, API call, and the arguments were recorded. From the MIST format, (a) the API call sequence and (b) the APICAT sequence were extracted independently, and consecutive four byte sequences were grouped as one N-gram. For each MIST format file, two separate N-gram files were prepared, where one consisted of API call sequences and the other comprised of APICAT sequences. After removing duplicate N-grams from each N-gram file that had different sequences, 7292 N-grams (generated using API call sequences) and 3017 N-grams (generated using APICAT sequences) were attained from all the N-gram files of the dataset. Since all these N-grams did not contribute to malware detection, the LSVC was applied to choose the distinct features and to ignore the noisy features. The LSVC recommended 667 N-grams as prominent features out of 7292 N-grams and 429 N-grams as significant features out of 3017 to construct two separate Dynamic Final Feature Sets.

In the present work, the features present in the Dynamic Final Feature Set and the Static Final Feature Set were amalgamated to build a Final Feature Set in five different combinations to validate the efficiency of the proposed HFMDS. The six composite forms of the Final Feature Sets were: 1) only API calls, 2) API calls with OH (API+OH), 3) API calls with DOSH (API+DOSH), 4) API calls with FH (API+FH), 5) OH and DOSH with FH (OH+DOSH+FH), and 6) API calls, OH, and DOSH with FH (API+OH+DOSH+FH).

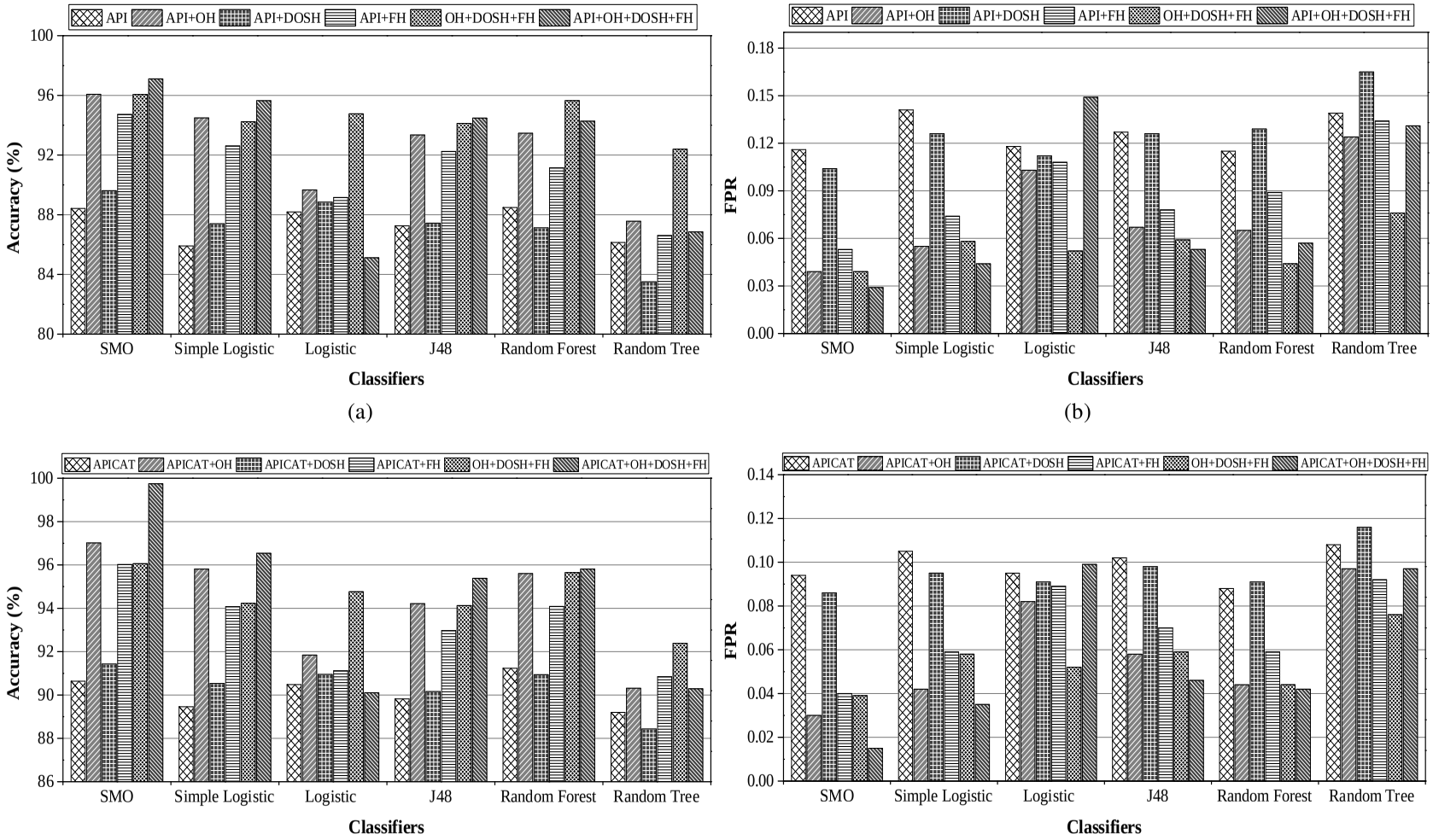


Figure 5.2: Performance evaluation of the classifiers for different combination of hybrid features set

Similarly, experiments were also carried out by considering dynamic features such as APICAT with other static features to ensure the effectiveness of the HFMDS in detecting malware. The experiments performed were reasonably able to prove that better detection accuracy can be achieved by the hybrid features compared with merely considering either dynamic features or static features.

5.3.5 Analysis of HFMDS Based on Evaluation Metrics

The prime goal of the experiments was to check the feasibility of the HFMDS in the detection and classification of unknown malware.

Initially, the experiments were carried out with only the dynamic features (invoked API calls). Accordingly, from Figure 5.2a and Figure 5.2b it can be noticed that the RF classifier obtained maximum detection accuracy of 88.488% with 0.115 FPR. However, the detection accuracy of the other classifiers such as SMO, Simple Logistic, Logistic, J48, and Random Tree achieved less accuracy. Meanwhile, experiments were also conducted and comparative analysis performed to show that better detection accuracy can be attained when APICAT is considered as dynamic features. Consequently, the RF classifier gained maximum detection accuracy of 91.234% with 0.088 FPR as shown in Figure 5.2c and Figure 5.2d. Further, classifiers such as SMO and Logistic yielded significantly less accuracy of 90.638% with 0.094 FPR and 90.482% with 0.095 FPR, respectively. On the other hand, Simple Logistic, J48, and Random Tree attained least accuracy of 89.458% with 0.105 FPR, 89.821% with 0.102 FPR, and 89.198% with 0.108 FPR, respectively.

From the experimental observations depicted in Figure 5.2a and Figure 5.2b, the SMO classifier achieved maximum accuracy of 96.071% and 0.039 FPR with the Final Feature Set corresponding to a combination of API+OH. Further, the Simple Logistic, RF, and J48 classifiers also performed well by producing an accuracy of 94.489% with 0.055 FPR, 93.464% with 0.065 FPR, and 93.348% with 0.067 FPR, respectively. Relatively the overall performance of the other classifiers such as Logistic and Random Tree reported lowest accuracy and their corresponding values were 89.665% with FPR 0.103 and 87.564% with FPR 0.124, respectively. Similarly, experiments were also performed to examine effectiveness in obtaining better detection accuracy when the Final Feature Set comprising of APICAT+OH was considered as the hybrid features. The SMO clas-

sifier achieved better detection accuracy of 97.017% with 0.03 FPR (see Figure 5.2c and Figure 5.2d). It can also be seen from Figure 5.2c that the second highest accuracy of 95.811% with 0.958 FPR was yielded by the Simple Logistic classifier, followed by other classifiers such as RF (95.604% with 0.044 FPR), J48 (94.216% with 0.058 FPR), Logistic (91.843% with 0.082 FPR), and Random Tree (90.313% with 0.097 FPR).

The accuracy achieved by the different classifiers with the Final Feature Set corresponds to a combination of API+DOSH as shown in Figure 5.2a and Figure 5.2b. From Figure 5.2a and Figure 5.2b, it can be observed that the highest detection accuracy of 89.613% with 0.104 FPR was obtained by the SMO classifier. However, the performance of the other classifiers was not remarkable. The second highest accuracy of 88.848% was accomplished by the Logistic classifier with 0.112 FPR. The J48 classifier recorded lowest accuracy of 87.435% with 0.126 FPR. Moreover, other classifiers such as Simple Logistic, RF, and Random Tree underperformed by achieving least accuracy of 87.396% with 0.126 FPR, 87.124% with 0.129 FPR, and 83.506% with 0.165 FPR, respectively. Further experiments were conducted to examine the detection ability of the classifiers for the Final Feature Set consisting of APICAT+DOSH as the hybrid features. From Figure 5.2c and Figure 5.2d, it can be noticed that the SMO classifier outperformed the other classifiers and achieved detection accuracy of 91.428% with 0.086 FPR. However, classifiers such as the Logistic, Simple Logistic, J48, and RF recorded detection accuracy of 90.949% with 0.091 FPR, 90.534% with 0.095 FPR, 90.158% with 0.098 FPR, and 90.936% with 0.091 FPR, respectively. Minimum performance was shown by the RF classifier by attaining a detection accuracy of 88.433% with 0.116 FPR.

For the combination of API+FH, the SMO classifier achieved highest accuracy of 94.722% with 0.053 FPR. Further, it can be noticed that both the Simple Logistic and J48 classifiers seem to be equipotent in obtaining detection accuracy of 92.608% with 0.074 FPR and 92.245% with 0.078 FPR. The performance of other classifiers such as the Logistic, RF, and Random Tree was not appreciable and yielded least accuracy of 89.159% with 0.108 FPR, 91.143% with 0.089 FPR, and 86.605% with 0.134 FPR, respectively, as depicted in Figure 5.2a and Figure 5.2b.

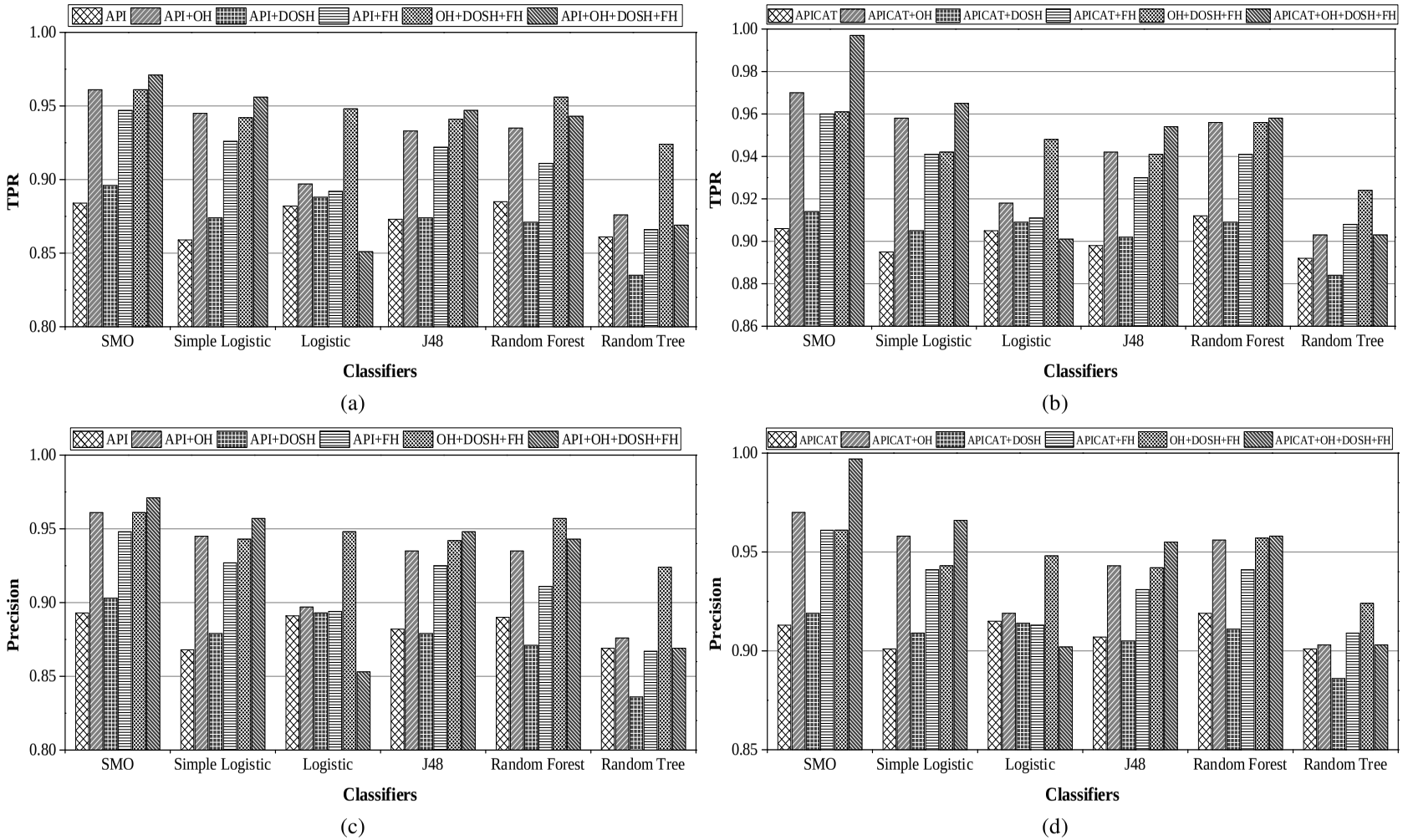


Figure 5.3: Performance evaluation of the classifiers for different combination of hybrid features set (i) TPR, and (ii) Precision.

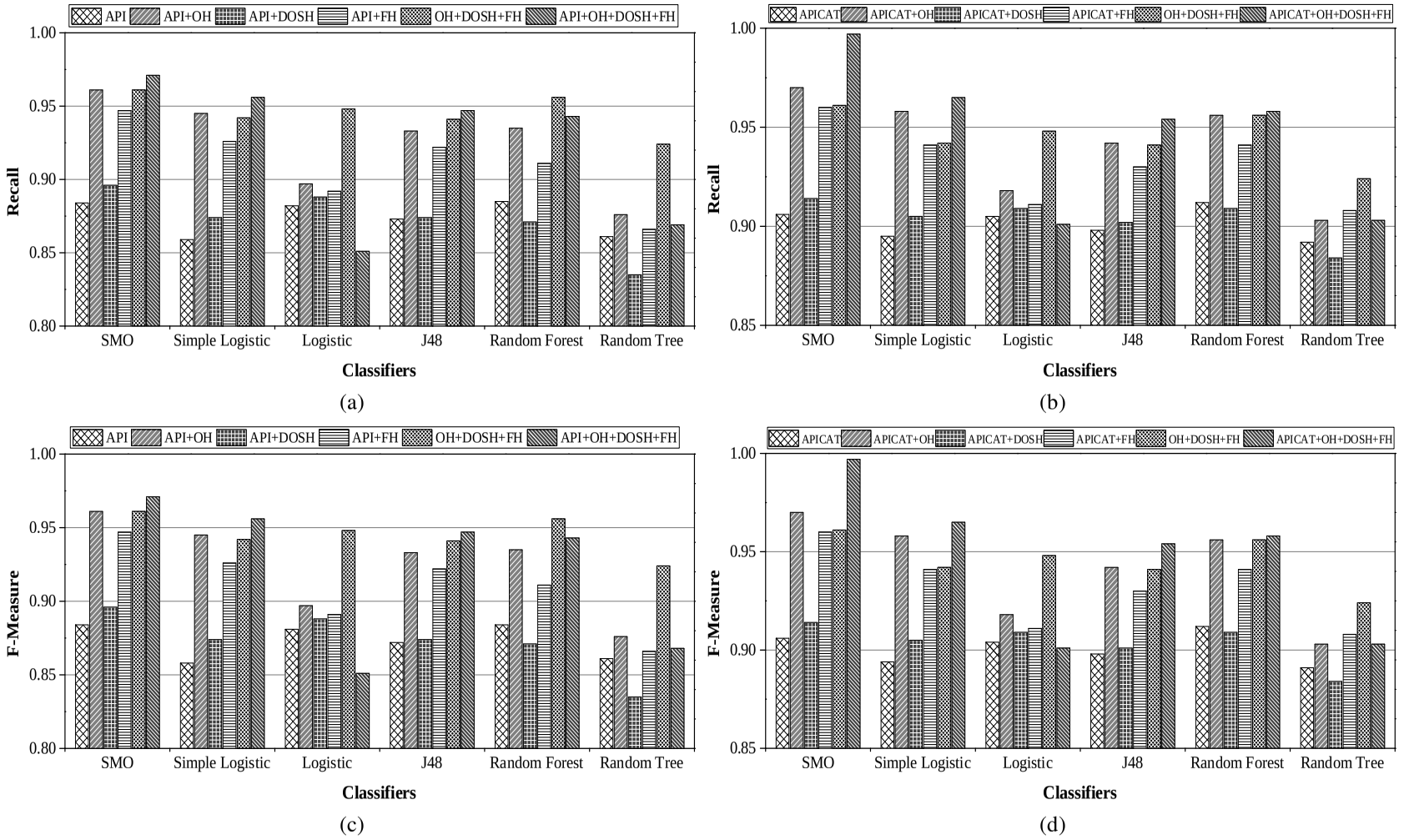


Figure 5.4: Performance evaluation of the classifiers for different combination of hybrid features set (i) Recall, and (ii) F-Measure.

Relatively, experiments were also conducted for a combination of APICAT+FH hybrid features. In this case, it was observed that the SMO classifier obtained maximum detection accuracy of 96.019% with 0.04 FPR. Meanwhile, classifiers such as the Simple Logistic and RF achieved significantly less detection accuracy of 94.074% with 0.059 FPR and 94.087% with 0.059 FPR, respectively. Other classifiers such as the Logistic, J48, and Random Tree underperformed and achieved less accuracy of 91.117% with 0.089 FPR, 92.972% with 0.070 FPR, and 90.845% with 0.092 FPR, respectively, as shown in Figure 5.2c and Figure 5.2d.

Later, the Final Feature Set built with OH, DOSH, and FH features was also considered. Consequently, Figure 5.2a and Figure 5.2c demonstrated that the SMO classifier attained highest accuracy of 96.058% with 0.039 FPR, followed by the RF classifier with 95.643% with 0.044 FPR. The three classifiers such as the Simple Logistic, Logistic, and J48 showed favourable performance by yielding an approximately equivalent accuracy of 94.229% with 0.058 FPR, 94.761% with 0.052 FPR, and 94.126% with 0.059 FPR, respectively. Subsequently, the performance of the Random Tree classifier was not remarkable.

Similar to the above sets of experiments, the next observation was made on the Final Feature Sets constructed with a combination of API calls, OH, DOSH, and FH features. It can be seen from Figure 5.2a and Figure 5.2b, that the highest accuracy of 97.108% with 0.029 FPR was reported by the SMO classifier. In addition, it can also be noticed that the Simple Logistic classifier was able to achieve the second highest accuracy of 95.643% with 0.044 FPR and substantiated its stable performance. Meanwhile, the J48 and RF classifiers showed competitive performance, but resulted in achieving less accuracy of 94.478% with 0.053 FPR and 94.268% with 0.057 FPR, respectively. In comparison, other classifiers such as the Logistic and Random tree produced least accuracy of 85.114% with 0.149 FPR and 86.851% with 0.131 FPR, respectively. Another set of experiments were also conducted on the Final Feature Sets constructed with a combination of APICAT, OH, DOSH, and FH features as depicted in Figure 5.2c and Figure 5.2d. Accordingly, the SMO classifier showed maximum detection accuracy of 99.743% with 0.015 FPR. The Simple Logistic classifier also performed well by producing an accuracy of 96.537% with 0.035 FPR. Further, the J48 and RF classifiers achieved remarkably less accuracy of 95.383% with 0.046 FPR and 95.811%

with 0.042 FPR, respectively. The overall performance of the other classifiers such as the Logistic and Random Tree reported lowest accuracy and their corresponding values were 90.106% with 0.099 FPR and 90.287% with 0.097 FPR.

Table 5.1 and Table 5.2 demonstrate the effectiveness of the classifiers for the hybrid features and each one represents the 'k' fold cross-validation results, where, k = 2, 3 . . . 10. From the thorough and substantial analysis conducted, it can be observed that most of the classifiers attained best accuracy for k=10. Further, it can be noticed that both the SMO and Simple Logistic classifiers performed well on all the composite forms of the Final Feature Sets. Moreover, through the analysis carried out, it was possible to discern that the SMO classifier provided competitive performance, and finally, achieved better accuracy. Apparently, the SMO classifier performed the best, and the Random Tree classifier exhibited the worst performance.

Experiments were also conducted with different Final Feature Sets to know their best detection rate. In this direction, the obtained and analyzed results proved that the Final Feature Set with the combination of APICAT+OH+DOSH+FH achieved better malware detection rate of 99.743% with low FPR 0.015 compared with other Final Feature Sets comprising of different combination of static and dynamic features.

For any MDS, highest TPR signifies its effectiveness in the precise detection of malware. Figure 5.3a provides details of the TPR achieved by the different classifiers for the 10-fold cross-validation tests. From the observation, it can be noticed that the SMO classifier yielded the highest TPR of 0.884, 0.961, 0.896, 0.947, 0.961, and 0.971 on the different combinations of the Final Feature Sets such as the API, API+OH, API+DOSH, API+FH, OH+DOSH+FH, and API+OH+DOSH+FH, respectively, compared with the other classifiers. Similarly from Figure 5.3b, it can be observed that the SMO classifier also outperformed and attained maximum TPR of 0.906, 0.970, 0.914, 0.960, and 0.961 on all the various blends of the Final Feature Sets such as the APICAT, APICAT+OH, APICAT+DOSH, APICAT+FH, and OH +DOSH+FH, respectively. It can also be seen from Figure 5.3a and Figure 5.3b that for the same above different combinations of the Final Feature Sets, the second highest TPR was produced by the Simple Logistic classifier. Similarly, it can be seen that the RF classifier was also equipotent in obtaining TPR nearly equal to one for all those combinations of the Final Feature Sets. Furthermore,

Table 5.1: Performance evaluation of the classifiers in terms of accuracy (%) with k fold cross-validation test results (k=2 to 10) for different combinations of hybrid features.

cross validation Folds	Classifiers					
	SMO	Simple Logistic	Logistic	J48	Random Forest	Random Tree
(a) API features						
2	87.192	84.430	83.769	85.234	85.973	83.406
3	88.216	85.597	86.297	85.960	87.308	84.469
4	87.995	85.286	87.231	86.738	87.879	84.586
5	88.449	86.064	87.140	86.764	88.125	85.001
6	88.190	85.701	87.632	86.829	87.918	85.027
7	88.190	85.675	87.892	87.218	88.397	85.156
8	88.225	86.219	88.099	87.218	88.358	85.468
9	88.449	86.038	87.801	86.803	88.203	86.025
10	88.423	85.908	88.190	87.256	88.488	86.142
(b) API+OH features						
2	94.774	93.750	87.668	92.492	92.349	85.282
3	95.695	94.152	90.093	92.881	92.907	86.540
4	95.759	94.035	88.926	92.997	93.386	86.709
5	95.889	94.152	88.952	92.985	93.348	87.396
6	95.915	94.437	88.861	93.244	93.361	87.344
7	96.058	94.528	89.328	93.244	93.451	86.916
8	96.200	94.437	89.613	93.309	93.335	88.005
9	95.928	94.177	89.639	93.412	93.374	88.303
10	96.071	94.489	89.665	93.348	93.464	87.564
(c) API+DOSH features						
2	88.822	87.059	84.569	86.255	85.827	81.146
3	88.757	87.357	86.605	85.827	86.060	83.065
4	89.004	87.279	87.889	86.670	86.358	82.780
5	89.159	87.409	88.005	86.981	87.007	82.520
6	89.445	87.448	88.031	86.877	87.240	83.441
7	89.406	86.877	88.291	86.864	86.735	83.402
8	89.172	87.188	88.420	87.357	86.799	82.715
9	89.471	87.733	88.667	86.955	86.968	83.545
10	89.613	87.396	88.848	87.435	87.124	83.506
(d) API+FH features						
2	93.568	91.545	80.627	90.599	90.689	84.063
3	94.009	92.155	86.955	91.428	90.988	86.229
4	94.203	92.712	87.746	91.506	91.260	86.164
5	94.333	92.194	88.615	91.779	91.871	86.657
6	94.670	92.985	90.832	91.623	91.636	85.490
7	94.502	93.062	89.224	92.012	91.506	86.618
8	94.644	92.933	88.783	91.882	91.519	86.242
9	94.722	92.790	88.109	92.012	91.766	85.982
10	94.722	92.608	89.159	92.245	91.143	86.605
(e) OH+DOSH+FH features						
2	94.968	93.270	94.515	92.907	94.955	90.988
3	96.006	93.944	94.709	93.737	95.422	91.208
4	95.889	93.840	94.359	93.581	95.344	92.077
5	95.915	94.255	94.592	93.983	95.539	92.168
6	96.071	94.294	94.839	93.853	95.746	92.907
7	95.980	94.385	94.800	94.074	95.721	92.855
8	96.006	94.216	94.528	93.892	95.785	92.842
9	96.097	93.372	95.072	94.100	95.669	92.518
10	96.058	94.229	94.761	94.126	95.643	92.388
(f) API+OH+DOSH+FH features						
2	96.421	94.981	80.485	94.009	92.959	85.464
3	96.797	95.759	80.627	93.866	93.659	86.579
4	96.913	95.513	81.314	94.592	94.424	86.618
5	97.017	95.643	83.610	94.463	93.866	86.799
6	96.978	95.863	83.311	94.761	94.022	86.657
7	97.069	95.798	83.545	94.696	94.320	87.305
8	97.030	95.759	83.571	94.942	94.216	87.525
9	96.978	95.941	83.765	94.891	94.216	86.968
10	97.108	95.643	85.114	94.748	94.268	86.851

Table 5.2: Performance evaluation of the classifiers in terms of accuracy (%) with k fold cross-validation test results (k=2 to 10) for different combinations of hybrid features.

cross validation Folds	Classifiers					
	SMO	**Simple Logistic**	**Logistic**	**J48**	**Random Forest**	**Random Tree**
(a) APICAT features						
2	90.041	89.354	87.876	88.316	89.898	88.550
3	90.261	88.835	89.056	88.900	90.158	88.809
4	90.417	89.276	89.821	89.367	91.014	89.082
5	90.650	89.302	90.261	89.146	91.065	89.250
6	90.547	89.548	90.650	89.315	91.130	89.717
7	90.560	89.367	90.249	89.587	90.923	89.380
8	90.456	89.847	90.430	89.393	91.338	89.548
9	90.676	89.315	90.249	89.678	91.208	89.587
10	90.638	89.458	90.482	89.821	91.234	89.198
(b) APICAT+OH features						
2	96.097	94.891	85.114	92.829	94.100	88.161
3	96.628	95.396	92.803	93.555	95.020	89.963
4	96.758	95.254	91.260	93.944	95.280	89.263
5	96.926	95.785	91.467	93.957	95.098	89.613
6	96.784	95.733	91.312	94.385	95.072	89.808
7	96.952	95.837	91.545	94.126	95.254	90.702
8	96.939	95.591	91.766	93.983	95.357	90.313
9	96.952	95.708	91.740	94.139	95.163	89.847
10	97.017	95.811	91.843	94.216	95.604	90.313
(c) APICAT+DOSH features						
2	90.650	90.404	89.613	88.641	89.613	87.357
3	90.936	89.808	89.432	89.587	90.404	87.033
4	91.325	90.404	90.767	90.028	90.689	87.461
5	91.234	90.663	90.638	90.261	90.573	87.837
6	91.480	90.443	90.728	90.145	90.884	87.850
7	91.221	90.638	90.793	90.067	90.910	87.772
8	91.377	90.599	91.117	90.326	90.793	88.005
9	91.260	90.508	90.845	90.443	90.949	87.837
10	91.428	90.534	90.949	90.158	90.936	88.433
(d) APICAT+FH features						
2	95.332	93.361	87.357	91.714	92.894	88.692
3	95.513	93.775	90.443	92.699	93.464	89.808
4	95.643	93.672	91.260	93.036	94.255	89.458
5	95.863	93.866	91.416	92.686	94.255	89.613
6	95.798	94.009	90.962	92.972	94.164	90.547
7	95.876	93.983	91.390	93.153	93.970	90.274
8	95.811	94.087	91.027	93.088	94.346	90.236
9	95.798	94.139	91.325	93.010	94.190	90.547
10	96.019	94.074	91.117	92.972	94.087	90.845
(e) OH+DOSH+FH features						
2	94.968	93.270	94.515	92.907	94.955	90.988
3	96.006	93.944	94.709	93.737	95.422	91.208
4	95.889	93.840	94.359	93.581	95.344	92.077
5	95.915	94.255	94.592	93.983	95.539	92.168
6	96.071	94.294	94.839	93.853	95.746	92.907
7	95.980	94.385	94.800	94.074	95.721	92.855
8	96.006	94.216	94.528	93.892	95.785	92.842
9	96.097	93.372	95.072	94.100	95.669	92.518
10	96.058	94.229	94.761	94.126	95.643	92.388
(f) APICAT+OH+DOSH+FH features						
2	99.108	95.954	88.615	93.996	94.917	88.083
3	99.302	96.395	84.647	94.774	95.798	88.161
4	99.562	96.447	83.311	94.917	95.565	89.211
5	99.666	96.447	85.204	95.085	95.941	88.433
6	99.640	96.589	88.303	95.332	95.759	90.274
7	99.678	96.719	89.458	95.409	95.928	90.352
8	99.704	96.473	89.159	95.267	96.006	89.808
9	99.717	96.576	89.691	95.189	96.356	89.678
10	99.743	96.537	90.106	95.383	95.811	90.287

Table 5.3: Receiver Operating Characteristics area with k fold cross-validation Results (k=2 to 10) for different combination of hybrid features

cross validation Folds	Classifiers					
	SMO	Simple Logistic	Logistic	J48	Random Forest	Random Tree
(a) API features						
2	0.872	0.905	0.852	0.879	0.923	0.855
3	0.882	0.913	0.891	0.890	0.929	0.866
4	0.880	0.908	0.903	0.896	0.933	0.867
5	0.884	0.916	0.904	0.898	0.936	0.871
6	0.882	0.912	0.911	0.899	0.935	0.872
7	0.882	0.912	0.910	0.899	0.938	0.874
8	0.883	0.915	0.917	0.899	0.936	0.877
9	0.884	0.913	0.912	0.899	0.936	0.886
10	0.884	0.916	0.917	0.900	0.938	0.886
(b) API+OH features						
2	0.948	0.984	0.924	0.940	0.971	0.853
3	0.957	0.985	0.946	0.946	0.976	0.865
4	0.958	0.985	0.931	0.944	0.977	0.867
5	0.959	0.986	0.926	0.945	0.977	0.874
6	0.959	0.985	0.923	0.946	0.978	0.873
7	0.961	0.987	0.929	0.949	0.980	0.869
8	0.962	0.986	0.934	0.947	0.978	0.880
9	0.959	0.985	0.932	0.950	0.979	0.883
10	0.961	0.986	0.935	0.948	0.982	0.876
(c) API+DOSH features						
2	0.888	0.946	0.873	0.906	0.925	0.835
3	0.888	0.947	0.903	0.907	0.932	0.851
4	0.890	0.946	0.924	0.912	0.933	0.853
5	0.892	0.946	0.927	0.914	0.934	0.845
6	0.894	0.948	0.925	0.916	0.936	0.856
7	0.894	0.945	0.928	0.912	0.936	0.856
8	0.892	0.945	0.935	0.921	0.935	0.850
9	0.895	0.949	0.933	0.915	0.939	0.858
10	0.896	0.947	0.937	0.917	0.938	0.859
(d) API+FH features						
2	0.936	0.972	0.821	0.921	0.959	0.841
3	0.940	0.974	0.897	0.928	0.961	0.863
4	0.942	0.975	0.908	0.926	0.961	0.862
5	0.943	0.974	0.921	0.927	0.963	0.866
6	0.947	0.976	0.939	0.928	0.963	0.855
7	0.945	0.977	0.927	0.932	0.963	0.866
8	0.946	0.977	0.913	0.933	0.963	0.862
9	0.947	0.977	0.912	0.933	0.965	0.859
10	0.947	0.976	0.921	0.933	0.962	0.866
(e) OH+DOSH+FH features						
2	0.950	0.984	0.985	0.949	0.988	0.910
3	0.960	0.987	0.985	0.957	0.988	0.912
4	0.959	0.986	0.983	0.957	0.989	0.921
5	0.959	0.987	0.985	0.963	0.989	0.922
6	0.961	0.987	0.986	0.959	0.989	0.929
7	0.960	0.987	0.985	0.960	0.990	0.929
8	0.960	0.987	0.985	0.961	0.991	0.929
9	0.961	0.987	0.986	0.963	0.991	0.925
10	0.961	0.987	0.985	0.963	0.989	0.924
(f) API+OH+DOSH+FH features						
2	0.964	0.990	0.830	0.954	0.978	0.855
3	0.968	0.992	0.832	0.954	0.981	0.866
4	0.969	0.991	0.840	0.960	0.984	0.866
5	0.970	0.992	0.869	0.959	0.984	0.868
6	0.970	0.993	0.866	0.964	0.984	0.867
7	0.971	0.992	0.865	0.958	0.983	0.873
8	0.970	0.993	0.866	0.961	0.985	0.875
9	0.970	0.992	0.871	0.962	0.984	0.870
10	0.971	0.992	0.876	0.960	0.984	0.869

Table 5.4: Receiver Operating Characteristics area with k fold cross-validation Results (k=2 to 10) for different combination of hybrid features

cross validation Folds	Classifiers					
	SMO	Simple Logistic	Logistic	J48	Random Forest	Random Tree
(a) APICAT features						
2	0.900	0.949	0.904	0.924	0.953	0.915
3	0.903	0.944	0.922	0.927	0.954	0.921
4	0.904	0.947	0.932	0.934	0.958	0.921
5	0.907	0.948	0.938	0.931	0.959	0.926
6	0.905	0.949	0.942	0.932	0.959	0.930
7	0.906	0.949	0.938	0.934	0.958	0.926
8	0.905	0.950	0.940	0.936	0.958	0.927
9	0.907	0.946	0.939	0.938	0.959	0.930
10	0.906	0.949	0.941	0.936	0.959	0.923
(b) APICAT+OH features						
2	0.961	0.990	0.885	0.945	0.982	0.882
3	0.966	0.991	0.968	0.949	0.985	0.900
4	0.968	0.991	0.943	0.956	0.985	0.893
5	0.969	0.992	0.949	0.953	0.986	0.897
6	0.968	0.992	0.950	0.957	0.986	0.898
7	0.970	0.992	0.952	0.955	0.985	0.908
8	0.969	0.992	0.953	0.955	0.987	0.904
9	0.970	0.992	0.954	0.957	0.986	0.899
10	0.970	0.992	0.957	0.958	0.987	0.904
(c) APICAT+DOSH features						
2	0.907	0.968	0.936	0.935	0.957	0.907
3	0.909	0.966	0.935	0.940	0.961	0.901
4	0.913	0.968	0.954	0.950	0.963	0.905
5	0.912	0.970	0.952	0.951	0.964	0.910
6	0.915	0.970	0.953	0.951	0.965	0.911
7	0.912	0.970	0.955	0.952	0.965	0.908
8	0.914	0.970	0.957	0.950	0.965	0.912
9	0.913	0.970	0.955	0.954	0.965	0.911
10	0.914	0.970	0.958	0.950	0.965	0.915
(d) APICAT+FH features						
2	0.953	0.983	0.909	0.956	0.971	0.887
3	0.955	0.985	0.953	0.958	0.971	0.897
4	0.956	0.986	0.962	0.962	0.975	0.895
5	0.959	0.986	0.957	0.958	0.977	0.897
6	0.958	0.987	0.953	0.960	0.974	0.907
7	0.959	0.986	0.956	0.962	0.975	0.904
8	0.958	0.987	0.957	0.960	0.975	0.904
9	0.958	0.987	0.958	0.959	0.975	0.906
10	0.960	0.986	0.956	0.962	0.977	0.910
(e) OH+DOSH+FH features						
2	0.950	0.984	0.985	0.949	0.988	0.910
3	0.960	0.987	0.985	0.957	0.988	0.912
4	0.959	0.986	0.983	0.957	0.989	0.921
5	0.959	0.987	0.985	0.963	0.989	0.922
6	0.961	0.987	0.986	0.959	0.989	0.929
7	0.960	0.987	0.985	0.960	0.990	0.929
8	0.960	0.987	0.985	0.961	0.991	0.929
9	0.961	0.987	0.986	0.963	0.991	0.925
10	0.961	0.987	0.985	0.963	0.989	0.924
(f) APICAT+OH+DOSH+FH features						
2	1	0.993	0.943	0.953	0.988	0.881
3	1	0.995	0.877	0.961	0.990	0.882
4	1	0.994	0.857	0.959	0.990	0.892
5	1	0.995	0.880	0.966	0.991	0.884
6	1	0.995	0.921	0.965	0.992	0.903
7	1	0.995	0.942	0.966	0.991	0.904
8	1	0.995	0.935	0.963	0.991	0.898
9	1	0.995	0.939	0.965	0.992	0.897
10	1	0.995	0.946	0.965	0.991	0.903

the malware detection performance of all the other classifiers for the aforementioned combinations is depicted in Figure 5.3a and Figure 5.3b. The lowest TPR was shown by the Random Tree classifier on all these different blends of the Final Feature Sets.

In particular, the proposed HFMDS was successful in achieving the highest TPR of 0.997 for the Final Feature Set, which included the APICAT+OH+DOSH+FH features by the SMO classifier (see Figure 5.3b). Apparently, the Simple Logistic classifier also achieved a nearly equivalent highest TPR of 0.965, which justifies that the HFMDS is efficient in identifying unknown malware. The RF classifier also achieved a relatively equivalent TPR of 0.958, which is less when compared with the SMO and Simple Logistic classifiers. Other classifiers such as the Logistic, J48, and the Random Tree attained less TPR of 0.901, 0.954, and 0.903, respectively.

The effectiveness of the classifiers was also measured using other evaluation metrics such as Precision, Recall, and F-Measure. Where, Precision is a measure of exactness and Recall is a measure of completeness. Usually, if the value of these performance metrics is closer to one, it signifies as ideal MDS. However, this is difficult to attain in reality.

The obtained experimental results shown in Figure 5.3c, Figure 5.4a, and Figure 5.4c demonstrate that the SMO classifier was able to attain high Precision, Recall, and F-Measure of 0.971 for the combination of API+OH+DOSH+FH features. Similarly, it can be noticed from Figure 5.3d, Figure 5.4b, and Figure 5.4d that the same SMO classifier achieved high Precision, Recall, and F-Measure of 0.997 for the combination of APICAT+OH+DOSH+FH features. The recorded values for classifiers which are approximately equal to one justify that the HFMDS is superior in the detection of malware. Based on the results depicted in Figure 5.3c, Figure 5.3d, Figure 5.4a, Figure 5.4b, Figure 5.4c, and Figure 5.4d it can be seen that even the Simple Logistic classifier was successful in attaining Precision, Recall, and F-Measure values almost equal to one, i.e., 0.957 (combination of API+OH+DOSH+FH features) and 0.966 (combination of APICAT+OH+DOSH+FH features), which is another evidence that the HFMDS is proficient in identifying unknown malware. The other classifiers such as the Logistic, J48, RF, and Random Tree achieved low Precision, Recall, and F-Measure values.

The ROC curve is used in the present work to compare the classification capabil-

ity of the different classifiers. The higher ROC value, i.e., closer to 1 signifies that the classifier categorizes all malware files as malware and benign files as benign with zero FPR. The performance of each classifier tabulated in Table 5.3 and Table 5.4 is appraised separately with different hybrid features set and each sub-table denotes the 'k' fold cross-validation experimental results, where, k = 2 to 10. From the experimental analysis, it can be observed that the SMO classifier achieved the highest ROC of 1 for all the 'k' fold cross-validation experiments when the classifier provided hybrid features' sets comprised of APICAT+OH+DOS+FH, while the Random Tree reported the lowest ROC value. Furthermore, it can also be observed from Table 5.3 and Table 5.4, that the Simple Logistic classifier produced a reasonably higher value of ROC for all the hybrid feature sets compared with the other classifiers.

5.4 AN EMPIRICAL STUDY TO ESTIMATE THE STABILITY OF RANDOM FOREST CLASSIFIER ON THE HYBRID FEATURES RECOMMENDED BY FILTER BASED FEATURE SELECTION TECHNIQUE

The main aim of the present work is to measure the detection ability of the proposed HFMDS with the RF classifier. The reason for selecting the RF classifier is that it is widely used, efficient, and can operate over large datasets (Cutler et al., 2012; Alam and Vuong, 2013). However, only few works show how many DTs should be employed to compose an RF classifier (Oshiro et al., 2012; Ajay Kumara and Jaidhar, 2017). Therefore, in the present work investigations were made by conducting various set of experiments to know what number of DTs does the RF classifier needs to achieve consistent accuracy for hybrid features.

The proposed HFMDS utilized the DH, SH, FH, OH, and IF extracted from the PE files as static features, whereas the API+CAT gathered from the behavioural report generated by the Cuckoo Sandbox was treated as dynamic features. To acquire a set of imperative features from the original feature set, FSTs such as Chi-Square$\tilde{\chi}^2$ (Belaoued and Mazouzi, 2015), Gain-Ratio (Moskovitch et al., 2008), Max-Relevance and Min-Redundancy (mRMR) (Peng et al., 2005), and Max-Relevance (Max-Rel) (Sakar et al., 2012) were employed. A relative analysis of the chosen FSTs in recognizing the best one was carried out using the RF classifier.

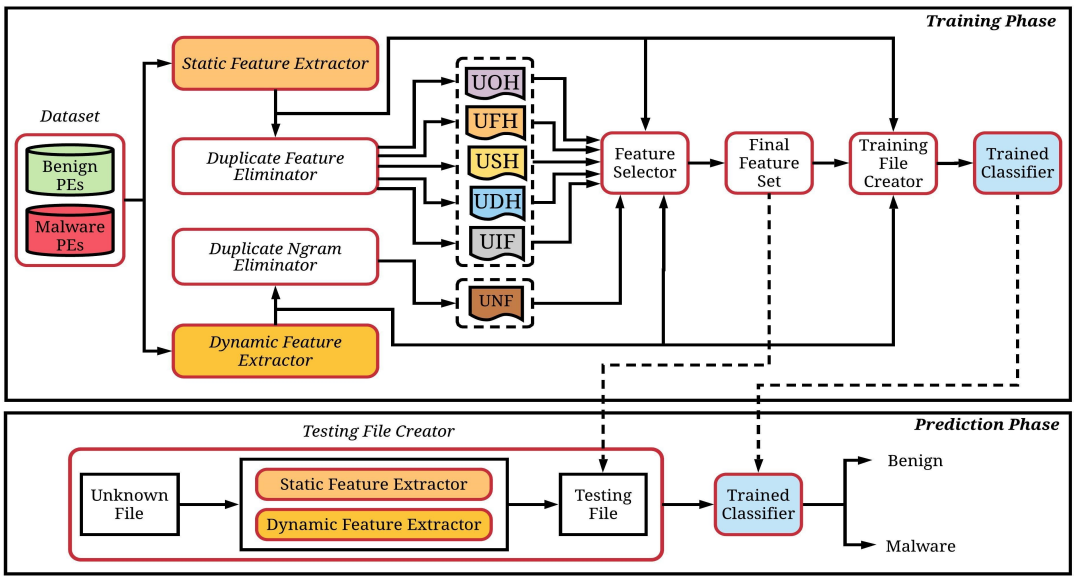


Figure 5.5: An architecture overview of the proposed HFMDS

5.4.1 Overview of the Proposed HFMDS

The proposed HFMDS was implemented directly on to the PE files and was categorized into two phases such as the Training phase and the Prediction phase as illustrated in Figure 5.5. The detailed procedures of the training phase and prediction phase are described in the following sections.

5.4.2 Training Phase

A training phase is a basic building block of the HFMDS and involves static and dynamic features extraction, followed by selection of prominent features in order to train the HFMDS. An adequate number of benign and malware PE files are processed to derive the relevant static and dynamic features from each PE file. The HFMDS essentially adopts the Static Feature Extractor to acquire static features, namely, DH, SH, FH, OH, and IF, whereas the Dynamic Feature Extractor component of the HFMDS is used to gather the execution time behavioural report and then, to derive the dynamic features. Since the extracted original hybrid feature set dimensionality is quite large, the feature selector is utilized to choose the prominent features and these are considered to build a training file.

5.4.2.1 Dynamic Feature Extractor

In order to evade detection by the anti-malware solution, sophisticated malware generate a large number of runtime features. All the monitored runtime characteristics are

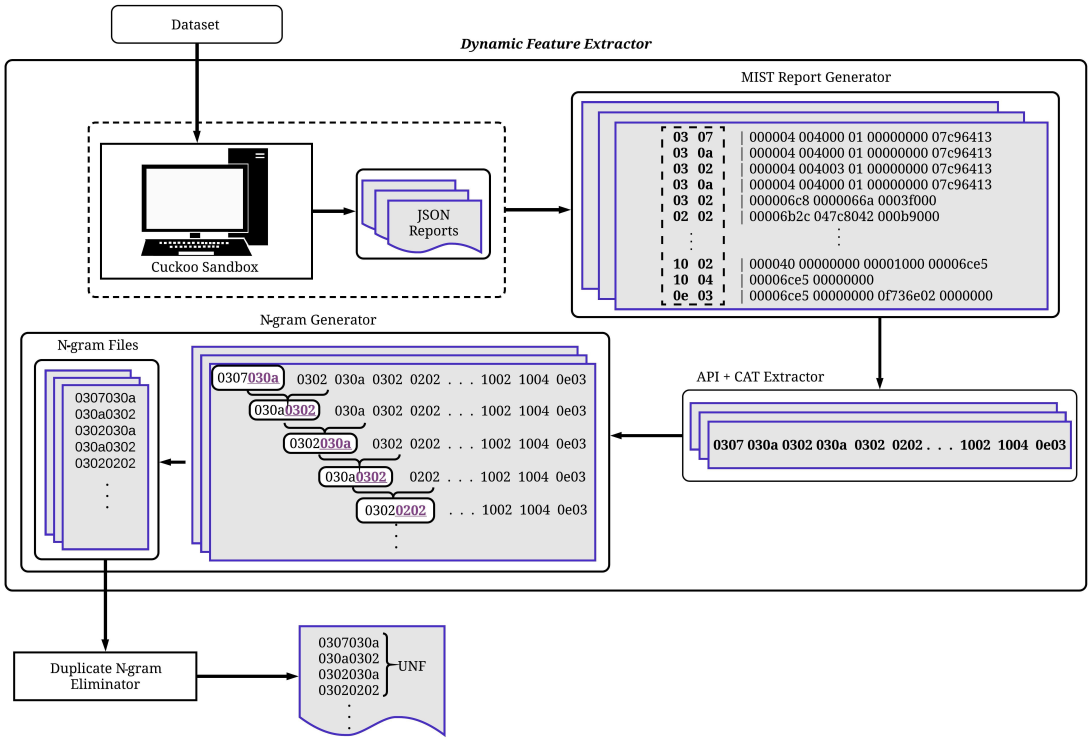


Figure 5.6: Dynamic Feature Extractor

computationally expensive for consideration. Therefore, recognizing the most prominent runtime features is essential to ascertain the potential of the malware. Many of the anti-malware detection techniques examine only the API calls from the behavioural pattern of the executing PE files to discriminate whether the source PE file is malware or benign (Faruki et al., 2012; Salehi et al., 2014; Kawaguchi and Omote, 2015; Salehi et al., 2017). The proposed HFMDS focused on acquiring a combination of API calls and their CAT from the behavioural report to generate dynamic features in the form of N-grams. The Dynamic Feature Extractor executes the source PE files one at a time onto the Cuckoo Sandbox to obtain the runtime behavioural report needed to derive the dynamic features (Tsyganok et al., 2012; Qiao et al., 2014; Sethi et al., 2018). Its subcomponents as shown in Figure 5.6 are: (1) Cuckoo Sandbox, (2) MIST Report Generator, (3) API + CAT Extractor, and (4) N-gram Generator.

Figure 5.6 depicts the operation carried out by each component of the Dynamic Feature Extractor. The description related to the subcomponents such as the Cuckoo Sandbox, MIST Report Generator, and API + CAT Extractor has been discussed in Section 4.3.1.1, Section 4.3.1.1, and Section 4.4.1.2, respectively.

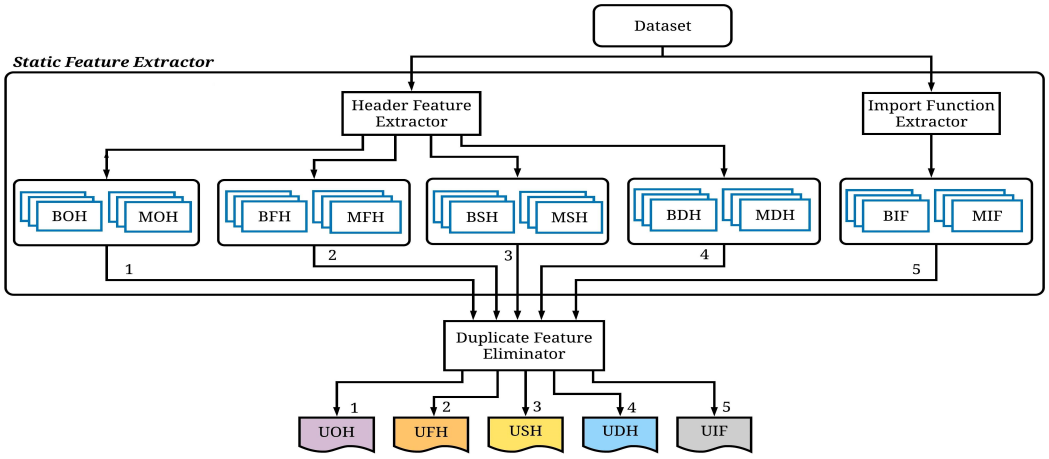


Figure 5.7: Static Feature Extractor

N-gram Generator

In the present work, the extracted API + CAT from the MIST report is represented in overlapping substrings based on the sliding window approach to generate the N-grams as shown in Figure 5.6. As previous works have demonstrated that N-grams of size four bytes show encouraging results (Darshan et al., 2016; Ajay Kumara and Jaidhar, 2017), the values in the category and operational field of each MIST report were extracted to prepare N-grams (Kolter and Maloof, 2006; Masud et al., 2008) of size four bytes in the present work. For instance, for byte sequence 0307 030a 030b 0306 0201, the corresponding N-grams would be 0307030a, 030a030b, 030b0306, and 03060201.

Duplicate N-gram Eliminator

The N-gram-based technique generates a large number of N-grams that comprise of duplicate N-grams. Therefore, all the generated N-grams cannot be used to prepare a training file, since the redundant N-grams(features) consume storage space, high processing time, and degrade performance by achieving high misclassification rate. To overcome these issues, it is necessary to eliminate the duplicates and construct a distinct N-gram features' set. Therefore, a union operation was applied to generate a Unique N-gram Features (UNF) set from the Benign N-gram Files (BNFs) $[BNF_1, BNF_2, BNF_3, \ldots, BNF_n]$ and Malware N-gram Files (MNFs) $[MNF_1, MNF_2, MNF_3, \ldots, MNF_m]$ as represented in Eq. 5.1.

$$UNF = \{BNF_1 \cup BNF_2 \cup BNF_3 \cup \ldots \cup BNF_n \cup MNF_1 \cup MNF_2 \cup MNF_3 \cup \ldots \cup MNF_m\} \quad (5.1)$$

Table 5.5: Datatype specifications defined for the attributes to build a training file

Features	Static Features					Dynamic Features
	OH	DH	FH	SH	IF	N-grams
<datatype>	numeric	numeric	numeric	Binary{0,1}	numeric	Binary{0,1}

5.4.2.2 Static Feature Extractor

The Static Feature Extractor shown in Figure 5.7 extracts certain static features from the PE files to create an integrated static feature set that includes header related information and IF calls. Its subcomponents are: 1) Header Feature Extractor, and 2) Import Function Extractor.

Header Feature Extractor

The Header Feature Extractor extracts PE Header related information such as DH, FH, SH, and OH from all the PE files present in the dataset (Section 5.4.4.2). The Python module called pefile (Ero, 2017) is used to extract the OH, FH, and DH information and an open source tool called the ClaMP (urwithajit9, 2016) is used to acquire SH information. The Header Feature Extractor maintains a separate output directory and stores the output in the appropriate directory. For instance, the extracted OH related information is stored in the OH directory.

Import Function Extractor

The DLL function used by the malware provides good insight into its functionality. Most malware require the use of system-level functionalities to perform their intended task. These functionalities are available in the form of DLL import functions. Hence, the access of system-level functionality must be done through appropriate DLL functions (Bai et al., 2014). However, it is difficult for malware authors to avoid the DLL import functions to carry out malicious activities. Therefore, IF is used as static features for malware detection in the proposed HFMDS. The IF was extracted using the Python module called PEframe (Amato, 2016), and the extracted features were stored in the specific output file and in a specific output directory as depicted in Figure 5.7.

Duplicate Feature Eliminator

It is used to prepare a unique feature set separately for each type of static features by considering both the benign and malware files of the relevant type as shown in Figure

5.7. This step is essential since the extracted original features set may include redundant features leading to substantial memory consumption, performance degradation of the classifier, and wrong prediction of the unknown PE file. To obtain unique features necessary for the Feature Selector, a union operation was applied on each type of acquired static features as shown in Eq. 5.2,

$$\begin{aligned} UOH &= \{BOH_1 \cup ... \cup BOH_n \cup MOH_1 \cup ... \cup MOH_m\} \\ UFH &= \{BFH_1 \cup ... \cup BFH_n \cup MFH_1 \cup ... \cup MFH_m\} \\ USH &= \{BSH_1 \cup ... \cup BSH_n \cup MSH_1 \cup ... \cup MSH_m\} \\ UDH &= \{BDH_1 \cup ... \cup BDH_n \cup MDH_1 \cup ... \cup MDH_m\} \\ UIF &= \{BIF_1 \cup ... \cup BIF_n \cup MIF_1 \cup ... \cup MIF_m\} \end{aligned} \tag{5.2}$$

Where, U specifies unique, B indicates benign, and M denotes malware. As shown in Figure 5.7, UOH, UFH, USH, UDH, and UIF represent the unique feature sets for OH, FH, SH, DH, and IF features, respectively.

The generated unique static features sets (UOH, UFH, USH, UDH, and UIF) and the UNF set are utilized to prepare either an ARFF file or a Comma Separated Values (CSV) file as per the requirement of the Feature Selector. The ARFF or CSV file is a file that characterizes a list of instances sharing a set of attributes (Ajay Kumara and Jaidhar, 2017). To obtain better malware detection rate, the attributes' data type have been defined as numeric for the selected OH, FH, DH, and IF static features and binary values 0, 1 for SH and N-gram features as shown in Table 5.5. The specific reason for this assignment is that OH(30), FH(07), and DH(19) consist of a predefined set of features, and it makes no sense to substitute the binary values as 0 or 1 to show whether those features exist or not. Therefore, the correspondent values obtained for the selected features of UOH, UFH, and UDH were substituted during the extraction of the features. For UIF features, their frequency count, if present or '0' otherwise was substituted, therefore, the attribute data type of the UIF features was numeric.

5.4.2.3 Feature Selector

The dimensionality of the original feature set in numerous applications present a great challenge. In most cases, some irrelevant features exist that do not assist in the ac-

curacy of the classification task and degrades the performance of the classifier (Huda et al., 2016). The Feature Selector was used in the present work to recognize the imperative and noisy features from the original feature set. Although the filter-based FST recommends best relevant features based on highest computed score, there is no guarantee that the features suggested by a single FST is more informative and makes a better prediction for malware detection. Therefore, in the proposed HFMDS, four popular filter-based FSTs such as Chi-Square$\tilde{\chi}^2$ (Belaoued and Mazouzi, 2015; Ajay Kumara and Jaidhar, 2017), Gain-Ratio (Moskovitch et al., 2008), mRMR (Peng et al., 2005), and Max-Rel (Huda et al., 2018) were used to identify the crucial features. The four FSTs computed separate scores for each of the features that belonged to OH, FH, SH, DH, IF, and N-grams. Based on the highest feature score, the topmost features were acquired resulting in a list of top scored features from each FST. The topmost scored features advised by each FST was acquired individually with the purpose of identifying the better one. The description for the Chi-Square$\tilde{\chi}^2$ FST is provided in Section 4.4.2. The explanation concerning the other FSTs such as Gain-Ratio, mRMR, and Max-Rel is described below.

Gain-Ratio

Gain-Ratio (Moskovitch et al., 2008) was introduced to compensate for the bias of IG and it was calculated using Eq. 5.5. It estimates the foreseen depletion of entropy due to portioning the examples based on the selected feature. If E(S) is the entropy, it assesses the irrelevant features in a collection of features, and then quantifies the significance of a feature in classifying the training data. The entropy was calculated as per Eq. 5.4. It represents the entropy of a set of items S, considering C subsets of S, given by S_c. IG evaluates the expected reduction in entropy due to portioning the examples based on attribute A, where, 'V' is the set of values of 'A', as shown in Eq. 5.3.

$$IG(S, A) = E(S) - \sum_{v \in V(A)} \frac{\mid S_v \mid}{S} . E(S_v) \tag{5.3}$$

$$E(S) = \sum_{c \in C} -\frac{\mid S_c \mid}{S} . \log_2 \frac{\mid S_c \mid}{S} . \tag{5.4}$$

IG estimates features with a high variety of values by comparing with those only

few. Gain-Ratio resolves this issue by including how the feature splits the data. Eq. 5.5 and Eq. 5.6 are used to compute the Gain-Ratio score.

$$Gain\text{-}Ratio(S, A) = \frac{IG(S, A)}{SI(S, A)} \tag{5.5}$$

$$SI(S, A) = -\sum_{i=1}^{d} \frac{\mid S_i \mid}{\mid S \mid} . log_2 \frac{\mid S_i \mid}{\mid S \mid}. \tag{5.6}$$

Where, S_i are d subsets of examples obtained from portioning S by the d-valued feature A.

Max-Relevance and Min-Redundancy

Max-Rel FST (Peng et al., 2005; Sakar et al., 2012) employs MI (Gulgezen et al., 2009) to select the most predominant features and provides more information about the class variable. If S is a set of features, $\{F_i \mid F_i \in S : i = 1, 2, 3, ...\}$ and class variable is c, Max-Rel as defined in (Peng et al., 2005) as shown in Eq. 5.7.

$$max\ T(S, c), \quad T = \frac{1}{\mid S \mid} \sum_{F_i \in S} I(F_i; c). \tag{5.7}$$

Where, $I(F_i; c)$ is MI between feature F_i, and class c. It is probable that Max-Rel chooses features that are highly relevant to the class. When two features are highly dependent on each other, the corresponding class discriminative power does not cause much affect, if one of them is removed. Thus, to avoid redundancy in Max-Rel, the following redundancy function was incorporated to select mutually exclusive features (Gulgezen et al., 2009).

$$max\ L(S), \quad L = \frac{1}{\mid S \mid^2} \sum_{F_i, F_j \in S} I(F_i; F_j). \tag{5.8}$$

Where, $I(F_i; F_j)$ is the Mutual Information between the features: F_i, F_j. Thus, mRMR (Sakar et al., 2012) is defined as per Eq. 5.8. The T and L is combined as

$\Phi(T, L)$ and is defined in the simplest form as per Eq. 5.9.

$$max\ \Phi(T, L), \quad \Phi = T - L. \tag{5.9}$$

5.4.2.4 Final Feature Set

The Final Feature Set comprises a blend of crucial dynamic and static features (hybrid) advised by the FST. These features are used as foremost features since there is no further elimination of features.

5.4.2.5 Training File Creator

The Training File Creator constructs a training file necessary to train the classifier. It parses the training dataset that consists of N-gram files, OH files, SH files, FH files, DH files, and IF files corresponding to the benign and malware PE files with the features of the Final Feature Set to prepare a training file.

5.4.3 Prediction Phase

In the prediction phase, the Testing File Creator is utilized to prepare testing files essential to assess the classification ability of the trained classifier. It utilizes the Final Feature Set and the outcome of the Dynamic Feature Extractor and the Static Feature Extractor to obtain a testing file. The created testing file is provided as input to the trained classifier, which predicts the given test input file as either malware or benign.

5.4.4 Experimental Results and Discussion

In the present work, the proposed HFMDS was evaluated and validated with the intention of demonstrating the ability of the hybrid features set recommended independently by four FSTs in a direction to find the best one. Further, the performance of the RF classifier was investigated by considering a variable number of DTs in detecting unknown Windows malware. With this, the following research goals are presented as four questions:

- Does the hybrid feature set, amalgamated as a combination of static and dynamic features, boost the efficiency of the Malware Detection System in detecting unknown malware?

- Is it possible to classify a test file(s) accurately as benign or malware with less number of features as recommended by the FSTs?
- Which Final Feature Set features recommended by the FST significantly increases the detection accuracy of the classifier?
- Investigate the prediction capability (accuracy) of the RF classifier for a variable number of DTs to know how many DTs should be employed to compose an RF classifier?

To obtain the answers to the above questions, a wide set of experiments were carried out.

5.4.4.1 Experimental Set-up

In the present work, an RF classifier available in WEKA (Frank et al., 2009) was used as an ensemble-based technique for classification tasks throughout the experiments. It runs efficiently on larger datasets and estimates crucial features, which are essential for better predictability (Breiman, 2001; Shabtai et al., 2009; Oshiro et al., 2012; Cutler et al., 2012; Alam and Vuong, 2013; Ajay Kumara and Jaidhar, 2017). Every DT in the RF classifier attains the output by each individual tree on the basis of most frequent values in the class of datasets. The RF classifier amalgamates the DTs with a bagging (Breiman, 2001) method, where every DT is constructed individually by running bootstrap sample features of the input training set. However, the bootstrap features are selected by recursive arbitrary sub-sampling with the replacement of the original training set. Assume a training set 'T_a' with 'f' features, 'n' instances, then 'T_b' a bootstrap training set defined to be a training set sampled from 'T_a' with replacement that contains 'r' random features where, $(r \geq f)$ with 'n' instances. Thus, the RF classifier comprises a collection of tree structured classifiers $\{h_i(x, T_i)\}$, where, $i = 1, 2, ..., N$, and T_i are independent and uniformly distributed random features, and each tree casts a unit vote for the most popular class as input x. Further, each node of the DT in the RF classifier is made of arbitrarily selected features that help to reduce the correlation between the features and is thus less susceptible to irrelevant data. The number of arbitrary features 'r' determined by each decision node in a tree determines the misclassification rate of the classifier when performing the prediction. The misclassification rate of the RF classifier relies upon the interdependence between two trees and the prediction ca-

pability of each tree. If the selected arbitrary features 'r' are less, then the correlation between the tree and the potency of each tree is also less. On the other hand, increasing the random features 'r' enhances the correlation between the trees and the proficiency of each tree. The Out-of-Bag Error (OOBE) rate manifests how well the RF classifier exhibits its predictive performance on the dataset. In the RF classifier, about one-third of the input dataset is left out to build the N^{th} tree from the bootstrap samples for each individual tree. The one-third sample is utilized to test the N^{th} tree and the outcome of the incorrect classification is averaged over all the trees.

The main reason to select the RF classifier is to explore its efficiency in the classification of known and unknown malware by attaining hybrid features recommended by different filter-based FSTs. In Peng et al. (2005), experiments were carried out to measure the proficiency of the mRMR FST by using three different classifiers such as Naive Bayes, Support Vector Machine, and Linear Discriminate Analysis. The authors claimed that the mRMR FST is capable of identifying the best features for the classification task. However, the mRMR FST performance is required to be measured with tree-based classifiers. Thus in the present work, the efficiency of the mRMR (Sakar et al., 2012) was investigated and compared with other FSTs such as Chi-Square$\tilde{\chi}^2$, Gain-Ratio, and Max-Rel using the RF classifier. In addition, the following observations were made to evaluate the efficiency such as (i) number of DTs required for the RF classifier to achieve high accuracy, (ii) effect of more number of DTs, and (iii) examining the stability of the RF classifier accuracy. Therefore, the performance of the RF classifier was tested by considering a range of DTs from 5 up to 5120 and with different features' combination in additions of 25 to 100 (i.e., 25, 50, 75, and 100). The evaluation was conducted using the 10 fold cross-validation tests for each experiment and all the experiments were conducted on the host system as described in Section 2.7.

5.4.4.2 Data Collection

The proposed HFMDS is appropriate for the Windows PE files, as today Windows malware monopolize malicious codes (VirusTotal, 2004b; Amin, 2016). The evaluation was accomplished by utilizing two datasets, namely, Dataset-1 consisting of 200 malware and 200 benign PE files and Dataset-2 consisting of 3856 malware and 3856 benign PE files. All the collected files, existing in either Dataset-1 or Dataset-2, were

non-identical, unpacked, and in the Windows PE file format. The malware samples were downloaded from a public source (VirusShare, 2011; CNET, 1996) and the collected set included seven distinct kinds of malware such as Backdoor, Exploit, Flooder, Rootkits, Trojan, Virus, and Worm. Majority of the benign PE files were collected from newly installed Windows virtual machines that included Windows-XP, Windows-7, and Windows-8 operating system files, and some benign PE files were gathered from free online software archives (CNET, 1996).

5.4.4.3 Result Analysis

Initially, the experiments were performed by involving Dataset-2. The hybrid features derived from Dataset-2 formed the original features set and consisted of 3096 hybrid features. The dynamic features and static features extracted formed separate original features set. The constructed original features set subject to static, dynamic, and hybrid features were used to create three different training files to train the classifier, and the obtained results are shown in Table 5.6. The evaluation was carried out by choosing different classifiers, namely, J48, RF, Random Tree, Logistic, SMO, and Simple Logistic to determine which classifier could provide better detection accuracy. It was found that hybrid features can achieve better detection accuracy than by merely using static or dynamic features alone. Another observation was that the RF classifier achieved maximum accuracy than the other five chosen classifiers. Thereby, the proposed approach provided empirical evidence to be used as a substitute to either static features-based (classical method) or dynamic features-based techniques, and this led to further investigate the hybrid features. Although better results were observed when all the hybrid features in the original features set were involved, the highest accuracy of 93.926% accomplished by the RF classifier was not appreciable.

The reason in achieving less accuracy was due to the existence of insignificant features, which mislead the predictability of the classifier. In this regard, to select the best imperative features from the original feature set, the Feature Selector was employed in the proposed approach, and four different FSTs were chosen to demonstrate their effectiveness with the empirical results. Each FST computed the score for every feature individually. Based on the highest feature score, the topmost 25, 50, 75, and 100 features recommended by each FST were selected separately to obtain four individual

Table 5.6: Relative analysis of the hybrid features with static features and dynamic features involving all features in their respective original features set

Classifiers	Accuracy (%)		
	Static Features alone	Dynamic Features alone	Hybrid Features
SMO	89.872	89.989	92.993
Simple Logistic	91.182	89.354	92.837
Logistic	91.636	87.746	91.882
J48	90.876	89.652	93.317
Random Forest	92.043	91.065	93.926
Random Tree	90.800	89.600	91.286

Final Feature Sets to verify which Final Feature Set achieved more accuracy. For the topmost 25, 50, 75, and 100 features suggested by each FST, the RF classifier yielded maximum accuracy compared with the other classifiers as shown in the last column of Table 5.7. Due to this, the detection accuracy of the RF classifier was further examined by considering different number of DTs with the varied topmost features' set size. Accordingly, two sets of experiments (Experiment I and Experiment II) were conducted.

5.4.4.4 Experiment I

The first set of experiments was conducted using Dataset-1, and each PE file of Dataset-1 was executed one at a time onto the Cuckoo Sandbox to capture the runtime behavioural reports. The obtained reports were in the JSON file format, and these were converted into MIST format to extract a sequence of API+CAT. The gathered sequence of API+CAT was used to generate N-grams of length 4 bytes, and each N-gram was treated as an individual dynamic feature. After removal of the redundant N-grams, 4948 distinct N-grams were obtained. Static features, namely, OH, FH, DH, SH, and IF were also extracted from the same set of PE files. In case of static features, all the 30 OHs, seven FHs, and 19 DHs were considered as distinct features and their correspondent values were obtained during the preparation of the training file. The reason to consider feature correspondent value is that all the benign and malware PE files consist of the same features as per the PE Header format. In contrast, after the removal of duplicates from each original features set of SH and IF, 10 features were attained from each. Since the proposed HFMDS uses both static and dynamic features, the size of the

original feature set comprised of 5024 features, which required more time to prepare a training file. The analyses of these experiments are discussed in Section 5.4.5.1.

5.4.4.5 Experiment II

The second set of experiments was performed on Dataset-2. Steps similar to Experiment I were followed in Experiment II to extract static features. For Dataset-2 PE files also, all the 30 OHs, 19 DHs, and seven FHs were used. Further, the SH features consisting of Section Names as nine features and 14 IF features were retained after the removal of irrelevant features from each of the respective original feature sets. The execution of all 7712 PE files onto the Cuckoo Sandbox in order to capture the execution time behavioural report is a tedious task and takes a long time to compute. Therefore, the MIST reports available in the public source (Konrad, 2015) were used to extract the API+CAT as dynamic features (N-grams). After removal of repetitive N-gram features, 3017 distinct N-grams were obtained. The gathered original hybrid feature set of size 3096 was quite large and considering all of them directly for classification would certainly degrade the performance of the classifier. The analyses of these experiments are explained in Section 5.4.5.2.

5.4.5 Performance Analysis of the RF Classifier

Experiments were conducted using the RF classifier by differing the number of DTs as per Eq. 5.10, where, $x = 1, 2, 3, ..., 11$. In each experiment, accuracy was chosen as a performance metric to analyze the results and it was calculated as per Eq. 2.1.

$$Number\ of\ DTs = 5 * 2^{x-1} \tag{5.10}$$

5.4.5.1 Analysis-I

The accuracy accomplished by the RF classifier for Dataset-1 and Dataset-2 is illustrated in Table 5.8 and Table 5.9, respectively. The accuracy accomplished by the RF classifier for the topmost 25 features recommended by each of the four chosen FSTs with variable number of trees has been presented in Table 5.8. It can be noticed that for the top 25 features recommended by the Chi-Square$\tilde{\chi}^2$ FST, the RF classifier achieved lowest accuracy of 98.167% with five DTs and the accuracy increased to 98.429% when five more DTs were added.

Table 5.7: Relative analysis of accuracy(%) achieved by the different classifiers for hybrid feature set derived from Dataset-2 where Random Forest classifier is evaluated with 10 Decision Trees

Feature Selection Technique	**Topmost Features Count**	**Classifiers**					
		SMO	Simple Logistic	Logistic	J48	Random Tree	Random Forest
No-FST	*	92.993	92.837	91.882	93.317	91.286	93.926
Chi-Square$\tilde{\chi}^2$	25	93.983	94.242	93.646	96.343	95.695	97.380
Gain-Ratio	25	91.480	92.284	92.271	92.634	92.777	92.933
mRMR	25	93.931	94.372	93.646	96.628	95.578	96.836
Max-Rel	25	94.761	95.215	95.202	95.500	94.904	97.536
Chi-Square$\tilde{\chi}^2$	50	95.176	95.513	95.435	96.978	95.137	97.756
Gain-Ratio	50	91.869	92.453	92.440	92.712	92.933	93.244
mRMR	50	95.189	95.409	95.422	96.875	95.111	97.173
Max-Rel	50	95.617	95.876	95.863	95.643	94.722	98.016
Chi-Square$\tilde{\chi}^2$	75	95.228	95.422	95.565	96.939	94.904	98.029
Gain-Ratio	75	93.036	94.074	93.892	95.110	95.383	95.928
mRMR	75	95.306	95.682	95.565	96.978	95.150	97.432
Max-Rel	75	95.448	95.941	95.915	95.811	94.346	97.821
Chi-Square$\tilde{\chi}^2$	100	95.254	95.591	95.669	96.991	94.709	97.743
Gain-Ratio	100	93.023	93.970	93.814	95.135	95.461	96.113
mRMR	100	95.228	95.513	95.643	96.939	94.930	97.264
Max-Rel	100	95.474	95.915	95.798	95.785	94.242	97.704

* All the features of original features set

Table 5.8: Detection accuracy(%) achieved by the Random Forest classifier for hybrid feature set derived from Dataset-1 for different number of Decision Trees

Feature Selection Technique	**Topmost Features Count**	**Number of Decision Trees**										
		5	10	20	40	80	160	320	640	1280	2560	5120
Chi-Square$\tilde{\chi}^2$	25	98.167	98.429	98.691	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.952
Gain-Ratio	25	98.429	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
mRMR	25	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
Max-Rel	25	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
Chi-Square$\tilde{\chi}^2$	50	98.167	98.167	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
Gain-Ratio	50	98.691	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
mRMR	50	98.952	98.691	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
Max-Rel	50	98.429	98.691	98.691	98.952	98.952	98.952	98.952	98.691	98.952	98.952	98.952
Chi-Square$\tilde{\chi}^2$	75	98.691	99.214	**99.214**	98.691	98.429	98.429	98.429	98.429	98.429	98.429	98.429
Gain-Ratio	75	98.691	98.691	98.691	98.952	98.952	98.691	98.952	98.952	98.952	98.952	98.952
mRMR	75	98.691	98.691	98.952	98.952	98.952	98.952	98.952	98.952	98.691	98.691	98.691
Max-Rel	75	98.429	98.429	98.167	98.429	98.691	98.691	98.429	98.691	98.691	98.691	98.691
Chi-Square$\tilde{\chi}^2$	100	98.167	98.429	98.429	98.691	98.691	98.691	98.429	98.429	98.429	98.429	98.429
Gain-Ratio	100	98.167	98.429	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952	98.952
mRMR	100	98.691	98.429	98.691	98.952	98.691	98.691	98.691	98.691	98.691	98.691	98.691
Max-Rel	100	98.691	98.691	98.691	98.691	98.691	98.691	98.691	98.429	98.429	98.691	98.691

Further, an identical accuracy of 98.691% for the number of DTs 20 and 40 can be observed. Besides, steady accuracy of 98.952% can be discerned as the number of DTs augmented to 80, 160, 320, 640, 1280, 2560, and 5120. Similarly, for the topmost 25 features advised by the Gain-Ratio FST, accuracy of 98.429% and 98.691% was obtained when the RF classifier was set with five and ten DTs, respectively. However, stable accuracy of 98.952% was obtained as the DTs in the RF classifier were appended in terms of 20, 40, 80, 160, 320, 640, 1280, 2560, and 5120. Moreover, when the topmost 25 features suggested by the mRMR FST were employed, the RF classifier was able to attain accuracy of 98.691% with five DTs. Later, when different number of DTs, i.e., 10, 20, 40, 80, 160, 320, 640, 1280, 2560, and 5120 was considered, at each computation, the RF classifier yielded stable detection accuracy. Correspondingly, for the topmost 25 features endorsed by the Max-Rel FST, the RF classifier with five DTs achieved detection accuracy of 98.691%, and as the number of DTs was incremented by 10, 20, 40, 80, 160, 320, 640, 1280, 2560, and 5120, the accuracy accomplished by the RF classifier was identical, i.e., 98.952%.

To measure the effect of the number of DTs on the predictability of the RF classifier, additional experiments were carried out by considering topmost 50, 75, and 100 features recommended by each of the FSTs. When the topmost 50 features were taken into account as recommended by the Chi-Square$\tilde{\chi}^2$ and Gain-Ratio FSTs separately, the RF classifier achieved an accuracy of 98.167% and 98.691% when examined with 5 and 10 DTs, respectively. Contrarily, the RF classifier obtained steady accuracy of 98.952% when the number of DTs were set to 20, 40, 80, 160, 320, 640, 1280, 2560, and 5120 for the crucial 50 features advised by the Chi-Square$\tilde{\chi}^2$ and Gain-Ratio FSTs. Further, it was noticed that the RF classifier gained detection accuracy of 98.952% with five DTs for the significant 50 features suggested by the mRMR FST. However, for the same features, the detection accuracy of the RF classifier declined slightly, i.e., 98.691% when the classification process was performed with 10 and 20 DTs. Further, for the same features, the detection accuracy obtained was the same, i.e., 98.952% when the RF classifier was executed with 40, 80, 160, 320, 640, 1280, 2560, and 5120 DTs. Lastly, when 50 imperative features recommended by the Max-Rel FST were considered, the RF classifier with five DTs achieved an accuracy of 98.429% and showed further improvement by gaining an accuracy of 98.691% when the number of DTs was increased

to 10 and 20. Later, the accuracy of 98.952% stabilized as the number of DTs increased (see Table 5.8).

Experiments were further pursued to examine the efficiency of the RF classifier when the top 75 relevant features were selected by each of the FSTs. It can be seen from Table 5.8 that for the top 75 features recommended by the Chi-Square$\tilde{\chi}^2$ FST, the RF classifier attained detection accuracy of 98.691% with five DTs. However, as the number of DTs increased to 10 and 20, interestingly, the RF classifier achieved highest accuracy of 99.214%. Later, the accuracy decreased to 98.691% when the RF classifier was set with 40 DTs. However, for the same 75 features, the accuracy decreased, i.e., 98.429% when the RF classifier was set with DTs of size 80, 160, 320, 640, 1280, 2560, and 5120. For the best 75 features provided by the Gain-Ratio FST, stable accuracy of 98.691% was accomplished when the RF classifier was evaluated with a different number of DTs such as 5, 10, and 20. When the RF classifier was set with 40 and 80 DTs, an increase in accuracy to 98.952% was observed. Further, a drop in accuracy, i.e., 98.691% was found when the number of DTs was 160. Moreover, the detection accuracy of 98.952% became steady when the number of DTs was fixed at 320, 640, 1280, 2560, and 5120. Subsequently, the topmost 75 features, insisted by the mRMR FST, were also utilized to find the detection accuracy of the RF classifier with 5 and 10 DTs, and it gained an accuracy of 98.691%. Further, a gradual increase in accuracy, i.e., 98.952% was noticed, which then stabilized to a certain extent as the number of DTs increased to 20, 40, 80, 160, 320, and 640. Furthermore, when the number of DTs was set to 1280, 2560, and 5120, the accuracy slightly declined, i.e., 98.691%. Additionally, the RF classifier with 5 and 10 DTs was examined with crucial uppermost 75 features provided by the Max-Rel FST and achieved a detection accuracy of 98.429%. On succeeding further by varying the number of DTs to 20, 40, 80, 160, 320, 640, 1280, 2560, and 5120, the detection accuracy recorded by the RF classifier is as shown in Table 5.8, and stable accuracy obtained was 98.691%.

The proficiency of the RF classifier was also examined with the topmost 100 features recommended by each of the FSTs (See Table 5.8). For the top 100 features endorsed by the Chi-Square$\tilde{\chi}^2$ FST, the RF classifier with five DTs accomplished a detection accuracy of 98.167%, followed by 98.429% when the number DTs was set to 10 and 20. Further, as the number of DTs increased to 40, 80, and 160, slightly im-

proved accuracy of 98.691% was noticed and thereafter, constant accuracy of 98.429% was achieved, irrespective of the number of DTs. The RF classifier with five DTs was evaluated by considering the topmost 100 features advised by the Gain-Ratio FST and the achieved accuracy was 98.167%. Later, when the number of DTs was increased to 10, better accuracy of 98.429% could be observed. Consecutively, the performance of the RF classifier was analyzed by increasing the number of DTs sequentially in terms of 20, 40, 80, 160, 320, 640, 1280, 2560, and 5120, and it was observed that the RF classifier accomplished stable accuracy of 98.952%. In comparison, for the best 100 features recommended by the mRMR FST, the RF classifier with a different number of DTs achieved different accuracies, i.e., for 5 DTs, the accuracy achieved was 98.691%, for 10 DTs, the accuracy was slightly minimized to 98.429%, for 20 DTs, the accuracy increased, i.e., 98.691%, and for 40 DTs, the achieved accuracy was 98.952%. However, the accuracy declined to 98.691% and became stable on further increase in the number of DTs (see Table 5.8). Finally, for the topmost 100 relevant features recommended by the Max-Rel FST, the RF classifier attained a steady accuracy of 98.691% for varying number of DTs, except when the number of DTs was set to 640 and 1280.

5.4.5.2 Analysis-II

Table 5.9 demonstrates the performance analysis of the RF classifier for Dataset-2. It can be seen that for the top 25 features suggested by the Chi-Square$\tilde{\chi}^2$ FST, the detection accuracy achieved by the RF classifier was found to sequentially increase in terms of 96.888%, 97.380%, 97.549%, 97.782%, 97.899%, and 98.029% when the number of DTs was set to 5, 10, 20, 40, 80, and 160, respectively. Similarly, the performance of the RF classifier was observed when the top 25 features recommended by the Gain-Ratio FST were utilized. As the number of DTs was increased from 5 to 80, the detection accuracy also increased - 92.842%, 92.933%, 92.959%, 92.966%, and 92.985%. In this case also, it was recognized that increasing the number of DTs after a certain threshold did not show any improvement in the detection accuracy. When the 25 significant features advised by the mRMR FST were employed, the RF classifier produced an accuracy of 96.473%, 96.836%, 96.991%, 97.095%, 97.152%, and 97.186% when it was set to 5, 10, 20, 40, 80, and 160 DTs, respectively.

Table 5.9: Detection accuracy(%) achieved by the Random Forest classifier for hybrid feature set derived from Dataset-2 for different number of Decision Trees

Feature Selection Technique	**Topmost Features Count**	**Number of Decision Trees**										
		5	10	20	40	80	160	320	640	1280	2560	5120
Chi-Square$\tilde{\chi}^2$	25	96.888	97.380	97.549	97.782	97.899	98.029	97.977	97.925	97.964	97.912	97.925
Gain-Ratio	25	92.842	92.933	92.959	92.966	92.985	92.972	92.985	92.985	92.985	92.985	92.977
mRMR	25	96.473	96.836	96.991	97.095	97.152	97.186	97.173	97.134	97.095	97.095	97.108
Max-Rel	25	97.134	97.536	97.614	97.990	98.029	98.016	98.029	97.977	97.964	97.990	97.997
Chi-Square$\tilde{\chi}^2$	50	97.523	97.756	98.184	98.236	98.379	98.301	98.327	98.314	98.314	98.288	98.301
Gain-Ratio	50	93.153	93.244	93.205	93.166	93.257	93.205	93.218	93.231	93.244	93.244	93.231
mRMR	50	96.667	97.173	97.251	97.445	97.627	97.666	97.653	97.666	97.627	97.653	97.588
Max-Rel	50	97.484	98.016	98.145	98.158	98.262	98.288	98.314	98.275	98.288	98.288	98.275
Chi-Square$\tilde{\chi}^2$	75	97.264	98.029	98.184	98.340	98.314	98.379	98.418	98.444	98.418	98.431	98.431
Gain-Ratio	75	96.019	95.928	96.006	96.084	96.045	96.084	96.148	96.148	96.110	96.174	96.174
mRMR	75	96.810	97.432	97.588	97.704	97.821	97.756	97.730	97.717	97.704	97.704	97.691
Max-Rel	75	97.406	97.821	98.106	98.314	98.340	98.392	98.353	98.327	98.340	98.379	98.353
Chi-Square$\tilde{\chi}^2$	100	97.264	97.743	98.236	98.301	98.366	**98.521**	98.482	98.431	98.431	98.431	98.444
Gain-Ratio	100	96.110	96.113	96.122	96.174	96.226	96.278	96.213	96.252	96.226	96.213	96.226
mRMR	100	96.563	97.264	97.678	97.787	97.795	97.925	97.925	98.003	97.925	97.925	97.951
Max-Rel	100	97.121	97.704	98.132	98.327	98.262	98.301	98.340	98.418	98.469	98.444	98.444

Even though experiments were carried out to analyze the performance of the RF classifier by increasing the number of DTs to 320, 640, 1280, 2560, and 5120, an improvement in detection accuracy was not noticed. Finally, the performance of the RF classifier was analyzed for the 25 features suggested by the Max-Rel FST, where the detection accuracy consistently improved until the DTs within the RF was raised to 80. However, incrementing the number of DTs after 80 led to a decrease in accuracy of the RF classifier.

Appendix C, D, E, and F show the topmost hybrid features suggested by four different FSTs, respectively.

For the topmost 50 features suggested by the Chi-Square$\tilde{\chi}^2$ FST, the RF classifier with five DTs obtained an accuracy of 97.523%. Further improvement in accuracy as the DTs increased to 10, 20, 40, and 80 was found. Consequently, the tree size was increased to 160, 320, 640, 1280, 2560, and 5120, in order to see improvement in accuracy than the previous, however, poor accuracy was attained as reported in Table 5.9. The efficiency of the RF classifier was examined by considering the top 50 features advised by the Gain-Ratio FST. The detection accuracy showed a continuous improvement until the number of DTs was raised to 80. The detection accuracy yielded was 93.153%, 93.244%, 93.205%, 93.166%, and 93.257%. In this case also, drop in accuracy was observed when the number of DTs was set to 160, 320, 640, 1280, 2560, and 5120. Further, the proficiency of the RF classifier was checked for the 50 features recommended by the mRMR FST. The accuracy obtained by the RF classifier was 96.667%, 97.173%, 97.251%, 97.445%, 97.627%, and 97.666% when the number of DTs was set to 5, 10, 20, 40, 80, and 160, respectively. Meanwhile, when the number of DTs was increased from 320 to 5120, less accuracy was observed. Moreover, the detection accuracy achieved by the RF classifier for the Max-Rel recommended features was also better when the number of the DTs was increased until 320 and maximum accuracy of 98.314% was achieved (see Table 5.9). Further, the accuracy yielded by the RF classifier was not remarkable when the number of DTs was set to 640, 1280, 2560, and 5120.

The analysis was continued for the top 75 features recommended by the chosen FSTs. For the top 75 features advised by the Chi-Square$\tilde{\chi}^2$ FST, gradual increase in

accuracy was determined when the DTs were consistently varied from 5 up to 640 trees. The obtained detection accuracy was 97.264%, 98.029%, 98.184%, 98.340%, 98.314%, 98.379%, 98.418%, and 98.444%. However, further increase in the number of DTs did not show much affect on detection accuracy. When the RF classifier was executed with the top 75 features suggested by the Gain-Ratio FST with five DTs, detection accuracy of 96.019% was achieved. Later, on increasing the DTs to 10, an immediate decrease in accuracy, i.e., 95.928% was found. Further, hike in detection accuracy, i.e., 96.006% and 96.084% was observed when the number of DTs was set to 20 and 40, respectively. Moreover, the recorded accuracy in the Table 5.9 shows that the detection ability was variable as the number of DTs was increased to 80, 160, 320, 640, 1280, 2560, and 5120. Maximum accuracy of 96.174% was accomplished when the number of DTs was set to 2560 and 5120. Furthermore, the analysis was pursued independently with the 75 features insisted by the mRMR FST and Max-Rel FST. It was observed that the detection accuracy achieved by the RF classifier gradually increased as the number of DTs increased from 5 up to 80. The accuracy computed by the RF classifier for the features recommended by the mRMR and Max-Rel FSTs did not show any improvement when the number of DTs was set to 160, 320, 640,1280, 2560, and 5120 trees (see Table 5.9).

Similar to the previous experiments, the topmost 100 features suggested by the chosen FSTs were selected to analyze the ability of the RF classifier. From Table 5.9, it can be observed that for the topmost 100 features advised by the Chi-Square$\tilde{\chi}^2$ FST, the detection accuracy showed an improvement each time when the number of DTs was varied until 160, and the detection accuracy declined on further increasing the number of DTs. Besides, for the top 100 features recommended by the Gain-Ratio FST, the detection accuracy was increased continuously until the number of DTs was set to 160 (see Table 5.9), and later, the detection accuracy did not show much improvement on increasing the number of DTs. Finally, the RF classifier was evaluated for the 100 significant features recommended by the mRMR and Max-Rel FSTs. The detection accuracy recorded in Table 5.9 describes that as the number of DTs increased from 5 up to 640, the detection accuracy also increased. Moreover, when the number of DTs was set to 1280, 2560, and 5120, there was not much impact on the detection accuracy.

5.4.6 Out-of-Bag Error

The OOBE rate (Khoshgoftaar et al., 2007) determines the error rate of the RF classifier. The error rate is due to the correlation between two trees and the discrimination ability of every DT (Leistner et al., 2009; Ajay Kumara and Jaidhar, 2017). However, the RF classifier error rate was evaluated considering the aggregation of the Out-of-Bag prediction.

In the present work, the OOBE rate was used to evaluate the predictive performance of the RF classifier according to the number of DTs considered as per Eq. 5.10. Figure 5.8a, Figure 5.8c, Figure 5.9a, and Figure 5.9c illustrate the OOBE rate for the top features suggested by the four different chosen FSTs for Dataset-1. Figure 5.8b, Figure 5.8d, Figure 5.9b, and Figure 5.9d represent the OOBE rate achieved with the top crucial features recommended by the four FSTs for Dataset-2.

Figure 5.8a demonstrates the OOBE rate attained when the DTs were increased to 80 for the topmost 25 features recommended by the Chi-Square$\tilde{\chi}^2$ and mRMR FSTs. Similarly, for the top 25 features suggested by the Gain-Ratio FST, lower OOBE rate was achieved at 160 DTs. Likewise, for the top 25 features advised by the Max-Rel FST, lower OOBE rate was obtained at 320 DTs. In this case, the error rate stabilized after reporting lower OOBE rate as 0.0105 at 80 DTs, 0.0105 at 160 trees, and 0.0105 at 320 trees.

The OOBE rate was achieved for the top 50 features suggested by the Chi-Square$\tilde{\chi}^2$ FST at 40 DTs and by the Max-Rel FST when the DTs were increased to 80, whereas for the top 50 features advised by the Gain-Ratio FST, lower OOBE rate was obtained at 160 DTs. Similarly, the OOBE rate was attained when the DTs were increased to 320 for the topmost 50 features recommended by the mRMR FST. As shown in Figure 5.8c the error rate stabilized after yielding lower OOBE rate as 0.0105 at 40 DTs, 0.0131 at 80 trees, 0.0105 at 160 trees, and 0.0131 at 320 trees.

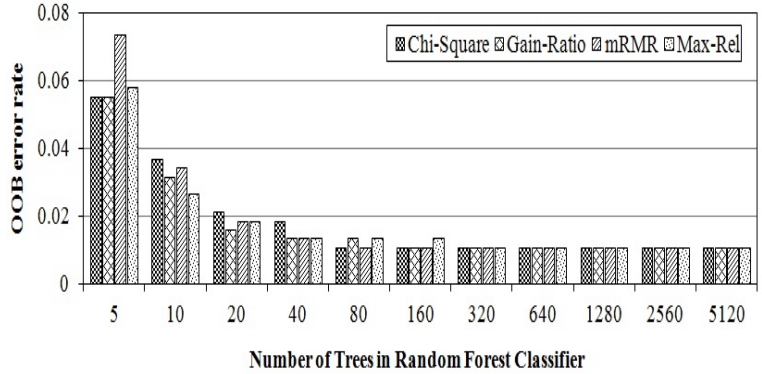
The OOBE rate obtained for the top 75 features is illustrated in Figure 5.9a, where lower OOBE rate for the 75 features advised by the Chi-Square$\tilde{\chi}^2$ FST was attained at 80 DTs, and by Max-Rel FST at 80 DTs. Similarly, for the top 75 features recommended by the Gain-Ratio FST, lower OOBE rate was achieved at 40 DTs, and for the mRMR FST, when the DTs were increased to 80. In this case, the error rate was

stabilized after gaining minimum OOBE rate as 0.0105 at 80 DTs, 0.0131 at 80 trees, 0.0105 at 40 trees, and 0.0131 at 80 trees for the top 75 features recommended by the Chi-Square$\tilde{\chi}^2$, Max-Rel, Gain-Ratio, and mRMR FSTs, respectively.

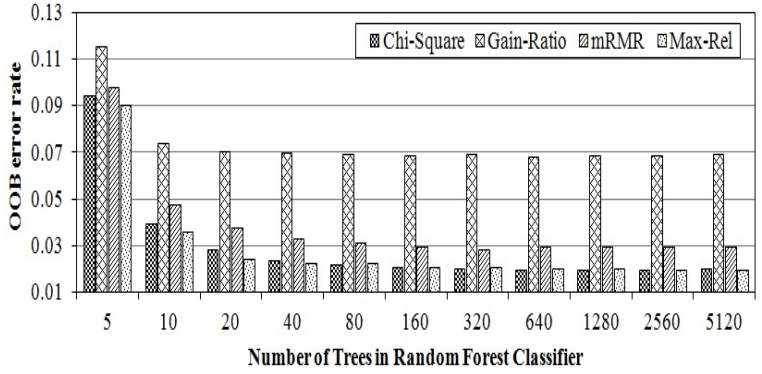
Figure 5.9c describes the lower OOBE rate as 0.0157 obtained for the top 100 features recommended by the Chi-Square$\tilde{\chi}^2$ FST at 80 DTs. For the top 100 features advised by the Gain-Ratio FST, lower OOBE rate 0.0105 was attained at 40 DTs. Similarly, lower OOBE rate of 0.0131 was achieved for the top 100 features suggested by the mRMR FST at 80 DTs, and for the top 100 features insisted by Max-Rel FST, lower OOBE rate of 0.0105 was achieved at 40 DTs.

Additionally, a thorough analysis was carried out to know the predictive performance of the RF classifier while measuring the OOBE rate from the experiments conducted using Dataset-2. It can be seen from Figure 5.8b, Figure 5.8d, Figure 5.9b, and Figure 5.9d, that the evaluation was done by noticing the OOBE rate when the DTs were increased from 5 to 5120. Figure 5.8b depicts that lower OOBE rate of 0.0197 was achieved at 320 DTs for the top 25 features recommended by the Chi-Square$\tilde{\chi}^2$ FST. Similarly, for the top 25 features advised by the Gain-Ratio FST, lower OOBE rate of 0.0683 was obtained at 160 DTs. Likewise, for the 25 features suggested by the mRMR FST and Max-Rel FST, lower OOBE rate of 0.0281 and 0.0198 was attained at 320 and 640 DTs, respectively.

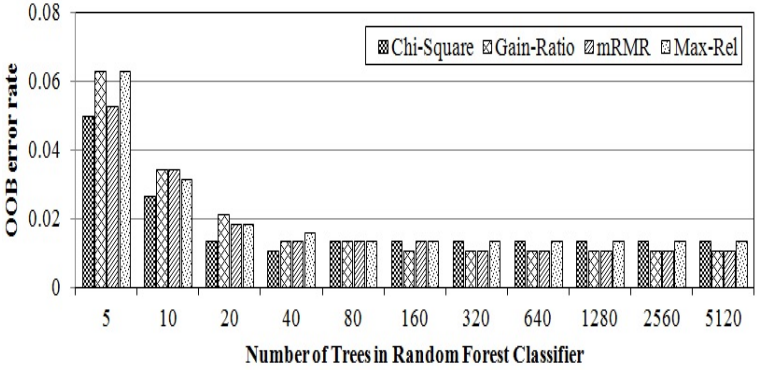
When the OOBE rate was measured for the top 50 features suggested by the Chi-Square$\tilde{\chi}^2$ FST, error rate of 0.0171 was obtained at 80 DTs as shown in Figure 5.8d. The OOBE rate of 0.0681 was achieved at 40 DTs for the top 50 features recommended by the Gain-Ratio FST. Similarly, lower OOBE rate of 0.024 was obtained at 80 DTs and 0.0172 at 160 DTs for the top 50 features advised by the mRMR and Max-Rel FSTs, respectively. Figure 5.9b illustrates the lower OOBE rate achieved for the top 75 features suggested by the chosen four FSTs. Accordingly, lower OOBE rate of 0.0175 was obtained at 80 DTs for the top 75 features suggested by the Chi-Square$\tilde{\chi}^2$ FST. When lower OOBE rate was measured for the top 75 features suggested by the Gain-Ratio FST, the error rate of 0.0368 was achieved at 40 DTs. Further, the OOBE rate for the top 75 features advised by the mRMR FST was 0.0244 at 80 DTs. Similarly, lower OOBE rate of 0.0218 was accomplished at 40 DTs for the top 75 features recommended by the Max-Rel FST.



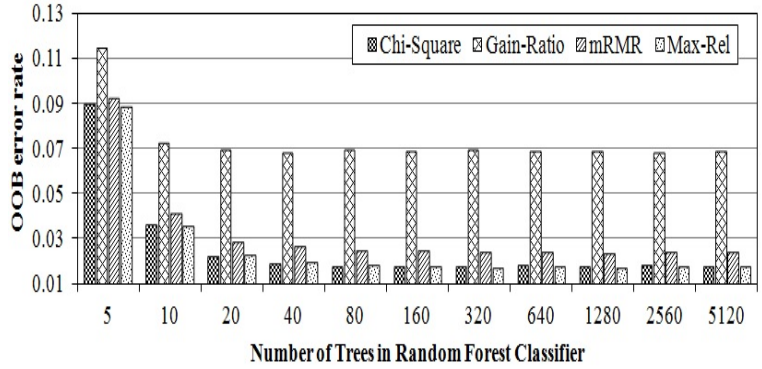
(a) Dataset-1: Accuracy (25 Features)



(b) Dataset-2: Accuracy (25 Features)

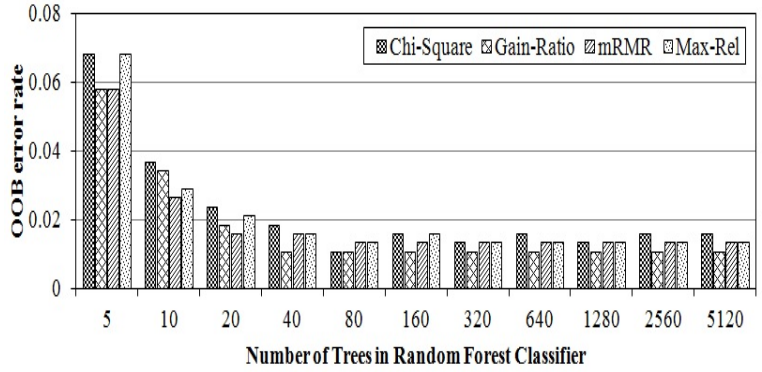


(c) Dataset-1: Accuracy (50 Features)

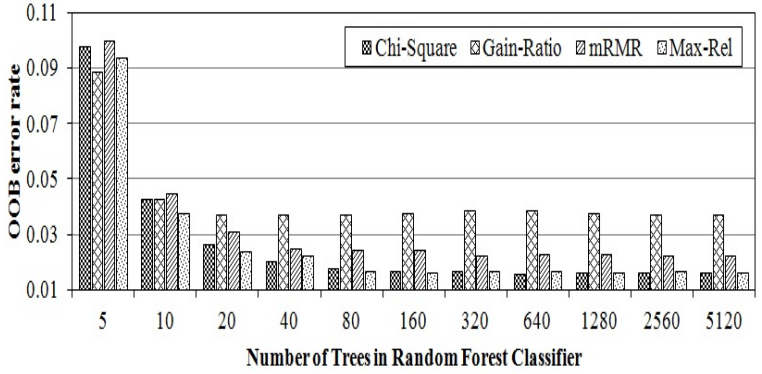


(d) Dataset-2: Accuracy (50 Features)

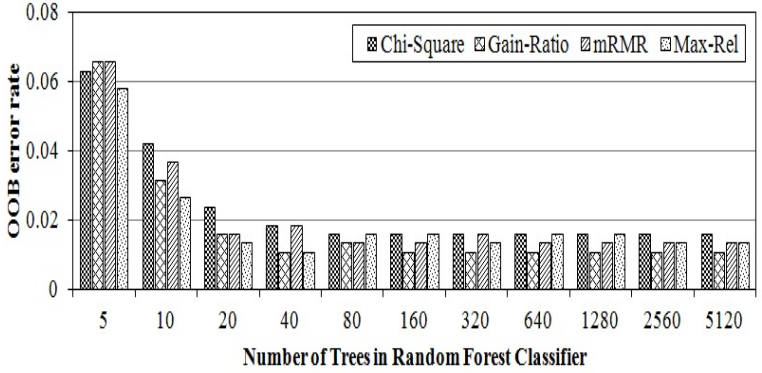
Figure 5.8: Out-of-Bag Error (OOBE) rate achieved by the Random Forest classifier for Dataset-1 and Dataset-2



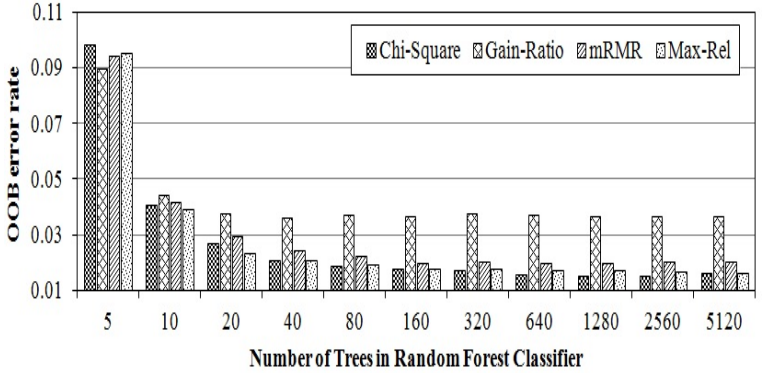
(a) Dataset-1: Accuracy (75 Features)



(b) Dataset-2: Accuracy (75 Features)



(c) Dataset-1: Accuracy (100 Features)



(d) Dataset-2: Accuracy (100 Features)

Figure 5.9: Out-of-Bag Error (OOBE) rate achieved by the Random Forest classifier for Dataset-1 and Dataset-2

Table 5.10: Comparison of the proposed approach with existing research works

Related Work	**# Executables used**		**Feature Selection Technique**	**Features Type**	**Accuracy (%)**
	Malware	**Benign**			
Islam et al. (2013)	2398	541	Not Mentioned	Function Length Frequency, Printable String Information, and API calls	97.05
Santos et al. (2013)	1000	1000	Information-Gain	Information of the execution trace of the executable, and Operational code	96.60
Ahmadi et al. (2013)	806	306	Fisher Score	API calls sequences and iterative patterns	98.10(AUC)
Salehi et al. (2014)	826	385	Relief	API calls with their arguments	98.40
Awan and Saqib (2016)	1600	600	Information-Gain	API calls, Printable Strings, and PE Header Information	97.20
Kumar et al. (2019)	2722	2488	Not Mentioned	PE files Header field's raw value and derived values	98.40
Proposed Work HFMDS	200	200	Chi-Square$\tilde{\chi}^2$	OH, FH, SH, DH, IF, and API + CAT	99.214
	3856	3856	Chi-Square$\tilde{\chi}^2$	OH, FH, SH, DH, IF, and API + CAT	98.521

Table 5.11: Comparison of proposed approach with the recent related works

Recent Works	**Accuracy (%)**
Jain and Singh (2017)	73.47
Aman et al. (2017)	93.30
Proposed work (HFMDS)	98.52

Finally, the lower OOBE rate was measured for the 100 features recommended by the Chi-Square$\tilde{\chi}^2$, Gain-Ratio, mRMR, and Max-Rel, FSTs, and the error rate achieved was 0.0205, 0.0359, 0.0219, and 0.0187, with the increased number of DTs as 40, 40, 80, and 80, respectively (see Figure 5.9d).

From the analysis, it can be understood that when less number of features are considered, maximum number of DTs are required to stabilize the OOBE rate. Contrarily, if the number of features is more, then less number of DTs is required by the RF classifier to stabilize the OOBE rate.

In order to substantiate the proposed approach, the present work was compared with some of the earlier research works (see Table 5.10). It can be noticed that the proposed HFMDS outperforms previous works with an improvement in accuracy that varies from 0.18% to 2.61% for Dataset-1 and from 0.12% to 1.92% for Dataset-2.

Similarly, the present work was compared with some of the research works of recent years as shown in Table 5.11. Jain and Singh (2017) propounded an integrated features-based malware detection framework that utilized selected static and dynamic features for detecting malware. They experimented to demonstrate the relative analysis of the integrated technique with only static features and only dynamic features to analyze which could provide better detection accuracy. They used three machine learning-based classifiers, namely, Naive Bayes, Support Vector Machine, and RF to evaluate their framework. The results obtained manifested that the integrated approach was better and attained detection accuracy of 73.47% by the RF classifier. Aman et al. (2017) presented a malware discrimination system based on multiple features such as static, API calls, and regular expression. They achieved an accuracy of 93.30% by the RF classifier with 100 trees.

From Table 5.11, it can be inferred that the proposed approach obtained an accuracy of 98.521% by the RF classifier with 160 trees, which is more than the accuracy of

(Jain and Singh, 2017) and (Aman et al., 2017). The enhancement in accuracy might be because of the use of API+CAT and IF along with the PE header features (DH, FH, SH, and OH), and the increased number of DTs.

5.5 Discussion

In the present work, the feasibility of the proposed HFMDS in detecting Windows malware was explored. The FSTs which were employed in the work were compared to determine which FST increased the RF classifier's accuracy. Finally, experiments were carried out to seek the stability of the RF classifier accuracy and also to ascertain the optimal number of DTs needed by the RF classifier. Four research questions mentioned in Section 5.4.4 were investigated to measure the efficiency of the HFMDS. To answer them, several experiments were designed and conducted. In Section 5.4.4.4, the effectiveness of the HFMDS was demonstrated using both static and dynamic features in detecting unknown malware. The experimental results proved the efficiency of the proposed HFMDS in achieving highest accuracy of 99.214% by the RF classifier with 10 DTs for the top 75 features recommended by the Chi-Square$\tilde{\chi}^2$ FST with respect to Dataset-1. Further, it was possible to attain an accuracy of 98.521% by the RF classifier with 160 DTs for the 100 features suggested by the Chi-Square$\tilde{\chi}^2$ FST corresponding to Dataset-2, which promised the proficiency of the HFMDS. Another observation was that the RF classifier achieved an acceptable accuracy for the mRMR and Max-Rel FSTs with different features size obtained for Dataset-1 and Dataset-2. Although, the mRMR suggested features were found to achieve better detection accuracy in (Peng et al., 2005) when tested by the SVM classifier, it failed to recommend the best significant features and resulted in less significant accuracy gain by the RF classifier. Further, from Analysis-I and Analysis-II, one can reasonably infer that as the number of DTs increased, it did not always mean that the performance of the RF classifier was better with fewer trees within the RF classifier. In general, the number of DTs was fixed on trial-and-error basis. However, based on the observations on the experiments conducted, it can be understood that merely doubling the number of DTs is of no use since it is also possible to define a threshold for the number of DTs, as beyond the threshold there was no significant improvement in accuracy. The analysis performed by employing two datasets demonstrated that beyond 160 DTs, there was no significance in increasing the number of DTs, to 640, 1280, 2560, and 5120. Therefore based on the experimental

results, it is possible to recommend that the DTs should range from 80 to 160 for the RF classifier to attain better detection accuracy.

The proposed hybrid apparoach achieves False Positive due to the following reasons: (i) While execution, some of the sophisticated malware do not exhibit their malicious action until certain condition met. Extracting the features from behavioural report that do not provide the malicious action(s) never provide essential features to recognize malware accurately. (ii) Feature(s) present in the Final Feature Set do not exist in the features list that is extracted from the testing file also leads to misclassification.

5.6 Summary

In Section 5.1, an HFMDS that utilizes both static and dynamic features to precisely classify unknown malware has been presented. The API calls triggered by the PE files during their execution were treated as dynamic features and the OH, FH, and DOSH related data was considered as static features. The effectiveness of the API calls alone, combination of API calls and static features, and combination APICAT and static features were investigated in the detection of unknown malware to know which blend would provide better detection accuracy. The LSVC was applied onto static features set as well as dynamic features set individually to obtain potential features that would increase the performance of the classifier. The LSVC recommended features were treated as the final features to evaluate the effectiveness of the HFMDS. Experiments are conducted with real-world malware samples, and the obtained empirical results demonstrated that the proposed HFMDS was efficient in analyzing and detecting malware with an accuracy of 99.743% as accomplished by the SMO classifier comprising of APICAT+OH+DOSH+FH as the hybrid features.

In Section 5.4, the proposed HFMDS examined the hybrid features of the PE files that included PE header information, IF, and API+CAT to accurately classify the test input file as either benign or malware. The selection of the most significant hybrid features was achieved by utilizing well-known filter-based FSTs, and a thorough analysis was made to identify the best FST, which could assist the classifier to achieve better detection accuracy. A wide set of experiments was conducted by considering a smaller dataset (Dataset-1) as well as a larger dataset (Dataset-2) to evaluate the detection proficiency of the proposed HFMDS with the RF classifier. The obtained empirical results

demonstrated that the RF classifier achieved highest detection accuracy when the number of DTs was set in the range of 80 to 160, and it was noticed that merely doubling the number of DTs did not have any significant effect on the predictive performance of the RF classifier.

Chapter 6

Conclusions and Future Work

The existing Windows malware detection techniques fall short of tackling the challenges posed by obscure malware leading to escalating information, time, and financial losses. This necessitates the development of new or enhanced Windows malware defence techniques. In this direction, a detailed study of the different Windows malware detection techniques was carried out that consider static features alone, behavioural features alone, and a combination of both. In the present research work, Windows malware detection approaches were proposed to accurately identify benign PE files as benign and malware PE files as malware. This chapter discusses concluding remarks related to the proposed approaches.

Most of the conventional Windows antimalware defensive solutions are static features-based approach. However, the discovery of obscure malware in real-time is still a critical issue. In this regard, the PEOHF-based MDS was designed for discerning the Windows malware by considering OH and their corresponding values as static features. To obtain the most significant PEOHF features from the original feature set, the Single-Stage-Feature-Selector was utilized. Experiments were conducted, and comparative analysis was performed to measure the effectiveness of the chosen filter-based FSTs in recommending the best features to obtain better accuracy by the classifiers.

The modern Windows malware is well versed with evasion techniques and can easily elude the conventional static features-based technique. Therefore, a malware detection technique using the Cuckoo Sandbox was implemented to gather system-level behaviour of the PE files. The system calls sequence, invoked by the PE files, was extracted from the MIST reports obtained by pre-processing the JSON reports. The sliding window approach was used to represent the extracted system calls sequence as N-grams, and the IG FST was employed, which suggested prominent N-gram features required to construct the Final Feature Set. An evaluation of the machine learning-based classifier was carried out, and the Spegagos classifier ensured better detection rate for N-gram lengths of three and four bytes.

The performance of the classifier depends purely on the final features used in the

training file. FST plays an essential role in selecting the best features from the original feature set. It computes the score for each N-gram (features) and takes an enormous amount of time to produce a score when the total number of N-grams is vast. To address this issue, a multiprocessing model that computes the IG score for each N-gram was developed. The efficiency of the proposed multiprocessing model was measured through a set of the experiments and the obtained empirical evidence demonstrated that it was 80% faster than the sequential model.

The dimensionality of the original feature set exhibited significant effect on the predictive performance of the machine learning-based classifier. In order to identify and eradicate noisy features from the original feature set, the LSVC was employed as feature selector. The effectiveness of the behavioural features suggested by the LSVC in identifying unknown Windows malware was investigated. Two different types of behavioural features, namely, API calls and Category+API calls were considered to know which type of features provided better detection rate. From the obtained experimental results, it was observed that maximum accuracy was achieved for the final features set comprising of API calls as N-gram features of size 4 bytes as recommended by the LSVC. Thus, the empirical results manifested that the LSVC is proficient in recommending the best N-gram features, which boost the detection rate of the machine learning-based classifier.

The behavioural-based malware detection approaches with machine learning techniques have been widely adopted as a profound solution to defend against malware. Though machine learning-based malware detection techniques have exhibited success in detecting malware, their shallow learning architecture is still deficient in identifying sophisticated malware. Therefore, a CNN-based Windows malware detector was proposed that used the execution time behavioural features of the PE files to detect and classify obscure malware. The runtime behaviour features (CAT-API) of the PE files, as advised by the Feature Selection Technique was used to create images, and thereby, it was demonstrated that the malware detection problem can be transformed into an image classification problem. Further, to prevent the Neural Network from learning the associations between the irrelevant features, experiments were conducted to demonstrate that feature selection at the input level of CNN is necessary. The obtained empirical results manifested that the detection accuracy achieved by the proposed approach is

predominant than that of the detection accuracy gained by the machine learning-based classifiers. Thereby, the proposed approach was proficient in the detection of Windows malware.

The amalgamation of the dynamic and static analysis can provide vital information for analyzing malware. In this regard, the HFMDS was proposed to identify unknown Windows malware that used the LSVC as feature selector. A wide set of experiments were conducted to demonstrate the benefits of combining both static and dynamic features. The obtained experimental results demonstrated that the proposed HFMDS was able to attain maximum detection accuracy for the combination of API-CAT+OH+DOSH+FH hybrid features.

The effectiveness of the hybrid features-based detection approach depends on the types of features used to identify the malware. In this regard, the proposed HFMDS investigated the hybrid features of the PE files that included PE header information, IF, and API+CAT to precisely detect and classify the PE file. The HFMDS was trained with prominent hybrid features advised by the filter-based FST. The detection ability of the proposed HFMDS was evaluated with the RF classifier. In-depth analysis was carried out to determine the optimal number of DTs required by the RF classifier to achieve consistent accuracy. Besides, the performance of the four popular FSTs was also analyzed to determine which FST recommended the best features. From the experimental analysis, it was inferred that increasing the number of DTs after 160 within the RF classifier did not make any significant difference in attaining better detection accuracy.

The proposed approaches are robust in detecting the Windows malware. As future work, efforts should be made to map the proposed approaches to discern and classify non-Windows malware. The proposed CNN-based Windows malware detector showed advantages in accuracy, but performed slightly slower than the machine learning-based classifiers. In this regard, it is required to reduce the detection time in the future. A novel approach has to be introduced whereby the Sandbox can monitor and recognize kernel-based malware.

Appendix A: Top 25 PEOHF features (static features) selected by each FST for BD

Sl. No.	DFS	Sl. No.	MI
1	BaseOfCode-4096	1	BaseOfCode-4096
2	CheckSum-0	2	CheckSum-0
3	DllCharacteristics-0	3	DllCharacteristics-0
4	DllCharacteristics-32768	4	DllCharacteristics-32768
5	ImageBase-16777216	5	ImageBase-16777216
6	ImageBase-268435456	6	ImageBase-4194304
7	ImageBase-4194304	7	MajorImageVersion-0
8	MajorImageVersion-0	8	MajorImageVersion-5
9	MajorImageVersion-5	9	MajorLinkerVersion-6
10	MajorLinkerVersion-2	10	MajorLinkerVersion-7
11	MajorLinkerVersion-6	11	MajorOperatingSystemVersion-4
12	MajorLinkerVersion-7	12	MajorOperatingSystemVersion-5
13	MajorOperatingSystemVersion-4	13	MinorImageVersion-0
14	MajorOperatingSystemVersion-5	14	MinorImageVersion-1
15	MinorImageVersion-0	15	MinorLinkerVersion-0
16	MinorImageVersion-1	16	MinorLinkerVersion-10
17	MinorLinkerVersion-10	17	MinorOperatingSystemVersion-0
18	MinorOperatingSystemVersion-0	18	MinorOperatingSystemVersion-1
19	MinorOperatingSystemVersion-1	19	SizeOfHeaders-1024
20	SizeOfHeaders-1024	20	SizeOfHeaders-4096
21	SizeOfHeaders-4096	21	SizeOfStackReserve-1048576
22	SizeOfStackReserve-1048576	22	SizeOfStackReserve-262144
23	SizeOfStackReserve-262144	23	SizeOfUninitializedData-0
24	Subsystem-2	24	Subsystem-2
25	Subsystem-3	25	Subsystem-3

Sl. No.	CPD	Sl. No.	DIA
1	SizeOfUninitializedData-225280	1	AddressOfEntryPoint-107871
2	SizeOfUninitializedData-241664	2	AddressOfEntryPoint-24748
3	SizeOfUninitializedData-24576	3	AddressOfEntryPoint-31475
4	SizeOfUninitializedData-262144	4	AddressOfEntryPoint-36633
5	SizeOfUninitializedData-3592192	5	AddressOfEntryPoint-4205
6	SizeOfUninitializedData-36864	6	AddressOfEntryPoint-9360
7	SizeOfUninitializedData-40960	7	AddressOfEntryPoint-9369
8	SizeOfUninitializedData-413696	8	AddressOfEntryPoint-9621
9	SizeOfUninitializedData-45056	9	BaseOfCode-40960
10	SizeOfUninitializedData-49152	10	BaseOfData-565248
11	SizeOfUninitializedData-512	11	CheckSum-134908
12	SizeOfUninitializedData-528384	12	CheckSum-448067
13	SizeOfUninitializedData-53248	13	CheckSum-70053
14	SizeOfUninitializedData-57344	14	MinorImageVersion-55
15	SizeOfUninitializedData-589824	15	MinorLinkerVersion-55
16	SizeOfUninitializedData-61440	16	SizeOfCode-101376
17	SizeOfUninitializedData-65536	17	SizeOfCode-219136
18	SizeOfUninitializedData-69632	18	SizeOfCode-36352
19	SizeOfUninitializedData-70144	19	SizeOfImage-314811
20	SizeOfUninitializedData-73728	20	SizeOfImage-331776
21	SizeOfUninitializedData-77824	21	SizeOfInitializedData-12173312
22	SizeOfUninitializedData-86016	22	SizeOfInitializedData-290816
23	SizeOfUninitializedData-90112	23	SizeOfInitializedData-590848
24	SizeOfUninitializedData-98304	24	SizeOfStackCommit-65536
25	Subsystem-0	25	SizeOfUninitializedData-57344

Appendix B: Top 25 PEOHF features (static features) selected by each FST for UBD

Sl. No.	DFS	Sl. No.	MI
1	BaseOfCode-4096	1	BaseOfCode-4096
2	CheckSum-0	2	CheckSum-0
3	DllCharacteristics-0	3	DllCharacteristics-0
4	DllCharacteristics-32768	4	DllCharacteristics-32768
5	FileAlignment-32	5	FileAlignment-512
6	ImageBase-16777216	6	ImageBase-16777216
7	ImageBase-4194304	7	ImageBase-4194304
8	ImageBase-65536	8	ImageBase-65536
9	MajorImageVersion-0	9	MajorImageVersion-0
10	MajorImageVersion-5	10	MajorImageVersion-5
11	MajorLinkerVersion-6	11	MajorLinkerVersion-6
12	MajorLinkerVersion-7	12	MajorLinkerVersion-7
13	MajorOperatingSystemVersion-4	13	MajorOperatingSystemVersion-4
14	MajorOperatingSystemVersion-5	14	MajorOperatingSystemVersion-5
15	MinorImageVersion-0	15	MinorImageVersion-0
16	MinorImageVersion-1	16	MinorImageVersion-1
17	MinorLinkerVersion-10	17	MinorLinkerVersion-10
18	MinorOperatingSystemVersion-0	18	MinorOperatingSystemVersion-0
19	MinorOperatingSystemVersion-1	19	MinorOperatingSystemVersion-1
20	SectionAlignment-32	20	SectionAlignment-4096
21	SizeOfHeaders-1024	21	SizeOfHeaders-1024
22	SizeOfStackReserve-1048576	22	SizeOfStackReserve-1048576
23	SizeOfStackReserve-262144	23	SizeOfStackReserve-262144
24	Subsystem-1	24	Subsystem-1
25	Subsystem-3	25	Subsystem-3

Sl. No.	CPD	Sl. No.	DIA
1	SizeOfUninitializedData-176128	1	AddressOfEntryPoint-107871
2	SizeOfUninitializedData-19532	2	AddressOfEntryPoint-15209
3	SizeOfUninitializedData-2048	3	AddressOfEntryPoint-24748
4	SizeOfUninitializedData-241664	4	AddressOfEntryPoint-25152
5	SizeOfUninitializedData-262144	5	AddressOfEntryPoint-31475
6	SizeOfUninitializedData-32768	6	AddressOfEntryPoint-36633
7	SizeOfUninitializedData-36864	7	AddressOfEntryPoint-9360
8	SizeOfUninitializedData-413696	8	AddressOfEntryPoint-9369
9	SizeOfUninitializedData-45056	9	AddressOfEntryPoint-9621
10	SizeOfUninitializedData-49152	10	BaseOfData-565248
11	SizeOfUninitializedData-512	11	CheckSum-134908
12	SizeOfUninitializedData-528384	12	CheckSum-75729
13	SizeOfUninitializedData-53248	13	MinorImageVersion-55
14	SizeOfUninitializedData-57344	14	MinorLinkerVersion-55
15	SizeOfUninitializedData-589824	15	MinorLinkerVersion-69
16	SizeOfUninitializedData-61440	16	NumberOfRvaAndSizes-17
17	SizeOfUninitializedData-65536	17	SizeOfCode-101376
18	SizeOfUninitializedData-69632	18	SizeOfCode-219136
19	SizeOfUninitializedData-70144	19	SizeOfCode-6400
20	SizeOfUninitializedData-73728	20	SizeOfImage-304856
21	SizeOfUninitializedData-77824	21	SizeOfImage-331776
22	SizeOfUninitializedData-86016	22	SizeOfInitializedData-290816
23	SizeOfUninitializedData-90112	23	SizeOfInitializedData-590848
24	SizeOfUninitializedData-98304	24	SizeOfStackCommit-65536
25	Subsystem-0	25	SizeOfUninitializedData-19532

Appendix C: Top 25 hybrid features recommended by Chi-Square$\tilde{\chi}^2$ FST

Type	Features
API calls + Category (N-grams) (3)	02020a01, 11031102, 030a0307
OPTIONAL_HEADER (12/30)	DllCharacteristics MajorLinkerVersion MajorOperatingSystemVersion MajorSubsystemVersion SizeOfCode SizeOfInitializedData BaseOfCode BaseOfData MinorOperatingSystemVersion MajorImageVersion CheckSum AddressOfEntryPoint
DOS_HEADER (1/17)	e_lfanew
FILE_HEADER (3/7)	NumberOfSections, TimeDateStamp, Characteristics
SECTION_HEADER (4)	.text, .rdata, .reloc, .data
IMPORT FUNCTION CALL (2)	KERNEL32.dll, USER32.dll

Appendix D: Top 25 hybrid features recommended by Gain-Ratio FST

Type	Features
API calls + Category (N-grams) (18)	03070306, 030a0307, 030a1103, 0307030a 11031103, 0c011102, 03060303, 0306030a 0a010201, 120c0602, 030b0306, 030a0302 0b030e01, 03040303, 02021103, 030a0301 120c0202, 02020a06
OPTIONAL_HEADER (5/30)	MajorSubsystemVersion MinorOperatingSystemVersion MinorSubsystemVersion MajorOperatingSystemVersion DllCharacteristics
DOS_HEADER (0/17)	Not Recommended by Gain-Ratio FST
FILE_HEADER (0/7)	Not Recommended by Gain-Ratio FST
SECTION_HEADER (2)	.reloc, .text
IMPORT FUNCTION CALL (0)	Not Recommended by Gain-Ratio FST

Appendix E: Top 25 hybrid features recommended by Max-Rel FST

Type	Features
API calls + Category (N-grams) (4)	030a0307, 0307030a, 02020a01, 0c011102
OPTIONAL_HEADER (11/30)	DllCharacteristics MajorLinkerVersion MajorOperatingSystemVersion MajorSubsystemVersion SizeOfCode BaseOfData BaseOfCode CheckSum MajorImageVersion SizeOfInitializedData MinorOperatingSystemVersion
DOS_HEADER (1/17)	e_lfanew
FILE_HEADER (3/7)	Characteristics, NumberOfSections, TimeDateStamp
SECTION_HEADER (4)	.reloc, .text , .rdata, .data
IMPORT FUNCTION CALL (2)	KERNEL32.dll, USER32.dll

Appendix F: Top 25 hybrid features recommended by mRMR FST

Type	Features
API calls + Category (N-grams) (9)	030a0307, 02020a01, 0c011102, 03070306 11021104, 030a1103, 0307030a, 030a0301 09050902
OPTIONAL_HEADER (7/30)	DllCharacteristics MajorSubsystemVersion MajorOperatingSystemVersion MajorLinkerVersion MajorImageVersion MinorOperatingSystemVersion BaseOfCode
DOS_HEADER (0/17)	Not Recommended by mRMR FST
FILE_HEADER (2/7)	Characteristics, NumberOfSections
SECTION_HEADER (5)	.reloc, .text, .rdata, .data, .rsrc
IMPORT FUNCTION CALL (2)	KERNEL32.dll, ole32.dll

References

Ahmadi, M., Sami, A., Rahimi, H., and Yadegari, B. (2013). "Malware detection by behavioural sequential patterns". *Computer Fraud and Security*, *2013*(8), 11–19.

Ajay Kumara, M. A. and Jaidhar, C. D. (2017). "Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor". *Digital Investigation*, *23*, 99–123.

Alam, M. S. and Vuong, S. T. (2013). "Random forest classification for detecting android malware". In *Proc. International Conference on Green Computing and Communications (GreenCom), and IEEE Internet of Things (iThings/CPSCom) and IEEE Cyber, Physical and Social Computing*, 663–669. IEEE.

Albelwi, S. and Mahmood, A. (2017). "A Framework for Designing the Architectures of Deep Convolutional Neural Networks". *Entropy*, *19*(6), 242.

Aman, N., Saleem, Y., Abbasi, F. H., and Shahzad, F. (2017). "A Hybrid Approach for Malware Family Classification". In *Proc. International Conference on Applications and Techniques in Information Security*, volume 719, 169–180. Springer.

Amato, G. (2016), *"Open Source Tool to Perform Static Analysis on Portable Executable"*. https://github.com/guelfoweb/peframe/ [Accessed: 9 February 2018].

Amin, M. (2016). "A Survey of Financial Losses Due to Malware". In *Proc. Second International Conference on Information and Communication Technology for Competitive Strategies*, 145:1–145:4. ACM.

Andreas, M. (2018), *"AV-TEST GmbH"*. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2017-2018.pdf [Accessed: 20 July 2018].

Andreas, M. (2019), *"AV-TEST GmbH"*. https://www.av-test.org/en/statistics/malware/ [Accessed: 3 June 2019].

Arefkhani, M. and Soryani, M. (2015). "Malware Clustering Using Image Processing Hashes". In *Proc. 9th Iranian Conference on Machine Vision and Image Processing (MVIP)*, 214–218. IEEE.

Awan, S. and Saqib, N. A. (2016). "Detection of Malicious Executables Using Static and Dynamic Features of Portable Executable (PE) File". In *Proc. International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage (SpaCCS)*, volume 10067, 48–58. Springer.

Bai, J., Wang, J., and Zou, G. (2014). "A malware detection scheme based on mining format information". *The Scientific World Journal*, *2014*, 1–11.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., and Nazario, J. (2007). "Automated classification and analysis of internet malware". In *Proc. International Workshop on Recent Advances in Intrusion Detection*, volume 4637, 178–197. Springer.

Baldangombo, U., Jambaljav, N., and Horng, S.-J. (2013). "A static malware detection system using data mining methods". *CoRR*, *abs/1308.2831*, 1–13. http://arxiv.org/abs/1308.2831.

Bayer, U., Moser, A., Kruegel, C., and Kirda, E. (2006). "Dynamic analysis of malicious code". *Journal in Computer Virology*, *2*(1), 67–77.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., and Hamzeh, A. (2013). "A survey on heuristic malware detection techniques". In *Proc. 5th Conference on Information and Knowledge Technology (IKT)*, 113–120. IEEE.

Belaoued, M. and Mazouzi, S. (2015). "A Real-Time PE-Malware Detection System Based on CHI-Square Test and PE-File Features". In *Proc. 5th IFIP International Conference on Computer Science and its Applications*, volume 456, 416–425. Springer.

Borello, J.-M. and Mé, L. (2008). "Code obfuscation techniques for metamorphic viruses". *Journal in Computer Virology*, *4*(3), 211–220.

Bounouh, T., Brahimi, Z., Al-Nemrat, A., and Benzaid, C. (2017). "A Scalable Malware Classification Based on Integrated Static and Dynamic Features". In *Proc. International Conference on Global Security, Safety, and Sustainability*, volume 630, 113–124. Springer.

Breiman, L. (2001). "Random forests". *Machine learning*, *45*(1), 5–32.

CNET (1996), *"Internet download directory website"*. https://download.cnet.com/ [Accessed: 16 May 2017].

Coronado-De-Alba, L. D., Rodríguez-Mota, A., and Escamilla-Ambrosio, P. J. (2016). "Feature Selection and Ensemble of Classifiers for Android Malware Detection". In *Proc. 8th IEEE Latin-American Conference on Communications (LATINCOM)*, 1–6. IEEE.

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., and Chen, J. (2018). "Detection of Malicious Code Variants Based on Deep Learning". *IEEE Transactions on Industrial Informatics*, *14*(7), 3187–3196.

Cutler, A., Cutler, D. R., and Stevens, J. R. (2012). "Random forests". In *Ensemble machine learning* 157–175. Springer.

Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., and Stamp, M. (2017). "A comparison of static, dynamic, and hybrid analysis for malware detection". *Journal of Computer Virology and Hacking Techniques*, *13*(1), 1–12.

Darshan, S. S., Kumara, M. A., and Jaidhar, C. (2016). "Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm". In *Proc. 11th International Conference on Industrial and Information Systems (ICIIS)*, 534–539. IEEE.

Das, S., Liu, Y., Zhang, W., and Chandramohan, M. (2016). "Semantics-based online malware detection: Towards efficient real-time protection against malware". *IEEE transactions on information forensics and security*, *11*(2), 289–302.

David, B., Filiol, E., and Gallienne, K. (2017). "Structural analysis of binary executable headers for malware detection optimization". *Journal of Computer Virology and Hacking Techniques*, *13*(2), 87–93.

David, O. E. and Netanyahu, N. S. (2015). "DeepSign: Deep Learning for Automatic Malware Signature Generation and Classification". In *Proc. International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE.

Deng, L. (2012). "Three classes of deep learning architectures and their applications: a tutorial survey". *APSIPA transactions on signal and information processing*. http://www-edlab.cs.umass.edu/cs697l/readings/Three%20Classes%20of%20Deep%20Learning%20Architectures.pdf.

Ding, Y., Dai, W., Yan, S., and Zhang, Y. (2014). "Control flow-based opcode behavior analysis for Malware detection". *Computers and Security*, *44*, 65–74.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). "A survey on automated dynamic malware-analysis techniques and tools". *ACM computing surveys (CSUR)*, *44*(2), 6:1–6:42.

Ero, C. (2017), *"Python module to read and work with PE files"*. https://code.google.com/p/pefile/ [Accessed: 24 July 2017].

Eugene, K. (2019), *"Kaspersky"*. https://usa.kaspersky.com/about/press-releases/2019_kaspersky-lab-financial-cyberthreats-report [Accessed: 10 March 2019].

Faruki, P., Laxmi, V., Gaur, M. S., and Vinod, P. (2012). "Behavioural detection with API call-grams to identify malicious PE files". In *Proc. First International Conference on Security of Internet of Things*, 85–91. ACM.

Firdausi, I., Erwin, A., Nugroho, A. S., et al. (2010). "Analysis of machine learning techniques used in behavior-based malware detection". In *Proc. Second International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT)*, 201–203. IEEE.

Frank, E., Hall, M., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I. H., and Trigg, L. (2009). "Weka-a machine learning workbench for data mining". In *Data mining and knowledge discovery handbook* 1269–1277. Springer.

Fujino, A., Murakami, J., and Mori, T. (2015). "Discovering Similar Malware Samples Using API Call Topics". In *Proc. 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, 140–147. IEEE.

Gandotra, E., Bansal, D., and Sofat, S. (2014). Integrated framework for classification of malwares. In *Proc. 7th International Conference on Security of Information and Networks*, 417:417–417:422. ACM.

Geden, M. and Happa, J. (2018). "Classification of Malware Families Based on Runtime Behaviour". In *Proc. International Symposium on Cyberspace Safety and Security*, volume 11161, 33–48. Springer.

Goodall, J. R., Radwan, H., and Halseth, L. (2010). "Visual Analysis of Code Security". In *Proc. 7th International Symposium on Visualization for Cyber Security (VizSec)*, 46–51. ACM.

Guarnieri, C., Tanasi, A., Bremer, J., and Schloesser, M. (2012), "Automated Malware Analysis-Cuckoo Sandbox". https://cuckoosandbox.org/.

Gulgezen, G., Cataltepe, Z., and Yu, L. (2009). "Stable and accurate feature selection". In *Proc. Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 5781, 455–468. Springer.

HaddadPajouh, H., Dehghantanha, A., Khayami, R., and Choo, K.-K. R. (2018). "A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting". *Future Generation Computer Systems*, *85*, 88–96.

Han, K. S., Lim, J. H., Kang, B., and Im, E. G. (2015). "Malware analysis using visualized images and entropy graphs". *International Journal of Information Security*, *14*(1), 1–14.

Hansen, S. S., Larsen, T. M. T., Stevanovic, M., and Pedersen, J. M. (2016). "An approach for detection and family classification of malware based on behavioral analysis". In *Proc. International Conference on Computing, Networking and Communications (ICNC)*, 1–5. IEEE.

Hardy, W., Chen, L., Hou, S., Ye, Y., and Li, X. (2016). "DL4MD: A deep learning framework for intelligent malware detection". In *Proc. International Conference on Data Mining (DMIN)*, 61–67. WorldComp.

Huda, S., Abawajy, J., Alazab, M., Abdollalihian, M., Islam, R., and Yearwood, J. (2016). "Hybrids of support vector machine wrapper and filter based framework for malware detection". *Future Generation Computer Systems*, *55*, 376–390.

Huda, S., Islam, R., Abawajy, J., Yearwood, J., Hassan, M. M., and Fortino, G. (2018). "A hybrid-multi filter-wrapper framework to identify run-time behaviour for fast malware detection". *Future Generation Computer Systems*, *83*, 193–207.

Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). "Classification of malware based on integrated static and dynamic features". *Journal of Network and Computer Applications*, *36*(2), 646–656.

Jain, A. and Singh, A. K. (2017). "Integrated Malware analysis using machine learning". In *Proc. 2nd International Conference on Telecommunication and Networks (TELNET)*, 1–8. IEEE.

Jin, W. and Srihari, R. K. (2007). "Graph-based text representation and knowledge discovery". In *Proc. ACM symposium on Applied computing*, 807–811. ACM.

Kakisim, A. G., Nar, M., Carkaci, N., and Sogukpinar, I. (2018). "Analysis and Evaluation of Dynamic Feature-Based Malware Detection Methods". In *Proc. International Conference on Security for Information Technology and Communications*, volume 11359, 247–258. Springer.

Kan, Z., Wang, H., Xu, G., Guo, Y., and Chen, X. (2018). "Towards Light-weight Deep Learning based Malware Detection". In *Proc. 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 600–609. IEEE.

Kancherla, K. and Mukkamala, S. (2013). "Image Visualization based Malware Detection". In *Proc. IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, 40–44. IEEE.

Kawaguchi, N. and Omote, K. (2015). "Malware function classification using APIs in initial behavior". In *Proc. 10th Asia Joint Conference on Information Security (AsiaJCIS)*, 138–144. IEEE.

Khoshgoftaar, T. M., Golawala, M., and Van Hulse, J. (2007). "An empirical study of learning from imbalanced data using random forest". In *Proc. 19th IEEE international conference on Tools with Artificial Intelligence (ICTAI)*, volume 2, 310–317. IEEE.

Kolosnjaji, B., Eraisha, G., Webster, G., Zarras, A., and Eckert, C. (2017). "Empowering Convolutional Networks for Malware Classification and Analysis". In *Proc. International Joint Conference on Neural Networks (IJCNN)*, 3838–3845. IEEE.

Kolosnjaji, B., Zarras, A., Webster, G., and Eckert, C. (2016). "Deep Learning for Classification of Malware System Call Sequences". In *Proc. Australasian Joint Conference on Artificial Intelligence*, volume 9992, 137–149. Springer.

Kolter, J. Z. and Maloof, M. A. (2006). "Learning to Detect and Classify Malicious Executables in the Wild". *Journal of Machine Learning Research*, *7*, 2721–2744.

Kononenko, I. (1994). "Estimating Attributes: Analysis and Extensions of RELIEF". In *Proc. European Conference on Machine Learning (ECML)*, volume 784, 171–182. Springer.

Konrad, R. (2015), *"Malheur Dataset"*. https://github.com/rieck/malheur/commit/801410c653e41782d04139972e178288dd09bac1 [Accessed: 29 October 2016].

Kumar, A., Kuppusamy, K., and Aghila, G. (2019). "A learning model to detect maliciousness of portable executable using integrated feature set". *Journal of King Saud University-Computer and Information Sciences*, *31*, 252–265.

Kumar, R., Xiaosong, Z., Khan, R. U., Ahad, I., and Kumar, J. (2018). "Malicious Code Detection based on Image Processing Using Deep Learning". In *Proc. International Conference on Computing and Artificial Intelligence (ICCAI)*, 81–85. ACM.

Lee, J., Im, C., and Jeong, H. (2011). "A study of malware detection and classification by comparing extracted strings". In *Proc. 5th International Conference on Ubiquitous Information Management and Communication*, 75:1–75:4. ACM.

Leistner, C., Saffari, A., Santner, J., and Bischof, H. (2009). "Semi-supervised random forests". In *Proc. 12th International Conference on Computer Vision*, 506–513. IEEE.

Lengyel, T. K., Maresca, S., Payne, B. D., Webster, G. D., Vogl, S., and Kiayias, A. (2014). "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system". In *Proc. 30th Annual Computer Security Applications Conference*, 386–395. ACM.

Li, Y., Li, T., and Liu, H. (2017). "Recent advances in feature selection and its applications". *Knowledge and Information Systems*, *53*(3), 551–577.

Lin, C.-H., Pao, H.-K., and Liao, J.-W. (2018). "Efficient dynamic malware analysis using virtual time control mechanics". *Computers and Security*, *73*, 359–373.

Lindorfer, M., Kolbitsch, C., and Comparetti, P. M. (2011). "Detecting environment-sensitive malware". In *Proc. International Workshop on Recent Advances in Intrusion Detection*, volume 6961, 338–357. Springer.

Malwr (2010), *"Malwr - Malware Analysis by Cuckoo Sandbox"*. https://malwr.com [Accessed: 12 September 2016].

Masud, M. M., Khan, L., and Thuraisingham, B. (2008). "A scalable multi-level feature extraction technique to detect malicious executables". *Information Systems Frontiers*, *10*(1), 33–45.

Miller, C., Glendowne, D., Cook, H., Thomas, D., Lanclos, C., and Pape, P. (2017). "Insights gained from constructing a large scale dynamic analysis platform". *Digital Investigation*, *22*, S48–S56.

Mohaisen, A., Alrawi, O., and Mohaisen, M. (2015). "AMAL: High-fidelity, behavior-based automated malware analysis and classification". *Computers and Security*, *52*, 251–266.

Moser, A., Kruegel, C., and Kirda, E. (2007). "Limits of Static Analysis for Malware Detection". In *Proc. 23rd Annual Computer Security Applications Conference (ACSAC)*, 421–430. IEEE.

Moskovitch, R., Elovici, Y., and Rokach, L. (2008). "Detection of unknown computer worms based on behavioral classification of the host". *Computational Statistics and Data Analysis*, *52*(9), 4544–4566.

Moskovitch, R., Feher, C., Tzachar, N., Berger, E., Gitelman, M., Dolev, S., and Elovici, Y. (2008). "Unknown malcode detection using opcode representation". In *Proc. European conference on intelligence and security informatics*, volume 5376, 204–215. Springer.

Narouei, M., Ahmadi, M., Giacinto, G., Takabi, H., and Sami, A. (2015). "DLLMiner: structural mining for malware detection". *Security and Communication Networks*, *8*(18), 3311–3322.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). "Malware Images: Visualization and Automatic Classification". In *Proc. 8th International Symposium on Visualization for Cyber Security (VizSec)*, 4:1–4:7. ACM.

Ni, S., Qian, Q., and Zhang, R. (2018). "Malware Identification Using Visualization Images and Deep Learning". *Computers and Security*, *77*, 871–885.

OKane, P., Sezer, S., and McLaughlin, K. (2011). "Obfuscation: The Hidden Malware". *IEEE Security Privacy*, *9*(5), 41–47.

Oshiro, T. M., Perez, P. S., and Baranauskas, J. A. (2012). "How many trees in a random forest?". In *Proc. International Workshop on Machine Learning and Data Mining in Pattern Recognition*, volume 7376, 154–168. Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research*, *12*, 2825–2830.

Pektaş, A. and Acarman, T. (2018). "Malware classification based on API calls and behaviour analysis". *IET Information Security*, *12*(2), 107–117.

Pektaş, A., Eriş, M., and Acarman, T. (2011). "Proposal of n-gram based algorithm for malware classification". In *Proc. The Fifth International Conference on Emerging Security Information, Systems and Technologies*, 7–13. IARIA.

Peng, H., Long, F., and Ding, C. (2005). "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". *IEEE Transactions on pattern analysis and machine intelligence*, *27*(8), 1226–1238.

Qiao, Y., Yang, Y., He, J., Tang, C., and Liu, Z. (2014). "CBM: Free, Automatic Malware Analysis Framework Using API Call Sequences". In *Proc. Knowledge Engineering and Management*, volume 214, 225–236. Springer.

Quist, D. A. and Liebrock, L. M. (2009). "Visualizing Compiled Executables for Malware Analysis". In *Proc. 6th International Workshop on Visualization for Cyber Security (VizSec)*, 27–32. IEEE.

Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., Tracy, A., McLean, M., and Nicholas, C. (2018). "An investigation of byte n-gram features for malware classification". *Journal of Computer Virology and Hacking Techniques*, *14*(1), 1–20.

Reddy, D. K. S. and Pujari, A. K. (2006). "N-gram analysis for computer virus detection". *Journal in Computer Virology*, *2*(3), 231–239.

Rhode, M., Burnap, P., and Jones, K. (2018). "Early-Stage Malware Prediction Using Recurrent Neural Networks". *Computers and Security*, *77*, 578–594.

Rieck, K., Holz, T., Willems, C., Düssel, P., and Laskov, P. (2008). "Learning and Classification of Malware Behavior". In *Proc. 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 5137, 108–125. Springer.

Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). "Automatic analysis of malware behavior using machine learning". *Journal of Computer Security*, *19*(4), 639–668.

Saeed, I. A., Selamat, A., and Abuagoub, A. M. (2013). "A survey on malware and malware detection systems". *International Journal of Computer Applications*, *67*(16), 25–31.

Sakar, C. O., Kursun, O., and Gurgen, F. (2012). "A feature selection method based on kernel canonical correlation analysis and the minimum Redundancy–Maximum Relevance filter method". *Expert Systems with Applications*, *39*(3), 3432–3437.

Salehi, Z., Sami, A., and Ghiasi, M. (2014). "Using feature generation from API calls for malware detection". *Computer Fraud and Security*, *2014*(9), 9–18.

Salehi, Z., Sami, A., and Ghiasi, M. (2017). "MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values". *Engineering Applications of Artificial Intelligence*, *59*, 93–102.

Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S., and Hamze, A. (2010). "Malware detection based on mining API calls". In *Proc. ACM symposium on applied computing*, 1020–1025. ACM.

Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., and Bringas, P. G. (2010). "Idea: Opcode-sequence-based malware detection". In *Proc. International Symposium on Engineering Secure Software and Systems*, volume 5965, 35–43. Springer.

Santos, I., Brezo, F., Ugarte-Pedrero, X., and Bringas, P. G. (2013). "Opcode sequences as representation of executables for data-mining-based unknown malware detection". *Information Sciences*, *231*, 64–82.

Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. (2013). "Opem: A static-dynamic approach for machine-learning-based malware detection". In *Proc. International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*, volume 189, 271–280. Springer.

Saxe, J. and Berlin, K. (2015). "Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features". In *Proc. 10th International Conference on Malicious and Unwanted Software*, 11–20. IEEE.

Schultz, M. G., Eskin, E., Zadok, F., and Stolfo, S. J. (2001). "Data Mining Methods for Detection of New Malicious Executables". In *Proc. Symposium on Security and Privacy (SandP)*, 38–49. IEEE.

Sethi, K., Chaudhary, S. K., Tripathy, B. K., and Bera, P. (2018). "A Novel Malware Analysis Framework for Malware Detection and Classification using Machine Learning Approach". In *Proc. 19th International Conference on Distributed Computing and Networking (ICDCN)*, 49:1–49:4. ACM.

Shabtai, A., Moskovitch, R., Elovici, Y., and Glezer, C. (2009). "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey". *information security technical report*, *14*(1), 16–29.

Sharma, A. and Sahay, S. K. (2014). "Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey". *International Journal of Computer Applications*, *90*(2), 7–11.

Shijo, P. and Salim, A. (2015). "Integrated static and dynamic analysis for malware detection". *Procedia Computer Science*, *46*, 804–811.

Simeon, M. and Hilderman, R. (2008). "Categorical proportional difference: A feature selection method for text categorization". In *Proc. 7th Australasian Data Mining Conference*, volume 87, 201–208. Australian Computer Society, Inc.

Singh, B., Kushwaha, N., Vyas, O. P., et al. (2014). "A Feature Subset Selection Technique for High Dimensional Data Using Symmetric Uncertainty". *Journal of Data Analysis and Information Processing*, *2*(04), 95–105.

Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., and Yagi, T. (2016). "Malware Detection with Deep Neural Network Using Process Behavior". In *Proc. 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, 577–582. IEEE.

Trinius, P., Holz, T., Gobel, J., and Freiling, F. C. (2009). "Visual Analysis of Malware Behavior Using Treemaps and Thread Graphs". In *Proc. 6th International Workshop on Visualization for Cyber Security (VizSec)*, 33–38. IEEE.

Tsyganok, K., Tumoyan, E., Babenko, L., and Anikeev, M. (2012). "Classification of polymorphic and metamorphic malware samples based on their behavior". In *Proc. Fifth International Conference on Security of Information and Networks*, 111–116. ACM.

urwithajit9 (2016), *"ClaMP (Classification of Malware with PE headers)"*. https://github.com/urwithajit9/ClaMP [Accessed: 6 February 2018].

Uysal, A. K. and Gunal, S. (2012). "A novel probabilistic feature selection method for text classification". *Knowledge-Based Systems*, *36*, 226–235.

Vinod, P., Laxmi, V., and Gaur, M. S. (2011). "Scattered feature space for malware analysis". In *Proc. International Conference on Advances in Computing and Communications*, volume 190, 562–571. Springer.

VirusShare (2011), *"VirusShare.com - Because Sharing is Caring"*. https://virusshare.com [Accessed: 23 March 2017].

VirusTotal (2004a), *"VirusTotal"*. https://www.virustotal.com/ [Accessed: 26 March 2017].

VirusTotal (2004b), *"VirusTotal File statistics"*. https://www.virustotal.com/en/statistics/ [Accessed on January 2018].

Willems, C., Holz, T., and Freiling, F. (2007). "Toward Automated Dynamic Malware Analysis Using CWSandbox". *IEEE Security and Privacy*, *5*(2), 32–39.

Yakura, H., Shinozaki, S., Nishimura, R., Oyama, Y., and Sakuma, J. (2018). "Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism". In *Proc. 8th ACM Conference on Data and Application Security and Privacy (CODASPY)*, 127–134. ACM.

Yan, G., Brown, N., and Kong, D. (2013). "Exploring Discriminatory Features for Automated Malware Classification". In *Proc. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 7967, 41–61. Springer.

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

Yang, Y. and Pedersen, J. O. (1997). "A Comparative Study on Feature Selection in Text Categorization". In *Proc. 14th International Conference on Machine Learning (ICML)*, 412–420. Morgan Kaufmann Publishers Inc.

Ye, Y., Chen, L., Wang, D., Li, T., Jiang, Q., and Zhao, M. (2009). "SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging". *Journal in computer virology*, *5*(4), 283–293.

Ye, Y., Li, T., Chen, Y., and Jiang, Q. (2010). "Automatic malware categorization using cluster ensemble". In *Proc. 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 95–104. ACM.

Ye, Y., Li, T., Jiang, Q., and Wang, Y. (2010). "CIMDS: adapting postprocessing techniques of associative classification for malware detection". *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *40*(3), 298–307.

Ye, Y., Wang, D., Li, T., Ye, D., and Jiang, Q. (2008). "An intelligent PE-malware detection system based on association mining". *Journal in computer virology*, *4*(4), 323–334.

Yoo, I. (2004). "Visualizing Windows Executable Viruses Using Self-Organizing Maps". In *Proc. ACM workshop on Visualization and data mining for computer security*, 82–89. ACM.

You, I. and Yim, K. (2010). "Malware obfuscation techniques: A brief survey". In *Proc. International conference on broadband, wireless computing, communication and applications*, 297–300. IEEE.

Zhang, J., Qin, Z., Yin, H., Ou, L., and Hu, Y. (2016). "IRMD: Malware Variant Detection using opcode Image Recognition". In *Proc. 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, 1175–1180. IEEE.

Zhang, Y., Huang, Q., Ma, X., Yang, Z., and Jiang, J. (2016). "Using Multi-features and Ensemble Learning Method for Imbalanced Malware Classification". In *Proc. IEEE Trustcom-BigDataSE-ISPA*, 965–973. IEEE.

Publications

Journal Papers

1. **Shiva Darshan S.L** and Jaidhar C.D (2018), "Windows Malware Detection System based on LSVC Recommended Hybrid Features", ***Journal of Computer Virology and Hacking Techniques***, 15(2), 127-146, DOI:10.1007/s11416-018-0327-9.

2. **Shiva Darshan S.L** and Jaidhar C.D (2019), "Windows Malware Detector using Convolutional Neural Network based on Visualization Images", ***IEEE Transactions on Emerging Topics in Computing***, DOI: 10.1109/TETC.2019.2910086.

3. **Shiva Darshan S.L** and Jaidhar C.D (2019), "An Empirical Study to Estimate the Stability of Random Forest Classifier on the Hybrid Features Recommended by Filter Based Feature Selection Technique", ***International Journal of Machine Learning and Cybernetics***, DOI: https://doi.org/10.1007/s13042-019-00978-7

Conference Papers

1. **Shiva Darshan S.L**, Ajay Kumara M.A, and Jaidhar C.D (2016), "Windows Malware Detection based on Cuckoo Sandbox Generated Report using Machine Learning Algorithm", In *Proc. of the* 11^{th} ***International Conference on Industrial and Information Systems****, (ICIIS 2016)*, 534-539, DOI: 10.1109/ICIINFS.2016.8262998.

2. **Shiva Darshan S.L**, Ajay Kumara M.A, and Jaidhar C.D (2017), "Information Gain Score Computation for N-Grams using Multiprocessing Model", In *Proc. of the* ***Asia Security and Privacy Conference****, (ISEASP 2017)*, 1-7, DOI: 10.1109/ISEASP.2017.7976984.

3. **Shiva Darshan S.L** and Jaidhar C.D (2017), "Empirical Study on Features Recommended by LSVC in Classifying Unknown Windows Malware", In *Proc. of the* 7^{th} ***International Conference on Soft Computing for Problem Solving****, (SocPros 2017)*, 577-590, DOI: https://doi.org/10.1007/978-981-13-1595-4_46.

4. **Shiva Darshan S.L** and Jaidhar C.D (2018), "Performance Evaluation of Filter-based Feature Selection Techniques in Classifying Portable Executable Files", In *Proc. of the* 6^{th} ***International Conference on Smart Computing and Communications****, (ICSCC 2018)*, 346-356, DOI: https://doi.org/10.1016/j.procs.2017.12.046.

Curriculum Vitae

Mr. Shiva Darshan S.L
Full-Time Research Scholar
Department of Information Technology
National Institute of Technology Karnataka
P.O. Srinivasanagar, Surathkal
Mangalore-575 025

Permanent Address

Shiva Darshan S.L
#49, Banashankari Nilaya
1^{st} Main, 1^{st} Cross, K.R Extension
Tiptur (Post), Tumkur (Dist)
Karnataka (State), PIN-572 201
Email: darshkottur@gmail.com
Mobile: +91-9743413637.

Academic Records

1. M.Tech. in Computer Science and Engineering, Jawaharlal Nehru National College of Engineering (JNNCE) Shivamogga, 2013.

2. B.E. in Computer Science and Engineering, Kalpataru Institute of Technology (KIT), Tiptur, 2011.

Research Interests

Malware Detection Techniques
Machine Learning
Deep Learning
Feature Selection Techniques

Programming Languages

C, CPP, Java, Matlab, Python, Scripts.