Automated Malware Detection and Classification Using Supervised Learning

Raja Muhammad Khurram Shahzad

Department of Computer Science
Blekinge Institute of Technology

Doctoral Dissertation Series no. 2024:03
Malware has been one of the key concerns for Information Technology security researchers for decades. Every year, anti-malware companies release alarming statistics suggesting a continuous increase in the number and types of malware. This is mainly due to the constant development of new and more sophisticated malicious functionalities, propagation vectors, and infection tactics for malware. To combat this ever-evolving threat, anti-malware companies analyze thousands of malicious samples on a daily basis, either manually or through semi-automated means, to identify their type (whether it’s a variant or zero-day) and family. After the analysis, signature databases or rule databases of anti-malware products are updated in order to detect known malware. However, due to the ever-growing capabilities of malware, the malware analysis process is challenging and requires significant human effort. As a result, researchers are focusing on data-driven approaches based on machine learning to develop intelligent malware detectors with high accuracy. Specifically, they are focused on extracting static features from malware in the form of n-grams for experimental purposes. However, the previous research is inconclusive in terms of optimal feature representation and detection accuracy. Thus, the primary objective of this thesis is to present state-of-the-art automated techniques for detecting and classifying malware using supervised learning algorithms. In particular, the focus is on two critical aspects of supervised learning-based malware detection: optimal feature representation and improved detection accuracy.

Malware detection can be accomplished using two methods: static analysis, which extracts patterns without executing malware, and dynamic analysis, which captures behaviors through executing malware. This thesis focuses on static analysis instead of dynamic analysis because static analysis requires fewer computing resources. An additional benefit of static analysis is that present-day malware cannot evade it. To achieve the goals of this thesis, two new feature representations for static analysis are proposed. Furthermore, three customized ensembles are introduced to enhance malware detection accuracy, and their feasibility is experimentally demonstrated.

The experiments incorporate customized malware data sets, including Spyware, Adware, Scareware, and Android malware samples, and public malware data sets from Microsoft having samples from nine distinct malware families. Artificially generated data sets are employed to mitigate class imbalance issues and represent inter-family and intra-family examples. Reverse engineering is performed to transform the data sets as feature data sets using both byte code and assembly language instructions. Further, existing and new feature representations along with various feature selection algorithms and feature fusion techniques are explored. To enhance detection accuracy, different decision theories from social choice theory, such as veto and consensus, are integrated into customized ensembles. The experimental results indicate that the proposed methods are capable of detecting known and zero-day malware. The proposed ensembles are also tested on the UCI public data sets, such as Forest CoverType, and the results demonstrate their effectiveness in classification. Further, these methods are designed to be portable and adaptable to different operating systems, and they can also be scaled for multi-class malware detection.
Automated Malware Detection and Classification Using Supervised Learning

Raja Muhammad Khurram Shahzad
Abstract

Malware has been one of the key concerns for Information Technology security researchers for decades. Every year, anti-malware companies release alarming statistics suggesting a continuous increase in the number and types of malware. This is mainly due to the constant development of new and more sophisticated malicious functionalities, propagation vectors, and infection tactics for malware. To combat this ever-evolving threat, anti-malware companies analyze thousands of malicious samples on a daily basis, either manually or through semi-automated means, to identify their type (whether it's a variant or zero-day) and family. After the analysis, signature databases or rule databases of anti-malware products are updated in order to detect known malware. However, due to the ever-growing capabilities of malware, the malware analysis process is challenging and requires significant human effort. As a result, researchers are focusing on data-driven approaches based on machine learning to develop intelligent malware detectors with high accuracy. Specifically, they are focused on extracting static features from malware in the form of \( n \)-grams for experimental purposes. However, the previous research is inconclusive in terms of optimal feature representation and detection accuracy. Thus, the primary objective of this thesis is to present state-of-the-art automated techniques for detecting and classifying malware using supervised learning algorithms. In particular, the focus is on two critical aspects of supervised learning-based malware detection: optimal feature representation and improved detection accuracy.

Malware detection can be accomplished using two methods: static analysis, which extracts patterns without executing malware, and dynamic analysis, which captures behaviors through executing malware. This thesis focuses on static analysis instead of dynamic analysis because static analysis requires fewer computing resources. An additional benefit of static analysis is that present-day malware cannot evade it. To achieve the goals of this thesis, two new feature representations for static analysis are proposed. Furthermore, three customized ensembles are introduced to enhance malware detection accuracy, and their feasibility is experimentally demonstrated.

The experiments incorporate customized malware data sets, including Spyware, Adware, Scareware, and Android malware samples, and public malware data sets from Microsoft having samples from nine distinct malware families. Artificially generated data sets are employed to mitigate class imbalance issues and represent inter-family and intra-family examples. Reverse engineering is performed to transform the data sets as feature data sets using both byte code and assembly language instructions. Further, existing and new feature representations along with various feature selection algorithms and feature fusion techniques are explored. To enhance detection accuracy, different decision theories from social choice theory, such as veto and consensus, are integrated into customized ensembles. The experimental results indicate that the proposed methods are capable of detecting known and zero-day malware. The proposed ensembles are also tested on the UCI public data sets, such as Forest CoverType, and the results demonstrate their effectiveness in classification. Further, these methods are designed to be portable and adaptable to different operating systems, and they can also be scaled for multi-class malware detection.

**Keywords:** Malicious software, Malware detection, Malware classification, Static Analysis, Supervised learning, Ensemble learning.
Automated Malware Detection and Classification Using Supervised Learning

Raja Muhammad Khurram Shahzad

Doctoral Dissertation in Computer Science

Department of Computer Science
Blekinge Institute of Technology
SWEDEN
Abstract

Malware has been one of the key concerns for Information Technology security researchers for decades. Every year, anti-malware companies release alarming statistics suggesting a continuous increase in the number and types of malware. This is mainly due to the constant development of new and more sophisticated malicious functionalities, propagation vectors, and infection tactics for malware. To combat this ever-evolving threat, anti-malware companies analyze thousands of malicious samples on a daily basis, either manually or through semi-automated means, to identify their type (whether it’s a variant or zero-day) and family. After the analysis, signature databases or rule databases of anti-malware products are updated in order to detect known malware. However, due to the ever-growing capabilities of malware, the malware analysis process is challenging and requires significant human effort. As a result, researchers are focusing on data-driven approaches based on machine learning to develop intelligent malware detectors with high accuracy. Specifically, they are focused on extracting static features from malware in the form of \( n \)-grams for experimental purposes. However, the previous research is inconclusive in terms of optimal feature representation and detection accuracy. Thus, the primary objective of this thesis is to present state-of-the-art automated techniques for detecting and classifying malware using supervised learning algorithms. In particular, the focus is on two critical aspects of supervised learning-based malware detection: optimal feature representation and improved detection accuracy.

Malware detection can be accomplished using two methods: static analysis, which extracts patterns without executing malware, and dynamic analysis, which captures behaviors through executing malware. This thesis focuses on static analysis instead of dynamic analysis because static analysis requires fewer computing resources. An additional benefit of static analysis is that present-day malware cannot evade it. To achieve the goals of this thesis, two new feature representations for static analysis are proposed. Furthermore, three customized ensembles are introduced to enhance
malware detection accuracy, and their feasibility is experimentally demonstrated.

The experiments incorporate customized malware data sets including Spyware, Adware, Scareware, and Android malware samples, and public malware data sets from Microsoft having samples from nine distinct malware families. Artificially generated data sets are employed to mitigate class imbalance issues and represent inter-family and intra-family examples. Reverse engineering is performed to transform the data sets as feature data sets using both byte code and assembly language instructions. Further, existing and new feature representations along with various feature selection algorithms and feature fusion techniques are explored. To enhance detection accuracy, different decision theories from social choice theory, such as veto and consensus, are integrated into customized ensembles. The experimental results indicate that the proposed methods are capable of detecting known and zero-day malware. The proposed ensembles are also tested on the UCI public data sets, such as Forest CoverType, and the results demonstrate their effectiveness in classification. Further, these methods are designed to be portable and adaptable to different operating systems, and they can also be scaled for multi-class malware detection.
I owe a high debt of gratitude and thanks to my primary advisor, Professor Niklas Lavesson, for his trust and for triggering my interest in Machine Learning. Throughout my research work, Niklas has been a great mentor and a source of encouragement. He has provided advice and guidance and taught me innumerable things, from basic research to developing new ideas. He has inculcated in me how to approach each research problem from a different angle and ascertain the validity of the applied approach. Thank you, Niklas.

I am indebted to Professor Bengt Carlsson whose unwavering support, inspiration, and guidance have been instrumental in shaping the quality of my research. His ideas and feedback have been crucial in guiding my research, and for that, I am grateful. I am also grateful to my supervisor, Dr. Martin Boldt, for his timely corrections, insights, and discussions that have been instrumental in my research journey.

Lastly, I’m grateful for the unwavering support of my beloved parents and family, who have stood by me throughout my Ph.D. journey, especially during the most challenging times. Their encouragement and support have been a constant source of inspiration, and I’m grateful for their unwavering love and support.
The thesis comprises of seven papers, which have been published in peer-reviewed conference proceedings and journals. The author of the thesis is also the main author of all the papers. While the included versions of Paper I, Paper II, Paper III, and Paper VII contain minor modifications and expansions in the text, they are made solely to correct any errors found in the original publications.


As the principal author, the thesis author is responsible for conducting research studies, performing experiments, and writing papers. Paper I is an extension of the author’s master thesis and is co-authored by a fellow student and the master thesis’s supervisor. Papers II, III, IV, and V are co-authored by senior researchers at Blekinge Institute of Technology. These co-authors have provided feedback and constructive advice on research study design, analysis, and paper writing to help organize and present the work effectively.
Contents

Acknowledgments iii
Preface v
Contents vii

1 Introduction 1
  1.1 Background 1
  1.2 Terminology 8
  1.3 Thesis Outline and Structure 11

2 Approach 13
  2.1 Related Work 13
  2.2 Aim & Scope 16
  2.3 Research Questions 16
  2.4 Methodology 17
  2.5 Validity Threats 19

3 Results 23
  3.1 Results and Contributions 23
  3.2 Discussion 30
  3.3 Lessons Learned 33
3.4 Conclusions and Future Work ........................................... 35

4 Detection of Spyware by Mining Executable Files .......................... 37
*Raja Khurram Shahzad, Syed Imran Haider, Niklas Lavesson*
4.1 Introduction ............................................................... 38
4.2 Background .............................................................. 41
4.3 Proposed Method .......................................................... 45
4.4 Results ................................................................. 51
4.5 Discussion ............................................................... 54
4.6 Conclusions and Future Work ........................................... 56

5 Accurate Adware Detection using OpCode Sequence Extraction .............. 59
*Raja Khurram Shahzad, Niklas Lavesson, Henric Johnson*
5.1 Introduction .............................................................. 60
5.2 Background .............................................................. 62
5.3 Related Work ............................................................ 64
5.4 Method ................................................................. 65
5.5 Evaluation Metrics ......................................................... 73
5.6 Experimental Procedure ................................................ 74
5.7 Results ................................................................. 74
5.8 Analysis ................................................................. 75
5.9 Conclusion and Future work ............................................. 78

6 Detecting Scareware by Mining Variable Length Instruction Sequences .......... 81
*Raja Khurram Shahzad, Niklas Lavesson*
6.1 Introduction .............................................................. 82
6.2 Related Work ............................................................ 87
6.3 Methodology ............................................................. 91
6.4 Experiment ............................................................. 94
6.5 Results ................................................................. 97
6.6 Analysis ................................................................. 99
6.7 Conclusions and Future work ........................................... 100


CONTENTS

7 Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection 103
Raja Khurram Shahzad, Niklas Lavesson
7.1 Introduction ............................................ 104
7.2 Background ............................................. 107
7.3 Veto-based Classification ............................. 111
7.4 Pre-processing Techniques ......................... 115
7.5 Experiment I .......................................... 117
7.6 Experiment II ......................................... 126
7.7 Conclusion and Future Work ...................... 133

8 Consensus Decision Making in Random Forests 135
Raja Khurram Shahzad, Mehwish Fatima, Niklas Lavesson, Martin Boldt
8.1 Introduction ............................................ 136
8.2 Background ............................................. 137
8.3 Consensus Decision-Making in Random Forests Algorithm 139
8.4 Experiment .......................................... 143
8.5 Results and Analysis ................................ 147
8.6 Conclusion ............................................ 149

9 A Hybrid Approach For Malware Classification Using Secondary Features Fusion 151
Raja Khurram Shahzad, Muhammad Mustaqeem, Haroon Elahi
9.1 Introduction ............................................ 152
9.2 Experimental Setup .................................. 156
9.3 Experiments .......................................... 168
9.4 Results and Discussion ............................. 169
9.5 Conclusion ............................................ 175

10 Android Malware Detection Using Feature Fusion and Artificial Data 177
Raja Khurram Shahzad
10.1 Introduction ............................................ 178
10.2 Background .......................................... 181
10.3 Theoretical Explanation .................................. 182
10.4 Experiment ............................................. 188
10.5 Results and Analysis ..................................... 193
10.6 Conclusion and Future Work ........................... 198

References ...................................................... 199
Chapter 1

Introduction

This thesis focuses on data-driven approaches to detect malware. The broader task addressed in this thesis is software classification within the information technology (IT) security domain. The thesis investigates how to classify/detect several types of malicious software (malware). The thesis addresses one task-oriented applied IT security research question and two solution-oriented applied machine learning research questions.

Information Technology security has become crucial for organizations and individuals in securing business and personal information across computing devices. The failure of IT security may have severe repercussions on business processes, reputation, and business relations for companies and personal images, and personal ties for individuals. One of the primary and growing threats against IT security is malicious software (malware), which is aimed at corporate or personal computing resources. The computing resources consist of computers, mobile phones, and tablets running Windows operating system (OS), iOS, and Android OS [1]. During the pandemic, most companies reshaped their employees’ working atmosphere and supported remote working or working from home. This change has placed invariable pressure on
1. **Introduction**

Companies’ IT infrastructure to provide secure access to corporate resources. Malware authors have leveraged the situation for their benefit and have been creating and distributing malware at an alarming rate [2].

Over the past few decades, the techniques employed by malware to propagate and launch attacks have become more sophisticated. These methods are utilized to infiltrate computer systems and execute harmful and unlawful acts, putting the safety of saved data and computational resources at risk, frequently leading to financial gain for the perpetrators [3, 4, 5]. Thus, motivated by the prospect of monetary benefits, malware writers are increasingly spreading malware samples to infect computing devices. This is evident from recent reports published by different anti-malware companies, such as MalwareBytes\(^1\), Sophos\(^2\), TrendMicro\(^3\), Avast\(^4\), etc., about different malware such as Spyware and Adware. In March 2023, Trend Micro has reported the detection of 146 billion threats for their customers in 2022, which highlights a 55% increase from the previous year [6]. Furthermore, Avast and Malware Bytes’s reports indicate a rising spread of Spyware and Adware in the last quarter of 2022, while Sophos warns of Spyware becoming an increasing threat for Android-based devices [7, 8, 9].

Malware can be divided into two broad categories: traditional malware, which consists of viruses, worms, and Trojan horses, and programs that have potentially unwanted functionality, commonly referred to as PUPs. Traditional malware is destructive in nature and can cause extensive damage to the data stored on computer systems. On the contrary, the objective of an undesirable function can have a wide spectrum ranging from pursuing financial gain to gaining strategic or tactical advantages by compromising users’ information. It is worth noting that the emergence of malware construction kits\(^5\), such as the "Next Genera-

\(^1\)https://www.malwarebytes.com
\(^2\)https://www.sophos.com
\(^3\)https://www.trendmicro.com
\(^4\)https://www.avast.com
tion Virus Creation Kit”, has made it possible for even the most inexperienced individuals to produce harmful programs that can wreak havoc on unsuspecting victims. These kits are capable of generating PUPs that can infiltrate and compromise individuals’ personal banking information [10]. Moreover, malware authors have used a combination of different propagation vectors and attack strategies to develop different types of malware, especially PUPs like Trojans, Spyware, adware, scareware, and ransomware, to target both Windows-based personal computers and Android-based smartphones [11]. This situation has warranted the need for automated malware analysis and classification systems. Historically, the anti-malware industry has addressed malware detection problem through a reactive approach. This typically involves using either signature-based detection, which involves pattern matching, or heuristic-based detection, which relies on a set of predetermined rules. These approaches can detect circa 30%, and 39% of malware instances in the wild [12, 13]. These approaches, however, are incapable of detecting zero-day malware. Thus, there is an arms race between the malware writers and the signature and rule producers.

To address problems in heuristic-based approaches, researchers have investigated the usage of machine learning, specifically supervised learning, to enhance the efficacy of malware detection methods. In supervised learning, the malware detection task is generally approached as a two-step process, i.e., feature extraction and classification [5]. It is expected that class-indicating features can be extracted from both benign and malware samples. These features can be utilized in learning algorithms to construct accurate classifiers that can detect both known and zero-day malware.

This thesis focuses on extending heuristic-based detection by using supervised learning. The challenges of supervised learning for the malware detection task lie in the feature extraction (data preparation) step and in obtaining the accuracy of the classifiers for classifying the given samples into distinct classes using the extracted features.
This thesis addresses different aspects of data preparation and improving the classifier’s accuracy as follows:

**C1:** This thesis contributes with respect to the representation of a reverse-engineered binary file in terms of extracted features such as fixed size $n$-grams, non-adjacent $n$-grams, variable size $n$-grams, and variable-length $n$-grams.

**C2:** Multiple feature selection methods are investigated for malware detection. Moreover, subsequent feature selection (primary and secondary feature selection) and feature fusion are also investigated.

**C3:** This thesis addresses the challenges associated with the class imbalance problem and the problem of the absence of inter-family or intra-family examples using artificially generated data.

**C4:** To improve the classification accuracy, this thesis demonstrates the usage of customized ensembles and investigates the different decision strategies from the field of the social choice theory that can be seamlessly incorporated into ensembles.

**C5:** This thesis presents a multi-layered architecture for malware detection and shows how it can be used for malware detection.

**C6:** This thesis demonstrates that proposed techniques for malware detection on one computing platform can be seamlessly adapted and implemented on other platforms with equal effectiveness.

### 1.1 Background

The users of computing devices and the Internet are increasing exponentially due to the availability of different online services such as e-banking, communication, education, and online social life\(^6\). For these

\(^6\)https://www.itu.int
users, malware is one of the main IT security threats, which has increased its impact from the early days. The evolution of malware may be broadly divided into two eras. The initial era, in which the malware authors have written the malware with intentions to damage the data or gain fame, has been written for the computers only [5]. The later part of the first era depicts a shift in malware authors’ intentions from damages to stealing users’ sensitive information for different advantages such as monetary gains in the form of ransom and crypto-currency [6, 14]. For example, Ramnit virus\(^7\), which is captured in 2010, has been altered and currently has features of Zeus\(^8\) malware, is used to steal 4,50,000 Facebook accounts for monetary benefits [5]. It is reported by Symantec’s\(^9\) that stolen legitimate information such as Netflix\(^{10}\) accounts, gaming accounts, and social media accounts such as Facebook accounts are sold by malware authors at the online black/underground markets for the monetary benefits. Moreover, the availability of the Internet makes it easy for malware authors to develop and spread malware remotely and gain monetary benefits while remaining anonymous. In 2022, LockBit is used to demand an enormous known ransom of 50 million dollars [8]. Furthermore, the malware authors are using malware as the primary source of their malicious activities, such as launching a security attack, which may compromise networks, sending Spam emails, destroying critical infrastructure, and creating command and control networks [9, 15].

During the second malware era, one significant development is the usage of mobile computing devices and cloud services along with traditional computing devices. Mobile computing platforms such as mobile phones have got more computational power and are converted into smartphones, which allow users to browse through websites, use email, perform banking operations, and other required services along with the standard mobile phone features such as calls and messages. This new trend has also lured malware authors, and the first smartphone malware

\(^7\)https://www.f-secure.com/v-descs/virus_w32_ramnit.shtml
\(^8\)https://usa.kaspersky.com/resource-center/threats/zeus-virus
\(^9\)https://www.symantec.com/security-center/threat-report
\(^{10}\)https://www.netflix.com/
1. Introduction

Cabir\textsuperscript{11} is released in 2004. Initially, mobile malware have functionality similar to malware for computer systems. However, due to advances in mobile technology, mobile malware also has evolved \cite{16}. In 2008, the first smartphone with an Android Operating System is launched\textsuperscript{12}. Android OS is based on Linux kernel and owned by Google Inc\textsuperscript{13}. From year 2008-2009, Android based devices have gained popularity. The current share of Android devices in the smartphone market is approximately 75\% and more than 60\% in other smart devices market such as tablets\textsuperscript{14,15}. In August 2010, the first malware, i.e., DROIDSMS.A\textsuperscript{16} for Android OS is released. Since then, the number of malware for the Android OS are increasing at an increasing rate. It is reported that, currently, 90\% of mobile malware are released for Android OS\textsuperscript{17}. It is worth mentioning that Android malware may have both privacy endangering characteristics and traditional malware characteristics, i.e., spreading the malware on other devices \cite{17}. It is also reported that 60\% of Android malware causes financial losses\textsuperscript{18}. In the 2022 annual report, McAfee has identified four types of threats for Android malware, which are for monetary benefits \cite{18}.

To address the malware problem, anti-malware software are used as a countermeasure, which mainly relies on either static (code) malware analysis or dynamic (behavior) malware analysis. For the dynamic malware analysis, the captured sample of malware is executed in an isolated virtual environment, i.e., sandbox, to reveal its characteristics and behavior \cite{19}. However, a malware can detect the virtual environment and can evade it. Moreover, the dynamic analysis may not provide indications

\textsuperscript{11}http://about-threats.trendmicro.com/us/archive/malware/SYMBOLS_CABIR.A
\textsuperscript{12}http://en.wikipedia.org/wiki/Smartphone
\textsuperscript{13}https://www.google.com/
\textsuperscript{14}www.gartner.com/
\textsuperscript{15}www.statsoft.com/
\textsuperscript{16}http://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/ANDROIDOS_-DROIDSMS.A
\textsuperscript{17}http://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-malware-is-on-android-this-is-the-easy-way-you-stay-safe/
\textsuperscript{18}http://www.kaspersky.com/about/news/virus/2014/sixty-per-cent-of-Android-attacks-use-financial-malware
about polymorphism and metamorphism characteristics of a malware and is resource intensive [20]. On the contrary, for the static analysis, malware characteristics (features) and behavior are extracted from the code by reverse engineering the file. The extracted features, such as function calls, application programming interface (API) calls, byte code, or structural fingerprints (assembly instructions) may be used to develop the signatures for signature based detection or rules for heuristic detection. However, the static detection process is time consuming, and requires human intervention and interpretation [20]. Consequently, malware analysts may spend much time manually analyzing the captured example of malware. Moreover, malware authors can bypass the majority of anti-malware tools, which rely on static detection by using obfuscation techniques, i.e., changing the instructions’ order or inserting dead code [21].

To increase IT security, address the above mentioned problem and to develop intelligent malware detectors, Researchers have reported the use of supervised learning [22,23,24,25]. The supervised learning methods can identify valuable patterns from a given data set, which consists of both malware and benign examples. These methods can identify different malware families [26,27]. It is worth noting that supervised learning based detectors mainly differ from each other in terms of feature sets, feature selection methods, and learning algorithms. The majority of Researchers have extracted static features from the malware, and benign samples without executing them [1]. This helped Researchers in representing the sample files in the data set by using a selected feature representation. The extracted features are byte codes, the file’s structural fingerprints, Windows API calls, readable text strings, flow graphs, and file features such as file size. From the extracted features, a subset of features (also referred to as the training set) is derived by applying the feature selection methods, which helps in reducing the dimensionality. The reduced feature set with class labels is fed to the learning algorithms to generate the detection models or classifiers. These classifiers are evaluated for their performance by testing them on unseen examples using the same feature representation as the training set. Similarly, after the
deployment of the classifier, a sample is fed into the model using the same feature representation and feature selection method for the prediction of its class. It is reported that the generated model performs better than signature-based detection methods and simple heuristic-based detection methods as it is capable of detecting the variants of a particular malware and zero-day malware up to a reasonable extent.

1.2 Terminology

The following given terminologies are relevant to the thesis and may be defined in a (slightly) different way as per the problem at hand. Thus, the following explanations of these terminologies are pertinent to the primary focus of this particular thesis.

**Privacy** According to Aristotle, the private life of a person is the fundamental human right and is referred to as privacy [28]. In literature, the basic definition of privacy is "The right to be let alone" [29]. However, in the digital age, computer and mobile users need this right digitally as they are concerned about the unauthorized usage of their data and personal information. Consequently, the concept of privacy has been further extended as follows [30]:

"To control, edit, manage, and delete information about them [selves] and decide when, how, and to what extent information is communicated to others."

The word privacy in this thesis refers to information privacy, which indicates the legal right of a user to decide, with informed consent, the extent of the collection, storage, and sharing of their personal information in digital form.

**Informed consent** is a legal term that is derived from two words, i.e., informed and consent [31]. The term informed means that accurate information is provided to the user who is able to interpret that specific information accurately. The term consent refers to an opportunity given to
1.2. Terminology

a rational user to accept or reject something offered to him without any pressure or lure. It is a common practice for software vendors to obtain users’ informed consent by providing the information in the end user license agreement (EULA). EULA is a software license agreement between the software manufacturer and the user, which describes the permissions, rights, and restrictions about using that particular software. In the case of computer malware (especially PUPs) and Android malware, the information about the unwanted functionalities may be concealed in the EULA. The obscure information in EULA sets new challenges, especially about the informed consent of the user.

Static Analysis  Static analysis is a method of examining an executable file without executing it. In the static analysis, the actual code or instructions of the file in hand are not available. However, the executable file may be disassembled using a disassembler such as IDA\(^{19}\) to obtain the assembly instructions of different file sections. The disassembled instructions can provide indicators about the intent and behavior of the file, which can be further used to generate signatures or rules. It is worth noting that static analysis is capable of providing syntactic information of instructions and may not provide the semantic information [32].

Signature  A signature or malware signature is a string of unique values, i.e., byte pattern or hash value, which is capable of indicating the presence of a particular malware. Anti-malware software stores these signatures in their database to detect a specific malware.

Machine Learning  is formally defined as [33]:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".

In the context of malware detection, the above given definition [33] can

\(^{19}\)https://hex-rays.com
be elaborated as follows:

**Task T**: Recognizing and classifying malware from the set of benign and malware files.

**Performance Measure P**: Percentage of malware and benign files correctly classified.

**Training Experience E**: A data set of malware and benign files’ features with their known classes.

Learning in the context of malware detection implies an ability to recognize complex patterns, especially patterns, which indicate malicious actions, and decide based on previously seen data. One of the main challenges for machine learning in malware classification is to discover and generalize the knowledge obtained from a limited malware example set and give a decision about the unseen malware. For this purpose, ML algorithms use statistical and computational principles.

**Supervised learning** for malware classification is learning a function, which can map a given file as input to an output class based on previously seen examples of input-output pairs. Supervised learning algorithms for malware classification are used to generate a function, i.e., a model, which is generally referred to as a classifier that maps inputs to distinct outputs. These algorithms are trained by a limited set of benign and malware examples and their respective outputs. Further, they are used to predict the output (class) for future inputs, i.e., malware or benign files.

**File Representation** For malware detection or classification, data sets can be prepared by using different representations of executable files. An executable file can be represented by a specific type of extracted features present in the file. The extracted features may be further used to create *n*-grams, which in turn can be viewed as words or terms if the learning problem is defined as a text categorization problem.

**N-gram** *N*-gram is a sequence of *n* items such as *n* bytes or *n* operation codes (OpCodes), which may or may not be contiguous in a file. For the
malware detection task, an $n$-gram is a sub-string of the program code. The $n$-gram is generally a fixed size string with a length of $N$. However, for this particular thesis, the size of a $n$-gram may be variable depending on the base unit. The minimum size of a $n$-gram for this particular thesis is greater than one.

**Classification** A learning algorithm tries to find the relationship between instances and their respective classes by using different strategies, such as decision trees, depending on the family of the algorithm [33,34]. The mapped relationships can be interpreted, and all the relationships found in a single data set are summarized in the form of a model called a classifier. Later, the classifier can be applied to other data sets, where the class value of instances is unknown to obtain a prediction about the target class for each instance in the data. This process is referred to as classification. Thus, for the malware detection task, to classify an unknown file, initially, a classifier can be constructed by using the training set. Later, a classifier can be applied over unknown file to predict their classes.

### 1.3 Thesis Outline and Structure

Chapter 2 provides an overview of the Related Work, the Aim & Scope of the research, along with the Research Questions (RQs) and the Research Methodology. Following this, Chapter 3 presents the contributions of the studies, which are then followed by an overall discussion and lessons learned. The chapter concludes with recommendations for future work. Finally, from Chapter 4 to Chapter 10, publications are presented.
Chapter 2

Approach

2.1 Related Work

At the end of the 80s, theoretical evidence is presented that accurate malware detection is impossible [35, 36]. However, some security experts argue that this may not be an actual case in real-life [37]. Malware detection has gone through different phases, which are from signature-based detection to behavioral-based detection [38]. Initial malware, i.e., viruses are used to infect files, boot sectors and are easy to detect using signature-based detection methods up to the mid of ‘80s. With the development of more sophisticated malware such as the worm in the mid of 80s [39], and Trojan horses, which exploit the vulnerabilities in the operating system and software such as web servers, more sophisticated detection techniques are required. Heuristic-based detection and Behavior-based anomalies detection techniques are used to address the challenge [40,41]. However, till that time, the malware industry has generally relied on the signature-based detection technique [42]. Thus, in an effort, the creation of the generic signature for a family of malware is also investigated [43]. However, these techniques face the challenge of execution speed, central processing unit (CPU) load, and false positive rate. Moreover, for the success of these techniques, vulnerability
knowledge is required in advance, which may not be available. To address these issues, especially prior vulnerability knowledge, test cases are generated, which produce models [44, 45]. The generated models are further used to identify vulnerabilities. However, these models are also used by malware writers to fabricate evading techniques [46].

To generalize the known malwares’ detection and detect unknown malware especially zero-day attack, the use of data mining and machine learning methods are first investigated in 1994 [47]. The use of supervised learning techniques in ML for malware detection is first investigated in 2001 [48]. In the training phase of supervised learning, a batch of malicious and legitimate examples is provided to the learning method [23, 49]. The batch of examples contains the patterns that are used for representing executable files (i.e., file representation) [50]. These patterns are generally referred to as file features, which are extracted from the executable files by applying different reverse engineering techniques [22, 23, 24]. The file features are considered static features (morphological), and supervised learning is usually applied to extracted file features to extend the static analysis [49, 50, 51, 52]. The malware detection researchers primarily focus on static feature extraction (commonly referred to as structural fingerprints) from executable files and their analysis. Static features are typically used to generate the signature profiles and behavioral profiles using $n$-gram [51, 53]. The concept of $n$-gram is widely investigated in the domain of text categorization [54], authors profiling, text comparison, and speech recognition [55] and have shown promising results, independent of any specific language [56]. The concept of $n$-gram is also applied for malware detection in the form of tri-grams extracted from the virus signatures [47]. N-grams are also fed into the neural network based classifier for detecting viruses [57, 58]. Similarly, several neural network based classifiers are also generated on the $n$-gram for virus detection [59].

Due to the increasing number of Android malware, many researchers have investigated Android security covering vulnerability assessments and Android malware analysis and detection/classification. For mal-
ware detection, different researchers have extracted different information from APK files. In a study, 11 different features such as permissions, intents, API calls, and code related information are extracted [60] from a large number of benign and malware files. However, the number of malware files is only 8% in the data set. These features are used to train classifiers. The authors have reported approximately 99% accuracy. However, the imbalance between the benign and malware classes is not considered. Moreover, only one ensemble is used, i.e., Random Forest. In another study, authors have reported two different approaches for malware detection. For the first approach, they used a combination of permissions and intent filters, and for the second approach, they used source code [61]. In a similar study, a combination of services, methods, and API calls are used as a feature. They also used Java code to generate specific size features [62]. These studies are able to obtain more than 95% accuracy. However, for the experimental purpose, they have used features from one source, i.e., manifest or dex. They have not investigated the fusion of features from both sources. Similarly, features from manifest files are used to suggest a security service for Android OS, which certifies the application [63]. The security service relies on the set of rules, which are derived on the basis of the system’s permissions or intent filters in manifest files, to determine the class/type of the application. However, the difference between information extracted from benign and malicious applications is subtle. Thus, the security service might have a false positive rate. Other researchers suggested various methods of using the same static features, i.e., system’s permissions from manifest files and API calls from the SMALI files [64, 65, 66, 67, 68, 69]. Only a few researchers have focused on OpCode as static features from the dex files. In a study, OpCode signatures are used to detect Android malware [70]. The authors have automated the process of malware collection and generated OpCode based signatures at different levels, such as the application and class levels. However, OpCode signatures are unable to indicate the behavior of the application.
2. Approach

2.2 Aim & Scope

This thesis explores data-driven approaches to detect malware. The broader task addressed is software classification within the IT security domain. The thesis investigates how to detect several types of malware. The thesis centers on one task-oriented applied IT security research question and two solution-oriented applied machine learning research questions:

2.3 Research Questions

In this thesis, we aim to address the following three research questions:

RQ I  \textit{Which types of patterns should be exploited to achieve reliable detection of common types of malware?}

The main objective of malware detection/classification is to determine the correct class of a given sample. This can be achieved by knowing the intent of the given sample, which may be helpful, such as providing a certain functionality, or malicious, such as destruction or theft of data, monetary losses, or misuse of computing resources. Each executable file contains patterns, which can be indicative of the intent of the file, and these patterns can be used for malware detection. It is worth investigating, which type of patterns can be extracted from the malware and benign files and used for generating the training and testing data sets.

RQ II  \textit{What is the most suitable representation of binaries for each detection task?}

Each executable file can be disassembled into different formats, such as binary code, hexadecimal code, or assembly instructions. These disassembled files can be further transformed into different representations and can provide different information. Thus, it is of interest to know which representation is most suitable for
achieving classification accuracy and providing valuable information about the file.

**RQ III** *What ensemble approach yields the highest accuracy for each detection task?*

Ensembles are expected to perform better than a single classifier. Ensembles can be created using different methodologies for the problem at hand to improve the classification accuracy. For malware detection, the cost of misclassification may be higher for a user in terms of loss. However, it is unclear which ensemble approach may yield the highest classification accuracy for malware detection. Thus, it is of interest to know which ensemble method produces the highest accuracy for malware detection.

### 2.4 Methodology

To address the malware detection issue, researchers generally conduct empirical investigation, which falls in the quantitative research category. Quantitative research can be carried out using different research designs such as the experimental method or the case study. This thesis employs the experimental method for all included studies to provide answers to the research questions presented in the section 2.3. The experimental method is commonly referred to as the scientific method as it follows a systematic approach to investigation and deriving knowledge [71]. An experimental method is defined as follows [72]:

"*A study in which an intervention is deliberately introduced to observe its effects*"

The experimental method aims to identify a cause and effect relationship between the variables being studied. In an experiment, all variables except one (known as the independent variable) are controlled. This is why experiments are also known as controlled experiments. Other variables involved in a controlled experiment include dependent variables, control groups, and constants.
2. **Approach**

Thus, to conduct controlled experiments for each study, research questions are formulated to identify the research objectives and design the experimental steps, including the identification of an independent variable. Further, data is collected and pre-processed, and relevant features are extracted and selected. The data and current literature are analyzed to choose suitable supervised learning algorithms. For experimental purposes, the majority of studies in this thesis used the learning algorithms with their default parameters’ values. However, in some studies, such as in Chapter 9, the parameters’ values are determined in a systematic manner. Finally, learning algorithms are used to build classifiers or models. Subsequently, the results and performance of each classifier are validated.

All the variables, in the context of supervised learning based malware detection experiments, are explained as follows:

- **Independent Variable** is generally referred to as the cause or experimental conditions or treatments for a study. In this thesis, an independent variable, such as valuable extracted features using a particular feature representation or decision mechanism, is used to drive or vary the experiment to investigate its effects. It is worth noting that each study uses a specific independent variable in the experiments.

- **Dependent Variable** is generally referred to as the effect. The value of a dependent variable is dependent upon the independent variable. The dependent variable in this thesis is the accuracy of the classifier, which is the percentage of correctly identified malware and benign files.

- **Control Group** is a group that either receives no treatment or standard treatment and is then contrasted to an Experimental Group. In some studies, this thesis uses a classifier generated by the ZeroR algorithm, which predicts the class of an instance by always assigning the majority class, as a member of the control group.
Thus, all classifiers generated by different learning algorithms are compared against the classifier generated by the ZeroR algorithm to assess their performance.

- **Constants** are variables that remain the same throughout the experiments. Constants may not directly impact the outcome of the experiment. The constants for the thesis are tools/software which help in conducting experiments. Thus, operating system, data collection software, machine learning API, disassembly, and feature extraction routines.

During the experiments conducted for Chapter 4 to 10 a positive correlation is found between the independent variable, e.g., the number of valuable extracted feature sets, and the dependent variable, e.g., the classifier’s accuracy. It is common in the field of machine learning to have a larger data set (that lowers the variance across the different data sets) to improve the accuracy. Thus, the increase in the independent variable, i.e., increase in the number (and or quality) of valuable extracted features up to a certain extent may help in improving the classifier accuracy.

## 2.5 Validity Threats

There may be possible threats to the experimental method’s validity, which may have different concerns such as reliability, generalization, and applicability in real-world settings. These threats are divided into four main categories, i.e., internal, external, construction, and conclusion, where each type may address a specific methodological question [73].

*Internal Validity* threats are related to experimental settings, where they mainly concern the effect of an independent variable on the dependent variable, and the impact of any change in the experimental settings to its outcome [74, 75]. For the experiments in the thesis, there are two main internal validity threats. The first threat is concerned with the
2. **Approach**

selection of sample input during the training stage. This threat is addressed by using random assignment during the training phase. The second threat is concerned with the selection of features in the input. To mitigate the selection threat, statistical methods are used. Another relevant threat may be related to the extraction of features from files as extraction routines may not be able to extract features from all the programs. This threat is mitigated by using uncompressed and unpacked files and using different representations.

*External Validity threats* are concerned with the generalization of conclusions or results from a particular (controlled) experiment to the more general population or in the real-world settings [74]. For the studies in this thesis, the ratio of malware and benign programs in the wild is an example of the external validity threat. However, an inherited problem with this threat is that there may not be a consensus for the ratio of malware and benign programs. However, this threat is addressed by investigating the different ratios in Chapter 4, Chapter 5, and Chapter 6. For the rest of the studies, an equal number of samples from both families is used for the experimental purpose to give an equal representation. Another external validity threat for the thesis is that the data set used in experiments may not represent all families of malware in the sample, so the results may not be generalized. Correct sampling allows generalization and mitigates the external validity threats; statistical sampling and replacing methods are used in experiments.

*Construct Validity threats* are threats to the experiment itself and threats to the data preparation stage in the experiment; hence, they assess the quality of the experiment design [73]. Construct validity threats are mitigated by performing the statistical analysis of the experimental procedure and the relation between the variables [75]. For evaluating the relation between variables, variables shall be defined in a quantifiable way. However, these threats do not apply to the experiments in the thesis. Moreover, the experimental procedures and data preparation methods are standards within the domain.
Conclusion Validity threats concern the reason-ability of conclusions drawn from the experiment [73]. Unfortunately, there is no standard data set available for malware detection. The available data sets, which are used in different studies, are old and contain traditional malware only. Thus, it is impossible to compare the work with state of the art. Thus, to mitigate this threat, in all experiments in the thesis, statistical test such as the paired t-test is used. The T-test is a parametric test, which is used to obtain the ratio of the difference between means of two groups against variability between groups [74].
Chapter 3

Results

3.1 Results and Contributions

The following sections elaborate on the contributions of this thesis in general and the particular results and contributions of the published articles that constitute Chapter 4 to 10. The details of contributions in each chapter, according to the research questions (given in section X), as well as similarities and differences with other chapters are as follows:

Chapter 4 presents a malware detection approach using a specific malware, i.e., Spyware. This chapter investigates the possibility of using an ML-based detector to detect Spyware that is concealed or embedded within a program as its functionality. The byte codes (hexadecimal code) are extracted from programs or files in the data set without considering the file structure, i.e., from a specific section of the file. The extracted features are similar to signatures used for malware detection. The byte codes are further used to generate non-overlapping fixed size $n$-grams with different configurations, ranging from bi-gram to octa-gram, which is further used to generate signature profiles. Two feature reduction methods are suggested to obtain a subset of features. Five different supervised learning algorithms from different families, such as decision tree based, are applied to all the configurations to generate classifiers.
The experimental results suggest that the proposed detector is capable of detecting embedded malware. This chapter contributes to both RQ I and RQ II. For both RQ I and RQ II, the results suggest that byte code contains patterns, which supervised learning algorithms can use to generate classifiers to detect known and zero-day malware detection. However, byte code features cannot provide helpful information about the functionality or behavior of the program. In addition, this chapter suggests that using larger $n$-gram sizes is more effective for detecting malware. Further, another contribution of this chapter is that two feature reduction methods are proposed and investigated. The experimental results suggest that better detection results can be achieved by reducing features based on their frequency. Moreover, this chapter suggests that machine learning based malware detectors can indicate the presence of Spyware (or malware) with a high degree of accuracy, even when the malicious files are scarce.

Chapter 5 proposes an algorithm for the detection of malware, specifically Adware. The algorithm employs machine learning techniques to effectively identify and mitigate the risks of Adware. The assembly instructions are extracted from the data set without considering the file structure. Further, OpCodes are extracted from each instruction and are used to generate non-overlapping fixed size $n$-grams with different configurations, i.e., bi-gram to octa-gram for the classification. The generated $n$-grams are further used to generate the signature profiles for both Adware and benign files. Primary and secondary feature selection is performed using different configurations to acquire a total of 63 data sets. In order to obtain an effective and accurate detector, six data mining algorithms are evaluated on these data sets. Empirical results show that the proposed approach has the capability to accurately identify both new and previously identified malware instances. This chapter contributes to RQ I and RQ II. This chapter contributes to RQ I as the experimental results indicate that OpCode $n$-grams are more effective for malware detection than the byte code $n$-grams. The results also extended the results of the previous study that larger $n$-gram size is a better choice for malware detection. Regarding RQ II, it indicates that
larger OpCode \( n \)-grams may provide information about the program’s behavior. Another contribution of this chapter is that the impact of feature selection is evaluated in two stages, i.e., primary feature selection using the frequency of a feature and secondary feature selection using the statistical importance of a feature, to find an optimal percentage of features. The experimental results indicate that an effective feature selection strategy feature selection strategy has a positive correlation with classifiers’ accuracy.

Chapter 6 extends the traditional heuristic detection methods to an automated process for extracting the behavior of a particular type of malware i.e., Scareware, and legitimate (benign) software in the form of variable-length instruction sequences, which are further used to generate behavioral profiles for both Scareware and benign files. Extracted fragments of behavior are analyzed in order to improve the existing knowledge about malware detection. This chapter contributes to RQ I in terms of the features used. Regarding RQ II, the experimental results suggest that utilizing variable length \( n \)-gram is an effective method for capturing the entirety of a valuable function within the program without disrupting it beyond a predetermined length. In addition, this chapter also provides pointers for the feature selection as the primary feature selection and secondary feature selection from the text mining are evaluated. Further, it is worth noting that results from both Chapter 5 and Chapter 6 are in favor of using quad-gram.

Chapter 7 presents a generic malware detection approach. This chapter proposes two OpCode based novel feature extraction techniques, a layered architecture, a trust-based ensemble algorithm, and an empirical evaluation of the proposed algorithm. The results of this chapter contribute to R.Q I and RQ III. Regarding RQ I, the suggested extraction methods have the capability of presenting more extensive behavioral data through larger \( n \)-grams. For RQ III, the empirical evaluation of the impact of combining inductive biases and the effect of different voting schemes on the ensemble’s output. Moreover, the experimental results suggest that different voting strategies shall be adopted for distinct tasks,
3. Results

depending on the nature of the problem. It has been observed that combining the inductive biases of multiple algorithms trained on varying representations can lead to more accurate malware detection compared to combining biases of different algorithms trained on the same dataset.

Chapter 8 proposes and investigates a decision strategy from the field of social choice theory to enhance the prediction accuracy of the Random Forest algorithm (an ensemble). Additionally, signature profiles of each malware sample are used to extend heuristic-based detection methods. Thus, OpCode features are extracted from the data set in the form of quad-grams. Further, the frequency based feature selection methods are applied to select the valuable features subset. To enhance the accuracy of the classifier, the Random Forest’s majority voting decision method has been substituted with a consensus-based decision method. To validate the generalization of the suggested decision method, the learning algorithm is also tested with binary data sets from the UCI\textsuperscript{1} repository, in addition to the malware data set. This chapter contributes to RQ III as the experimental results suggest that the majority voting decision method may be replaced by a better decision strategy, which combines the output of different supervised learning algorithms to improve the classifier’s accuracy for the malware detection task.

Chapter 9 explores the task of malware detection as binary classification and multi-class classification. For this purpose, one of the largest available malware data sets containing nine different malware families is used. The data set is extended to include the benign class. For the file representation, the features are extracted from a specific section of malware samples. The extracted primary features are APIs, DLLs and OpCodes. APIs and DLLs are used to generate bi-grams after a combinational analysis. OpCodes are used to generate quad-grams and variable length grams. The feature selection is performed in two steps, i.e., primary features selection and secondary feature selection. For primary feature selection, dictionary based approach is used to eliminate non-standard features and frequency analysis is performed to find common-

\textsuperscript{1}https://archive.ics.uci.edu/ml/datasets.html
3.1. Results and Contributions

alities among different classes and keep valuable features. For the sec-
ondary feature selection, filter, wrapper, and embedded methods are
plied. For the wrapper method, a customized algorithm is also pro-
poused. Later, feature fusion is investigated. Further, a customized en-
semble with a decision method is suggested to improve the classification
accuracy. The chapter contributes to RQ I, RQ II, and RQ III. For RQ I,
this chapter contributes that OpCodes based feature contains patterns,
which can indicate the presence of malware. However, other features,
such as APIs and DLL may also provide valuable information. Thus,
a fusion of these valuable features is suggested. For RQ II, the experi-
mental results suggest that variable-length based OpCode n-grams can
capture a program’s behavior statically and are more indicative than the
fixed size n-grams or other features. For RQ III the experimental results
suggest that a voting-based ensemble can improve detection accuracy.

Chapter 10 presents a malware detection approach for Android OS
that investigates the possibility of using ML based detector to detect
malware for Android devices. This chapter extends the techniques pre-
sented in the preceding studies to Android OS. For the features, Op-
Code based features similar to earlier studies are extracted from Android
based applications. Statistical importance based feature reduction meth-
ods are investigated to obtain a subset of data. Moreover, feature fusion
is investigated to obtain a combined set of all those features, which are
considered important by different measures sets. The previously investi-
gated supervised learning algorithms from different families are used to
generate classifiers. Further, the algorithm fusion is investigated, and the
majority voting decision method is used. The empirical results suggest
that the proposed detector can detect malware within Android based
applications. This chapter contributes to RQ I, RQ II, and RQ III. For RQ
III this chapter contributes that a decision strategy for combining differ-
ent classifiers’ decisions outperforms a single classifier in terms of accu-
curacy. Moreover, the empirical results contribute that supervised learning
based malware detection techniques are portable and can be used across
different OS and architectures. Another significant contribution of this
chapter is that it investigates the possibility of using artificial data sets
for malware detection and suggests an optimal ratio of synthetic examples in the data set.

It is worth noting that the presented studies exhibit both similarities and differences.

- Most of them are conducted on Windows OS, except for the last one, which is conducted on Android OS.

- To obtain the benign data set for Chapter 4 - 9, files are downloaded from download.com, or a fresh Windows OS is installed on a computer, and its executables are copied.

- For the experiments in Chapters 4 to 8, the file structure is not considered while extracting features. However, for experiments in Chapters 9 - 10, file structures are taken into account.

- The experiments given in Chapter 4 - 7 employ similar learning algorithms, which have been selected from the literature.

Chapter 4 serves as the foundation for the other chapters, and it is the only study that utilizes byte code as a feature and introduces two distinct feature reduction techniques. To conduct experiments, Spyware samples obtained from the website spywareguide.com are used in this chapter. The chapter’s findings show that a larger \( n \)-gram leads to a significant improvement in detection accuracy. Chapter 5 explores the use of Adware from Lavasoft\(^2\) for experimental purposes. The chapter utilizes fixed-size \( n \)-gram OpCodes as features and for generating feature representation. To maintain a balance between the extracted features, the chapter attempts to balance the samples’ size. Unlike the Chapter 4, this one performs feature reduction in two stages. Additionally, this chapter extends the previous one by suggesting that larger \( n \)-grams are a better choice for malware detection. The optimal size of \( quad \)-grams is indicated as the ideal choice, but this is still inconclusive. Chapter 6 differs

\(^2\)http://www.lavasoft.com
3.1. Results and Contributions

from the previous chapters in terms of malware types and feature representation. In this chapter, Scareware is used for experimental purposes, and Opcodes are used as features. However, for feature representation, variable length n-grams are generated. Similar to the previous chapter, feature selection is accomplished in two stages. The chapter suggests that a quad-gram is a valuable n-gram size for malware detection. The results presented in this chapter are a valuable addition to the results discussed in Chapter 5.

In Chapter 7, fixed-size n-grams are used as a way to represent features, which is similar to Chapter 5. However, two new feature representations are introduced. Previous chapters did not provide a clear choice of algorithm for detecting specific types of malware. To address this gap, this chapter proposes a multi-layer architecture that incorporates different representations for different malware types and uses a different classifier for each. Furthermore, to improve detection accuracy, the use of a customized ensemble made up of top algorithms is suggested. Moreover, in Chapter 8, the use of customized ensembles is further explored, along with an investigation of the generalizability of these approaches and the change in the decision mechanism of a predefined ensemble. It is worth noting that a general similarity between chapters 4 - 8 is the use of a limited data set. Moreover, the majority of learning algorithms used in these studies are similar.

Chapter 9 both builds upon and diverges from the previous chapters in several ways. In this chapter, one of the largest multiclass public data sets is used and further extended for experimental purposes. To extract features, the file structure is considered, and different features are extracted from a specific part of the files. Additionally, the feature representations presented in previous chapters are used together in this chapter. Similar to Chapters 4 - 6, feature selection is performed in two stages. Furthermore, a combinational analysis is conducted. This chapter also complements the previous chapters by investigating the usage of customized ensembles. Finally, Chapter 10 investigates the portability of the methods suggested in previous chapters on Android OS. It also
3. Results

compares the best feature representations based on OpCode to determine the most optimal one. To address the issue of missing malware examples, this study utilizes synthetic data and determines the optimal percentage required in the data set. Similar to Chapter 9, feature fusion is also performed. Moreover, the chapter explores the possibility of using an ensemble to improve the detection accuracy, similar to previous chapters.

3.2 Discussion

Machine learning has proven to be an effective tool in analyzing data patterns across various fields. On an abstract level, the application of ML for a problem can be categorized into three stages: problem definition, data processing, and modeling and evaluation. For the first stage, the problem statement is well-defined as the malware detection/classification problem. The main contributions of the thesis are in the remaining two stages. For the second stage of a machine learning solution, the features extracted from the data set may affect the classification accuracy. However, for the malware detection problem, this has not been thoroughly investigated. The majority of researchers have extracted a feature set in the form of either byte code or OpCode to generate representations similar to representations used in text mining problems. Moreover, the most commonly extracted feature sets are similar to signature profiles of relevant files. Further, to extract the behavior of a file, either API calls or flow graphs are used. Thus, there is a need to investigate the potential ways in which the extracted features can be used to generate representation/s, which can significantly improve the accuracy. This thesis investigates different feature representations and compares them to find an optimal representation. Some of the investigated feature representations can provide information about the behavior of the file without executing it. These representations may not accurately depict the flow of the file. However, they are capable of providing information about all the functionality of the file. The captured malicious functionality can be further used in a dynamic malware detector, especially for detecting packed or
3.2. Discussion

encrypted malware, which releases the behavior during the execution.

An inherited problem with malware is the abundance of malware in the wild and their ratio to benign programs. An inherited problem with malware is the abundance of malware in the wild and their ratio to benign programs. According to an estimate, in 2023, there will be over 1.2 billion malware in the wild\(^3\), and around 450,000 new malware (including potentially unwanted programs) are identified every day. Moreover, in the second quarter of 2023 alone, 5,704,599 malicious programs (including potentially unwanted programs) are detected for Android OS\(^4\). The number of malware in the wild is increasing at an exponential rate. However, an accurate ratio with benign programs is still uncertain, and it causes an imbalance. This imbalance is a critical factor for conducting malware detection experiments. This thesis addresses this problem by investigating the class imbalance problem by generating multiple data sets with different ratios of benign and malware.

Another associated problem is the inability to perform experiments on every type or family of malware. It is impossible to capture a representative sample that represents all types, variants, and families of malware. Although the presented techniques in the thesis are capable of detecting malware variants, which may be generated by obfuscation or polymorphism up to a reasonable degree, this can be achieved by using different variants of malware in experiments. However, due to the absence of a large number of inter/intra family examples in the data set used in experiments, the classifier’s generalization remains questionable. Moreover, the available data set for the experimental purpose is limited in terms of the number of examples in the data set. To a considerable extent, the aforementioned issues can be resolved through the utilization of artificially generated examples. The artificially generated examples are capable of representing intra or inter-family examples. This thesis investigates the usage of artificial examples for Android malware and suggests the ratio of artificial/synthetic samples in the data set. The

\(^3\)https://www.av-test.org/en/statistics/malware/
\(^4\)https://securelist.com/it-threat-evolution-q2-2023-mobile-statistics/110427/
experimental results have indicated that artificial examples may help to increase the generalization of the classifier.

The work presented in this thesis also contributes to two distinct dimensions, i.e., for novice users and malware analysts. For the users, the generated models may detect malware and, in the case of PUP, may provide information about the purpose of the program. This empowers users to make informed decisions about whether or not to use a particular software. This empowerment will help to enhance the privacy of users. For the experts, this thesis presents an automated process to ease their jobs. The techniques presented in this thesis will help the experts to analyze a file, enabling them to determine its behavior and class. However, one main drawback of the proposed techniques is that they are incapable of detecting packed or encrypted malware. The encrypted part of the file cannot be decrypted without the key; thus, human intervention is required. However, the presence of an encrypted module in a program can be considered an indication. Another associated problem with the proposed technique is updating the generated classifiers. Although experimental results indicate that generated classifiers are capable of detecting zero-day attacks; however, due to the emergence of new examples and families of malware, users need to update the classifiers. However, this differs from updating the traditional signature or rules database, where new signatures or rules are added to an existing database daily, which consequently increases the database size. For the proposed techniques, to update the classifiers, classifiers are generated again on the new data set, which contains the latest malware samples. The classifier generation is time-consuming and shall be avoided at the user end. The job of the classifier generation can be performed at a central location. Later, users can be informed about a new classifier and fetch the new classifier, or this can be done automatically by the system. The size of this update is relatively smaller than the size of the signature database and its updates used in traditional detection techniques. Moreover, the frequency of new classifier generation and updating at the user end is also lower than updating the signature or rule databases.
This section summarizes the lessons learned during writing this dissertation. The section is divided into two segments: technical and personal lessons.

When this thesis idea was conceived, the malware phenomenon transitioned from being something purely destructive to encompassing privacy-invasive software variants. It was still relatively uncommon to apply machine learning as a means of detecting malware. If machine learning was used, the focus was on detecting existing types of malware rather than new breeds. We designed our first research study under the hypothesis that existing malware data sets could be used to extract patterns that could be used to detect novel malware variants. However, we realized that a few suitable data sets for this purpose existed, at least publicly. This situation forced us to invest more time in data set generation than anticipated. We contacted various anti-malware software companies to get support for creating data sets to use for research purposes. However, it was not evident for the anti-malware companies which data could be shared. We managed to get limited data sets that featured different variants of privacy-invasive software. To prepare these data for experimentation, we explored several approaches to engineering features and cleaning data. At this point, it wasn’t easy to compare our approaches against the state-of-the-art. We initially overlooked the difficulties in collecting relevant data. It subsequently led to data collection and feature transformation becoming significant activities for initial studies, limiting future studies’ scope.

Finding an optimal feature representation is another lesson learned during the research process. While we had designed our studies, we faced the challenge of not being able to identify a standard way of representing the instances in our data set for experimentation. The literature review uncovered that there is no specific feature representation that can be used as the gold standard. Therefore, we conducted extensive investigations into various feature representations, either by utilizing them or
3. Results

suggesting them, in order to find the optimal one. This experience highlighted the significance of considering all crucial elements while formulating research studies. One valuable lesson we learned was the cruciality of accurately estimating experiment time frames. Despite our best efforts to estimate the optimal timeline, we faced unforeseeable issues that made it challenging to remain on schedule. Unpredictable factors, such as power outages and limited resources (e.g. RAM), caused interruptions to our experiments. Additionally, we did not run our experiments in batches, which proved to be a waste of time when failures occurred. Our experiments were also delayed due to sharing a server with other computation-heavy processes. Moving forward, we realized the importance of estimating for best and worst-case scenarios, as unforeseeable circumstances can arise. During our experiments, we learned an important lesson about presenting our findings. We followed the conventional path of presenting results in terms of accuracy and detection rate, similar to other researchers. However, we realized that presenting results using other performance measuring parameters, such as precision and recall, can also provide valuable insights into the results. One important lesson we learned was about predicting the future of malware. Viruses were the first type of malware and they still share similar characteristics today. We hypothesized the same would be true for Spyware, adware, and other types of malware. However, due to technological advancements and the increasing potential for financial gains, the characteristics of different malware types started to overlap, making it difficult to distinguish between them. We were also unable to anticipate the future of PUP over different operating systems. With the evolution of technology, particularly with Android-based smartphones, many malware authors have shifted their focus to creating malware for these devices. The rate of Android-based malware releases surged until 2016 but this has since begun to decline. Nevertheless, it is unclear what the future holds in terms of trends.

Throughout this period, the author learned several personal lessons, but one of them was particularly important. The author realized the importance of having a realistic approach to time management. Initially, an
overly optimistic approach was taken while calculating the time required for each activity. However, the author understood that unforeseen work or unexpected activities must be considered during the initial calculations. Another valuable lesson was related to language and presentation. As the author is not a native English speaker, their mother tongue has a different grammar and presentation style from English or similar languages. Therefore, to improve the articles, a significant amount of time was dedicated to refining the language by correcting grammatical errors and improving the overall presentation.

### 3.4 Conclusions and Future Work

This thesis comprises seven research papers that aim to present automated techniques for detecting and classifying malware using supervised learning algorithms. The approach taken in this thesis is systematic and aims to enhance the accuracy of malware detection by addressing features, their representation, and selection. The first three studies demonstrate the automated extraction of features to create different representations and investigate the usage of feature selection methods. The knowledge derived from these studies is utilized to suggest new feature representations and investigate selection methods to find an optimal solution. Furthermore, the thesis explores the use of synthetic data to improve detection accuracy and proposes customized ensembles. This is investigated by suggesting different decision mechanisms and evaluating their performance. In particular, this thesis proposes three decision-making methods and a new strategy is implemented in an existing ensemble. The thesis also demonstrates that the malware detection problem can be treated as a multi-class problem and presents the portability of the proposed techniques.

For future work, it is recommended to explore diverse data/text mining approaches for feature extraction and detecting the malware and its category. This thesis has demonstrated the potential to extend the detection process to multiple classes. To increase the reliability of the
3. **Results**

predictions, the incorporation of conformal prediction methods may be investigated. Moreover, for malware detection, there are two inherited problems in using learning algorithms. The first problem is that learning algorithms can have inductive bias, which may result in overlooking critical characteristics. Furthermore, the nature of malware is constantly evolving, causing shifts in its indicative features over time. The second problem is that malware classification experiments often treat the malware class as a minority class, leading to a reduction in indicators or training points from this group. Moreover, the overall accuracy of the learning algorithm is assessed with a 5% error rate, which may obscure its performance on the minority class. In the future, these two problems can be addressed by incorporating Fairness, Accountability, Transparency, and Ethics in learning algorithms.
Detection of Spyware by Mining Executable Files

Raja Khurram Shahzad, Syed Imran Haider, Niklas Lavesson

Abstract

Spyware represents a serious threat to confidentiality since it may result in loss of control over private data for computer users. This type of software might collect the data and send it to a third party without informed user consent. Traditionally, two approaches have been presented for the purpose of Spyware detection: Signature-based Detection and Heuristic-based Detection. These approaches perform well against known Spyware, but have not been proven to be successful at detecting new Spyware. This paper presents a Spyware detection approach by using Data Mining (DM) technologies. Our approach is inspired by DM-based malicious code detectors, which are known to work well for detecting viruses and similar software. However, this type of detector has not been investigated in terms of how well it is able to detect Spyware. We extract binary features, called $n$-grams, from both Spyware and legitimate software and apply five different supervised learning algorithms to train classifiers that are able to classify unknown binaries by an-
Salyzing extracted $n$-grams. The experimental results suggest that our method is successful even when the training data is scarce.

4.1 Introduction

Programs that have the potential to violate the privacy and security of a system can be labeled as Privacy Invasive Software [76]. These programs include: Spyware, adware, trojans, greyware, and backdoors. They may compromise confidentiality, integrity, and availability of the system and may obtain sensitive information without informed user consent [77,78]. This information is valuable for marketing companies and also generates income for advertisers from online ads distribution through adware. This factor works as a catalyst for elevating the Spyware industry [76]. Traditionally, advertisements for computer users are spread by sending spam messages, but such advertisements are not targeted toward a specific segment of users as no information about the users is available to the spammers. On the other hand, data collected by Spyware may be used for customized ads spread through adware to an individual user.

Originally, viruses represented the only major malicious threats to computer users and since then much research has been carried out in order to successfully detect and remove viruses from computer systems. However, a more recent type of malicious threat is represented by Spyware and this threat has not been extensively studied. According to the Department of Computer Science and Engineering at the University of Washington, Spyware is defined as "Software that gathers information about use of a computer, usually without the knowledge of the owner of the computer, and relays the information across the Internet to a third party location" [79]. Another definition of Spyware is given as "Any software that monitors user behavior, or gathers information about the user without adequate notice, consent, or control from the user" [76]. The major difference between the definitions involves user consent, which we regard as an important concept when it comes to understanding the difference between Spyware and other malicious software.
4.1. Introduction

Unlike viruses, which are always unwanted, Spyware can sometimes be installed with the users expressed consent, since it may provide some useful functionality either on its own or by an accompanying software application. Due to this reason Spyware overlaps the boundaries of what is considered legal and illegal software and thus falls in a grey-zone. However, in most cases, the Spyware vendors do not seem to provide the user with any realistic opportunity to give an informed consent or to reject the installation of a software application in order to prevent Spyware. Vendors embed Spyware in regular software, which is installed with the application or by using hacking methods [80]. The installed Spyware may be capable of capturing keystrokes, taking screen shots, saving authentication credentials, storing personal email addresses and web form data, and thus may obtain behavioral and personal information about users. It may also communicate system configuration including hardware and software, system accounts, location information, and information about other aspects of the system to a third party. This can lead to financial loss, as in identity theft and credit card fraud\(^1\). The symptoms of Spyware infection vary, but Spyware may, e.g., show characteristics like nonstop appearances of advertisement pop-ups, open a web site or force the user to open a web site, which has not been visited before, install browser toolbars without seeking acceptance from the user, change search results, make unexpected changes in the browser, display error messages, and so forth. Furthermore, other indications of Spyware may include a noticeable change in computer performance after installation of new software, auto-opening of some piece of software or the default home page in a web-browser, a changed behavior of already installed software, the occurrence of network traffic without any request from the user, and increased disk utilization even in perceivably idle conditions [80]. Some researchers have predicted that advanced Spyware can possibly take control of complete systems in the near future [81].

The awareness about Spyware and its removal is considered low and

\(^1\)www.us-cert.gov/reading_room/Spywarehome_0905.pdf [accessed 18-05-2009]
outside the competence of normal users [82,83]. Even if users have anti-virus software installed, it may not be helpful against Spyware until it is designed particularly for this threat, as Spyware differ from regular viruses, e.g., in that it uses a different infection technique [84]. Viruses normally replicate themselves by injecting their code into executable files and spread in this way, which is not the case for most Spyware. Specific anti-Spyware tools have been developed as countermeasures, but there seem to be no single anti-Spyware tool that can prevent all existing Spyware because, without vigilant examination of a software package, the process of Spyware detection has become almost impossible [85]. Current anti-Spyware tools make use of signature-based methods by using specific features or unique strings extracted from binaries or heuristic-based methods by using on the basis of rules written by experts who define behavioral patterns as approaches against Spyware. These approaches are often considered ineffective against new malicious code [48,84]. Moreover, since most heuristic approaches have been developed in order to detect viruses, it is not certain whether they would be capable of detecting new types of Spyware because Spyware use stealth systems and they do not employ any specific routines like viruses, which may be associated explicitly with Spyware [84].

This paper presents a Spyware detection method inspired by data mining-based malicious code detection. In this method, binary features are extracted from executable files. A feature reduction method is then used to obtain a subset of data, which is further used as a training set for automatically generating classifiers. This method is different from signature-based or heuristic-based methods since no specific matching is performed. In this method, the generated classifiers are used to classify new, previously unseen binaries as either legitimate software or Spyware. In our experiments, we employ 10-fold cross-validation in order to evaluate classifiers on unseen binaries. We use accuracy and the Area under Receiver Operating Characteristic (ROC) curve as metrics for the evaluation of classifier performance.
4.2 Background

The term Spyware first appeared in a Usenet post on October 16, 1995 about a piece of hardware that could be used for espionage. In 2000, the founder of Zone Labs, Gregor Freund, used the term in a press release for Zone Labs firewall product\(^2\). Since then, Spyware has spread rapidly and several attempts to prevent this spread have been made. In 2001, the use of data mining was investigated as an approach for detecting malware [48] and this attempt attracted the attention of many researchers. Since then, several experiments have been performed to investigate the detection of traditional malicious software such as viruses, worms, and so forth, by using data mining technologies.

The objective of the aforementioned data mining experiment [48] was to detect new and unseen malicious code from available patterns. Data mining is the process of analyzing electronically stored data by automatically searching for patterns [34]. Machine Learning algorithms are commonly used to detect new patterns or relations in data, which are further used to develop a model, i.e., a classifier or a regression function. Learning algorithms have been used widely for different data mining problems to detect patterns and to find correlations between data instances and attributes. In order to represent malware instances in a suitable format for data mining purposes, many researchers have used \(n\)-grams or API calls as their primary type of feature. An \(n\)-gram is a sequence of \(n\) elements from a population. It can, e.g., represent a character or a word. The length of an \(n\)-gram can be either fixed (e.g., \textit{unigrams}, \textit{bigrams}, and \textit{trigrams}) or variable. In experiments for the detection of malware, sequences of bytes extracted from the hexadecimal dump of the binary files have been represented by \(n\)-grams. In addition to the use of such byte sequences, some experimental studies have been conducted using data from End User License Agreements, network traffic, and honeypots.

\(^2\)http://zonealarm.com
The 2001 data mining study of malicious code [48] used three types of features, i.e., Dynamic-link Library resource information, consecutive printable characters (strings) and byte sequences. The data set consisted of 4,266 files out of which 3,265 were malicious and 1,001 were legitimate or benign programs. A rule induction algorithm called Ripper [86] was applied to find patterns in the DLL data. Naive Bayes (NB), a learning algorithm based on Bayesian statistics [34], was used to find patterns in the string data and $n$-grams of byte sequences were used as input data for the Multinomial Naive Bayes algorithm [34]. A data set partitioning was performed in which two data sets were prepared, i.e., a test data set and a training data set. This is to allow for performance testing on data that are independent from the data used to generate the classifiers. The Naive Bayes algorithm, using strings as input data, yielded the highest classification performance with an accuracy of 97.11%. The study also implemented a signature-based algorithm and compared its results to those of the data mining algorithms. The data mining-based detection rate of new malware was twice as high in comparison to the signature-based algorithm.

Following this study, a large number of researchers [87,88,89,90,91,92,93] have devoted their efforts for encountering malicious code, which is in most of the cases either viruses or worms, by using data mining. Only two studies [84, 94] focused specifically on Spyware. References [90, 95, 96] used $n$-grams of byte code as features while others [88, 91] used OpCodes. They all were successful in having more than 97% of accuracy. In a different study [89], an experiment was performed on network traffic filtered by network scanner, but still having suspicious malicious code. Two different types of features were used: $n$-grams of size 5 and Windows Portable Executable header data. This study was successful in achieving an Area under the ROC curve score of 0.983. Reference [87] performed an experiment for detection of viruses on a data set of 3,000 files. The study performed experiments on sequence lengths ranging from 3 to 8. The best result was obtained using a sequence length of 5. The results indicated that classifier performance could be increased by using shorter sequences. Reference [91] performed an ex-
4.2. Background

Experiment for the detection of Trojans. In this study, instruction sequences were used as features. The primary data set contained 4,722 files. Out of these, 3,000 files were Trojans and the rest were benign programs. A detection of compilers, common packer was also performed on data set and the feature set was also systematically reduced. Three types of algorithms were analyzed; Random Forest (RF), Bagging, and Decision Trees (DT). The study used ROC as analysis for measuring performance and the best results for false positive rate, overall accuracy and area under the ROC curve were achieved with the Random Forest classifier.

Reference [84] has replicated the work of [48] but with a focus on Spyware collected in 2005. The purpose was to specifically evaluate the suitability of the technique for Spyware detection. The data set consisted of 312 benign executables and 614 Spyware executables. These Spyware applications were not embedded (bundled) with any other executables. The Naive Bayes algorithm was evaluated, using a window size of 2 and 4, with 5-fold cross-validation. Cross-validation is a statistical method that is used to systematically divide the available data into a predetermined number of folds, or partitions [34]. Prediction models, or classifiers, are generated by applying a learning algorithm to \( n-1 \) folds and then evaluated on the \( n \)th fold. The process is repeated until all folds have been used for evaluation once. Even though criticism has been directed towards (over) belief in the cross-validation performance estimates [97], the method is still widely regarded as a reasonable and robust performance estimation method, especially when the data is scarce. The experiment showed that the overall accuracy was higher when using a window size of four.

The Spyware problem is also different from that of detecting viruses or worms as vendors of Spyware-hosting applications usually include them in bundles with popular free software. The End User License Agreement (EULA) may very well mention the Spyware, in order for the Spyware vendors to avoid legal consequences, but this information is given in a way that makes it difficult for the average user to make an informed consent. In addition, the EULAs from both legitimate software vendors and
Spyware vendors normally contain thousands of words, and this makes it hard for users to interpret the information. Reference [94], therefore, investigated the possibility to automatically detect Spyware by mining the EULA. This study is similar to the studies carried out on spam detection by using data mining. The studied data set contained 996 EULAs out of which 9.6% were associated with Spyware. The study applied 17 learning algorithms on two data sets, represented by a bag-of-words and meta EULA model, respectively. The performances of the 17 classifiers were compared with a baseline classifier, ZeroR, which predicts the class of an instance by always assigning the majority class, e.g., the class that the majority of the instances in the training data set belong to. ZeroR is commonly used as a baseline when evaluating other learning algorithms. The results indicated that the bag-of-words model is better than the meta EULA model. Results also indicated that it is indeed possible to distinguish between legitimate software and Spyware by automatically analyzing the corresponding EULAs.

A majority of the reviewed studies use n-grams to represent byte sequences. Except for Reference [84] and Reference [94], all the studies were performed on malware or viruses. Moreover, some studies [48,84,91] featured data sets with almost a double number or an equal number of malicious files than benign files. Other studies [89,93] use a population in which a third consists of malicious files. This situation is arguably unrealistic, since in real life, the number of malicious files compared to benign files is much lower. Most of the studies used standard data sets available for malware or virus research. These data sets contain individual malicious executables. Thus, the executables are not embedded or bundled with other executables, which is the common situation for Spyware. We have only been able to find one malicious code detection study that focuses on Spyware [84]. This study performed experiments using n-grams for data representation, in particular, with $n = 2$ and $n = 4$. The latter configuration yielded the best results. However, other experiments on malicious code [89] have shown better results for $n = 5$. We, therefore, argue that a larger set of $n$-values need to be evaluated for the Spyware domain.
4.3 Proposed Method

The focus of our analysis is executable files for the Windows platform. We use the Waikato Environment for Knowledge Analysis (Weka) [98] to perform the experiments. Weka is a suite of machine learning algorithms and analysis tools, which is used in practice for solving data mining problems. First, we extract features from the binary files, and we then apply a feature reduction method in order to reduce data set complexity. Finally, we convert the reduced feature set into the Attribute-Relation File Format (ARFF). ARFF files are ASCII text files that include a set of data instances, each described by a set of features [34]. Figure 4.1 shows the steps involved in our proposed method.

4.3.1 Data Collection

Our data set consists of 137 binaries out of which 119 are benign and 18 are Spyware binaries. The benign files were collected from CNET Download\(^3\), which certifies the files to be free from Spyware. The Spyware files were downloaded from the links provided by SpywareGuide.com\(^4\), which hosts information about different types of Spyware and other types of malicious software. The rather low amount of gathered Spyware is attributed to the fact that most of the links provided by Spy-

\(^3\)http://download.com
\(^4\)http://Spywareguide.com
wareGuide.com were broken, i.e., these links did not lead to pages where the Spyware executables could be downloaded. We have yet to find, or build a larger Spyware data set.

### 4.3.2 Malicious File Percentage

Reference [88] has shown that for their particular study, the MFP needed to be equal to, or lower than 15% of the total population in order to yield a high prediction performance. Relating to our data set, the MFP is almost 14%. However, it is important to stress that we have yet to uncover evidence to support that the recommended MFP leads to improved results in the general case.

### 4.3.3 Byte Sequence Generation

We have opted to use byte sequences as data set features in our experiment. These byte sequences represent fragments of machine code from an executable file. We use `xxd`\(^5\), which is a UNIX-based utility for generating hexadecimal dumps of the binary files. From these hexadecimal dumps, we may then extract byte sequences, in terms of \(n\)-grams of different sizes.

### 4.3.4 \(n\)-gram Size

A number of research studies have shown that the best results are gained by using an \(n\)-gram size of 5 [87, 89]. In the light of previous research, we chose to evaluate three different \(n\)-gram sizes (namely: 4, 5, and 6) for the experiments.

### 4.3.5 Parsing

We first extract byte sequences of the desired \(n\)-size. Each row contains one \(n\)-gram and the length of a single row is thus equal to the size of \(n\).

\(^5\)http://linux.about.com/library/cmd/blcmdl1_xxd.htm
4.3. Proposed Method

4.3.6 Feature Extraction

The output from the parsing is further subjected to feature extraction. We extract the features by using two different approaches: the Common Feature-based Extraction (CFBE) and the Frequency-based Feature Extraction (FBFE). The purpose of employing two approaches is to evaluate two different techniques that use different types of data representation, i.e., the occurrence of a feature and the frequency of a feature. Both methods are used to obtain Reduced Feature Sets (RFSs), which are then used to generate the ARFF files.

Table 4.1: Feature Statistics

<table>
<thead>
<tr>
<th>Size / Features</th>
<th>$n = 4$</th>
<th>$n = 5$</th>
<th>$n = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign Features</td>
<td>26,474,673</td>
<td>21,179,768</td>
<td>17,649,809</td>
</tr>
<tr>
<td>Spyware Features</td>
<td>8,357,458</td>
<td>6,685,971</td>
<td>5,571,645</td>
</tr>
<tr>
<td>Total Features</td>
<td>34,832,131</td>
<td>27,865,739</td>
<td>23,221,454</td>
</tr>
<tr>
<td>FR = 1 - 49</td>
<td>26,269,292</td>
<td>21,987,533</td>
<td>18,746,618</td>
</tr>
<tr>
<td>FR = 50 - 80</td>
<td>5,282</td>
<td>3,226</td>
<td>2,286</td>
</tr>
<tr>
<td>FR = 81 - 500</td>
<td>6,018</td>
<td>3,929</td>
<td>2,788</td>
</tr>
<tr>
<td>CFBE</td>
<td>536</td>
<td>514</td>
<td>322</td>
</tr>
</tbody>
</table>

FR = Frequency Range.

Common Feature-based Extraction

In CFBE, the common $n$-grams (byte sequences) are extracted from the binary files, one class at a time.

Frequency-based Feature Extraction

The word frequency can be defined in various ways. In statistics, it basically represents the number of occurrences or repetitions of some observation at a specific time or from some specific category. In our study, the word frequency means the number of occurrences of some specific
4. Spyware Detection

An $n$-gram in a certain class or the number of repetitions of some specific $n$-gram in a particular class. In FBFE, all the $n$-grams were sorted and the frequency of each $n$-gram in each class is calculated. All $n$-grams, within a specified frequency range, are extracted and the rest is discarded. In the frequency calculation, we discovered that there were a few uninteresting $n$-grams, e.g., 0x0000000000, 0xFFFFFFFFFF, and 0x0000000001. Even though these instances were few (less than 10), their frequencies were high (more than 10,000). Thus, the frequency analysis helped us to define three suitable frequency ranges: 1-49, 50-80, and 81-500. The number of $n$-grams in the 50-80 frequency range tends to be almost equal to the number of $n$-grams in the 81-500 range.

4.3.7 Feature Reduction

Features were reduced in both the CBFE and FBFE method. In CBFE, the common features gained from all files were sorted. Only one representation of each feature was considered in one class. CBFE has produced a better reduced feature set. For example, the reduced feature set for $n = 4$ contains only 536 features compared to 34,832,131 for the complete set. In FBFE, the frequency of each $n$-gram is calculated. Reduced features were obtained with three frequency ranges 1-49, 50-80, and 81-500. After analysis of the number of $n$-grams in each frequency range, it was decided that the 1-49 frequency range will not be included in the experiments, since the number of $n$-grams even in the reduced feature set was too high. For example, for an $n$-gram size of 5, the total number of $n$-grams was 27,865,739 and the number of $n$-grams included in the reduced feature set for the 1-49 frequency range was 21,987,533, which indicated the presence of a large amount of uninteresting features. Table 4.1. shows some statistics regarding the number of features in each class, the total number of features, and the reduced feature sets, based on different frequency ranges for both FBFE and CFBE.
4.3. Proposed Method

Figure 4.2: Comparison of Accuracy with $n = 6$

4.3.8 ARFF Generation

Two ARFF databases based on frequency, and common features were generated. All input attributes in the data set are represented by Booleans, i.e., either a certain $n$-gram or the $n$-grams within a certain frequency range are represented by either 1 or 0 (present or absent).

4.3.9 Classifiers

Previous studies are not conclusive about which learning algorithm generates the best classifiers for problems similar to the studied problem. However, the results have provided us with a basis for choosing ZeroR, Naive Bayes, Support Vector Machine (SVM) algorithm, i.e., Sequential Minimal Optimization (SMO), C4.5 Decision Tree (J48), Random Forest and JRip as candidates for our study. ZeroR is used only as a baseline for comparison. For our purpose, it can be viewed as a random guesser, modeling a user who makes an uninformed decision about a piece of software. A Naive Bayes classifier is a probabilistic classifier based on Bayes theorem with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other. This of course, is seldom true for real-life applications. Never-
4. Spyware Detection

theless, the algorithm has shown good performance for a wide variety of complex problems. SVMs, which are used for classification and regression, work by finding the optimal hyperplane, which maximizes the distance/margin between two classes. J48 is a decision tree-based learning algorithm. During classification, it adopts a top-down approach and traverses a tree for classification of any instance. Moreover, Random Forest is an ensemble learner. In this ensemble, a collection of decision trees are generated to obtain a model that may give better predictions than a single decision tree. Meanwhile, JRip is a rule-based learning algorithm.

4.3.10 Performance Evaluation Criteria

We evaluate each learning algorithm by performing cross-validation tests to ensure that the generated classifiers are not tested on the training data. From the response of the classifiers, the relevant confusion matrices were created. Four metrics define the elements of the matrix: True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN). TP represents the correctly identified benign programs while FP represents the incorrectly classified Spyware programs. Correspondingly, TN represents the correctly identified Spyware programs and FN represents the wrongly identified benign programs. The performance of each classifier was evaluated using the AUC metric and the (overall) Accuracy (ACC) metric. The later is defined in Equation (4.1). AUC is essentially a single point value derived from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [34]. Therefore, it plots the True Positive Rate (TPR, see Equation (4.1)) on the x-axis in function of the False Positive Rate (FPR, see Equation (4.2)) on the y-axis at different points. TPR is the ratio of correctly identified benign programs while FPR is the ratio of wrongly identified Spyware programs. ACC is the percentage of correctly identified programs. In many situations, ACC can be a reasonable estimator of performance (the performance on yet unseen data). However, AUC has the benefits of being independent of class distribution and cost [99]. In many real world problems, the classes are not equally distributed and the cost of misclas-
sifying one class may be different to that of misclassifying another. For such problems, the ACC metric is not a good measure of performance, but it may be used as a complementary metric.

\[ TPR = \frac{TP}{TP + FN} \] (4.1)
\[ FPR = \frac{FP}{TN + FP} \] (4.2)
\[ Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \] (4.3)

### 4.4 Results

Table 4.2. and Table 4.3. show the results for each \( n \)-gram size for both the CFBE and the FBFE method. Two feature sets were produced as a result of the FBFE approach. The first feature set includes instances from the frequency range of 50-80 and the second set includes instances from the 81-500 frequency range. Each table shows the results of the baseline classifier and five different learning algorithms. As stated earlier, we represent classification performance using two metrics: AUC and ACC. Algorithms that are significantly better or worse than the baseline in terms of AUC according to the corrected paired t-test (confidence 0.05, two-tailed) are also indicated. It is not the main objective of this study to determine the optimal algorithm or the highest possible performance. Hence, we did not tune the parameters of the learning algorithms, i.e., all algorithms are default configured.

#### 4.4.1 Results for \( n = 4 \)

Using the feature set produced by the CFBE feature selection method for \( n = 4 \), the J48 decision tree classifier achieves the highest accuracy results. However, it only performs slightly better than the Random Forest model and the JRip classifier. In comparison, the accuracy of Naive Bayes is mediocre while the support vector machines classifier (SMO) achieved the lowest accuracy. In summary, all included algorithms have
<table>
<thead>
<tr>
<th>Method</th>
<th>ZeroR</th>
<th>NaiveBayes</th>
<th>SMO</th>
<th>J48</th>
<th>Random Forest</th>
<th>JRip</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBFE</td>
<td>86.92 (2.72)</td>
<td>88.25 (6.0)</td>
<td>98.72 (7.35)</td>
<td>88.04 (6.13)</td>
<td>89.29 (4.13)</td>
<td>88.04 (6.72)</td>
</tr>
<tr>
<td>FR 81-500</td>
<td>86.92 (2.72)</td>
<td>88.25 (6.0)</td>
<td>98.72 (7.35)</td>
<td>88.04 (6.13)</td>
<td>89.29 (4.13)</td>
<td>88.04 (6.72)</td>
</tr>
<tr>
<td>FR 50-80</td>
<td>86.92 (2.72)</td>
<td>88.25 (6.0)</td>
<td>98.72 (7.35)</td>
<td>88.04 (6.13)</td>
<td>89.29 (4.13)</td>
<td>88.04 (6.72)</td>
</tr>
<tr>
<td>FR 81-500</td>
<td>86.92 (2.72)</td>
<td>88.25 (6.0)</td>
<td>98.72 (7.35)</td>
<td>88.04 (6.13)</td>
<td>89.29 (4.13)</td>
<td>88.04 (6.72)</td>
</tr>
<tr>
<td>FR 50-80</td>
<td>86.92 (2.72)</td>
<td>88.25 (6.0)</td>
<td>98.72 (7.35)</td>
<td>88.04 (6.13)</td>
<td>89.29 (4.13)</td>
<td>88.04 (6.72)</td>
</tr>
<tr>
<td>FR 81-500</td>
<td>86.92 (2.72)</td>
<td>88.25 (6.0)</td>
<td>98.72 (7.35)</td>
<td>88.04 (6.13)</td>
<td>89.29 (4.13)</td>
<td>88.04 (6.72)</td>
</tr>
</tbody>
</table>

Table 4.2: Accuracy results of the experiment.

FR = Frequency Range.
4.4. Results

4.4.2 Results for $n = 5$

Similarly to the results for $n = 4$, the feature set produced by the CFBE feature selection method was suitable as a data representation for the J48 algorithm, which again yielded the best ACC results. In comparison, the SMO produced the worst ACC result of all included algorithms for this particular data set. In fact, the results of the SMO are even lower than the baseline. In contrast, the difference between the results of J48,
Naive Bayes and JRip is small and all algorithms performed better than the baseline. In terms of the AUC, the Random Forest algorithm was the best performer, followed by the J48. For the 50-80 frequency range, the Random Forest algorithm yielded the highest ACC results. However, it only slightly outperformed the JRip rule inducer. J48 performed mediocre at this range. Moreover, in the 81-500 range, the SMO yielded the highest ACC results while NB and RF were mediocre. In terms of the area under the ROC curve, the Random Forest algorithm outperformed the other algorithms for both the 50-80 and the 81-500 range. When comparing results obtained for both frequency results, the 50-80 frequency range seems to be more suitable on average than the 81-500 frequency range.

### 4.4.3 Results for $n = 6$

The data sets generated for $n = 6$ proved to be most successful in terms of both accuracy and the area under the ROC curve. The feature set produced by the CFBE feature selection algorithm was used in conjunction with the J48 decision tree algorithm to yield a top accuracy score of 90.5%. It slightly outperformed the NB algorithm, which was followed by RF and JRip. The support vector machines algorithm yielded the lowest ACC score. All algorithms performed better than the baseline. For the AUC Random Forest was the best performer, yielding a top AUC score of 0.83.

### 4.5 Discussion

The feature sets generated by the CFBE feature selection method generally produced better results with regard to accuracy than the feature sets generated by the FBFE feature selection method. However, the reverse situation seems to be true when AUC is used as an evaluation metric. Overall, the results suggest that the two higher frequency ranges are more suitable than the lowest. The best AUC result was obtained by using the Random Forest algorithm, an $n$-gram size of 6, and the highest frequency range. However, the 50-80 frequency range yielded better
Table 4.3: Area Under ROC Curve results of the experiment

<table>
<thead>
<tr>
<th>n</th>
<th>Method</th>
<th>ZeroR</th>
<th>Naive Bayes</th>
<th>SMO</th>
<th>J48</th>
<th>Random Forest</th>
<th>JRip</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>FR 50_80</td>
<td>0.50(0.00)</td>
<td>0.62(0.20)</td>
<td>0.71(0.18)</td>
<td>0.62(0.16)</td>
<td>0.78(0.20)</td>
<td>0.60(0.16)</td>
</tr>
<tr>
<td></td>
<td>FR 81_500</td>
<td>0.50(0.00)</td>
<td>0.62(0.23)</td>
<td>0.67(0.19)*</td>
<td>0.61(0.17)</td>
<td>0.78(0.20)*</td>
<td>0.56(0.14)</td>
</tr>
<tr>
<td></td>
<td>CBFE</td>
<td>0.50(0.00)</td>
<td>0.62(0.17)*</td>
<td>0.58(0.16)</td>
<td>0.63(0.15)*</td>
<td>0.68(0.22)*</td>
<td>0.59(0.13)*</td>
</tr>
<tr>
<td>5</td>
<td>FR 50_80</td>
<td>0.50(0.00)</td>
<td>0.61(0.17)</td>
<td>0.66(0.17)*</td>
<td>0.64(0.20)*</td>
<td>0.76(0.23)*</td>
<td>0.61(0.16)*</td>
</tr>
<tr>
<td></td>
<td>FR 81_500</td>
<td>0.50(0.00)</td>
<td>0.61(0.16)*</td>
<td>0.67(0.17)*</td>
<td>0.62(0.24)</td>
<td>0.78(0.21)*</td>
<td>0.66(0.19)*</td>
</tr>
<tr>
<td></td>
<td>CBFE</td>
<td>0.50(0.00)</td>
<td>0.61(0.19)</td>
<td>0.62(0.17)</td>
<td>0.63(0.16)</td>
<td>0.75(0.22)</td>
<td>0.61(0.16)</td>
</tr>
<tr>
<td>6</td>
<td>FR 50_80</td>
<td>0.50(0.00)</td>
<td>0.60(0.16)</td>
<td>0.66(0.17)*</td>
<td>0.59(0.15)</td>
<td>0.82(0.17)*</td>
<td>0.57(0.12)</td>
</tr>
<tr>
<td></td>
<td>FR 81_500</td>
<td>0.50(0.00)</td>
<td>0.62(0.18)*</td>
<td>0.71(0.20)*</td>
<td>0.65(0.18)*</td>
<td>0.83(0.17)*</td>
<td>0.66(0.18)*</td>
</tr>
</tbody>
</table>

FR = Frequency Range.

*Significantly better at confidence 0.05 two tailed.
4. Spyware Detection

AUC results than the other frequency ranges when the size of $n$-grams was 4. When comparing the performance on the data sets generated by the FBFE method, it is clear that the ACC results for the 50-80 frequency range are better than those for the 81-500 frequency range. This can easily be viewed in Figure 4.2. Meanwhile, the AUC results are very close to each other for both ranges as shown in Figure 4.3. Consequently, more experiments, e.g., with larger amounts of data and a wider variety of learning algorithms are needed in order to fully understand which data representation and feature selection method is optimal for our purpose.

4.6 Conclusions and Future Work

Data mining-based malicious code detectors have been proven to be successful in detecting clearly malicious code, e.g., like viruses and worms. Results from different studies have indicated that data mining techniques perform better than traditional techniques against malicious code. However, Spyware has not received the same attention from researchers, but it is spreading rapidly on both home and business computers. The main objective of this study was, therefore, to determine whether Spyware could be successfully detected by classifiers generated from $n$-gram based data sets, which is a common data mining-based detection method for viruses and other malicious code.

In order to find a suitable data representation, we analyzed $n$-gram-based byte sequences of different sizes from a range centered on $n = 5$, which has proven to be an appropriate value that yields high performance for similar experiments. We then evaluated five common learning algorithms by generating classifiers and using 10-fold cross-validation and the corrected paired t-test. Moreover, two different feature selection methods were compared for all algorithms and $n$-gram sizes. Since no suitable Spyware data set was available, we collected Spyware and legitimate binaries and generated a small data set for the purpose of validating our approach. The experiments indicate that the approach is successful, achieving a 90.5 % overall accuracy with the J48 decision tree
algorithm when using \( n = 6 \) and the common \( n \)-gram feature selection method. The success of the approach is also indicated by an AUC score of 0.83 with the Random Forest algorithm when using \( n = 6 \) and the frequency-based feature selection method. Currently, the false positive rate is quite high for most combinations of algorithms and data sets. However, we believe that one of the primary reasons for this is that the data set is small. In particular, the number of Spyware files is too low. In data mining, it is believed that larger set of data can produce better results [34]. So a larger set of data can be tested to have better classification with higher ACC and lower false positive rate. With regard to AUC, which is our primary evaluation metric, all algorithms were statistically significantly better than the baseline, but with different combinations of \( n \)-gram size and feature selection method. Thus, from our experiments, we can conclude that it is possible to detect Spyware by using automatically generated classifiers to identify patterns in executable files.

We hope that data mining techniques can help the researcher community and security experts to label the software or the home user to have an informed decision before installation of any software. For future work, we plan to gather a larger collection of binary files, especially Spyware binaries as no standard data set of Spyware is currently available, and to evaluate our approach when the data set features represent OpCodes instead of arbitrary bytes. Additionally, we aim to develop a hybrid Spyware identification method that is based on a combination of EULA-based and executable-based detection techniques.
Chapter 5

Accurate Adware Detection using OpCode Sequence Extraction

Raja Khurram Shahzad, Niklas Lavesson, Henric Johnson

Abstract

Adware represents a possible threat to the security and privacy of computer users. Traditional signature-based and heuristic-based methods have not been proven to be successful at detecting this type of software. This paper presents an adware detection approach based on the application of data mining on disassembled code. The main contributions of the paper is a large publicly available adware data set, an accurate adware detection algorithm, and an extensive empirical evaluation of several candidate machine learning techniques that can be used in conjunction with the algorithm. We have extracted sequences of opcodes from adware and benign software and we have then applied feature selection, using different configurations, to obtain 63 data sets. Six data mining algorithms have been evaluated on these data sets in order to find an efficient and accurate detector. Our experimental results show that the proposed approach can be used to accurately detect both novel
5. Adware Detection

and known adware instances even though the binary difference between adware and legitimate software is usually small.

5.1 Introduction

The aim of this study is to investigate adware detection and to develop an algorithm that accurately detects known and unknown adware instances. Adware may be defined as software that is installed on the client machine with the objective of displaying ads for the user of that machine [100]. Basic adware may also be bundled with extra functionality or software to invade the privacy of a user by monitoring his or her surfing activities or preferences in order to display related pop-up or pop-under advertisements. However, advanced adware may, for example: read data from locally stored files, collect surfing or chat related information, and even create remote connections for transferring and installing software in the future by making a system vulnerable and compromised [100]. These and similar capabilities may even turn the adware into a Spyware or some other type of malicious software (malware). Arguably, adware compromises the confidentiality, and in some cases also the integrity and availability of computer systems. Analogously to computer viruses, which infect executable computer files, adware may be installed automatically when the user visits infected websites, installs freeware or shareware or when the user tries to open infected E-mail attachments [101]. The presence of adware is often mentioned in the End User License Agreement (EULA), but in a manner, which makes it difficult for the average users to comprehend or even notice [94]. As a consequence, the user’s informed consent is thus not obtained. Because of its presence in the EULA, adware vendors often claim that their software should be regarded as benign [94]. Such claims along with the differences in policies and regulations decided upon by different countries place adware in a grey zone in terms of legal status.

The adware problem is growing continuously due to the profound monetary gains for adware developers [100,102]. The users’ awareness about
adware and its potential consequences is generally considered to be low [101]. Currently, the major commercial anti-virus tools try to detect instances of adware by relying on static or dynamic analysis such as signature-based and heuristic approaches (which were developed for detection of viruses). These techniques have a deficiency in detecting unknown or new instances and can be bypassed in different ways [88]. Two popular commercial tools for adware detection are SpyBot\(^1\) and Ad Aware\(^2\), which rely on signature based approach. Hence they require frequent update of their signature database and can detect only known instances.

Consequently, in this paper, we present a (static) detection method based on data mining. We have proposed an automated means for extracting instruction sequences (ISes) from adware and benign files in order to capture the behavior of the corresponding software. Our method extracts the operation code (OpCode) from each instruction and then produces a data set in which each instance is described by sequences of opcodes. As the remainder of this paper will show, our approach is feasible for detecting adware despite the fact that the binary files of this type of software are sometimes quite similar to legitimate software.

### 5.1.1 Aim and Scope

In this paper, we present the results from an experimental study of adware detection. The aim is to determine the success of using data mining techniques for the detection of unseen and new instances of adware. Additionally, we investigate the relationship between OpCode \(n\)-gram size and the number of features required to generate accurate detection models. Our hypothesis is that: it is possible to find a balance between the size of \(n\)-grams (that are used to represent OpCode sequences) and the number of features (the number of \(n\)-grams) that yields a model of reasonable classification performance.

\(^1\)http://www.safer-networking.org/
\(^2\)http://lavasoft.com
5. Adware Detection

5.2 Background

Adware is different from other malware since it may be installed with or without the consent of the user [103]. Users may accept its presence knowingly for using freeware software or unknowingly when it is obfuscated in the EULA. The user may also be fooled into installing adware when trying to install other software or the installation of adware may be carried out as a background task without any human interaction at all [103]. Thus, it is important to be able to automatically detect adware. As mentioned earlier, traditional detection techniques, i.e., signature-based and heuristic methods have a deficiency in detecting novel instances of traditional malware, Spyware and adware. In the signature-based technique, specific features or unique strings are extracted from binaries, which are later used for detection of malware. However, a copy of the malware is required to extract and develop a signature for detection purposes. Due to this fact, signature-based techniques are usually not capable of detecting novel (unseen) instances of malware. In the heuristic technique, human experts define rules for detecting behavioral patterns for malware detection. This technique is capable of detecting novel instances albeit with limited capacity and may be prone to false alarms.

5.2.1 Data Mining-based Detection

To overcome the aforementioned deficiency in detection techniques, Machine Learning (ML) methods, as an alternative approach, were applied for malware detection in 2001 [48]. Since then, different studies have been conducted for detection of traditional malware such as viruses, worms, and so forth, by applying ML and Data Mining (DM) technologies. DM helps in analyzing the data, with automated statistical analysis techniques, by identifying meaningful patterns or correlations. The results from this analysis can be summarized into useful information and can be used for prediction [34]. ML algorithms are used for detecting patterns or relations in data, which are further used to develop a classifier or a regression function [34].
For DM purposes, researchers have prepared their experimental data sets either by using different representations of binary files or by extracting a certain type of features that is present in the files. A binary file may be converted into hexadecimal code, binary code or ISes as a means for representation. Moreover, these representations may be further used to create \( n \)-grams, which are fixed-size strings. Other features that are commonly present in files are printable text strings or calls to an application-programming interface. The use of opcodes as an alternative representation has also been suggested in [104]. An OpCode is a part of the instruction for an operation in machine language. It may or may not include one or more operands for performing an operation such as an arithmetical operation or transferring program control.

When the data set is prepared for machine learning classification tasks, a class imbalance problem may arise. Typically, the imbalance problem occurs in a data set when one class has significantly more instances in comparison to another class or other classes. Due to this problem, the generated classifier tends to misclassify instances of the least represented class(es) and thus the problem may result in degradation of classification performance. Therefore, it is necessary to address the imbalance problem during data set preparation. One approach is of course to try to ensure that all classes are equally represented. This approach, however, turns out to be practically impossible to adopt in many real world problems since there usually is a great shortage of data instances of certain classes.

### 5.2.2 Feature Selection

Another important task when preparing the data set is to reduce the data set complexity while maintaining or improving performance of the classification model. For this purpose, the most common approach is to apply a feature selection algorithm. The objective of feature selection is basically to apply a feature quality measure to prioritize the available features and then keep only the best features from the prioritized list. In
5. Adware Detection

the information retrieval domain, the bag-of-words model (in which the logical order of words has no importance) performs better than other models in representing text documents [105].

Different feature selection measures, such as Document Frequency, Gain Ratio, and Fisher Score, are commonly used for obtaining reduced data sets [34]. Categorical Proportional Difference (CPD) is a relatively new addition in the feature selection algorithm family for text classification tasks [106]. The experiments have shown that CPD outperforms common feature selection methods such as chi-square and information gain. CPD represents a measure of the degree to which a word contributes to differentiating a specific class from others [106]. The possible value of CPD is within the interval of -1 and 1. A CPD value close to -1 indicates that a word to large extent occurs in an equal number of instances in all classes and a value in proximity of 1 indicates that a word occurs only in one class. Given that $A$ is the number of times word $w$ and class $c$ occur together and $B$ is the number of times word $w$ occurs without class $c$, then we may define CPD for a particular word $w$ and class $c$ as follows (see Equation (5.1)):

$$CPD(w, c) = \frac{A - B}{A + B}$$ (5.1)

The reduced feature sets can then be used for data mining purposes and can be used as input to learning algorithms. Many types of learning algorithms are available. Therefore, it is important to choose suitable algorithms with respect to the problem at hand.

5.3 Related Work

Due to legal issues and lawsuits from adware vendors, anti-virus vendors are hesitant to classify any software as adware [102]. Therefore, we have not been able to find any specific approaches for detecting adware. However, we argue that it is important to detect adware to let users exercise the right to make an informed choice about the software they install. In previous work, opcodes have been used for detection of
different variants of worms and some types of Spyware [107]. From the original malware, opcodes were extracted and paired with labels. With these pairs, researchers developed signatures, which were matched with pairs of variants of malware. A three-stage scanning was performed, which was successful in detecting the different variants. In another study, an attempt to detect unknown malware was made by extracting opcodes from malware and then converting them into sequences of opcodes [88]. In their experiment, the researchers applied three classifiers out of which two were boosted and achieved 93 per cent accuracy. In yet another study, variable length instruction sequences were used as a representation for the detection of worms. This time, researchers applied Bagging and were successful in achieving 96 per cent accuracy [108]. In an attempt to detect Spyware, \textit{n}-grams of hexadecimal representation were used as features [109]. This attempt was successful in obtaining 90.5 \% accuracy. Most of the reviewed detection experiments on traditional malware were performed using hexadecimal \textit{n}-grams as features. Only a few researchers seem to have considered opcodes as features and then only from the code segment of the studied files [88, 108]. The files in these experiments were disassembled using commercial dissemblers to obtain the IS. Moreover, most of these studies have not considered the class imbalance problem, which may lead to unnecessarily high rates of misclassification. In conclusion, most of the work concerning malware detection focuses on viruses, worms, and trojans. It is not clear whether the same type of detection methods would be successful when dealing with adware, which is more similar to legitimate software than such types of malware. Nevertheless, adware represents a serious threat to privacy and, as such, the research on adware is important, especially in terms of detection approaches.

5.4 Method

We propose a static DM-based analysis method, which includes disassembling the adware and benign files during preprocessing. We aim to evaluate our proposed method for detecting unknown and new in-
5. Adware Detection

stances as well as existing instances of adware.

<table>
<thead>
<tr>
<th>n</th>
<th>Adware Total</th>
<th>Adware Unique</th>
<th>Benign Total</th>
<th>Benign Unique</th>
<th>Final (tf-idf)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4497344</td>
<td>35666</td>
<td>4381315</td>
<td>25921</td>
<td>1236</td>
</tr>
<tr>
<td>3</td>
<td>2998173</td>
<td>452915</td>
<td>2920818</td>
<td>228780</td>
<td>1340</td>
</tr>
<tr>
<td>4</td>
<td>2248586</td>
<td>876451</td>
<td>2190581</td>
<td>440580</td>
<td>1413</td>
</tr>
<tr>
<td>5</td>
<td>1798843</td>
<td>881768</td>
<td>1752439</td>
<td>565536</td>
<td>1518</td>
</tr>
<tr>
<td>6</td>
<td>1499012</td>
<td>804138</td>
<td>1460335</td>
<td>630851</td>
<td>1630</td>
</tr>
<tr>
<td>7</td>
<td>1284845</td>
<td>727570</td>
<td>1251705</td>
<td>656345</td>
<td>1676</td>
</tr>
<tr>
<td>8</td>
<td>1124215</td>
<td>660092</td>
<td>1095219</td>
<td>643148</td>
<td>1753</td>
</tr>
</tbody>
</table>

a. The n column shows the n-gram size.

b. Final column presents the vocabulary obtained on the basis of tf-idf

5.4.1 Overview

The focus of our analysis is Windows-based executable files, since the Windows operating system has been considered to be more vulnerable to adware as opposed to, say, Unix-based operating systems. When any software is disassembled, the generated output contains text, which may represent hexadecimal dumps, binary dumps or ISes. We argue that text categorization techniques can, therefore, be applied on disassembled output to distinguish between adware and benign software. Thus, we disassemble executable files to obtain ISes and then extract opcodes from those instruction sequences. The extracted opcodes are converted into a vocabulary data set. Each word in the vocabulary data set is an n-gram of a specific size, which represents a feature. Although the size of each word in a particular vocabulary set is fixed, the length is variable. For example, if we observe a data set where the n-gram size is 4 then each word is constructed by joining four opcodes, where each OpCode may have a different length. We use Term Frequency - Inverse Document Frequency (tf-idf) to measure the significance of every word in order to
extract significant features. The generated data is converted into the Attribute-Relation File Format (ARFF) data set file format. The ARFF files are further processed with CPD to obtain feature-reduced data sets, which are used as input to the Waikato Environment for Knowledge Analysis (Weka) [98] to perform the classification experiments. Weka is a suite that includes a large set of machine learning algorithms as well as analysis tools for solving data mining problems.

5.4.2 Data Set Generation

No public data set is available for use in adware detection experiments as opposed to what is available for, e.g., virus and intrusion detection. Therefore, we have created a data set with 600 files out of which 300 files are adware and 300 files are considered as benign. All files represent executable binaries for the Windows operating system. The benign files stem from two sources: a copy of the Windows XP operating system was installed on a clean computer to obtain benign files, e.g., small programs such as notepad, paint, clock, and so forth. Second, to represent files available on the Internet, programs were downloaded from CNET Download3. This website claims to provide Spyware free software; however, when downloaded data set was scanned with a commercial version of the F-Secure Client Security software4, some instances were found infected by so-called riskware. The infected instances were replaced by other benign files. Adware files were obtained from a malware database5.

5.4.3 File Size and Data Size Analysis

File size and data size analysis has to be performed to investigate potential imbalance problems. When adware and benign programs were collected, it was observed that the mean file size of the two software program groups was quite different. Therefore, it was necessary to avoid

---

3http://download.com
4http://f-secure.com
5http://lavasoft.com
an unbalanced number of instructions since different file sizes may produce a varying number of ISes. This may further lead to a class based difference in the generated vocabulary, which in turn may lead to an imbalance problem. Therefore, we decided to restrict the maximum file size to 512 KB for this particular study. It was also considered that the total number of files and the total size of these files should be approximately equal in both data sets.

5.4.4 Disassembly and OpCode Extraction

The collected programs were disassembled to get instruction sequences in assembly Language. This step was performed using the Netwide disassembler\(^6\) (Ndisasm), which is commonly available for UNIX/Linux operating systems. Ndisasm disassembles binary files without understanding (correctly processing) object file formats. The generated output contained the memory address of the instruction, the byte-based location in the file and the instruction itself, i.e., the combination of OpCode and operands. An application was further developed to extract the opcodes from the disassembled file. We did not just include the opcodes from the code segment of the files, but instead used opcodes extracted from any segment in the file.

5.4.5 Parsing and \(n\)-Gram Size

The extracted OpCode data were processed further with a parser that tokenized the data to produce vocabulary/words as per a selected \(n\)-gram size. In a previous research study, an \(n\)-gram size of 4 or 5 yielded promising results for the hexadecimal representation [87,89]. In another study, an \(n\)-gram size of 2 for OpCode representation yielded the best performance [88]. Therefore, we decided to use \(n\)-grams of sizes ranging from 2 to 8 while considering 4 and 5 as intermediary values. The purpose of selecting this range was to evaluate \(n\)-gram sizes in proximity of what has been considered adequate settings in previous research. We created seven master data sets using these \(n\)-gram sizes. Each row

\(^6\)http://http://www.nasm.us/
Table 5.2: Area Under ROC value for \( n \)-gram size of 4

<table>
<thead>
<tr>
<th>( n )-Size = 4</th>
<th>Naive Bayes</th>
<th>SMO</th>
<th>IBk</th>
<th>J48</th>
<th>JRip</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>0.820(0.059)</td>
<td>0.815(0.058)</td>
<td>0.884(0.047)</td>
<td>0.824(0.057)</td>
<td>0.817(0.061)</td>
</tr>
<tr>
<td>20%</td>
<td>0.752(0.066)</td>
<td>0.848(0.043)</td>
<td>0.886(0.043)</td>
<td>0.832(0.058)</td>
<td>0.818(0.064)</td>
</tr>
<tr>
<td>30%</td>
<td>0.879(0.046)</td>
<td>0.906(0.036)</td>
<td>0.920(0.032)</td>
<td>0.884(0.045)</td>
<td>0.889(0.048)</td>
</tr>
<tr>
<td>40%</td>
<td>0.901(0.039)</td>
<td>0.927(0.033)</td>
<td>0.926(0.033)</td>
<td>0.896(0.053)</td>
<td>0.906(0.036)</td>
</tr>
<tr>
<td>50%</td>
<td>0.892(0.036)</td>
<td>0.945(0.031)</td>
<td>0.934(0.031)</td>
<td>0.886(0.049)</td>
<td>0.903(0.045)</td>
</tr>
<tr>
<td>60%</td>
<td>0.863(0.042)</td>
<td>0.942(0.031)</td>
<td>0.945(0.024)</td>
<td>0.888(0.049)</td>
<td>0.906(0.041)</td>
</tr>
<tr>
<td>70%</td>
<td>0.838(0.042)</td>
<td>0.939(0.031)</td>
<td>0.949(0.024)</td>
<td>0.885(0.045)</td>
<td>0.901(0.036)</td>
</tr>
<tr>
<td>80%</td>
<td>0.828(0.044)</td>
<td>0.945(0.026)</td>
<td>0.935(0.029)</td>
<td>0.884(0.044)</td>
<td>0.898(0.043)</td>
</tr>
<tr>
<td>90%</td>
<td>0.822(0.046)</td>
<td>0.944(0.028)</td>
<td>0.934(0.031)</td>
<td>0.884(0.046)</td>
<td>0.911(0.035)</td>
</tr>
</tbody>
</table>
in these data sets represented one word, which is an \( n \)-gram of a specific size. Thus we obtained features of \( n \)-grams with seven different \( n \)-gram sizes. These data sets contain the features with different number of occurrences in each class. We also calculated the number of unique features in one class. Table 5.1. presents the vocabulary statistics for each class and data set.

5.4.6 Feature Selection

The main objective of our particular feature selection step was to obtain sets of features with a different amount of data that represents both adware and benign programs. The output obtained from the previous steps, contains huge vocabularies, which may lead to two problems, i.e., ML algorithms may not process this huge vocabulary and all words in vocabulary do not provide valuable information for classification. We used \( \text{tf-idf} \) for initial feature selection. The frequency \( n \) is number of times a word (or \( n \)-gram in our case) appears in single document, \( d_j \). It is not feasible to use this frequency as basis for selection of words as documents may be of different length so that some words will be more frequent regardless of their actual importance. For normalization purposes, we use Term Frequency (\( \text{tf} \)), which gives a measure of the importance of a word (also known as term) in document \( d_j \). This measure is obtained by dividing the frequency of a word, \( n_{ij} \), with the sum of all frequencies of all words in the document \( d_j \). For obtaining the general importance of word in a document set, \( D \), we use Inverse Document Frequency, \( \text{idf} \). For obtaining \( \text{idf} \), \( D \) is divided by the number of documents that include that particular word and then the logarithm of that value is taken. To get the final measure of a word and filter out common words, we use Term Frequency - Inverse Document Frequency. The \( \text{tf-idf} \) of a word is obtained by multiplying \( \text{tf} \) and \( \text{idf} \) of that particular word. By using \( \text{tf-idf} \), we obtained the final data sets. The total number of final words in every data set varies. In information retrieval, it is common to use a predefined number of words obtained from \( \text{tf-idf} \) (such as the top 1,000 words for both classes or each class). We argue that our problem is different from normal text classification in terms that when the \( n \)-gram
size increases, the number of unique words in each class is also increased (see Table 5.1). Therefore, we let the number of selected words depend on the data set in question instead of a predefined number. There is an additional benefit derived from this feature selection step: suppose a file in the benign data set may be infected with a zero-day threat or that the features extracted from some files are really part of the data segment instead of the code or images. For these cases, the corresponding features will be ignored due to their absence in other files of the same class.

Moreover, the output from the previous step was further processed using the CPD algorithm to create the final data sets. CPD has shown promising results in text classification, but has not been used previously for malware classification. We expected that the use of CPD would lead to better detection performance than other common feature selection methods. As the exact percentage of features to keep in order to yield optimal performance is not known beforehand, we chose to discard features in increments of 10 per cent for every generated data set. Nine final data sets for each $n$-gram size were created. These data sets can be downloaded from http://www.bth.se/com/rks.

5.4.7 Data mining algorithms

Previous studies on similar problems are not conclusive regarding which learning algorithm generates the most accurate classifiers. In a number of studies of malware detection, Ripper (JRip), C4.5 Decision Tree (J48), Support Vector Machines (SMO), and Naive Bayes (NB) performed better than other algorithms. In a previous study of text categorization [23], $k$-nearest neighbor (IB$k$) outperformed NB and other algorithms. Based on previous research, we selected these algorithms as candidates and compared them against ZeroR as a baseline.

ZeroR

ZeroR is a simple, deterministic rule-based algorithm. ZeroR resembles as a random guesser, which could be used to model a user that makes an
uninformed decision about software by always predicting the majority class [34]. This algorithm is frequently used as a baseline to measure the performance gain of other algorithms in classification against chance.

**JRip**

JRip is an implementation of the Ripper algorithm [110], which tries to generate an optimized rule set for classification. Rules are added on the basis of coverage (that is, how many data instances that are matched) and accuracy. Ripper includes intermediate and post pruning techniques to get increase the accuracy of the final rule set.

**J48**

J48 is a decision tree induction algorithm, extended from the ID3 algorithm, which uses the concept of information entropy [111]. Decision trees recursively partition instances from the root node to some leaf node and a tree is constructed.

**SMO**

SMO is an implementation of the support vector machines (SVM) algorithm using Platt’s sequential minimization optimization. During classification, SMO tries to find the optimal hyperplane, which maximizes the distance/margin between two classes thus defining the decision boundaries. It is used for classification and regression [112]. SMO has been generalized in order to be applicable for problems in which there are more classes than two.

**Naive Bayes**

Naive Bayes is based on Bayes theorem and generates a probabilistic classifier with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other [113]. Clearly, such an assumption is violated in most real-world data sets. Neverthe-
5.5 Evaluation Metrics

less, the Naive Bayes algorithm has proven to generate quite accurate classifiers for many problems.

**IBk**

IBk is an implementation of the $k$-nearest neighbor ($k$NN) algorithm, which computes the Euclidean distance between the instance to be classified and the instances included in the training set. Predictions from the neighbors is obtained and weighted according to their distance from the test instance. The majority class of the closest $k$ neighbors is assigned to the new instance [114].

**5.5 Evaluation Metrics**

We evaluated each learning algorithm by performing cross-validation tests. Confusion matrices were generated by using the responses from classifiers. The following four estimates defined the elements of such a matrix: True Positives (TP) represent the correctly identified adware programs. False Positives (FP) represent the incorrectly classified benign programs. True Negatives (TN) represent the correctly identified benign programs and False Negatives (FN) represent the incorrectly identified adware programs.

The performance of each classifier was evaluated using Detection Rate (DR), which is the percentage of correctly identified adware. False Alarm Rate (FAR), which is the percentage of wrongly identified benign programs and Accuracy (ACC), the percentage of correctly identified programs. We argue that, for our problem, False Negative Rate, which is the percentage of incorrectly identified adware programs, is more important than FAR. The last evaluation parameter was Area Under Receiver Operating Characteristic Curve (AUC). AUC is essentially a single point value derived from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [34]. Therefore, it plots the DR on the $x$-axis in function of the FAR on the $y$-axis at different
points. In many situations, ACC can be a reasonable estimator of predictive performance. However, AUC has the benefits of being independent of class distribution and cost [99]. In many real-world problems, the classes are not equally distributed and the cost of misclassifying one class may be different to that of misclassifying another. For such problems, the ACC metric is not a good measure of performance, but it may be used as a complementary metric.

5.6 Experimental Procedure

To investigate our hypothesis that it is possible to find a suitable combination of \( n \)-gram size and the number of features to yield a model of reasonable classification performance, a comprehensive set of evaluation runs was designed. Our experiment used seven different sizes of \( n \)-grams to create data sets and for each specific \( n \) there were nine sub sets ranging from 10 \% features to 90 \% features. In total, we conducted 630 10-fold cross-validation (CV) tests for each classifier, which resulted in 3,780 runs in total. Default configurations were used for all algorithms. We used corrected paired t-test (confidence 0.05, two tailed) to compare each classifier with the base line classifier ZeroR.

5.7 Results

Most of the algorithms performed well when using an \( n \)-gram size of 4 and the 70 \% features data set. The results of all algorithms were compared with the results of ZeroR, which achieves an AUC score of 0.50 (random guessing). Figure 5.1. shows the comparison of all algorithms in terms of AUC score for \( n \)-gram size of 4 with 70 \% features data set. The AUC scores for IB\( k \) for aforementioned data set are presented in Table 5.2. Considering AUC as the primary performance metric, the results clearly show that our proposed methodology is successful in detecting novel (unseen) instances of adware. IB\( k \) achieved the best result (AUC = 0.949, FNR = 0.022 and FAR = 0.115 with \( n=4 \) and 70 \% attributes kept). In terms of FNR on the 70 \% data set for different \( n \)-gram sizes, most
of the algorithms achieved the highest FNR at \( n \)-gram size of 2. NB has shown high variance among all \( n \)-gram sizes for FNR and FAR with highest FNR value of 0.475 for \( n \)-gram size 2 and highest FAR value of 0.335 for \( n \)-gram size of 3. All other algorithms gave their highest FNR and FAR for \( n \)-gram size of 2. The IB\( k \) achieved highest FAR, i.e., 0.462 for \( n \)-gram size of 8.

### 5.8 Analysis

The results clearly show the possibility to detect adware using data mining on disassembled code and thus strengthen the validity of our hypothesis. The aim of this study is twofold. Firstly, we need to evaluate our methodology for detection and secondly, we need to find a suitable combination of \( n \)-gram size with percentage of features. Results have shown that adware could be detected using \( n \)-grams of opcodes. We
5. Adware Detection

have used OpCode sizes ranging from 2 to 8. For our experiments, we have not considered an $n$-gram size of 1 since it has been concluded in a previous study that sequences of opcodes are more suitable than a single OpCode for representation [88]. We have considered the false negative rate (adware classified as benign) because this is more important for a user than the false alarm rate (benign classified as adware). We argue that, if a benign file is classified as adware it may not affect the system as much as if an adware application is classified as benign and thus installed on the system.

5.8.1 Algorithm Performance Analysis

The classifier generated from the IB$k$ algorithm has shown the most promising results in terms of AUC and accuracy for an $n$-gram size of 4 especially for higher percentages of kept features. $k$NN and SVM are effective when the data are noisy. $k$NN has an advantage that its classification performance is refined incrementally when new training samples are introduced. J48 also has shown variance in results for smaller percentages of data. This may be attributed to the fact that for small data sets or in presence of noise, J48 is prone to over-fitting the training data.

NB has not been successful in classification as compared with other classifiers. It is evident that, as $n$-gram size and percentage of data are increased, the performance of NB classifier is varying significantly. This may be because, as the $n$-gram size increases, the number of unique combinations of opcodes in each data set increases as shown in Table 5.1. NB assigns probability to each feature. These unique combinations may be present in only a few instances and so the probability of occurrence is determined to be low in one class. However, this is not the case when using an $n$-gram size of 2 since the occurrence of any combination can be high.

For the studied problem, we may draw the conclusion that an $n$-gram size of 4 seems to be reasonable for good detection. The reason for this could be that at this size each $n$-gram is representing combination of
5.8. Analysis

four instructions sequences, which may be referring a function or interesting feature in the file. This is also easy to track this combination in the malware or benign files for further analysis.

5.8.2 State-of-the-Art

In a previous study on OpCode-based malware detection [88], $n=2$ yielded the best results, but we argue that such short combinations of opcodes may not indicate any important function or set of instructions in the files. Due to these reasons it may be difficult to perform analysis. In another study [108], Bagging was used in conjunction with the Random Forests algorithm. However the basis for this selection was not reported. These experiments were performed on worms and viruses and have shown promising results for detection, but worms and viruses are quite different from adware in that they may exhibit clearly malicious routines; these routines can then easily be identified by human experts. But in the case of adware, the resemblance to benign software is greater. Normal characteristics of adware (such as: the displaying of ads in popup windows or the transferring of information over the network) are also present in several instances of legitimate software. Therefore, it is difficult for human experts to classify a piece of software as adware on the basis of such characteristics.

5.8.3 OpCode Analysis

We decided to use ISes rather than other common representations, such as: hexadecimal $n$-gram representations, printable strings, API calls, or messages because ISes include program control flow information. Moreover, a group of ISes may indicate an interesting function, which can be easily tracked back in the program for deeper analysis. In order to find interesting functions we analyzed models generated by SMO, JRip, NB and J48 for $n$-gram size of 4 with 70 % features. We found that most of the features that were linked to adware by other models were not considered by J48 (e.g., pushcallsbbinc and incaddandinc). This may be
because J48 is considered unstable as a small variation in data set results in selection of different attributes, which affect the descendent sub trees.

5.8.4 Practical Considerations

DM techniques have performed well in detecting adware. But in the case of advanced adware with encrypted functionalities, the static analysis method used in this paper may not be successful albeit the presence of an encrypted segment can potentially be considered as an indication. It could be the case that a dynamic analysis approach has to be applied to detect these instances of advanced adware.

In terms of converting our approach into a practical solution for general users or experts, we argue that IB\textsubscript{k} represents a good choice of algorithm. IB\textsubscript{k} is the simplest algorithm with respect to working as it classifies an instance on basis of majority vote of its k nearest neighbors. The k is a small positive integer due to which the duration for training and building classifier from IB\textsubscript{k} was less than tree based and rule based algorithms where new trees or rules are required to generate or update previous. JRip algorithm was most expensive algorithm in terms of time consumed to train and generate the model, due to which this may not be considered feasible option for users. J48 algorithm was better than JRip in terms of results and time consumed for the training, but still it was expensive than other classifiers, due to which it is also not suitable candidate. SMO was nearest to IB\textsubscript{k}, so it may be used as an alternative or to complement the results of IB\textsubscript{k}. Another alternative for adware detection may be combining DM techniques with EULA analyzer as many adware vendors mention the presence of their adware in EULA to avoid legal consequence. In this way, we argue that advanced adware with encrypted routines can also be detected.

5.9 Conclusion and Future work

Many papers have been devoted to the study of detection approaches for malware such as viruses, worms, and trojans. However, less work
5.9. Conclusion and Future work

has been done in the area of adware detection. We argue that this has little to do with the fact that adware is considered less harmful. Rather, it seems that the area of adware is avoided due to the fact that this type of software resides in a legal grey zone: some people regard adware as legitimate and others perceive adware as harmful. This paper considers the latter perception. We have presented a static file analysis method, based on operation code mining, for adware detection. A series of experiments with data sets generated using different $n$-gram sizes were performed. The experiments show promising results in terms of the area under the ROC curve (AUC) for the detection of novel instances of adware based on previously unseen examples, while maintaining a low false negative rate. The highest classification performance (AUC score of 0.949) was achieved by the k-nearest neighbor algorithm. Another conclusion inferred from these experiments is that, as the size of $n$-grams and the percentage of features are increased, the detection performance also increases. However, an $n$-gram size of 4 seems to represent a local optimum, at least for the studied algorithms. For future work, we plan to perform experiments on a larger collection of adware and benign files by introducing a hybrid identification method, which uses the combination of $n$-grams of opcodes and features extracted from EULAs. We plan to combine dynamic and static analysis techniques to be able to detect basic as well as advanced adware.
Detecting Scareware by Mining Variable Length Instruction Sequences

Raja Khurram Shahzad, Niklas Lavesson

Abstract

Scareware is a recent type of malicious software that may pose financial and privacy-related threats to novice users. Traditional countermeasures, such as anti-virus software, require regular updates and often lack the capability of detecting novel (unseen) instances. This paper presents a scareware detection method that is based on the application of machine learning algorithms to learn patterns in extracted variable length OpCode sequences derived from instruction sequences of binary files. The patterns are then used to classify software as legitimate or scareware, but they may also reveal interpretable behavior that is unique to either type of software. We have obtained a large number of real world scareware applications and designed a data set with 550 scareware instances and 250 benign instances. The experimental results show that several common data mining algorithms are able to generate accurate
6. **Scareware Detection**

models from the data set. The Random Forest algorithm is shown to outperform the other algorithms in the experiment. Essentially, our study shows that, even though the differences between scareware and legitimate software are subtler than between, say, viruses and legitimate software, the same type of machine learning approach can be used in both of these dissimilar cases.

6.1 **Introduction**

This paper addresses the problem of detecting scareware, i.e., scam software with different forms of malicious payloads\(^1\), and presents a machine learning-based approach for detection of this type of software. Many reports have been published in the media regarding malicious software (malware) such as viruses and worms. Such reports have arguably increased the awareness of novice computer users about basic security issues. Consequently, users are becoming more conscious about the security and privacy of their systems and data. Through media, friends, colleagues and security experts, users have been advised to install detection software such as anti-virus software or other protective measures. Success stories about the detection and mitigation of virus and worm threats have probably also played a part in enhancing the general security awareness. However, personal computers are becoming more and more attractive targets for cyber criminals due in part to the electronic financial transactions that are nowadays performed by private citizens from their personal computers. Notable examples of such transactions include Internet bank sessions and online credit card payments. This development has caused a shift in the focus of the malware authors from self-spreading malware such as worms, which are easily detectable due to their distribution techniques, to privacy invasive malware [115].

A recent addition to the family of privacy invasive malware is known as rogue software, rogueware, or scareware. In the remainder of this paper, the latter term will be used to denote this type of software. Scare-\(^1\)

\(^1\)http://www.microsoft.com/security/antivirus/rogue.aspx
ware represents scam applications that usually masquerade as security applications such as anti-malware software or more specifically anti-virus software. In reality, however, scareware provides a minimum level of security or no security at all\(^2\),\(^3\). This type of software is especially crafted to include fake scanning dialogs, artificial progress bars and fake alerts \(^{116}\). Scareware may display fake lists of infected files and sometimes such lists are so unsophistically generated that they include files that may not even exist on the computer or they may be incompatible with the operating system \(^{117}\). Figure 6.1. shows the fake scanning dialog of a particular scareware Rouge:W32/Winwebsec \(^{118}\). The fake scanning processes and the fake results are of course used to scare users into believing that their system has been compromised and that it is infected with malicious content. The fake presentations are essentially carried out to convince the user that they need anti-virus software or some other form of protection. As a remedy for the fake situation, scareware offers a free download, which may also be bundled with other malware (such as trojans) or it may facilitate the installation of additional malware. Scareware may also trick users into paying registration fees in order to scan their system more thoroughly to remove (fake) warnings. Such an example is shown in Figure 6.2., which is the screen-shot of payment screen displayed by Rouge:W32/Winwebsec \(^{118}\). The additional malware, which has been installed instead of protective software, remains on the targeted computer regardless of whether the registration fee is actually paid or not. Such additional malware is typically used to collect personal data of the user or to launch different forms of attacks.

### 6.1.1 Background

In 2003, Secure Works observed that spam advertisements for fake anti-virus software were being sent to users by the utilization of a vulnerability in the Microsoft Messenger Service\(^4\). Two years later, in 2005,

\(^{2}\)http://voices.washingtonpost.com/securityfix/2009/03/obscene_profits_fuel_rogue_ant.html
\(^{3}\)http://www.lavasoft.com/
\(^{4}\)http://www.secureworks.com/research/threats/popup-spam/
Microsoft reported about the presence of scareware on web sites and web servers. Since then and arguably due to the overwhelming financial incentive to malware authors, the scareware has been increasing. The scareware distribution mechanism is different from other malware (such as viruses or worms). Scareware reaches the target machine by employment of social engineering, stealth techniques, or both of these approaches. User interaction is required when scareware is distributed through social engineering. For this purpose, advertisements are either sent via spam e-mail or posted on popular social networking web sites. Scareware is misleadingly marketed as legitimate software and, with user interaction; it is downloaded and installed on personal computers. When it comes to stealth techniques, vulnerabilities in web browsers or other popular software are exploited in order to employ a so-called drive-by download mechanism. Essentially, scareware is downloaded and installed without any user interaction using such a mechanism. It has been reported that the monetary conversion rate of the fees obtained for fake scanning services can be as much as 1.36 \%, which can result in a gross income of $21,000 - $35000 for a period of 44 days \[115\]. Panda Labs reported that an approximate overall gross income of 34 million per
month is generated by scareware [119]. Late in 2009, Symantec Corporation reported about 43 million installation attempts from more than 240 distinct families of scareware\(^5\). Recently a Swedish newspaper Aftonbladet\(^6\) reported that according to U.S Department of Justice, 9, 60,000 users were victims of rouge, which caused a loss of 460 million krona. This alarming situation has received the attention of legitimate security software companies. CA Global Security Advisor\(^7\), Secure Works\(^8\) and Microsoft\(^9\) published advisories about scareware, which describe the general functionality of scareware and tips for identifying this type of software [117]. To reduce the probability of being fooled by scareware, novice users are advised to install legitimate anti-malware software. However, the problem with such software is that users need to update it on regular basis as novel families of scareware are continuously appearing.

### 6.1.2 Traditional Countermeasures

Current anti-malware programs primarily rely on either signature-based methods or heuristic-based methods for the detection of scareware; techniques that were originally developed for detecting computer viruses. The signature-based approach revolves around the use of signature databases that contain byte strings that are unique to different instances of software. If these databases are allowed to become more than a couple of weeks old, the detection rate will be significantly reduced due to the fact that the approach cannot detect recent scareware for which it lacks recorded signatures [117]. The second approach, which relies on heuristic-based methods, is based on more general rules that, e.g., may define malicious or benign behavior. For both methods, anti-malware vendors need to catch novel instances, analyze them, create new signatures or rules and then update their databases. It is safe to say that,
6. Scareware Detection

between database updates, users may be exposed to novel scareware instances. Thus, it is important for users to have up-to-date and reliable protective measures.

6.1.3 Scope and Aim

In this paper, we present results from an experimental evaluation of a new scareware detection method. The aim of this method is to extend the traditional heuristic detection approach by employing machine learning. The objectives of the study are to assess the performance of the proposed method, which can be described as an automated system for extracting typical behavior of scareware and benign software in the shape of variable length instruction sequences, and to analyze such fragments of behavior in order to improve upon the existing knowledge about scareware detection.

6.1.4 Outline

The remainder of this paper is organized as follows. Section 6.2 presents related work by first introducing necessary concepts and terminology in
Section 6.2.1 and then reviewing related studies in Section 6.2.2. Section 6.3 then describes the employed methodology and the data preprocessing steps. Section 6.4 reviews the experimental procedure. The subsequent sections present the experimental results and the analysis. Finally, Section 6.7 concludes the paper and gives some pointers to future work.

6.2 Related Work

6.2.1 Concepts and Terminology

To overcome the deficiency of traditional techniques concerning the detection of novel instances, the potential of various approaches, such as agent-based technologies and artificial neural networks have been investigated. Data mining (DM) and machine learning (ML) methods have been extensively investigated in the field of text classification and have showed promising results for many applications. As we shall see, it is possible to benefit from this area of research when addressing the scareware detection problem. However, the idea of using DM and ML methods for making the malware detection process automated and for extending the heuristic-based detection approach for traditional malware is not new; it originates from a study conducted in 2001 [48].

The process of ML-based malware classification essentially follows standard classification and can thus be divided into two sub stages: training and testing. During the training stage, classifiers are generated from training sets that feature some type of extracted malware and benign file information as well as the predetermined classification of each file and the predictive performance of the classifiers is then evaluated during the testing stage. For malware classification, data sets have been prepared using various representations of files and by using different features that are either present in the files or obtained from any kind of meta analysis (for example, runtime generated digital footprints). Features that are commonly extracted from a binary file include: byte code n-grams, printable strings and instruction sequences. The n-gram is a sequence of
n characters or n extracted words. Other features that are present in binary files and that may also be used include system calls (to application programming interfaces). The use of opcodes as an alternative form of representation has also been suggested [104]. An assembly instruction contains an operation code (OpCode) and maybe one or more operands for performing the operation. Opcodes or sequences of opcodes may be represented using n-grams, which, in turn, can be viewed upon as words or terms if the learning problem is defined as a text categorization problem.

In text categorization, text files are commonly represented using the bag of words model, which is based on Salton’s vector space model [120]. A vocabulary of words or terms is extracted from the so-called document set. For each term (t) in the vocabulary, its frequency (f) in a single document (d) and in the entire set (D) is calculated. A weight is assigned to each term, usually equal to its f in d; such weights are denoted term frequencies (tf). When the frequency (F) of each term is calculated in D, this is called Document Frequency (DF). The tf value of a term is further divided by the frequency of the most frequent term in the document, i.e., max(tf) to obtain a normalized Term Frequency (TF) within the range of [0-1] as shown in Equation (6.1). An extended version of TF-DF is TF Inverse Document Frequency (TF-IDF), which combines TF and DF as shown in Equation (6.2); where N is the number of documents in the entire data set and DF is number of d in which t appears.

\[
\text{TermFrequency} = \frac{tf}{\max(tf)} \quad (6.1)
\]

\[
\text{TFInverseDocumentFrequency} = TF \times \log \left( \frac{N}{DF} \right) \quad (6.2)
\]

The problem of n-gram-based malware classification in this context is perhaps different from the general text categorization case since a huge vocabulary or very large feature sets have to be produced. The size of the vocabulary creates two problems: most ML algorithms cannot directly process the vocabulary and a vast number of terms in the vocabulary do not provide any valuable information for classification. Therefore,
it is necessary to obtain a subset of features by applying feature selection. The Categorical Proportional Difference (CPD) algorithm is a rather recent example of such an algorithm. In a number of text categorization experiments, CPD has outperformed other traditional feature selection algorithms such as: chi-square, information gain, mutual information, and odds ratio [106]. CPD represents a measure of the degree to which a word contributes in discriminating a specific class from other classes [106]. The possible outcome of CPD falls between [-1…1] where a CPD value close to -1 indicates that a word occurs in an equal number of instances in all classes and a value of 1 or in proximity to 1 indicates that a word occurs only in one class. Let \( A \) be the number of times word \( w \) and class \( c \) occur together and let \( B \) the number of times word \( w \) occurs without class \( c \), then we may define CPD for a particular word \( w \) and class \( c \) as shown in Equation (6.3):

\[
CPD(w, c) = \frac{A - B}{A + B}
\]  

(6.3)

The reduced feature set can now be converted, e.g., into the Attribute-Relation File Format (ARFF). ARFF files are structured ASCII text files that include a set of data instances, each described by a set of features [34]. ARFF files are used as input to the Waikato Environment for Knowledge Analysis (Weka) [98] before applying learning algorithms in order to build and analyze classifiers. Essentially, Weka is a suite of machine learning algorithms and analysis tools for solving or analyzing data mining problems. There are, of course, many alternatives to Weka, but we argue that this workbench is particularly fitting for developing our approach since it is released as open source and may be tuned, extended, or changed in any way.

### 6.2.2 Related Directions of Research

Opcodes have already been used to build signature databases that can be searched to detect different variants of worms [107]. To avoid the problem of having to manually update the databases of the scanners, data mining algorithms were later used as part of a scientific study to build
6. Scareware Detection

a generic scanner [92]. In this study, experiments were performed on two different data sets: the first data set contained the OpCode of each instruction and the second data set contained the OpCode as well as the first operand of each instruction. The frequency of appearance in the virus class and in the benign class was used as a basis for feature selection. Results showed that the first data set produced better results than the second. In another study, OpCode $n$-grams of different sizes were constructed to detect novel malware. By addressing the class imbalance problem properly, an accuracy of 96 % was achieved [88]. The idea of using variable length instruction sequences was conceived as part of an attempt to detect worms. Frequently occurring instruction sequences were analyzed using ensemble learners to classify novel instances of worms. In an attempt to detect a more recent type of malware, called Spyware, hexadecimal $n$-grams were used to represent binary files [109]. The most common $n$-grams for each class together with overall high frequency $n$-grams were used as features for building the classifiers. The Spyware detection rate was recorded to be 90.5 %.

Hexadecimal $n$-grams have been used extensively as features in traditional malware classification problems. Experiments have been performed on viruses, worms and trojans. These types of malware are typically very distinct from the standard benign software program. Moreover, only a few studies have used only the OpCode from the instruction as the feature of choice [88,92,107]. Today, very little is known about the appropriateness of using OpCodes or instruction sequences as features when trying to detect the type of malware that is more similar to benign software in terms of behavior. In this paper, we investigate the concept of scareware which, to the best of our knowledge, has not been investigated in terms of how well it can be detected by mining instruction sequences.
6.3 Methodology

Generalizing the scareware detection method so it can detect novel instances can arguably be regarded as quite important for user protection. Another problem regarding the detection of scareware is that it may resemble legitimate software to such extents that it is difficult to detect differences. Recently, data mining classification algorithms have been heavily applied in order to automate and extend the heuristic-based methods for detection of traditional malware. It is, therefore, of interest to investigate how well such classification algorithms can detect scareware. Consequently, we present a static analysis method based on data mining, which extends the general heuristic detection approach. In this context, a dynamic analysis method is used to detect malware instances by investigating runtime footprints while static analysis is carried out on files without any runtime execution. Our data set contains Windows-based executable scareware and benign files and this choice was made since the Windows operating system is still the most commonly used operating system, especially for novice users, and it is often considered more vulnerable than, e.g., Unix-based operating systems. We have disassembled our initial file database into instruction sequences and then we extracted the OpCodes from each instruction. The extracted OpCodes were combined into ordered lists, instruction sequences (IS), to produce our vocabulary. Each word in vocabulary is of variable length. We have used TF-IDF and CPD for generating the final data sets.

6.3.1 File Sampling and Data Set Design

As the threat of scareware is relatively new compared to, say, viruses and worms, there is unfortunately no default or public data set available for researchers to build classification models from. Therefore, we have created a data set of 800 files out of which 550 are scareware (provided by Lavasoft\textsuperscript{10} from their scareware collection). The remaining 250 files are benign and were downloaded from the web site CNET Download\textsuperscript{11}.

\textsuperscript{10}https://lavasoft.com
\textsuperscript{11}http://download.com
6. Scareware Detection

This web site claims that the software provided is Spyware free. However, after downloading software from the web site and scanning it with a commercial version of the F-Secure Client Security software\textsuperscript{12}, we discovered that some files were actually infected by so-called riskware. The infected instances were removed from the data set.

6.3.2 Extraction and Data Preparation

For the purpose of our experiment, we needed to convert our data set to a format that could be processed by learning algorithms. We decided to represent files by using extracted instruction sequences as features. The advantage of using IS as a primary feature is that IS represent program control flow blocks, which cannot be presented by binary or hexadecimal \( n \)-grams or printable strings. Moreover, each IS in this study represents a function that can be located within the actual program for the purpose of deeper analysis, even though such a step is out of scope in the presented paper. We disassembled each program using the Netwide disassembler\textsuperscript{13} (Ndisasm), which was configured to auto-synchronous mode to avoid misalignment between the data and code segments. The generated output, from all the file segments, was stored in regular text files and each entry contains the memory address of the instruction field as well as the OpCode and the operands. The disassembled files were further processed through a parser to obtain the instruction sequences (ordered lists of OpCodes). During the extraction process, the end of an instruction sequence was determined by identifying a conditional or unconditional control transfer instruction or function boundary. It is worth noting that these identified control transfer instructions (such as: call, iret, jmp or jnz) were not included in the generated instruction sequences. In this way, we obtained variable length instruction sequences. Each row in output contains single IS. Figure 6.3. shows the instruction sequences extracted from a scareware Rouge:W32/Winwebsec\textsuperscript{14} along-with some other related information of this particular scareware.

\textsuperscript{12}http://www.f-secure.com/
\textsuperscript{13}http://nasm.us/
\textsuperscript{14}http://winsec.se/?cat=188
6.3. Methodology

popawxorimuladd
mulincaddaddpushpushimuldbpush
dbandmovinc
pushincaddmovaddincaddadd
inandmovpushpushpushpushpushpushpush
fisubincaddpushpushincadd
addpush
xlatband
addaddstdadcincaddrclpushpush
adcincaddpushpushand

Name: Rogue:W32/Winwebsec
Aliases: Program:Win32/Winwebsec (Microsoft)
Category: Riskware
Type: Rogue
Platform: W32

Figure 6.3: Instruction Sequence Extracted from Rogue:W32/Winwebsec

6.3.3 Feature Selection

Feature selection is performed to measure the correlation of each feature with its class (scareware or benign). It is also performed to estimate the role of that specific feature in classification task. The measures used for feature selection by any feature selection methods are not biased to any classification algorithm or class, which helps us in comparing the performances of different classification algorithm.

Our disassembled files were in text format and each file can be read as text string so we decided to use the bag of words model, since it has been proven to be a suitable model for similar problems. We used the String-ToWordVector filter in Weka to parse each string, extract the vocabulary and produce word vectors. For our experiment, each word represents a unique IS. We used TF-IDF for the weight calculation of each word. Our vocabulary features top 1,625 unique words. We decided to perform a secondary feature selection to eliminate features, which will not contribute significant in classification task. We applied CPD to obtain reduced feature sets. As it is difficult to know beforehand the optimal number of features to remove, we decided to generate a number of data sets where each set was generated by keeping a different number of at-
tributes. This process resulted in 19 reduced data sets for which 5-95 % of the original features were kept.

6.4 Experiment

The aim of the experiment is to evaluate classifier performance on the task of detecting scareware by learning from variable length instruction sequences and to assess the impact of feature selection using categorical proportional difference. Learning algorithms can be categorized according to their learning bias, that is, by the way their designs restrict the search space and dictate how this space is traversed. When categorizing the commonly used algorithms in this manner, a rather small number of algorithm families can be identified, e.g., tree inducers, rule set inducers, neural networks, instance based learners, and Bayesian learners. We have tried to select at least one representative algorithm from each family. As our study extends the heuristic based detection technique that uses rules set, so we used families of algorithms that either uses rules or help in developing rule set. These families of algorithms are rules based and decision tree. Except these families we also used support vector machine, Bayesian theorem based algorithms and nearest neighbor concepts for classification. All the algorithms were used at their default configuration in Weka.

6.4.1 Learning algorithms

ZeroR

ZeroR is a rule-based algorithm. ZeroR works as a random guesser, modeling a user that makes an uninformed decision about software by always predicting the majority class (the class to which most of the data instances belong) [34]. This algorithm is frequently used as a baseline to measure the performance gain of other algorithms in classification against chance.
6.4. Experiment

**JRip**

JRip is an implementation of the Ripper algorithm. This algorithm tries to generate an optimized rule set for classification. Rules are added on the basis of coverage (that is, how many data instances they cover) and accuracy [110]. A data instance is classified as positive if a rule matches; otherwise it is classified as negative. JRip also features an optimization step in which redundant or bad rules are discarded.

**J48**

J48 is a decision tree based learning algorithm, which uses the concept of information entropy [111]. Decision trees recursively partition instances from the root node to some leaf node and a tree is constructed. For partitioning, J48 uses the attribute with the highest information gain and stops if all instances of same class are present in the subset. In learning, they adopt top-down approach and traverse the tree to make a set of rules, which is used for classification.

**Sequential Minimal Optimization**

Sequential Minimal Optimization (SMO) belongs to support vector machines. During classification, SMO finds the optimal hyper-plane, which maximizes the distance/margin between two classes thus defining the decision boundaries. It is used for classification and regression [112].

**Naive Bayes**

Naive Bayes (NB) is based on Bayes theorem and generates a probabilistic classifier with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other. Therefore, presence (or absence) of a particular feature of a class is not dependent on the presence (or absence) of any other feature [113].
6. Scareware Detection

**IBk**

IBk is $k$-nearest neighbor classifier, which uses Euclidean distance [114]. Predictions from the neighbors is obtained and weighted according to their distance from test instance. Majority class of closest $k$ neighbors is assigned to new instance.

**Random Forest**

Random Forest (RF) is an ensemble learner. A specified number of decision trees are created and their mode is obtained for prediction predictions [121]. Being an ensemble learner, it has superiority of having combined decision, which is not the case for other algorithms, therefore, it is expected to produce better accuracy than single decision tree.

**6.4.2 Evaluation**

We tested each learning algorithm by performing 10 fold cross-validation (CV) tests to ensure that the generated classifiers are not tested on the training data. Confusion matrices were generated by using the responses from classifiers. The following four measures defined the elements of the generated confusion matrices: True Positives (TP) represent the correctly identified scareware programs, False Positives (FP) represent legitimate software that has been classified as scareware, True Negatives (TN) represent correctly identified legitimate programs and False Negatives (FN) represent scareware programs that were incorrectly classified as legitimate software applications. We argue that the false negatives carry the highest cost from the users’ perspective.

The performance of each classifier was evaluated using Detection Rate (DR), which is the percentage of correctly identified scareware, as shown in Equation (6.4). False Negative Rate, which is the percentage of wrongly identified malicious programs (see Equation (6.5)), and Accuracy (ACC), the percentage of correctly identified programs (see Equation (6.6)). The last evaluation measure used was Area Under Receiver Operating Characteristic Curve (AUC). AUC is essentially a single-point value derived
from a ROC curve, which is commonly used when the performance of a classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [34]. Therefore, it plots the DR on the $x$-axis in function of the False Positive Rate (FPR) on the $y$-axis at different points. FPR is the percentage of wrongly identified benign programs. The higher AUC of an algorithm indicates that this algorithm is more robust and better in classification. In many situations, accuracy can also be a reasonable estimator of performance (the performance on completely new data). However, AUC has the benefits of being independent of class distribution and cost [99] unless the skewness of the class distribution is extreme.

\[
DetectionRate = \frac{TP}{TP + FN} \quad (6.4)
\]

\[
FalseNegativeRate = \frac{FN}{TP + FN} \quad (6.5)
\]

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (6.6)
\]

### 6.5 Results

The main experimental results, that is, the AUC of the seven included algorithms on the 19 data sets, are shown in Table 6.1. In this table, ZeroR is used as a baseline (and can be regarded as a random guesser) with AUC of 0.500 (0.000) for all the data sets. All algorithms had performed better than base algorithm. Random Forest outperformed the other algorithms and its best performance (DR of 0.977, FNR of 0.023 and FPR of 0.197) was recorded at the 60 % keep level (a data set with 974 features). The Naive Bayes yielded an acceptable detection rate (0.857), but its FPR at different data sets was too high (i.e., up to 0.688) for practical use. Moreover, Naive Bayes also exhibited a high variance in performance related to the different data sets. Due to this behavior, it is not possible to consider this algorithm as reliable for the studied problem. On the same data set other algorithms also achieved either the highest AUC or near to the highest value with ignorable differences such as SMO achieved
## Table 6.1: Learning algorithms AUC results for different levels of feature selection

<table>
<thead>
<tr>
<th>Data Set</th>
<th>ZeroR</th>
<th>SMO</th>
<th>Naive Bayes</th>
<th>IB</th>
<th>J48</th>
<th>JRip</th>
<th>J48</th>
<th>JRip</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>0.500 (0.000)</td>
<td>0.717 (0.119)</td>
<td>0.787 (0.030)</td>
<td>0.778 (0.030)</td>
<td>0.657 (0.129)</td>
<td>0.500 (0.000)</td>
<td>0.781 (0.030)</td>
<td></td>
</tr>
<tr>
<td>10%</td>
<td>0.500 (0.000)</td>
<td>0.812 (0.038)</td>
<td>0.829 (0.044)</td>
<td>0.851 (0.036)</td>
<td>0.809 (0.040)</td>
<td>0.788 (0.031)</td>
<td>0.857 (0.036)</td>
<td></td>
</tr>
<tr>
<td>15%</td>
<td>0.500 (0.000)</td>
<td>0.860 (0.045)</td>
<td>0.802 (0.049)</td>
<td>0.896 (0.037)</td>
<td>0.817 (0.054)</td>
<td>0.869 (0.043)</td>
<td>0.937 (0.027)</td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td>0.500 (0.000)</td>
<td>0.878 (0.040)</td>
<td>0.814 (0.045)</td>
<td>0.928 (0.036)</td>
<td>0.868 (0.047)</td>
<td>0.877 (0.048)</td>
<td>0.958 (0.021)</td>
<td></td>
</tr>
<tr>
<td>25%</td>
<td>0.500 (0.000)</td>
<td>0.864 (0.042)</td>
<td>0.809 (0.045)</td>
<td>0.924 (0.034)</td>
<td>0.872 (0.044)</td>
<td>0.874 (0.054)</td>
<td>0.959 (0.020)</td>
<td></td>
</tr>
<tr>
<td>30%</td>
<td>0.500 (0.000)</td>
<td>0.883 (0.041)</td>
<td>0.804 (0.045)</td>
<td>0.908 (0.043)</td>
<td>0.885 (0.038)</td>
<td>0.876 (0.056)</td>
<td>0.960 (0.021)</td>
<td></td>
</tr>
<tr>
<td>35%</td>
<td>0.500 (0.000)</td>
<td>0.885 (0.041)</td>
<td>0.805 (0.045)</td>
<td>0.927 (0.036)</td>
<td>0.883 (0.044)</td>
<td>0.880 (0.053)</td>
<td>0.962 (0.022)</td>
<td></td>
</tr>
<tr>
<td>40%</td>
<td>0.500 (0.000)</td>
<td>0.880 (0.043)</td>
<td>0.815 (0.045)</td>
<td>0.938 (0.032)</td>
<td>0.885 (0.041)</td>
<td>0.901 (0.043)</td>
<td>0.964 (0.018)</td>
<td></td>
</tr>
<tr>
<td>45%</td>
<td>0.500 (0.000)</td>
<td>0.892 (0.041)</td>
<td>0.832 (0.044)</td>
<td>0.930 (0.034)</td>
<td>0.887 (0.043)</td>
<td>0.904 (0.047)</td>
<td>0.965 (0.021)</td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td>0.500 (0.000)</td>
<td>0.900 (0.036)</td>
<td>0.855 (0.043)</td>
<td>0.932 (0.031)</td>
<td>0.893 (0.043)</td>
<td>0.896 (0.048)</td>
<td>0.966 (0.019)</td>
<td></td>
</tr>
<tr>
<td>55%</td>
<td>0.500 (0.000)</td>
<td>0.906 (0.035)</td>
<td>0.897 (0.040)</td>
<td>0.928 (0.033)</td>
<td>0.893 (0.041)</td>
<td>0.896 (0.051)</td>
<td>0.969 (0.020)</td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td>0.500 (0.000)</td>
<td>0.910 (0.033)</td>
<td>0.923 (0.033)</td>
<td>0.935 (0.029)</td>
<td>0.894 (0.047)</td>
<td>0.900 (0.044)</td>
<td>0.972 (0.017)</td>
<td></td>
</tr>
<tr>
<td>65%</td>
<td>0.500 (0.000)</td>
<td>0.910 (0.031)</td>
<td>0.879 (0.056)</td>
<td>0.938 (0.028)</td>
<td>0.893 (0.042)</td>
<td>0.894 (0.046)</td>
<td>0.972 (0.017)</td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td>0.500 (0.000)</td>
<td>0.909 (0.031)</td>
<td>0.710 (0.056)</td>
<td>0.938 (0.028)</td>
<td>0.901 (0.038)</td>
<td>0.893 (0.047)</td>
<td>0.970 (0.019)</td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>0.500 (0.000)</td>
<td>0.909 (0.031)</td>
<td>0.667 (0.044)</td>
<td>0.938 (0.028)</td>
<td>0.898 (0.039)</td>
<td>0.893 (0.047)</td>
<td>0.968 (0.021)</td>
<td></td>
</tr>
<tr>
<td>80%</td>
<td>0.500 (0.000)</td>
<td>0.909 (0.031)</td>
<td>0.657 (0.043)</td>
<td>0.938 (0.028)</td>
<td>0.901 (0.039)</td>
<td>0.916 (0.041)</td>
<td>0.970 (0.021)</td>
<td></td>
</tr>
<tr>
<td>85%</td>
<td>0.500 (0.000)</td>
<td>0.909 (0.031)</td>
<td>0.656 (0.043)</td>
<td>0.938 (0.028)</td>
<td>0.898 (0.043)</td>
<td>0.915 (0.037)</td>
<td>0.971 (0.021)</td>
<td></td>
</tr>
<tr>
<td>90%</td>
<td>0.500 (0.000)</td>
<td>0.911 (0.031)</td>
<td>0.658 (0.042)</td>
<td>0.939 (0.027)</td>
<td>0.896 (0.039)</td>
<td>0.910 (0.036)</td>
<td>0.970 (0.019)</td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td>0.500 (0.000)</td>
<td>0.915 (0.031)</td>
<td>0.668 (0.044)</td>
<td>0.938 (0.028)</td>
<td>0.901 (0.039)</td>
<td>0.906 (0.038)</td>
<td>0.971 (0.022)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Learning algorithms AUC results for different levels of feature selection.
AUC 0.910 and highest AUC with 95% features was 0.915, IBk achieved 0.935 while the highest AUC was 0.938 for data set of 65% features. JRip and J48 both achieved AUC of 0.894 and 0.900 respectively while their highest AUC was 0.901 and 0.916 for data sets with 70% and 80% features. Due to these ignorable minor differences in results, we considered that data set with 60% features is a better option for our problem.

6.6 Analysis

We created 19 different data sets and each data set was having 5% less features than its successor. Experimental results indicated that a step of 5% was not enough to create significant difference in the result. NB has been an exception to this, which showed a random trend with increased or decreased percentage of features. However, if we look at the data sets created with the difference of 10% features then the difference in results is quite prominent. If the step is increased up to 20% difference of features, then a clear and understandable difference of results is present. If we review the overall performances on the various data sets, it is clear that the performance of most algorithms was quite high on the 60% feature selection level. It seems that number of kept features at this level is properly balanced with the number of instances from each class. In order to understand the classification process and to find interesting features, we analyzed the models generated by JRip, J48, and SMO. Models created by other algorithms cannot be visualized, so it was not possible to perform their analysis. We found three kinds of features i.e., features present only in scareware, features present only in legitimate, features which were treated differently by different algorithms.
Table 6.2. shows some selected features with a high impact on the classification decision. Features 1 and 7 are used to indicate scareware by all three models. However, features 2, 5, 8, and 9 were considered as a scareware indicative feature by two algorithms, but were ignored by the remaining algorithms. Features 2, 3, 6, and 10 seem to be considered as legitimate software indicative features by all algorithms. Finally, feature 4 is regarded as a legitimate indicator by JRip, but as a scareware indicator by SMO.

In order to demonstrate the information provided by a single feature, we traced the features from Table 6.2. to the disassembled binary files. One such example is provided in Figure 6.4. As per our understanding, the function in Figure 6.4. seems to indicate an attempt to transfer some specific string data to the user for display or transfer from user to some other end. The particular contents of the memory are not available to us since we are performing a static analysis and thus to get a deeper understanding, we would have to manually analyze a larger portion of disassembled code. However, it is clear that some functionality is present only in scareware instances, which would suggest that it is possible to differentiate them from benign files on a general level. However, it would be hard for a human expert to detect and analyze such subtle differences; therefore, we argue that our automatic approach is superior, especially when considering the fact that regular applications can contain several thousands of lines of code.

6.7 Conclusions and Future work

We have extended the heuristic-based detection technique using a variable length instruction sequence mining approach for the purpose of scareware detection. Since scareware is a rather recent software security threat, there are no publicly available data sets to generate classification models from. We have, therefore, obtained a large sample of scareware applications and designed an algorithm for extracting instruction sequences from these applications (and similarly for legitimate
6.7. Conclusions and Future work

Table 6.2: Selected features and their number of occurrence in each class

<table>
<thead>
<tr>
<th>F.No</th>
<th>Feature</th>
<th>Jrip</th>
<th>J48</th>
<th>SMO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>pushpushandoutsw</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>-0.1420</td>
</tr>
<tr>
<td>2</td>
<td>ormovadd</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>-0.0813</td>
</tr>
<tr>
<td>3</td>
<td>inswpopaw</td>
<td>&gt;=1</td>
<td>0</td>
<td>0.1940</td>
</tr>
<tr>
<td>4</td>
<td>incoutswoutsb</td>
<td>&gt;=1</td>
<td>0</td>
<td>-0.0195</td>
</tr>
<tr>
<td>5</td>
<td>addmovmovmovcmp</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>-0.0615</td>
</tr>
<tr>
<td>6</td>
<td>leadb</td>
<td>&gt;1</td>
<td>&gt;0</td>
<td>0.2016</td>
</tr>
<tr>
<td>7</td>
<td>dbdecmov</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>-0.0848</td>
</tr>
<tr>
<td>8</td>
<td>outswarplfs</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>-0.0223</td>
</tr>
<tr>
<td>9</td>
<td>movpushmov</td>
<td>&gt;0</td>
<td>&gt;0</td>
<td>-0.0572</td>
</tr>
<tr>
<td>10</td>
<td>pushaddpush</td>
<td>&gt;1</td>
<td>&gt;0</td>
<td>0.1704</td>
</tr>
</tbody>
</table>

S - Scareware, L - Legitimate software.

software). The data sets used in this study will be publicly available at http://www.bth.se/com/rks. The experimental results are promising: the Random Forest algorithm managed to yield an AUC score of 0.972 after the complete data set was processed using the categorical proportional difference feature selection algorithm. Moreover, the results also indicate that our method is trustworthy since the false negative rate (the rate of scareware classified as legitimate) is considerably low (0.023). For future work, we aim to conduct further experiments on an even larger collection of scareware and benign files. We also plan to employ a hybrid identification method, which would integrate variable length instruction sequences with features extracted from, e.g., the end user license agreement or the information about the system calls a particular program makes.
Chapter 7

Comparative Analysis of Voting Schemes for Ensemble-based Malware Detection

Raja Khurram Shahzad, Niklas Lavesson

Abstract

Malicious software (malware) represents a threat to the security and the privacy of computer users. Traditional signature-based and heuristic-based methods are inadequate for detecting some forms of malware. This paper presents a malware detection method based on supervised learning. The main contributions of the paper are two ensemble learning algorithms, two preprocessing techniques, and an empirical evaluation of the proposed algorithms. Sequences of operational codes are extracted as features from malware and benign files. These sequences are used to create three different data sets with different configurations. A set of learning algorithms is evaluated on the data sets. The predictions from the learning algorithms are combined by an
ensemble algorithm. The predicted outcome of the ensemble algorithm is decided on the basis of voting. The experimental results show that the veto approach can accurately detect both novel and known malware instances with higher recall in comparison to majority voting, however, the precision of the veto voting is lower than the majority voting. Veto voting is further extended as trust-based veto voting. A comparison of the majority voting, the veto voting, and the trust-based veto voting is performed. The experimental results indicate the suitability of each voting scheme for detecting a particular class of software. The experimental results for the composite F1-measure indicate that the majority voting is slightly better than the trusted veto voting while the trusted veto is significantly better than the veto classifier.

7.1 Introduction

Malicious software (malware) is a common computer threat and is usually addressed through the static and the dynamic detection techniques. Anti-malware tools are only able to detect known malware instances and the success rate is circa 30% [13] in the wild. In an effort, to extend both the static and dynamic approaches, some researchers apply machine learning (ML) algorithms to generate classifiers, which show promising results both in detecting the known and novel malware. To increase the detection accuracy, the output (prediction) of different classifiers is combined (based on different parameters) to form an ensemble [122]. Ensembles can be data dependent such as multiple algorithm trained on the same data set, or independent from the data, i.e., using statistical measures [123]. The prediction of each participating classifier in the ensemble may be considered as a vote for a particular class, i.e., benign class or malware class. The ensemble’s outcome is generally derived on the basis of different voting strategies. Different voting strategies may give different results depending upon different factors such as families of algorithms used. Among different voting strategies, the majority voting is generally used for different problems. The majority voting is considered as a simplest and effective scheme [124]. The majority voting scheme follows democratic rules, i.e., the class with highest
7.1. Introduction

number of votes is the outcome. Majority vote does not assume prior knowledge about the problem in hand or classifiers used and may not require training [124]. The majority voting scheme has different limitations such as a subset of classifiers (majority in number) may agree on the misclassification of an instance by a chance. An alternative voting scheme is the veto voting, i.e., one single classifier vetoes the decision of other classifiers. The veto voting scheme is used in the fault diagnosis, the author identification and the malware detection [125, 126]. For the malware detection a ML-based detection model is proposed in which the inductive biases of different algorithms are combined, and the final prediction is given on the basis of veto voting, i.e., if an algorithm predicts the instance as a malware, this prediction may veto all the other predictions and the outcome is malware [126]. The veto voting may also affects the performance of the ensemble as outcome may depend on one single algorithm. However, the veto voting can achieve higher classification accuracy on the assumption that the data set contains abundant instances of a particular class (favored by veto) [127]. Thus, it is worth to investigate which voting scheme, i.e., majority voting or veto voting is suitable for the malware detection. After the comparison of the majority voting and the veto voting, the veto voting is extended from a simple veto voting to the trust-based veto voting. The trust-based veto voting considers the trust of each algorithm to determine whether an algorithm or set of algorithms can veto the decision. The trust-based veto voting is also applied on the same data set and results are compared with the majority voting and the veto voting. The majority voting is suitable for detecting benign applications and lacks the accuracy in the detection of malware in comparison to the veto voting. Similarly, the veto voting is suitable for detecting the malware and is less accurate for the detection of benign. The experimental results indicate that the proposed trust-based veto voting algorithm may be used to overcome the deficiencies of both the majority voting and the veto voting up to some extent. The trust-based veto voting accurately detect both novel and known instances of malware better than the majority voting and accurately predicts about benign instances better than the veto voting. The experimental results also indicate that if composite measure is taken into account, majority
voting is slightly better than the trust-based veto voting.

### 7.1.1 Aim and Scope

The aim is to evaluate the malware detection methods that combine the output of a set of classifiers and provides a classification on the basis of the majority voting or the veto voting. A malware detection application is developed, which implements the majority voting, the veto voting and the trust-based veto voting. The prediction results from all voting strategies are compared with each other. To achieve better predictive results, the quality of information (i.e., the information, which provides valuable input for classification) derived from the data in the pre-processing stage is very important. Therefore, two pre-processing techniques are also proposed and investigated.

### 7.1.2 Contribution

The contributions are: first, a malware detection model is proposed and implemented, which combines the inductive biases of different algorithms and uses contrary voting strategies for the prediction. Second, an extension to a particular voting strategy, i.e., the veto voting is proposed and implemented. Third, the empirical results of different voting strategies are analyzed. Fourth, two pre-processing techniques are proposed to extract features from executable files. These pre-processing techniques can be used to extract both hexadecimal based features or assembly instruction based features of different sizes.

### 7.1.3 Outline

The remainder of the article is organized as follows: Section 7.2 discusses the background, terminology and related work. Section 7.3 presents the veto-based classification by discussing it’s architecture. Section 7.4 discusses the pre-processing techniques. Section 7.5 describes the experimental procedure for the first experiment (i.e., Experiment I) and compares the results of the majority voting and the veto voting. Section
7.2. Background

One of the challenges faced by computer users is to keep the information and communication away from unwanted parties who exploit vulnerabilities present in the operating system (OS) or third party software to endanger the communication and access the information. A popular way to exploit vulnerabilities remotely is by using a malware [128]. Traditionally, the malware detection is conducted either by using the static analysis, i.e., by matching specific patterns called signatures or on the basis of a rule set, or the dynamic analysis, i.e., changes occurring in the system due to the execution of a specific file. The main deficiency of these techniques is that they fail to detect the zero-day attacks.

The use of ML has been investigated in fields such as natural language processing, medical diagnosis, and malware detection. ML can be divided into two broad categories. In supervised learning, algorithms are used to generate a classifier or a regression function using the labeled data. To achieve the better predictive performance, a finite set of classifiers can be combined as an ensemble. The prediction of most ensembles is based on the majority voting [122]. If the data is incompletely labeled, unsupervised learning is used. To achieve better predictive performance in unsupervised learning, deep learning can be used. In deep learning, algorithms learn from different levels of representations to find the complex patterns in the data.

Any algorithm used in supervised or unsupervised learning has its own inductive bias. Inductive bias of learning algorithms refers to the set of assumptions that a learning algorithm uses for predicting the output of unseen inputs [33]. In other words, it is a set of assumptions that is used by a learning algorithm to prefer one hypothesis over the
other hypothesis in the search space, in order to find a suitable hypothesis which can provide better predictions for the problem in question. These factors affect the classifier performance [34]. In the case of malware classification, an algorithm may be more accurate in classifying viruses than adware. Due to these reasons, detection results may vary from one learning algorithm to another learning algorithm. Therefore, it may be appropriate to join the results of classifiers trained at different representations to achieve the improved accuracy in predicting the class of unseen instances.

7.2.1 Terminology

**Data set**

A piece of information from a particular object such as a binary file or an image in a specific format is called a feature. The format of the feature is called feature representation. Each instance present in the original data is represented by its specific features for the ML processable data sets. For the malware detection task, different features can be used to represent the binary files such as files may be converted into the hexadecimal code [48, 109] or assembly instructions [129] to produce the data sets. Features may also be extracted from the binary files such as printable strings or system calls [48] to generate the data set.

**Feature Selection**

To improve the accuracy of ML algorithms, complexity of the data sets is reduced. For this purpose, the most common approach is to apply a feature selection algorithm, which measures the quality of each feature and prioritize features accordingly [34]. Only features that can provide the valuable information for the classification are kept, and the rest are discarded. In the malware detection, an extensive feature set is produced from the binary files data set. This feature set contains many invaluable features, which can degrade the performance of a ML algorithm. Therefore, it is necessary to obtain a subset of valuable features by applying the feature selection.
Classification

In ML, the classification is divided into two stages, i.e., training and testing. Learning algorithms are applied on the training set to build a model (commonly called classifier) [33]. This stage is called training. During the testing stage, the generated classifier is used to predict the class of unseen instances.

Ensemble

Ensemble are capable of combining multiple models for the improved accuracy [34]. The different models for the ensemble may be generated from the same base algorithm on different subsets of the data or different algorithms on the same data set. Ensembles perform better than a single model due to the diversity of base models [130].

Trust

Trust is primarily related to the human behavior of believing a person to meet expectations. Trust has different meanings in different contexts. Trust can be defined as “Trust is quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context” [131]. For the problem in hand, the terms trustor and the trustee refers to an algorithm that quantify the trust. Trust can be quantified as +1 or -1; the increased or decreased value can assist in determining the extent of the trust. The trust can be quantified as the single trustor or the group trust. The trust especially the group trust is quantified for different computational problems such as for the authentication, in the peer-to-peer networks, in the mobile ad-hoc networks, for resisting the different network attacks and for spam emails.

7.2.2 Related Work

A significant amount of research for classification tasks has applied techniques ranging from statistical methods to machine learning like supervised learning and deep learning (DL). The use of DL has been inves-
7. Voting Schemes

investigated to learn different levels of the representation in order to model complex relationships in the data to classify patterns and objects [132]. DL has also been used to extract features and represent them at different layers with different representations or abstraction to be used for vision, face recognition, and handwritten digit recognition. Similarly, the layered architecture has also been used for detecting the malicious behavior [13]. In some cases, the decision from an individual model or even from several models may be insufficient to obtain the final prediction, especially when the cost of misclassifying one class is higher than misclassifying the other class. As a solution, a few researchers have used veto voting for automated identification of a disease pulmonary embolism [133] and authorship identification [134].

For the malware detection, several researchers have used ensemble based on the majority voting. The majority voting is compared with different ensemble methods on five different malware data sets [135]. The majority voting is also used to generate the rules, according to the Dempster-Shafer theory, to classify the malicious codes based on the n-gram features of the binary files [136]. Some researchers have applied the variation of majority voting such as the weighted majority voting for the malware detection [93]. The concept of the veto voting is not investigated for the malware detection. Therefore, it is worth to investigate the veto voting for the malware detection.

To support the veto voting, the concept of trust may be used. In an early work on the authentication in open networks, the trust is used to accept or reject an entity for a task [137]. A set of inference rules is used to determine the value of trust, i.e., $0 \leq \text{trust} \leq 1$ and derived value is further used for the decision. Both direct and group trusts are used. In a study, trust value is used for resisting the non-trivial attacks on the authentication of the origin of the message in the distributed system [138]. A quantitative framework based on the attack values for resisting the attack is proposed [139, 140]. The proposed framework uses the group trust metric and calculates a trust value for all the nodes simultaneously. The values are further used to build a distributed name server, verify the
meta-data in peer-to-peer networks, and resistance to the Spam e-mails. The authors also present a real world example, i.e., Advogato website\(^1\).

A common trust based algorithm for the peer-to-peer network is the EigenTrust algorithm [141]. The EigenTrust algorithm uses peer nodes to assign the trust to each node. The assigned trust is used to computes global trust values in a distributed manner and node-symmetric manner. Global trust value is also used to distinguish malicious nodes in the system. The priority is given to the opinion of high reputation nodes. The EigenTrust algorithm is used to proposed a non-manipulable trust system for peer-to-peer networks [142]. The authors propose a partitioning technique that is used to partition the peers in groups and incentives for the peers to share files. Mobile ad hoc networks are decentralized networks and nodes in such network cooperate with each other [143]. The trust value of each node is used to improve the security of network.

The authors use the local trust and recommendation trust, which are combined to obtain the combination trust. Finally, the combination trust value is used to evaluate the level of risk for ongoing tasks. In the field of multi-agent systems and ML, trust is used to make a reputation system for auction systems [144]. A generic ML based trust framework is proposed to calculate the trust of a transaction by an agent. The trust is calculated on the basis of previous successful transactions based on distinguishing features of successful and unsuccessful transactions. The distinguishing features are given as input into ML algorithms to extract the relationship. The extracted relationships are further used to predict about the success of the current transactions.

### 7.3 Veto-based Classification

In certain situations, the recommendation from more than one expert may be required. In such cases, a committee of experts is formed as it is expected that a committee always performs better than a single expert. Normally the committee uses the majority voting for combining the decisions of experts to reach a final conclusion. In some cases, the committee

\(^1\)http://www.advogato.org/
may grant the right to veto the decision of the committee to any member. In ML, multiple algorithms can be used to generate multiple classifiers (experts) for a classification task. Every classifier has its own inductive bias, which affects the predictive performance. Research results indicate that ensemble perform better than single classifier in fields of text categorization [145] and data classification, etc. Several rules such as majority voting, i.e., bagging [34], weighted combination where weights represent effectiveness of member classifiers such as boosting [34], dynamic classifier selection [146, 147] and the veto voting [133, 134] can be used for combining the decisions and having a final prediction. Veto voting is used to give importance to a single expert (classifier) who predicts against the majority.

In malware detection, ignoring the correct prediction about a malicious file from a single expert may incur a higher cost in terms of security violations. The security violations may cause serious data loss, privacy or monetary damages to the user. Therefore, a veto voting based classification model is more appropriate than a majority voting based model. A model is proposed, which combines the inductive biases of individual classifiers and the final decision is given on the basis of veto voting. For brevity, the veto voting based classification system is referred to as veto classifier in later sections.

7.3.1 Voting Rules

The main objective of veto based classification is to combine the multiple predictions to reach a final decision. Formally, a veto based classification system consists of candidate classifiers (C) set and vote set V. The set of candidate classifiers C, and the set of votes (V) is a finite set (C, V) with predetermined fixed number of maximum classifiers and votes. The vote from each classifier is considered on the basis of rules given below. Some rules mentioned below are also recommended for the general voting [148].

**Anonymity** All votes in the vote set (V, a finite set with a predetermined
number of votes) are treated equally, and the outcome of the classifier committee remains consistent with any permutation of the votes.

**Neutrality** All candidates in the classifier set (C, a finite set with a predetermined number of classifiers) are considered equally without any additional weighting.

**Independence** Candidate classifiers are independent of each other, and the outcome of the voting system remains consistent with any combination of classifiers with votes, i.e., \((C_1, V) \cup (C_2, V) \subseteq (C, V)\) where \(C_1\) and \(C_2\) are different combinations of classifiers.

**Completeness** All votes from the classifiers are considered and counted only once.

**Veto** Any vote indicating an instance as malware, alone can determine the outcome of the classification task regardless of the number of other votes.

**Consistency** The result of the voting remains consistent even if the base classifiers are split into two disjoint sets and each set vote separately. The votes from each subset create a single vote set. Formally this can be mentioned as \((C, V_1) \cap (C, V_2) \subseteq (C, V)\) where \(V_1\) and \(V_2\) are the partitions of votes.

**Usability** Voting procedure can be used for other similar problems.

**Verifiability** Any user of the voting system can verify the outcome of voting by counting the votes manually.

Ideally veto based voting performs better than the single classifier because if any classifier in the committee predicts the class of an instance as malware, it may veto all the other predictions from the other classifiers. For the neutrality, the results from all the classifiers are combined without any additional weighting or filtering. It is also possible to use weighting method [149] for the votes such as by assigning more weight...
to the vote of a classifier who outperformed all other classifiers in terms of accuracy during the training stage.

7.3.2 Architecture

The model can be implemented in two possible ways, i.e., N-layers implementation and parallel implementation. N-layers implementation is based on the serial implementation.

N-Layers implementation The model can be implemented in \( n \)-layers with any permutation of classifiers, see Figure 7.1. Each layer can be customized with different \( n \)-gram sizes, several feature representations, various feature selection algorithms and learning algorithms. It is recommended that different classifiers shall be used while maintaining their neutrality as much as possible to increase the effectiveness [130]. From the lower layer, the instances that are declared as benign are given to the upper layer for the reclassification. In each layer, all classifiers give their predictions about the instances from the lower layer (or data sources). If at any layer, an instance is classified as malware, it is not fed into the next layer. The classification results from all the layers are given to the veto classifier. The malware prediction at any layer for any instance may be considered as veto for that particular instance. However, the final decision about the class of particular instance is taken by the veto classifier.

Parallel Implementation Instead of using layers, all the possible permutations of classifiers can be implemented in a distributed or parallel manner. Each learning algorithm is trained over the entire data set for generating the classifiers. Instead of testing only positive instances by some classifiers, each classifier works independent of results from other classifiers. All the votes from the classifiers are collected at a central location where the veto classifier outputs the final prediction.
7.4 Pre-processing Techniques

The selection of representative features affects the predictive performance of a classifier. Consequently, two $n$-gram extraction techniques are proposed. Most of the research studies for the malware detection demonstrate the use of either the hexadecimal-based $n$-gram data set or the OpCode-based $n$-gram data set for an experiment. The proposed extraction techniques can be used to extract both representations. For this study the OpCode $n$-gram extraction is performed, therefore, the proposed techniques are explained in the context of OpCode $n$-grams only.

Traditionally the $n$-gram extraction is performed by using a fixed size
window with a fixed step; the step size is equal to the window size. The fixed size window traverse the data file to extract the specific size n-grams. The generated output contains adjacent n-grams. To explain this process, assume that a disassembled binary file contains the following given data. A pair of characters represents an OpCode (or a hexadecimal code). The task is to create bi-grams, i.e., n-gram of size 2 from this data file.

\[ \text{aa bb cc dd ee ff gg hh ii jj kk ll mm nn oo pp} \]

The generated bi-grams from this file are "aabb ccdd eeff gghh iijj kkll mmmn ooopp" and so on. The fixed size window is unable to extract some n-grams such as "bbcc" or "ddee". If the file size is large and the data is redundant then there is a probability to have missing combinations, but still missing n-grams cannot be produced in the appropriate frequency and can have less importance for the classification task.

### 7.4.1 Overlapping n-grams

To address the problems of missing n-grams, the use of configurable sliding window to generate overlapping n-grams is proposed. The window can be configured with two parameters, i.e., size and step. The size parameter defines the size of a n-gram to be extracted, and the step parameter defines the number of OpCodes to be skipped before extracting the next n-gram. It is expected that all possible combinations can be extracted by this extraction technique. Referring the example in Section 7.4, if the window is configured as following, size = 2, i.e., two adjacent OpCodes are extracted to form a n-gram and step = 1, i.e., after the n-gram extraction window skips one OpCode (first OpCode) to move forward. This configuration generates "aabb bbcc ccdd ddee eeff ffgg gghh hiii iiij jjkk" and so on.
7.4.2 Non-adjacent OpCode Extraction

Either by using the traditional n-gram extraction or the overlapping n-gram extraction, extracted n-grams can provide the information only about the dependencies of the adjacent OpCodes. It is valuable to look at the information provided by non-adjacent OpCodes. Non-adjacent OpCodes have dependencies such as they can be function header and tail. Some changes are proposed in the overlapping n-gram extraction method to explore the information both from non-adjacent OpCode and non-adjacent adjacent OpCode. The size parameter is changed to the start-end size parameter. The start-end size parameter defines the number of adjacent OpCodes to be extracted for the start and the end of a n-gram. The step size parameter defines the number of OpCodes to be skipped for extracting a new n-gram. A new parameter is introduced, i.e., gap size, which specifies the interval between start and end OpCode or number of OpCodes to be skipped between the start and the end OpCode of a n-gram. The example mentioned in the Section 7.4 can be used to describe this procedure. If the window is configured as following for extracting non-adjacent bi-grams, start-end size = 1, i.e., one OpCode for the start and one OpCode for the end of a n-gram are extracted, step = 1 and gap = 1, i.e., one OpCode between the start OpCode and the end OpCode of a n-gram is skipped. This configuration produces the bi-grams, which contains non-adjacent OpCodes. The generated output is "aacc bbdd ccee ddff eegg" and so on. To have non-adjacent adjacent OpCodes in a n-gram, the configuration can be changed as follow: start-end = 2, i.e., two adjacent OpCodes for the start and the end of a n-gram are extracted; the step size and the gap size are kept 1. The generated output is "aabbdee bbcceeff cccdgggghi" and so on. If the value of the gap size and the step size parameters is changed from 1 to 2, the generated output is "aabbeeff ccdggghh eeefiiijj" and so on.

7.5 Experiment I

The aim of the experiment is to evaluate the proposed veto voting based malware detection method and impact of the proposed data pre-processing
techniques and compare the results with the majority voting. The proposed method can be used to detect either a specific type of malware or different types of malware; however in this study a single family of malware is used. The experimental data set contains Windows-based executable files. Windows is a common OS for novice users and contains different vulnerabilities\(^2\), which can be exploited by a malware. When a binary file in the data set is disassembled, different file features such as assembly language instructions and printable strings, are produced in the text format, which are further processed to extract the assembly directives, i.e., OpCode. OpCodes are further processed to produce the bi-gram data sets using different strategies. Different text categorization techniques can be applied to the output generated in the previous step to get discriminating features of benign and malware. Term Frequency-


\(7. \textbf{Voting Schemes}

Table 7.1: Experiment with one data set and three algorithms

<table>
<thead>
<tr>
<th>Data Set(^a)</th>
<th>Algorithm</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>(R)</th>
<th>(P)</th>
<th>(F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)-gram</td>
<td>JRip</td>
<td>243</td>
<td>184</td>
<td>66</td>
<td>07</td>
<td>0.972</td>
<td>0.786</td>
<td>0.869</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>226</td>
<td>225</td>
<td>25</td>
<td>04</td>
<td>0.904</td>
<td>0.900</td>
<td>0.902</td>
</tr>
<tr>
<td></td>
<td>IB(^k)</td>
<td>222</td>
<td>225</td>
<td>25</td>
<td>24</td>
<td>0.896</td>
<td>0.899</td>
<td>0.897</td>
</tr>
<tr>
<td></td>
<td>Veto(^c)</td>
<td>243</td>
<td>203</td>
<td>47</td>
<td>07</td>
<td>0.972</td>
<td>0.837</td>
<td>0.900</td>
</tr>
<tr>
<td></td>
<td>Majority(^c)</td>
<td>223</td>
<td>233</td>
<td>17</td>
<td>26</td>
<td>0.895</td>
<td>0.929</td>
<td>0.912</td>
</tr>
<tr>
<td>Overlap</td>
<td>JRip</td>
<td>238</td>
<td>197</td>
<td>53</td>
<td>12</td>
<td>0.952</td>
<td>0.817</td>
<td>0.879</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>232</td>
<td>234</td>
<td>16</td>
<td>18</td>
<td>0.928</td>
<td>0.935</td>
<td>0.931</td>
</tr>
<tr>
<td></td>
<td>IB(^k)</td>
<td>224</td>
<td>224</td>
<td>26</td>
<td>26</td>
<td>0.896</td>
<td>0.896</td>
<td>0.896</td>
</tr>
<tr>
<td></td>
<td>Veto</td>
<td>246</td>
<td>208</td>
<td>42</td>
<td>04</td>
<td>0.984</td>
<td>0.854</td>
<td>0.914</td>
</tr>
<tr>
<td></td>
<td>Majority</td>
<td>230</td>
<td>240</td>
<td>10</td>
<td>20</td>
<td>0.920</td>
<td>0.958</td>
<td>0.938</td>
</tr>
<tr>
<td>S. Window</td>
<td>JRip</td>
<td>209</td>
<td>215</td>
<td>35</td>
<td>41</td>
<td>0.836</td>
<td>0.856</td>
<td>0.846</td>
</tr>
<tr>
<td></td>
<td>J48</td>
<td>215</td>
<td>205</td>
<td>45</td>
<td>35</td>
<td>0.860</td>
<td>0.826</td>
<td>0.843</td>
</tr>
<tr>
<td></td>
<td>IB(^k)</td>
<td>164</td>
<td>237</td>
<td>13</td>
<td>86</td>
<td>0.656</td>
<td>0.926</td>
<td>0.768</td>
</tr>
<tr>
<td></td>
<td>Veto</td>
<td>242</td>
<td>139</td>
<td>111</td>
<td>08</td>
<td>0.968</td>
<td>0.685</td>
<td>0.802</td>
</tr>
<tr>
<td></td>
<td>Majority</td>
<td>220</td>
<td>204</td>
<td>46</td>
<td>30</td>
<td>0.880</td>
<td>0.827</td>
<td>0.852</td>
</tr>
</tbody>
</table>

\(a\)The full names of data sets are \(n\)-gram data set, overlap data set and sliding window data set.

\(b\)R is Recall, P is Precision, and F1 is F-Measure.

\(c\)Veto is Veto Classifier and Majority is Majority voting.

\(d\)Veto Classifier and Majority voting both are applied on all the three data sets.
Inverse Document Frequency (tf-idf) is used to derive the significant features from the data sets. The extracted features are used to create Attribute-Relation File Format (ARFF) files. ARFF file is a structured ASCII text file that includes a set of data instances, each described by a set of features [34]. ARFF files are used as input to the proposed model, which uses Waikato Environment for Knowledge Analysis (Weka) application programming interface (API) [98] for applying learning algorithms to build and analyze classifiers. A pre-experiment is performed for the selection of learning algorithms. The first experiment is divided into two sub-experiments. In the first experiment, the inductive biases of the different classifiers built on the same data set are combined. Second experiment combines the inductive biases of individual classifiers built on different data sets. In both experiments, the results from all the classifiers are given to the veto classifier for the final prediction.

7.5.1 Feature Representation

OpCode is used for generating bi-grams as features. It is concluded in the previous studies that OpCode n-grams are better choice for the malware detection in comparison to other features such as printable strings, systems calls or byte code (hexadecimal) n-grams [129]. The OpCode n-grams are capable of providing the information about the program flow, structure and function that cannot be deduced from other representations.

7.5.2 Data Set Creation

For the experiment, scareware (rouge) software is selected as malware representation. The reason for this choice is, there is a subtle difference between scareware and benign. In case of traditional malware, presence of malicious payload distinguishes a malware from the benign. However, in scareware no specific malicious payload is available that can be used to differentiate a scareware from the benign. Absence of malicious payload may deceive human expert for the classification of a particular

\[\text{http://www.cs.waikato.ac.nz/ml/weka/arff.html}\]
7. Voting Schemes

software as scareware.

Scareware are scam software that usually masquerade as an anti-virus software and resembles the benign software in functionality. Scareware generates the false alarm about the presence of malware in the user’s machine. The false alarms are used to scare the users into disclosing their credit card information for buying the protection\textsuperscript{4}. No public data set e.g., virus, Trojan, and worm data sets is available for the scareware detection experiments. Therefore, a data set with 500 files is created; out of which 250 files are scareware, and 250 files are representing benign. The benign files are default applications of Windows OS such as notepad, paint and applications available online for download at CNET Download\textsuperscript{5}. All the benign files are scanned with commercial security software (anti-malware) to reduce the chances of malware presence in a benign file. Scareware files are obtained from the malware database of Lavasoft\textsuperscript{6}.

7.5.3 Pre-Processing

The disassembled file is a standard text file, which contains three fields, i.e., the memory address of the instruction, the byte-based location in the file and the instruction itself (combination of OpCode and operands). The next step is to extract only OpCodes from the files and discard irrelevant information, i.e., operands. The extracted OpCodes are saved in the original order. After OpCodes extraction from the disassembled files, three different procedures are used to tokenize the data to produce bi-grams for three different data sets. Each row in a data set represents a bi-gram, i.e., concatenation of two OpCodes. Hereafter, these three data sets are referred to as bi-gram data set, overlap data set and sliding window data set respectively to indicate the method used in creating that particular data set. The bi-gram size has yielded the best performance in a previous study [88] and possible combinations of OpCodes to produce

\begin{itemize}
\item \textsuperscript{4}http://news.bbc.co.uk/2/hi/8313678.stm
\item \textsuperscript{5}http://download.com
\item \textsuperscript{6}http://lavasoft.com
\end{itemize}
bi-grams are limited, depending upon the number of reserve words in the assembly language.

For the bi-gram data set, a fixed size window traverse each input file from top to bottom. In every step, a n-gram consisting of two OpCodees is extracted and recorded in another file having the similar file name, but different extension. The purpose of keeping the similar name is to keep track of benign files and scareware files, so each file can be represented at the same position in all three data sets and finally in the ARFF file. For overlap data set method mentioned in the Section 7.4.1 is followed with the configuration, i.e., size = 1 and step = 1. For the sliding window data set, start-end size and step parameters are kept one. To obtain the nonadjacent OpCode bi-grams, each file is processed in four consecutive passes with a gap size ranging from 1-4. Due to the changing gap size, the first generated bi-grams are having a gap of one OpCode between the start OpCode and the end OpCode, in the second pass there is a gap of two OpCodees and so on. The process of generating the sliding window data set is slower than generating the bi-gram data set and the overlap data set. However, the computational cost and memory requirements for generating the sliding window data set are lower than creating large size n-grams.

7.5.4 Feature Selection

Many real world problems are complex. To apply learning algorithms, the dimensionality of the complex problem is reduced by choosing a subset of significant features from the given set of (raw) features. The selected subset of features plays significant role in the increase/decrease of either classification and computational performance. Significant feature selection is done by using a feature selection algorithm, removing features that are deemed unlikely to improve the classification process. In the field of text classification, tf-idf shows promising results for the valuable features selection. In this experiment tf-idf is applied on data sets to limit the number of features to top 1000 features per data set. The tf-idf is a statistical measure of importance of a bi-gram in the entire data
7. Voting Schemes

set [150]. The $tf$ is the number of times a bi-gram occurs in a file; $df$ is the number of files in a class that contain a specific bi-gram. The $idf$ of a bi-gram is obtained by dividing the total number of files ($N$) by the $df$ and then taking the logarithm.

7.5.5 Performance Evaluation Criteria

Each learning algorithm is evaluated by performing cross-validation tests. Confusion matrices are generated by using the responses from the classifiers. The following four estimates define the elements of a confusion matrix: True Positive (TP) represents the correctly identified scareware programs. False Positive (FP) represents the incorrectly classified benign programs. True Negative (TN) represents the correctly identified benign programs, and False Negative (FN) represents the incorrectly identified scareware programs. The performance of each classifier is evaluated using Recall (R), which is the ratio of scareware programs correctly predicted from the total number of scareware programs, Precision (P), ratio of scareware programs correctly identified from the total number of programs identified as scareware. F-Measure (F1) is the harmonic mean of the precision and the recall and is the final evaluation measure.

7.5.6 Pre-Experiment for Algorithm Selection

A number of studies have addressed the similar problem with different learning algorithms; however, none of the authors is conclusive on the choice of algorithms either for the malware detection or according to the produced data set. In a number of studies Ripper (JRip) [86], C4.5 Decision Tree (J48) [111], k-nearest neighbor (IBk) [114] Naive Bayes [34] and SMO [112] outperformed other algorithms. Based on previous research, a pre-experiment is performed to evaluate all these algorithms on all the three data sets. The top three algorithms, i.e., JRip, J48 and IBk are considered as candidates, to combine their inductive biases for the final prediction in the proposed model.
7.5.7 Results and Discussion

In the first experiment, one data set is used to build classifiers from three different algorithms. In the second experiment, three data representations are used and one classifier is trained from each representation. Majority voting is compared to the veto voting. In the first experiment, the predictions from three classifiers are collected and given to the veto classifier and the majority voting. The predictions from all the classifiers including both voting strategies on each data set are shown in Table 7.1. In the second experiment, three algorithms, i.e., JRip for \( n \)-gram data set, JRip for the overlap data set, and J48 for the sliding window data set are selected on the basis of the recall in the first experiment. These algorithms are used to build three classifiers and the predictions about each instance from these classifiers is given to the veto classifier and the majority voting for the final prediction. The results of this experiment (see Table 7.2) indicate that the recall of the veto classifier is better than the recall values in the first experiment. Majority voting shows the similar behavior for the precision.

The experimental results indicate that combining the inductive biases of different algorithms trained on multiple representations predicts better for the malware detection than combining the inductive biases of different algorithms trained on the same data set. The experimental results of both voting strategies can be discussed in three dimensions by using three measures, i.e., recall, precision, and f-measure. The experimental results show that the veto classifier has better recall than the majority voting, i.e., veto classifier reduces the number of misclassified scareware. Recall is the key measure as the objective of the veto approach is to reduce the likelihood of malware misclassification while tolerating a percentage of false positives or decrease in the precision. If the system is tuned to predict all applications as malware, it will produce a high false positive rate, which is undesirable from a user’s point of view. Users need the accurate prediction both for the malware and benign applications. Therefore, the precision is also considered as a complimentary measure with the recall. The veto classifier shows a higher tendency
for the correct detection of scareware while the majority voting shows a tendency towards the detection of benign applications. Therefore, the precision rate is higher for the majority voting. There are few instances, which are misclassified by both voting schemes. Most of these instances are benign, but predicted as scareware by both the veto classifier and the majority voting. However, the number of such instances is minimal. The precision and the recall have an inverse relationship if the precision increases, the recall decreases. Therefore, another evaluation measure is required, which combines the precision and the recall. Thus, the final evaluation measure is F-measure, which evenly weights the precision and the recall. It may be argued that the arithmetic mean of the precision and the recall can also be used as a composite measure. However, the arithmetic mean is an inappropriate measure as with 100 % R and 0 % P or vice versa; the arithmetic mean is always 50 %. This is not the case with the harmonic mean as the harmonic mean is always less than or equal to the arithmetic mean [151]. If there is a difference between the value of R and the P such that the value of R is significantly smaller than P, the harmonic mean tends strongly towards the recall. F1 of the majority voting is higher than the veto classifier, which favors the use of majority voting for the problem in question. However, the recall for the majority voting, which is a key measure, is lower than the veto classifier so it may be argued that the veto is a better approach for the malware detection. Thus, the veto classifier shall be extended to increase the precision.

Bi-grams are used as the feature in the experiment because they are computationally inexpensive to produce. Generally such short combinations may not represent an important function or set of instructions in the files and are difficult to analyze. However, bi-grams in the sliding window data set can provide the valuable information for the scareware analysis due to the combination of non-adjacent OpCodes. Scareware resembles the benign applications such as displaying popup windows or alert messages, and showing the dialog boxes. Therefore, it is difficult for the human experts to predict about the scareware by analyzing the functionality of an application only. The proposed model helps the hu-
Table 7.2: Experiment with three data sets and one algorithm on each data set

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Algorithm</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>R (^b)</th>
<th>P (^b)</th>
<th>F1 (^b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-gram</td>
<td>JRip</td>
<td>243</td>
<td>184</td>
<td>66</td>
<td>07</td>
<td>0.972</td>
<td>0.786</td>
<td>0.869</td>
</tr>
<tr>
<td>Overlap</td>
<td>JRip</td>
<td>238</td>
<td>197</td>
<td>53</td>
<td>12</td>
<td>0.952</td>
<td>0.817</td>
<td>0.879</td>
</tr>
<tr>
<td>S. Window</td>
<td>J48</td>
<td>215</td>
<td>205</td>
<td>45</td>
<td>35</td>
<td>0.860</td>
<td>0.826</td>
<td>0.843</td>
</tr>
<tr>
<td></td>
<td>Veto(^d)</td>
<td>248</td>
<td>195</td>
<td>55</td>
<td>02</td>
<td>0.992</td>
<td>0.818</td>
<td>0.896</td>
</tr>
<tr>
<td></td>
<td>Majority(^d)</td>
<td>223</td>
<td>247</td>
<td>03</td>
<td>26</td>
<td>0.895</td>
<td>0.986</td>
<td>0.938</td>
</tr>
<tr>
<td></td>
<td>Trust Veto(^e)</td>
<td>235</td>
<td>230</td>
<td>20</td>
<td>15</td>
<td>0.940</td>
<td>0.922</td>
<td>0.931</td>
</tr>
</tbody>
</table>

\(^a\) The full names of data sets are n-gram data set, overlap data set and sliding window data set.
\(^b\) R is Recall, P is Precision, and F1 is F-Measure.
\(^c\) Veto is Veto Classifier and Majority is Majority voting.
\(^d\) Veto Classifier, Majority voting and Trust-based veto Classifier are applied on all the three data sets.
\(^e\) Trust Veto is Trust-based Veto Classifier.

man expert by automating the process of analyzing and predicting the scareware (malware). JRip and J48 algorithm are considered expensive algorithms in terms of time consumed to train and generate the model. However, it is easy to analyze the rules and trees generated to differentiate the scareware and benign.

The decisions of the different classifiers are combined to produce better results, and such combination shall not be considered as a substitute of a good classifier [152]. The proposed veto classifier follows the same principle. Veto classifier is neither a substitute of a good classifier nor replacing the majority voting. In the domain of decision theory, it has been suggested that different voting strategies shall be adopted for different tasks according to the problem in question. We argue that the veto classifier is a better choice for malware detection task as this approach addresses the problems of the majority voting. There are different problems related with the majority voting such as majority voting may ignore the right decision of the minority. While ignoring the decision from the minority votes, the total number of majority votes may have an ignorable difference in comparison with the total number of minority votes. Another problem of the majority voting is the choice of the number of candidate classifiers. If the number of selected classifiers is an odd, then
7. Voting Schemes

A simple majority can be obtained, but if the number of selected classifiers is an even then a situation may arise where equal numbers of votes are given to both the benign and malware classes. In the domain of ML, different variations of majority voting has been suggested such as restricted majority voting, enhanced majority voting, and ranked majority voting to address the problems of majority voting [153]; such problems are avoided with the proposed veto classifier.

The results of the veto classifier depend upon a suitable permutation of the algorithms. Some permutations may obtain 100% recall by just predicting all applications as malware. Some permutation can achieve 100% precision, if all the instances are predicted as benign applications. Before permutation, classifiers selection is a critical and complex task. For a small number of classifiers, an optimal combination can be found exhaustively, but as the number of classifiers increases, the complexity of selection is increased due to their different inductive biases, search limits and practical applicability. The classifier selection process can be improved by a static selection or dynamic selection method [154].

7.6 Experiment II

Results of the experiment I in the Section 7.5.7 suggest that the veto classifier is a better choice for the malware detection problem. Results also indicate that the majority voting skews towards benign applications and the veto classifier skews towards malicious applications. In the Experiment I, all algorithms are treated equal for the final prediction to assure the "neutrality" property for the veto classifier. Consequently, algorithms who generally demonstrate higher misclassification in comparison to other algorithms (generally referred to as weak learners), supersede the correct prediction. This phenomenon produces a high false positive rate and the precision of the veto classifier is significantly lower than the majority voting. To address the above mentioned problems in the veto classifiers, use of the algorithm’s "trust" is suggested and the veto classifier is extended as a trust-based veto classifier. The aim of this
7.6. Experiment II

The experiment is to evaluate the trust-based veto classifier for the malware detection and compare the performance of the proposed algorithm with the majority voting and the veto classifier.

7.6.1 Trust-based Veto algorithm

Trust as a quantitative measure can be quantified with integer values. The positive integer may represent the trust while a negative integer may be used to represent the distrust. The quantified trust value in computational problems can be calculated for participating nodes, algorithms, and agents. For the machine learning problems, different kinds of trust can be calculated for algorithms contending in the system. Consequently, an algorithm is proposed, which generate trust-based veto classifier for the malware detection. The trust-based veto algorithm involves three kinds of trust, i.e., local trust (can also be referred to as direct trust), recommended trust, and global trust. Each algorithm in the system calculates its trust level for other algorithms in the system, i.e., how much \textit{algorithm}_x trusts the \textit{algorithm}_y in terms of predicting the class of an instance, called local trust (\textit{t}), see Figure 7.2. The local trust value is further used to calculate the recommended trust (\textit{RT}) for each algorithm, see Figure 7.2. The recommended trust aids in calculating the global trust value (\textit{GT}) of each algorithm. The global trust value is used for having a veto decision.

Local Trust Calculation

Local trust of an \textit{algorithm}_x on \textit{algorithm}_y (\textit{ty} : \textit{algo}_x \rightarrow \textit{algo}_y) is calculated by comparing the predictions (\textit{d}) of both algorithms with each other and the actual class (\textit{C}) of the instance, see Algorithm 1. Suppose, from a data set of benign and malicious instances, an instance of benign class is given to the \textit{algorithm}_x and the \textit{algorithm}_y for predicting the class of the instance. There is a finite set of possible predictions, i.e., both algorithms may predict correct, or both algorithms may predict incorrect, or any one of the algorithms may predict the correct class. If both algorithms have the same prediction for the instance, either correct
or incorrect; trust is not affected. However, if the algorithm\(x\) predicts the incorrect class and algorithm\(y\) predicts the correct class, the algorithm\(x\) increases the trust level (\textit{sat}) of the algorithm\(y\) with +1. In case, the algorithm\(x\) predicts the correct class and the algorithm\(y\) predicts incorrect class, the algorithm\(x\) increase the distrust level (\textit{unsat}) of the algorithm\(y\) with +1. All the instances in the data set are given to both algorithms sequentially for the prediction. At the end of process, local trust of the algorithm\(y\) is calculated by dividing trust (\textit{sat}) with the sum of the trust (\textit{sat}) and the distrust (\textit{unsat}), see Algorithm 1.

**Recommended Trust Calculation**

The local trust shows the unique trust on a particular algorithm (e.g., algorithm\(y\)) from another algorithm (e.g., algorithm\(x\)). This value varies from the algorithm to the algorithm in the system and cannot be used as a final metric for deciding about a veto in the system. The Recommended trust is calculated to address this problem. The local trusts on an algorithm from all the other algorithms in the system are summed to calculate the recommended trust. The recommended trust value represents the combine trust of all algorithms in the system on that particular algorithm.
Algorithm 1 Trust Calculation

Require: Actual Class of Instance (C), prediction of algo\(_x\) (\(d_x\)), prediction of algo\(_y\) (\(d_y\))
Ensure: \(t: algo_x \rightarrow algo_y\)
Ensure: \(RT\)
Ensure: \(GT_y\)

function localTrust
repeat
if \(d_x = d_y\) then \(\triangleright\) Prediction of both algorithms may be correct or incorrect
movenext
end if
if \(d_x \neq d_y\) then \(\triangleright\) Compare the prediction with the C
if \(d_x = C\) then
unsat \(\leftarrow\) unsat(algo\(_x\), algo\(_y\)) + 1
else
dy \(\neq C\)
sat \(\leftarrow\) sat(algo\(_x\), algo\(_y\)) + 1
end if
end if
until !EOF
end function

\(t_y : algo_x \rightarrow algo_y\) \(\leftarrow\) \(\frac{\text{sat(algo}_x\text{, algo}_y)}{\text{sat(algo}_x\text{, algo}_y) + \text{unsat(algo}_x\text{, algo}_y)}\)

If the set of all algorithms is \(S = \{algo_0, algo_1, algo_2, \ldots, algo_n\}\) then we may have a two subsets \(S' = \{algo_0\}\) and \(S'' = \{algo_1, algo_2, \ldots, algo_n\}\). The subset \(S''\) is having all the algorithms in the system as members except the algorithm \(algo_0\) for which the \(RT\) is calculated. The algorithm \(algo_0\) is the member of the subset \(S'\). The \(RT\) is calculated by using the Equation (7.1).

\[ RT_y \leftarrow \sum_{n=1}^{n} (t_y : algo_n \rightarrow algo_y) \quad \forall algo_n \in S'' \quad (7.1) \]
7. Voting Schemes

**Global Trust Calculation**

The RT varies from algorithm to algorithm and may not be compared on the similar scale. Consequently, RT of an algorithm is normalized to obtain the global trust of that particular algorithm. The term normalization represents distinct, but related meanings in different contexts. The basic purpose of normalization is to convert the different values on a notionally standard scale to compare them equally with each other. The normalized GT value lies in the interval of the [0-1] and is calculated by using the Equation (7.2):

$$GT_y \leftarrow \frac{RT_y}{\sqrt{\sum_{n=1}^{n} RT_n^2}}$$  \hspace{1cm} (7.2)

**Veto Decision**

The calculated GT value is used for deciding a veto for the prediction of a set of algorithms by another algorithm or set of other algorithms. Suppose a system in which seven algorithms are participating for predicting the class (benign or malicious) of an instance that belongs to the malicious class. A subset M of four algorithms in the system predicted the class of the instance as benign, and a subset V of three algorithms predicted the class of the instance as malicious. The mean of both groups is calculated. If the mean of V is greater than the mean of M, the V can veto the decision of the M and the outcome will be the prediction of the V.

However, for this experiment, there is a change in the veto decision function due to less algorithms. The change is explained as following. There are three algorithms in the system, i.e., algo\_x, algo\_y, and algo\_z. If two algorithms, i.e., algo\_x, and algo\_z classify the instance as a benign and only one algorithm, i.e., the algo\_y classify the instance as a malware; the algo\_y can veto according to Equation (7.3). The change in the prediction strategy is to reduce the random decision errors.

$$Veto : GT_y \geq \frac{GT_x + GT_z}{2}$$  \hspace{1cm} (7.3)
7.6. Experiment II

Figure 7.3: Results

7.6.2 Performance Evaluation Measures

Performance evaluation measures are same as used in the Experiment I and described in the Section 7.5.5.

7.6.3 Results and Discussion

The classifiers combination to form an ensemble can be divided roughly into two categories, i.e., multiclassifier and multirepresentation [123]. In the multiclassifier approach, a set of classifiers is trained on the same representation of the data. In the multirepresentation approach, different classifiers are trained on the multiple representations. On the basis of the experimental results presented in the Section 7.5.7, the trust-based veto classifier is applied only for combining the inductive biases of several algorithms trained on the different representations. The experimental results are shown in the Table 7.2. Experimental results indicate two issues. First, the low TN of the veto classifier that leads to a high FP; the low TN is because of veto classifier’s skew towards malware. Second, the low TP of the majority voting that leads to a high FN; the low TP is because of the majority voting’s skew towards benign programs. The trust-based veto classifier performs better than the veto classifier in
7. Voting Schemes

terms of TN and reduces the FP. The trust-based veto classifier is also better than the majority voting in terms of TP and reduces the FN. However, TP of trust-based veto classifier is better than the majority voting and less than the veto classifier. Recall of the trust-based veto classifier is better than the majority voting and less than the veto voting. In terms of F-Measure, the difference between the values of f-measure of the majority voting and the trust-based veto classifier is minimal, so one can argue that the trust-based veto classifier is an optimal choice for the malware detection due to inherited skewness towards the malware. The majority voting and the veto classifier are computationally inexpensive as the prediction from each algorithm is counted for the outcome. In trust-based veto classifier, each algorithm evaluates the trust and maintains the trust info locally in a trust table without significantly increasing the processing overhead; however, the storage requirement is higher than the majority voting and the veto classifier as they do not store any information. The locally stored trust information is provided to the system for the decision purpose, when required. Trust-based veto classifier provides a direct experience of the trust on each algorithm. Due to direct experience, there is no central authority for maintaining the trust information, which makes the proposed algorithm a self-policing algorithm.

One property mentioned in the voting rule set is anonymity (see section 7.3.1). The trust-based veto classifier maintains the anonymity property as all votes are treated equally and no vote is discarded. However, the veto property is not followed as it is mentioned in section 7.3.1 for the trust-based veto classifier experiment. In the changed veto decision strategy, a single algorithm indicating the instance as the malware cannot affect the outcome of the detection task. Now for the veto, algorithm or set of algorithms need to meet certain criteria, which reduced the chances of errors and terminate the prediction of weak learners. However, with the proposed strategy, there is a probability that the prediction of weak learner/s may be always ignored, if the trust on that particular algorithm is significantly less than the trust on other algorithms. Suppose a detection system with five algorithms where two algorithms are weak learner with significantly low trust levels. This particular group
of algorithms may not veto the decision of all other algorithms for all
the cases, even if the prediction was correct. However, the changes in
the veto strategy will increase the robustness of trust-based classifier as
any number of algorithms can compete for the veto decision with any
number of algorithms.

To follow the veto strategy mentioned in the section 7.3.1, one alter-
native direction is to allocate the trust to each algorithm on the basis of
predetermined criteria such as previous performance. The trust of all
algorithms may be readjusted regularly on the basis of prediction per-
formance. All the algorithms vote for the decision. For the veto decision,
when an algorithm predicts the instance as malware the trust of that par-
ticular algorithm can be compared with a specific threshold to obtain the
final decision. However, it is worth to note that veto strategy does not
perform as expected for the encrypted malware, i.e., malware with the
encrypted malicious routine. The encrypted part of the malware cannot
be disassembled to obtain accurate instruction sequences or byte code.
The presence of encryption in a file can be considered as the indication
of the malicious behavior. The encrypted malware can be decrypted
or executed to decrypt in a controlled environment to obtain the data
files. The data files can be further disassembled to extract instruction
sequences or byte code.

7.7 Conclusion and Future Work

There are a several strategies to obtain the result of an ensemble such
as the majority voting and the veto voting. However, it is not inves-
tigated which decision strategy is optimal for the malware detection.
Most of the researchers have used the majority voting for the malware
detection. A veto-based classification was proposed that was able to
predict about malware better than the majority voting. A series of ex-
periments with \( n \)-gram data sets, generated from different strategies,
were performed. A recent threat, i.e., scareware was used as malware
representation. The results indicated that the proposed model reduced
7. Voting Schemes

the number of false negatives (malware detected as legitimate application), however, the false positive of proposed model was very high. The decision strategy of proposed model was improved, i.e., trust-based veto classifier. The experimental results indicated that the improved classifier perform better than the previous approach in terms of the false positive rate. The proposed trust-based veto classifier performed better in the recall than the majority voting. However, for the composite measure F1, the majority voting was slightly better than the trusted-veto classifier and the trusted veto classifier was better than the veto voting. The experimental results also indicated the suitability of each voting scheme for detecting a particular class of software. For the future work, the aim is to further improve the proposed model in two different directions, i.e., improvement in the selection of classifiers for the optimal results, and parameter tuning of the selected classifiers. The proposed model will also be tested for the detection of different types of malware and for the multi-class prediction.
Chapter 8

Consensus Decision Making in Random Forests

Raja Khurram Shahzad, Mehwish Fatima, Niklas Lavesson, Martin Boldt

Abstract

The applications of Random Forests, an ensemble learner, are investigated in different domains including malware classification. Random Forests uses the majority rule for the outcome, however, a decision from the majority rule faces different challenges such as the decision may not be representative or supported by all trees in Random Forests. To address such problems and increase accuracy in decisions, a consensus decision making (CDM) is suggested. The decision mechanism of Random Forests is replaced with the CDM. The updated Random Forests algorithm is evaluated mainly on malware data sets, and results are compared with unmodified Random Forests. The empirical results suggest that the proposed Random Forests, i.e., with CDM performs better than the original Random Forests.
8. Consensus Decision

8.1 Introduction

One of the main challenges for anti-malware vendors is to detect or classify an unknown malware. The unknown malware are also referred to as zero day malware. The existing detection techniques, i.e., signature based (pattern matching) and rule based are incapable of detecting a zero day malware. To address this problem, researchers have borrowed different approaches from different domains including supervised learning and investigated their applicability for the malware detection. In supervised learning, a model is generated from the labeled data set for the malware classification. The generated model is commonly referred to as classifier and is further used to classify malware and benign files. Experimental results have indicated that a combination of multiple classifiers (ensemble) such as Random Forests (RF) may perform better in comparison to single classifier. Random Forests\(^1\), is created by generating the desired number of decision tree classifiers (also known as base learners) [121]. For the classification, an instance is given to each tree in RF, which then returns its prediction about the given instance. The decision of each tree is considered as a vote for obtaining the final decision. The final decision of RF is based on the majority rule, i.e., the class obtaining the majority of votes is the output. As an ensemble, RF is expected to perform better in terms of prediction accuracy than single base learners [155]. Typically, the predictive performance of a classifier is evaluated by estimating its error rate, i.e., the proportion of misclassified instances. In case of RF, the error rate may be affected by the presence of noisy features in the data set and majority of votes to a wrong class [156,157]. The final decision of the RF algorithm is a multi-classifier decision-making process and is generally referred to as the group decision-making in the decision theory [158]. The final decision of the group may vary from the decision of an individual participant in the group. Thus, the group decision may not be supported by the whole group, which is a drawback of majority rule in context of unanimous agreement within the group. A unanimous decision is critical for problems such as malware classifi-

\(^1\)http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
cation, where the misclassification cost is very high in terms of losses to the user.

To address the problems of majority rule, a consensus decision-making (CDM) method is suggested [158]. For the CDM method, all group members strive to reach an optimal outcome while catering the concerns of each member as much as possible. We hypothesize that the prediction accuracy of the RF algorithm can be improved by incorporating CDM. For integrating CDM in RF, the majority decision of Random Forests is replaced by CDM. Later, modified RF algorithm is compared to the baseline, i.e., traditional Random Forests implementation in WEKA [98] on a number of data sets using different evaluation measures.

The remainder of this paper is organized as follows: Section 8.2 provides a background of Random Forests and decision-making theory. Section 8.2.1 discusses the related work. Section 8.3 discusses theoretical foundations regarding CDM. Section 8.4 presents the experimental setup, while Section 8.5 discusses the results from the experiment and Section 8.6 concludes the paper.

8.2 Background

Random Forests is an ensemble of decision trees. For creating a RF, a specific number of trees are generated through bootstrap sampling. Due to the random selection of nodes during each tree generation, each tree in RF varies in the classification accuracy [155,159]. Each generated tree in RF is tested with the out of bag data, i.e., the data, which is not used for the training. The average mis-classification of Random Forests is known as the out of bag error estimate. For the outcome RF uses the (un-weighted) decisions from each tree and provides a decision based on the majority rule. It is indicated that group decision-making, where a problem is decomposed into smaller sub-problems, performs better in multiple problem domains such as decision theory, artificial intelligence, fuzzy sets theory and expert systems [160].
The group decision-making includes either the majority rule, autocracy (veto), and consensus vote. Contrary to the majority rule, in the veto, a designated member of the group or group of specific members may decide the outcome of the group. Both methods, i.e., majority rule and veto have their benefits and drawbacks. However, in both methods, the decision from a subgroup of members is ignored and outcome of the group cannot be referred as a decision that is supported by each member of the group. To address this problem, consensus voting is suggested [158]. In general, the word consensus refers to an agreement

http://www.merriam-webster.com/dictionary/consensus

However, in case of the group decision-making, the word consensus is distinct from the common meaning and refers to a process in which all members of the set $C$ collaborate to reach a solution that is supported by all the members of the group regardless of their individual assessment [158].

8.2.1 Related Work

Random Forests: The ideas of random decision forests and Bagging are presented in 1995 and 1996 respectively [161, 162]. These two concepts served as the base for the development of RF in 1999. In 2001, it is presented that RF does not over-fit because of Law of large numbers [121]. In 2008, the basic theorem of RF is further enhanced by a series of theorems that established the universal consistency of averaging rules [155]. Since the idea of RF is presented, it has been used in different domains such as medical science, biology, bio-informatics, computer security, image processing, malware classification [129], and many others [159]. Different variations of Random Forests are also investigated by researchers to improve the prediction accuracy [163]. Some researchers have calibrated RF [164], while some have used Random Forests for the selection of important variables [165]. RF has also been improved for obtaining predictions from imbalance data sets [166].

Decision Making: A group of individuals, for solving a given task, may select a suitable option from the set of alternatives using different methods such as the majority rule [167], multiple winner, proxy voting, and

veto [168]. These concepts are further extended by the social choice theory, which is a theoretical framework to attain a group decision from the participants while considering the individual preferences [169]. Different theories such as social decision scheme [170], quantitative decisions [171] and consensus decision-making are investigated. The idea of consensus decision-making is presented for the selection of a suitable candidate among multiple candidates using both individual judgment and group judgment [158].

Malware Classification: The task of malware classification is generally considered as a binary classification problem. RF algorithm is used for detecting the malware on computer systems, mobile phones and network data streams. RF has outperformed other algorithms such as Bagging, boosting, and decision trees for the detection of unknown malware [109, 172].

8.3 Consensus Decision-Making in Random Forests Algorithm

This section explains the consensus decision-making algorithm. The consensus decision-making process can be divided into two stages, i.e., consensus stage and selection stage. The consensus stage consists of several rounds where the preferences of each expert for the alternatives are evaluated to reach the solution. The iterative nature of the process helps the experts to reduce the differences between their opinions before concluding an outcome. The selection stage provides the subset of the most suitable alternative or a solution set. Different strategies may be further applied, (if required), to obtain the solution set.

8.3.1 Consensus Stage

In the consensus stage, decisions from the individual classifiers are collected and each decision is given a weight, which is further used to calculate the weighted group decision. The obtained weighted decision is
re-evaluated iteratively until the convergence is achieved. The consensus stage is described as follows:

1. **Vector of Classifiers**: In Random Forests, \( k > 1 \) tree classifiers are generated from Bootstrapping and a vector of generated classifiers is obtained.

2. **Initial decisions (Predictions)**: Each classifier classifies the given instance and provides a vector of decisions about the \( n \) alternative classes. A decision matrix of \( k \times n \), which gives the decisions from all classifiers about all alternative classes is generated.

3. **Criterion Weights = Criteria × Predictions**: The prediction of each tree classifier is multiplied with the selected criterion to obtain its weighted decision. For the experimental purpose, the out of bag error (OOB) [121] is selected as the criterion to evaluate decisions of generated tree classifiers. If a classifier is able to correctly predict the classes and OOB for that particular classifier is zero, the decision of the classifier is multiplied by 1. In all other cases, the decision of the classifier is multiplied by \( 1 - \text{OOB} \). This procedure reflects the confidence level of the classifiers for each classification.

4. **Aggregation** The first aggregation step combines the values for each alternative class. This generates score vectors, one for each classifier and for each class after aggregation. These score vectors are used to generate the score matrix (S).

5. **Recursive Aggregation over Classifiers** The OOB of each classifier is used to indicate the confidence level of the classifier for predictions. Thus, each classifier defines a vector of confidence for alternative classes. These vectors are used to construct a matrix R. The diagonals of matrix R are the degree of confidence that each classifier has in the correctness of its prediction. The matrix R differs at each step. Consequently, for each step the weighted mean aggregation is applied over the result of the previous step:
8.3. Consensus Decision-Making in Random Forests Algorithm

\[ T = R \times S \]

6. **Next Iteration:** For the next aggregation step, following assignment is performed: \( S = T \). The process is repeated for each class until the convergence is achieved. Suppose the number of aggregation steps are \( p \), then after \( p \) aggregation steps:

\[ S_p = R_p^T \times \cdots \times R_1^T \times C^T = (C \times R_1 \times \cdots \times R_p)^T \]

7. **Iteration Termination:** After the individual judgment, multiple iterations are performed, which combines the multi-classifier scores per class to obtain the overall score for each class, which leads to final ranking of all the classes. However, it is necessary to determine, how the iterations will stop. For this particular study, if the difference between the sums of two iterations is less than \( 1 \times e^{-5} \), the iterations are stopped and convergence is achieved.

### 8.3.2 Selection Stage

The selection stage provides the outcome of the RF. The selection stage receives the weighted means in the form of decisions from the consensus stage and the class with the highest weight is considered as the outcome.

### 8.3.3 Example and Explanation of Algorithm

Assume, a RF of 10 trees is generated using WEKA from a malware data set, which contains both malware files and benign files. An instance of malware file is given to this Random Forests. The task is to perform a binary classification, in which ‘1’ represents the presence of a malware in the file (or a malware file), and ‘0’ represents a benign file. Each generated tree contains (assumed value) OOB and has predicted the class of the given instance (see Table 8.1). For simplicity, assume that the decision vector of each tree (classifier) contains only outcome. Table 8.1 also presents the other required information such as the weight given by the tree to its decision \( (W_1) \), i.e., \( 1 - OOB \), and weight given to rest of trees.
8. Consensus Decision

individually \((W_2)\), i.e., \(OOB/9\). These values are further used for calculating the weighted aggregated mean and remain constant throughout the process. To obtain the weighted decision, the prediction of each tree classifier is multiplied with its \(W_1\). It is worth noting that if the OOB of a particular tree is zero, the decision of the tree will be multiplied by ’1’. To calculate the score matrix for each tree; sum of the prediction \(P_n\) from each tree is multiplied by \(W_1\), and sum of decisions from other trees is multiplied with the OOB of current tree. The obtained score is divided by 10 for calculating the aggregated weighted mean. The score matrix of selected iterations, i.e., initial iteration, \(i\)th iteration and final iteration is shown in Table 8.2. The iterations are stopped, if the difference between the sum of \(i\)th iteration and \(i+1\) iteration is \(< 1 \times e^{-5}\). Moreover, the mean of last iteration is taken and the score below the mean is considered as the prediction for the class ’0’ and score \(>\) mean is considered as the prediction for the class ’1’. In this example initially, five trees have predicted the class as malware and five trees have predicted the class as benign. In such cases, majority rule cannot determine an outcome. When the majority rule is replaced with CDM, majority of trees have

Table 8.1: Initial State

<table>
<thead>
<tr>
<th>Trees</th>
<th>(P^*)</th>
<th>OOB**</th>
<th>(W^a_1)</th>
<th>(W^b_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0</td>
<td>0.25</td>
<td>0.75</td>
<td>0.027777778</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
<td>0.20</td>
<td>0.80</td>
<td>0.022222222</td>
</tr>
<tr>
<td>T3</td>
<td>0</td>
<td>0.15</td>
<td>0.85</td>
<td>0.016666667</td>
</tr>
<tr>
<td>T4</td>
<td>0</td>
<td>0.50</td>
<td>0.50</td>
<td>0.055555556</td>
</tr>
<tr>
<td>T5</td>
<td>1</td>
<td>0.30</td>
<td>0.70</td>
<td>0.033333333</td>
</tr>
<tr>
<td>T6</td>
<td>1</td>
<td>0.65</td>
<td>0.35</td>
<td>0.072222222</td>
</tr>
<tr>
<td>T7</td>
<td>1</td>
<td>0.30</td>
<td>0.70</td>
<td>0.033333333</td>
</tr>
<tr>
<td>T8</td>
<td>0</td>
<td>0.80</td>
<td>0.20</td>
<td>0.088888889</td>
</tr>
<tr>
<td>T9</td>
<td>1</td>
<td>0.25</td>
<td>0.75</td>
<td>0.027777778</td>
</tr>
<tr>
<td>T10</td>
<td>1</td>
<td>0.30</td>
<td>0.70</td>
<td>0.033333333</td>
</tr>
</tbody>
</table>

\(^a\) Initial Predictions, \(^**\) Mean Absolute Error

\(^a\) Self Weight, i.e., \(1 – OOB\)

\(^b\) Weight given to each classifier in RF (OOB\(_n\)/9)
8.4 Experiment

The aim of the experiment is two folds. First, to evaluate the impact of CDM in the Random Forests algorithm in comparison to the original RF algorithm for the task of malware classification. Secondly, to validate the generalizability of modified algorithm. The proposed changes may be used for the multi-class classification problems, however, the experiments are performed only for the binary classification. Hereafter, the modified Random Forests with the CDM will be referred to as Consensus Random Forests (CRF). Two types of experimental data sets are used, i.e., a generated data sets for the malware classification and two data sets taken from a machine learning repository [173]. These data sets may also be categorized according to their size, i.e., number of instances or features. All data sets are in the Attribute-Relation File Format (ARFF)³, which is a structured ASCII text file that includes a set of data instances along with a set of features for each instance [34]. ARFF files are used

---

<table>
<thead>
<tr>
<th>Trees</th>
<th>Initial Scores</th>
<th>Intermediate Scores</th>
<th>Final Scores</th>
<th>Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0.125</td>
<td>0.0560</td>
<td>0.000450078</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>0.100</td>
<td>0.0458</td>
<td>0.000371326</td>
<td>0</td>
</tr>
<tr>
<td>T3</td>
<td>0.075</td>
<td>0.0351</td>
<td>0.000287487</td>
<td>0</td>
</tr>
<tr>
<td>T4</td>
<td>0.250</td>
<td>0.0995</td>
<td>0.000781605</td>
<td>1</td>
</tr>
<tr>
<td>T5</td>
<td>0.190</td>
<td>0.0673</td>
<td>0.000524192</td>
<td>1</td>
</tr>
<tr>
<td>T6</td>
<td>0.295</td>
<td>0.1205</td>
<td>0.000941675</td>
<td>1</td>
</tr>
<tr>
<td>T7</td>
<td>0.190</td>
<td>0.0673</td>
<td>0.000524192</td>
<td>1</td>
</tr>
<tr>
<td>T8</td>
<td>0.400</td>
<td>0.1352</td>
<td>0.001079901</td>
<td>1</td>
</tr>
<tr>
<td>T9</td>
<td>0.175</td>
<td>0.0585</td>
<td>0.000450079</td>
<td>1</td>
</tr>
<tr>
<td>T10</td>
<td>0.190</td>
<td>0.0673</td>
<td>0.000524192</td>
<td>1</td>
</tr>
</tbody>
</table>

* If Score is < $1 \times e^{-5}$, prediction is class '0' otherwise prediction is class '1'.

indicated the presence of a malware, which helps in determining the outcome.

³http://www.cs.waikato.ac.nz/ml/weka/arff.html
as an input to models generated by CRF and RF. The experiments are performed with Random Forests of different number of trees.

Table 8.3: Classification Results with 10 trees in Random Forests

<table>
<thead>
<tr>
<th>UCI Data sets</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>Accuracy</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forest Covertype Data Set</td>
<td>RF(^1)</td>
<td>0.953</td>
<td>0.048</td>
<td>0.953</td>
<td>0.953</td>
<td>0.952</td>
<td>0.989</td>
</tr>
<tr>
<td></td>
<td>CRF(^2)</td>
<td>0.953</td>
<td>0.048</td>
<td>0.953</td>
<td>0.953</td>
<td>0.953</td>
<td>0.989</td>
</tr>
<tr>
<td>Blogger Data Set</td>
<td>RF</td>
<td>0.790</td>
<td>0.330</td>
<td>0.783</td>
<td>0.790</td>
<td>0.782</td>
<td>0.790</td>
</tr>
<tr>
<td></td>
<td>CRF</td>
<td>0.830</td>
<td>0.279</td>
<td>0.827</td>
<td>0.830</td>
<td>0.824</td>
<td>0.830</td>
</tr>
</tbody>
</table>

| Malware Data sets          |         |         |           |        |           |          |      |
| 4-gram Data set            |         |         |           |        |           |          |      |
| RF                        | 0.904   | 0.096   | 0.906     | 0.904  | 0.904     | 0.903    | 0.967 |
| CRF                       | 0.916   | 0.084   | 0.918     | 0.916  | 0.916     | 0.916    | 0.974 |

\(^1\) Random Forests with majority rule.
\(^2\) Consensus Random Forests.

8.4.1 Data sets

The data sets from UCI repository are Forest Coverttype data set\(^4\) and Blogger data set\(^5\). The Forest Coverttype data is generally considered as one of the largest data sets available for machine learning experiments [174]. This data set contains 581012 instances with 54 features. This data set is preprocessed to transform from multi-classification to binary classification problem\(^6\). While Blogger Data Set is relatively small and contains only 100 instances with six features [175]. The data set is used to classify users’ trends in cyberspace into two distinct groups, i.e., professional bloggers (regular bloggers) and seasonal bloggers (irregular bloggers).

\(^4\)http://archive.ics.uci.edu/ml/datasets/Covertype
\(^5\)http://archive.ics.uci.edu/ml/datasets/BLOGGER
\(^6\)http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
Malware Data Set is generated for malware classification. The malware data set contains 800 instances of both malware programs and benign programs for Windows operating system. Malware programs are gathered from Lavasoft repository and benign programs are gathered from two different sources i.e., an online software repository CNET Download.com and from a clean installation of Windows OS. The collected benign files are checked with commercial anti-virus products to eliminate the probability of false negatives. Out of 800 examples, the malware class contains 400 instances, and the benign class contains 400 instances. The generated data sets is disassembled using nsdisasm utility of Ubuntu to extract different pieces of information such as byte code, OpCode (operation code is the part of an assembly instruction that specifies the operation to be performed), text strings, and other information. The extracted information is further used to represent the malware and benign examples for the classification task [50]. However, OpCode representation as $n$-gram has produced better results than other representations. Some researchers showed that 4-gram, i.e., $n=4$ produces optimal results. Thus, the size of $n$-gram in the experimental data set is four, i.e., 4-gram [157].

From the disassembled output only OpCodes are extracted by using a parser and other information is discarded. Disassembly and parsing have generated a large number of OpCode sequences, i.e., $n$-grams. Many of these sequences may not have any useful contribution in the classification task. Thus, it is crucial to select the most contributing sequences without affecting the classifier accuracy. In the malware detection domain, each generated sequence is analogous to a term in a text document. Thus, to find the most valuable sequences, Term frequency-Inverse document frequency $Tf-idf$ from the text categorization field is used [120]. Top 1000 sequences of benign and malware are selected by using the $Tf-idf$. These sequences are further used as features to repre-
sent a malware or a benign instance and generate ARFF files.

Table 8.4: Classification Results with 100 trees in Random Forests

<table>
<thead>
<tr>
<th>UCI Data sets</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>Accuracy</th>
<th>AUC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forest Covertype Data Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF¹</td>
<td>0.953</td>
<td>0.048</td>
<td>0.953</td>
<td>0.953</td>
<td>0.953</td>
<td>0.963</td>
<td>0.989</td>
</tr>
<tr>
<td>CRF²</td>
<td>0.963</td>
<td>0.036</td>
<td>0.963</td>
<td>0.963</td>
<td>0.963</td>
<td>0.963</td>
<td>0.995</td>
</tr>
<tr>
<td>Blogger Data Set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>0.850</td>
<td>0.219</td>
<td>0.848</td>
<td>0.85</td>
<td>0.848</td>
<td>0.85</td>
<td>0.839</td>
</tr>
<tr>
<td>CRF</td>
<td>0.850</td>
<td>0.219</td>
<td>0.848</td>
<td>0.85</td>
<td>0.848</td>
<td>0.85</td>
<td>0.849</td>
</tr>
<tr>
<td>Malware Data sets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-gram Data set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>0.940</td>
<td>0.060</td>
<td>0.940</td>
<td>0.940</td>
<td>0.940</td>
<td>0.940</td>
<td>0.987</td>
</tr>
<tr>
<td>CRF</td>
<td>0.943</td>
<td>0.058</td>
<td>0.943</td>
<td>0.943</td>
<td>0.942</td>
<td>0.942</td>
<td>0.988</td>
</tr>
</tbody>
</table>

¹ Random Forests with majority rule.
² Consensus Random Forests.

8.4.2 Evaluation Measures

The performance is evaluated using 10-fold cross-validation. Confusion matrices are generated by using the response of both classifiers. The following four measures define the elements of a confusion matrix from algorithms used in experiment: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). These elements are used to determine the True Positive Rate ($TPR = \frac{TP}{TP+FN}$), False Positive Rate ($FPR = \frac{FP}{TN+FP}$), True Negative Rate, ($TNR = \frac{TN}{TN+FP}$), False Negative Rate ($FNR = \frac{FN}{TP+FN}$), Recall, and Precision, which are further used to calculate the composite measures, i.e., Accuracy ($ACC = \frac{TP+TN}{TP+TN+FP+FN}$), Area Under ROC (AUC), and F-Measure ($F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$). ACC is the percentage of correctly identified classes. For some data sets, ACC can be a reasonable estimator of performance (the performance on the novel data). However, if data sets are imbalanced, the ACC metric may be used as a complementary metric with area under ROC. ROC is plotted as a result of TPR on the x-axis in the function of FPR on the y-axis at different points. AUC is commonly used when the performance of a
classifier needs to be evaluated for the selection of a high proportion of positive instances in the data set [34]. However, AUC has the benefits of being independent of class distribution and cost [99]. F-Measure is the harmonic mean of precision and recall. These evaluation parameters are used to compare the individual performance of RF and CRF.

8.5 Results and Analysis

The experimental results are presented in Table 8.3 and Table 8.4. It is worth noting that presented results are the average result of both classes. Table 8.3 presents the results of RF and CRF with 10 trees. Table 8.4 presents the results of RF and CRF with 100 trees. Both tables present the results of TP rate, FP rate, Precision, Recall, ACC, AUC and F-Measure.

The result’s analysis can be broadly divided into two parts, i.e., overall discussion and interpretation of results on all data sets and data set specific discussion and interpretation. It is worth noting that there is no difference between RF and CRF in terms of generated number of trees and obtaining the classification. For both experiments, CRF performs better than RF by increasing the TP and TN and decreasing the FP and FN. Table 8.3 suggests the improvement in results for all data sets except the Forest cover type data set. The Forest cover type data set is a large data set with 500,000+ instances. Thus, hundreds of instances classified correctly or incorrectly may not affect the results of the composite measures if average results are presented. However, in case of middle size data sets and small size data sets such as malware data set and Blogger data set, the difference in the number of instances in the elements of a confusion matrix produces the significant difference in composite measures. When comparing the results of Table 8.3 and Table 8.4, it is clearly indicated that the increase in number of trees, improves the classification results for larger data sets. For the smaller data set, the difference in results is ignorable. Another factor, which is not presented in both tables is running time of experiments. Both experiments have consumed the similar amount of time for the selection of features and generating trees.
8. Consensus Decision

However, when the decision part of the CRF is executed, the execution time of algorithm is significantly affected with the number trees. In the CRF algorithm, the score matrix for each tree is calculated and this process is performed iteratively until the convergence is achieved. Among all the data sets, the longest time to obtain predictions is for the CRF on the Forest cover type data set as for every instance, iterations are performed. However, for the other data sets, if the resources consumed are compared with the improvement in results, the CRF with 100 trees may be recommended.

It is worth noting that in experiments, the convergence is achieved between 15 to 20 iterations. To verify this, the numbers of iterations are manually increased from the 20 iterations to 40 iterations; however, no difference in results is found. The performance of the RF and CRF depends upon the number of selected features and quality of selected features for the generation of trees. However, to keep the selection procedures similar for both algorithm, no further pre-processing was performed. For these experiments the number of selected features is small. For the Forest Cover type data set, six features out of 54 are selected for the classification. For the malware data set, 11 features are selected out of 1000. Another contributing factor towards the performance of CRF is OOB of generated trees, which is used for calculating the score matrix. If the generated set of trees in RF is having a zero OOB, then there may not be a difference in the performance of RF and CRF. However, if the data is noisy, and generates trees that contain high OOB, the CRF is more useful as it uses OOB value for the prediction. If OOB of a tree is high, generally the good prediction results may not be obtained using RF. However, CRF uses the OOB to change the decision of trees. When the predicted outcome of a tree is multiplied with its OOB, the distance between actual class and predicted class increases or decreases. This process continues, until the prediction of that particular tree falls clearly in one class. This characteristic of CRF moves the borderline cases to a distinct class case and improves the classification results.
8.6 Conclusion

Random Forests has attracted researchers due to its randomness during tree generation and improved classification results compared to single classifiers. However, for obtaining the final decision, RF uses majority rule, which is not an optimum option for all situations especially for the malware classification task. Thus, to improve the decision mechanism, this paper introduces the consensus decision making in RF. To empirically evaluate the effect of changes, two different experiments are performed. Each experiment contains a large, medium and small data set. Two different number of trees, i.e., 10 and 1000 are used for experiments. The experiment results suggest that the modified algorithm improves the classification accuracy. The results also suggest that the modified algorithm with increased number of trees may be used for larger data sets. While, the modified algorithm with 10 trees may be used for medium and small data sets. For the future work, we plan to extend our experiments for the multi-class malware classification with more parameterization.
Chapter 9

A Hybrid Approach For Malware Classification Using Secondary Features Fusion

Raja Khurram Shahzad, Muhammad Mustaqueem, Haroon Elahi

Abstract

The number of malware (either variant or novel) is rapidly increasing, making malware detection and mitigation a complex problem. One approach to improving malware mitigation is automatic detection and malware family classification. However, traditional malware detection methods cannot classify detected malware into their respective families, hindering effective malware mitigation. Consequently, this paper proposes a method to automate malware detection and classification of the detected malware into respective malware families. The proposed method uses feature fusion after extracting relevant malware features such as API calls and fixed and variable length n-grams with a customized feature selection method. Moreover, for the predictive model, a voting-based approach is proposed for algorithm fusion. For the experimental evaluation of the proposed method, both binary and
multi-class classification approaches are applied to the data set pro-
vided by Microsoft. Finally, the experimental results are compared
with the state of the art. The experimental results indicate the ef-
fectiveness and efficiency of the proposed approach with an AUC
of 0.989, accuracy of 99.72%, and a log loss of 0.01.

9.1 Introduction

Malware samples are increasing and evolving yearly due to associated
monetary gains from their use in activities such as illegal content dis-
tribution and cyber attacks on organizations. For example, in the first
half of 2022, 2.8 billion\(^1\) malicious software (malware) samples were col-
lected. Generally, most malware are variations of existing malware\(^2\).
To generate variations, malware authors use readily available tools that
apply polymorphism, obfuscation, or a combination. These techniques
change the pattern and behavior of the malware. Consequently, existing
anti-malware software cannot detect variated or zero-day malware [176].
Therefore, detecting and classifying malware before it serves its mali-
cious purposes is highly prioritized. Effective malware mitigation re-
quires detection, i.e., determining whether particular software is benign
or malicious, and family classification, i.e., identifying the respective
malware family of the detected malware.

Malware detection is generally considered a binary problem addressed
by static and dynamic analysis. The malicious file is disassembled for
static analysis, and the file’s control flow is investigated for malicious
patterns. On the contrary, for the dynamic analysis, the malicious file is
executed in a secure environment such as a sandbox, and the file’s be-
havior is observed for malicious patterns. Due to the limited capabilities
of static and dynamic analysis, since 2000, researchers have investigated
the usage of machine learning (ML) to generalize malware detection
and classification on different data sets ranging in size from small to

---

\(^1\)https://www.sonicwall.com/medialibrary/en/white-paper/mid-year-2022-cyber-
threat-report.pdf
\(^2\)https://www.av-test.org/
large from different resources such as VX Heaven\textsuperscript{3} or VirusShare\textsuperscript{4}. Most data sets contain malware examples of a particular family or type. Most of these studies focus on extending static analysis by finding novel features for malware detection and increasing detection accuracy. However, these features may be used for a particular family and may not be generalized to all families. Moreover, the change in the data set may degrade the detection rate. Further, the static features may not indicate a malicious file’s behavior. In 2015, Microsoft released one of the largest disassembled malware data sets as a Kaggle\textsuperscript{5} challenge to improve malware mitigation. This data set can be used for a multi-class problem, where malware can be classified into different families. Researchers have used the provided data set (without any modification) to highlight ML’s importance or evaluate the proposed solution for malware classification. However, the provided data set contains only malware samples and cannot be used to classify between benign and malware.

This work modifies and extends Microsoft’s data set with benign examples and proposes an ML-based solution for malware detection (determining if a sample is benign or malicious) and, subsequently, the detected malware’s classification to respective families. For the experimental purpose, the Microsoft data set is partially used to evaluate the proposed approach for malware classification. The proposed approach addresses two distinct stages of ML, i.e., feature engineering and modeling.

The contributions of this article are as follows:

- This study extends the Microsoft data set with benign examples to address binary and multi-class problems.
- This study uses application programming interface (API) calls, Dynamic Link Library (DLL) imports, and operation code mnemonics

\textsuperscript{3}Vxheaven.org  
\textsuperscript{4}https://virusshare.com  
\textsuperscript{5}https://www.kaggle.com/c/malware-classification
9. **Hybrid Approach**

![Feature Selection Process Diagram](image)

**Figure 9.1: Feature Selection Process**

(OpCodes) as primary features. The combinational analysis of API calls and DLL imports is also performed to obtain high accuracy and valuable features. OpCodes are used to generate secondary features, i.e., fixed length \(n\) and variable length-grams. All these features are combined to generate a feature set.

- Different feature selection methods, such as filter, wrapper, and hybrid methods, are evaluated. Further, for feature selection, a customized backward selection algorithm is proposed.

- After the feature selection, feature fusion is performed to get a data set with the best features. As per the authors’ best knowledge, no feature fusion representing all families of feature selection methods is investigated for malware classification.

- Algorithm Fusion is performed to suggest a weighted voting-based ensemble to determine the outcome.
9.1. Introduction

9.1.1 Related Work

For malware detection, the cost of misclassifying a malware is higher than the cost of misclassification of a benign file due to its impacts on the system. Thus, researchers have used machine learning methods such as feature engineering, feature selection, and generating ensembles to improve malware detection and classification accuracy on different platforms. For the Microsoft malware data set, two types of studies are conducted, i.e., highlighting the importance of machine learning for malware classification and evaluating the proposed approach [23]. The latter group may be further divided into two categories: evaluation of the proposed feature engineering approach and assessment or comparison of predictive models. Different studies have presented their work for extracting novel features and reducing features dimensionality [49, 51, 177, 178, 179]. However, a few authors have reported that using both modalities, i.e., byte sequence and disassembled code, may cause overfitting [53]. For the feature selection and dimensionality reduction step, different authors have used different methods [22, 180, 181, 182, 183, 184, 185]. For the classification, algorithms from different classes are used, i.e., AdaBoost, XGBoost, LG Boost, Random forest, Extra trees, Rotation trees, and Random Forest [49]. A few authors have reported customized ensembles and parameter tuning [53, 179]. In an attempt to select the best features, the researchers have used eight different filter-based feature selection methods [52]. Four filter methods commonly used in studies are information Gain, Odds Ratio, Chi-Square, and Inverse document frequency. The new methods used by researchers are Document frequency thresholding, M2, and Relevance frequency feature selection. Later, for the modeling, they used K-Nearest Neighbors (kNN), Naive Bayes (NB), Sequential Minimal Optimization (SMO), MLP, Random Forest (RF), C4.5, and Logistic Regression (LR) algorithms. The highest performance is 0.955, which is achieved with the SMO algorithm.

The Microsoft dataset is not only used for malware classification or as a stand-alone data set [186, 187, 188, 189, 190]. In an attempt to reduce
features, the researchers have used the data set generated by combining heartbeat and threat reports collected by Microsoft’s endpoint protection solution [191]. In another study, the authors have presented their work to classify a new malware family based on small historical samples using text classification methods [192]. The authors used the Microsoft data set, which inherits the properties of an imbalanced data set, to simulate two different scenarios. Their proposed method shows reasonable accuracy in detecting the new variant. However, their method is time-consuming.

9.2 Experimental Setup

9.2.1 Data set

Microsoft released the experimental data set for a challenge at Kaggle, titled "Microsoft Malware Classification Challenge"\(^5,6\). The data set contains nine different malware families: Ramnit, Lollipop, Kelihos_ver3, Vundo, Simda, Tracur, Kelihos_ver1, Obfuscator.ACY, and Gatak.

The data set contains both training and test data. The training data set consists of 186 GB of data. The training data set contains 10,868 files. The test data size is 189 GB, and contains 10873 examples. The released data set contains two types of disassembled files for each malware sample, i.e., byte code, which contains the hexadecimal dump (47.3 GB of train and test data respectively) of malware files and .asm files, which are disassembled files in assembly instructions form. The .asm files are obtained by using IDA Pro\(^7\). In this particular study, for the experimental purpose, the data set represented as .asm files is used because it is unsure how the hexadecimal files are generated. Further, the data set is modified by removing the obfuscated files and extended by adding 1,609 benign disassembled files (.asm). The benign files are downloaded from download.com\(^8\). The downloaded files are disassembled by using

---

\(^5\)https://www.kaggle.com/c/malware-classification
\(^6\)https://www.hex-rays.com/products/ida/
\(^7\)https://www.hex-rays.com/products/ida/
\(^8\)https://download.cnet.com/
IDA Pro to obtain .asm files. The purpose of including benign files is multi-fold. The first purpose is to change the multi-class classification problem to a binary classification problem between malware (all families) and benign, and the other fold is to recognize the exact family of the malware and distinct it from benign files. The modified data set inherited the class imbalance problem [176]. For example, the Ramnit class has only 1541 instances out of 10868. To avoid class imbalance, stratified random sampling with replacement is used. It is worth noting that the data set is available on request from the authors.

9.2.2 Feature Extraction

A .asm file is divided into different sections such as .data section, .text section, .idata section, etc. [176,193] Each section has particular features. However, the .text section is of particular interest. The .text section contains different types of valuable features/information, such as assembly instructions, hexadecimal code of each byte with memory segment, readable strings, API calls, DLL imports, headers information, etc. Previous studies are not conclusive about the best features. However, it is suggested that API calls, DLL imports, and assembly instructions are better features as an input for learning algorithms for the classification [194, 195, 196] compared to hexadecimal code and readable strings. Thus, three features, i.e., API Calls, DLL imports, and assembly instructions, are extracted to use as a feature by writing a parser in Python\(^9\) (and regex).

The extracted assembly instructions consist of two parts, i.e., OpCode and operands. The operands are architecture-dependent and cannot provide helpful information. One of the studies has suggested that the registers’ frequency of use may be used as a feature for malware detection [22]. However, the generalizability of the suggested approach is unsure. Thus, operands from the assembly instructions are discarded, and OpCodes are extracted in the form of uni-gram. The extracted OpCodes are saved in the order of occurrences. It is worth noting that the

---

\(^9\)www.python.org
extracted OpCodes, API Calls, and DLL imports are saved as separate feature sets in order of their occurrence.

9.2.3 Features

*N*-gram is a contiguous sequence of *n* items (e.g., words) (extracted from a data set), where *n* may range from one to *n* [34]. For experimental purposes, the extracted feature sets contain the OpCodes, API calls, and DLL imports in textual form, and each feature can be considered a word.

**OpCode** Each extracted OpCode may be considered a *uni*-gram, which may not provide helpful information about the file’s structure and functionality (purpose) [197]. Moreover, the extracted OpCode data set contains noise, i.e., unofficial or customized OpCodes. Thus, to remove the noise from extracted OpCodes, dictionary-based primary feature selection is applied (Please, refer to section 9.2.4). Further, to represent features in the input matrix, the *uni*-gram OpCodes are used to generate *quad*-grams and *variable-length* grams. In previous studies, *quad*-grams and *variable-length* grams have shown promising results compared to other sizes [197]. However, to the best of the authors’ knowledge, both *quad*-gram and *variable-length* *n*-grams have not been used together nor have their performances been compared.

**API and DLL** Application programming interface or Windows API provides an integration interface for users or software to communicate and integrate. A DLL file contains a program or a collection of programs, generally referred to as a library with relevant data or a data structure that other programs can use. For the malware classification task, it is suggested that API and DLL names are crucial features of the software, which may provide valuable information. However, for generating API and DLL *n*-grams, previous studies are not conclusive for

---

10https://docs.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

158
optimal size [51, 198] and have used them as a uni-gram. Thus, to find the optimal \(n\)-gram size, the combinational analysis of the API and DLL data sets is performed. The results have suggested that a bi-gram is an optimal size. There is an ignorable difference in the classification accuracy if tri-gram, quad-gram or larger \(n\)-grams are used as input compared to bi-gram. However, the computational cost is relatively high with each increasing size. Thus, the API and DLL data sets are used to generate API and DLL bi-grams as features for the input matrix.

**Algorithm 2 Proposed Backward Selection Algorithms**

```
procedure BACKWARDSELECTION(TrainSet, TargetClass, MinFeatures)
Require: MaximumAccuracy ← 0, BestFeatures ← ∅, Train ← ∅, Test ← ∅, Accuracy ← 0
repeat
    Train ← Split(TrainSet)  // Training set is split into 80/20 by using sampling with replacement
    Test ← Split(TrainSet)
    Accuracy ← CalculateAccuracy
    if Accuracy > MaximumAccuracy then
        MaximumAccuracy ← Accuracy
        Update(BestFeatures)
    end if
    FeaturesImportance ← CalculateFeaturesImportance(Train)
    TrainSet ← RemoveLeastImportantFeatures(TrainSet)
until NumberOfRemainingFeatures ≥ MinFeatures
return BestFeatures
end procedure
```

### 9.2.4 Feature Selection

Feature selection is generally considered a one-step problem. However, for this particular study, after a manual inspection of the data, feature selection is performed in two stages, i.e., primary and secondary fea-
ture selection. The primary feature selection is further divided into two stages, i.e., dictionary-based selection and frequency analysis. After the primary feature selection, each file’s representative features (Please, refer to section 9.2.3) are generated and used as input for the secondary feature selection. For the secondary feature selection, statistical measures and different machine learning based methods are used. The purpose of dividing one-step feature selection into two steps is to obtain the most valuable features. Additionally, each learning algorithm has its own inductive bias, meaning that the best features of each algorithm are different. Combining the best features obtained from different methods may help create a comprehensive input matrix and improve classifier performance. The feature selection process is explained in Figure 9.1.

**Primary Feature Selection**

The primary feature selection is performed to remove irregular and rarely used features from the extracted feature set and reduce the data set size significantly.

**Dictionary-based Selection** The 8086 architecture\(^{11}\) manuals by Intel and Microsoft websites are consulted to develop the dictionaries of regular OpCodes, API calls\(^{12}\). Intel’s architecture guideline contains 1094 regular OpCodes. After comparing OpCodes with the dictionary, many irregular OpCodes are found and discarded. A majority of irregular OpCodes are custom made OpCodes. After removing the irregular OpCodes, the reduced OpCode set contains standard 539 OpCodes per Intel’s guidelines. For the API calls and DLL imports, the total number of standard APIs is 8,068, and DLLs are 1,957. However, no non-standard API call or DLL import is found in the data set.

**Frequency Analysis** Frequency analysis is performed at two distinct stages. First, after the feature extraction, i.e., extracting OpCodes, APIs

\(^{11}\)https://www.intel.com/
\(^{12}\)https://docs.microsoft.com/
9.2. Experimental Setup

and DLLs from disassembled files. Secondly, after generating feature representation for the input matrix such as quad-gram. For the first stage, a frequency analysis of extracted features is performed to find the most commonly used features and remove rarely used features. For this purpose, a hash table is generated for every feature (OpCode, API, and DLL) for mapping their frequencies. For OpCodes, there is a significant decrease in frequency after the first 300 uni-grams. For the API features, the frequency analysis has suggested that after 1329th API, the frequency of API calls is 100. After 3,320th API, the frequency of API calls is ten or less than ten, and after 6,341th API, the frequency of API calls is only one. For DLLs, after the first 86 DLLs, the frequency count is 100, and after 434th DLL, the frequency of DLL import is ten or less than ten.

After generating feature representations, a secondary frequency analysis is performed to decide a threshold for generated representations, which suggests a frequency threshold of 50. Thus, all features with a frequency of less than 50 are discarded.

Secondary Feature Selection

For experimental purposes, this article focuses on three supervised feature selection strategies, i.e., filter, wrapper, and hybrid [199].

Filter method  Filter methods use uni-variate statistics to identify/measure the relevance of features by their correlation with the target variable without induction of a learning algorithm [199, 200]. To represent filter methods, Shannon Entropy is a widely used method for the feature selection [34]. Entropy is a measure of randomness in the feature’s possible outcome. Entropy is computed on all feature representations of each malware and benign sample.

Wrapper Method  Wrapper methods convert the feature selection problem into a search problem using a learning algorithm. Wrapper methods can have either a forward feature selection process or a backward feature
9. Hybrid Approach

selection/elimination [201, 202]. For this particular study, we propose a backward feature selection algorithm, as presented in Algorithm 1 and further explained below.

In the previous studies, the Random Forest (RF) algorithm has shown promising results for feature selection and malware classification [51]. Another algorithm, Regularized Greedy Forest (RGF) has shown promising results in different Kaggle’s competition [203]. Thus, we used both Random Forest and Regularized Greedy Forest algorithm\(^{13}\) as base algorithms for backward feature elimination. Both algorithms have used 10-fold cross-validation. For the hyperparameter tuning to improve ensemble performance, giving the manual value to both algorithms can be costly in terms of time and resources. Thus, an automated process is used to give a predefined list of parameter values, and the best parameters are selected and used for the feature selection. Finally, for the RF, 100 trees are generated without a depth limit. It is suggested that an RF with 100 trees may provide maximum accuracy [204]. If the number of trees is grown beyond this limit, they may increase the computation cost without significantly increasing accuracy. The Gini impurity is used for splitting the nodes, which is a criterion for calculating the information gain of a feature, and its value lies between 0 to 0.5 [34]. For RGF, a variation of RGF, i.e., RGF Sib algorithm\(^{14}\), which uses "minimum penalty regularization with the sum-to-zero sibling constraints" is used as a base algorithm. For the loss calculation, the square loss (LS) method is used, which can be calculated as follows:

\[
\text{SquareLoss} = \frac{(p - y)^2}{2} \tag{9.1}
\]

Moreover, a limitation of a maximum of 1000 leaf nodes is applied to the algorithm.

Embedded Method  
Embedded methods combine the features of filters and wrapper methods. To represent the embedded methods, Lasso

\(^{13}\)https://github.com/rgf-team/rgf

\(^{14}\)https://www.kaggle.com/carlmcbrideellis/introduction-to-the-regularized-greedy-forest
9.2. Experimental Setup

and eXtreme Gradient Boosting (XGBoost)\textsuperscript{15} algorithm is used. XGBoost is also a decision tree-based ensemble, which is designed for speed, flexibility and performance [205].

\textbf{Algorithm 3} Algorithms Fusion

\begin{verbatim}
procedure ALGORITHMSFUSION(ClassifiersSet, Instance, ClassifiersWeights)
Require: PredictionSet ← ⊘, NumberOfClassifiers ← NumberOfItemsInSet(ClassifiersSet), Probability ← 0, Outcome ← 0
repeat
    Probability ← ClassifiersSet(NumberOfClassifiers) ← Instance
    PredictionSet(NumberOfClassifiers) ← ClassifiersWeights(NumberOfClassifiers) × Probability
until NumberOfClassifiers > 0
Outcome ← \left(\prod_{i=1}^{NumberOfClassifiers} ClassifiersWeights_i\right)^{1/NumberOfClassifiers}
end procedure
\end{verbatim}

9.2.5 Feature Fusion

Feature fusion integrates multiple feature sets to obtain a single feature set, which can improve accuracy. Different feature fusion methods are suggested to stack all features in a single feature vector. Thus, to combine the best features from all representations, a union of all features subset is taken.

9.2.6 Algorithms

Two types of supervised algorithms are used for experimental purposes, i.e., base/basic algorithms and tree-based ensembles [206, 207]. The basic algorithms used in the experiment are Classification and Regression

\textsuperscript{15}https://github.com/dmlc/xgboost/
## 9. Hybrid Approach

### Table 9.1: Multi-class Experiment’s Results

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Features</th>
<th>Selected Features</th>
<th>Accuracy (%)</th>
<th>T. Time*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CART</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>95.76</td>
<td>0.015</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>96.28</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>97.24</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>94.76</td>
<td>0.3</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>99.1</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>96.98</td>
<td>2.33</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>99.45</td>
<td>3.5</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>99.4</td>
<td>2.7</td>
</tr>
<tr>
<td>SVM</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>98.11</td>
<td>89.4</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>95.32</td>
<td>421.38</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>98.19</td>
<td>300.5</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>97.2</td>
<td>248.13</td>
</tr>
<tr>
<td>kNN</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>96.4</td>
<td>18.55</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>92.33</td>
<td>60.87</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>96.8</td>
<td>44.49</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>97.79</td>
<td>50.4</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>94.14</td>
<td>25.25</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>91.2</td>
<td>123.15</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>95.53</td>
<td>43.14</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>97.7</td>
<td>52.16</td>
</tr>
<tr>
<td>Neural Network</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>98.2</td>
<td>3.15</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>96.12</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>99.68</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>99</td>
<td>7.25</td>
</tr>
<tr>
<td>Random Forest</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>97.3</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>95.43</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>99.72</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>98.9</td>
<td>0.1</td>
</tr>
<tr>
<td>Adaboost</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>43.43</td>
<td>8.44</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>34.56</td>
<td>24.1</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>95.45</td>
<td>10.98</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>90.2</td>
<td>18</td>
</tr>
<tr>
<td>XGBoost</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>97.3</td>
<td>15.25</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>95.43</td>
<td>40.56</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>99.5</td>
<td>21.34</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>98.9</td>
<td>30.28</td>
</tr>
<tr>
<td>Light Gradient Boosting</td>
<td>DLL bi-gram</td>
<td>1306</td>
<td>99.1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>API bi-gram</td>
<td>5371</td>
<td>98.4</td>
<td>22.4</td>
</tr>
<tr>
<td></td>
<td>variable-length Gram</td>
<td>2659</td>
<td>99.6</td>
<td>16.34</td>
</tr>
<tr>
<td></td>
<td>quad-gram</td>
<td>3466</td>
<td>99.4</td>
<td>19.19</td>
</tr>
</tbody>
</table>

*The given training time is in minutes.*
9.2. Experimental Setup

Trees (CART), which is a decision tree-based algorithm, Naive Bayes (NB) algorithm, which is based on Baye’s theorem and is used to generate a probabilistic classifier, Support Vector Machine (SVM) which is a robust learning algorithm based on Vapnik-Chervonenkis theory, Logistic Regression (LR) which is based on the concept of probability and uses a logistic function, k-Nearest Neighbors (kNN), which assumes that similar observations are present in close proximity and a basic neural network algorithm [34]. For the tree-based ensembles, Random Forests (RF), which is an ensemble of decision trees, decides about the class using majority voting; Boosting, which is a meta-learning ensemble and combines weak learners, are used [34]. CART is used as a base algorithm for RF (with 100 trees) and Bagging. Boosting can be either adaptive Boosting (AdaBoost) or Gradient Boosting (GBoost). Two variants of GBoost are eXtreme Gradient Boosting and Light Gradient Boosting Machine (LightGBM). The LightGBM differs from other GBoost algorithms in growing the tree criteria. Moreover, the configuration used in algorithms is as follows: The XGBoost and LGBoost are used with an estimator of 100, Logistic Regression is used with an alpha of 0.0001 and a limit of 10000 iterations; Neural Network is used with a single layer and a limit of 300 iterations; Random Forest is used with an estimator of 200. The pruning factor for CART is 0.001 and the number of neighbors for kNN is five. It is worth noting that the majority of these algorithms have used parallel processing on all cores.

Table 9.2: Results of Top 5 Classifiers

<table>
<thead>
<tr>
<th>Classifiers</th>
<th>Log Loss</th>
<th>Accuracy</th>
<th>AUC</th>
<th>Precision Micro</th>
<th>Precision Macro</th>
<th>Recall Micro</th>
<th>Recall Macro</th>
<th>F1 Micro</th>
<th>F1 Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGBoost</td>
<td>0.048</td>
<td>98.773 (0.05694)</td>
<td>0.987</td>
<td>0.988</td>
<td>0.924</td>
<td>0.988</td>
<td>0.975</td>
<td>0.988</td>
<td>0.932</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.075</td>
<td>98.59 (0.05475)</td>
<td>0.986</td>
<td>0.986</td>
<td>0.925</td>
<td>0.986</td>
<td>0.972</td>
<td>0.986</td>
<td>0.928</td>
</tr>
<tr>
<td>Light Gradient Boosting</td>
<td>0.075</td>
<td>98.736 (0.02856)</td>
<td>0.986</td>
<td>0.988</td>
<td>0.927</td>
<td>0.988</td>
<td>0.973</td>
<td>0.988</td>
<td>0.931</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.043</td>
<td>98.819 (0.05887)</td>
<td>0.992</td>
<td>0.989</td>
<td>0.927</td>
<td>0.989</td>
<td>0.985</td>
<td>0.989</td>
<td>0.938</td>
</tr>
<tr>
<td>Neural Network</td>
<td>0.075</td>
<td>98.736 (0.02856)</td>
<td>0.988</td>
<td>0.988</td>
<td>0.927</td>
<td>0.988</td>
<td>0.973</td>
<td>0.988</td>
<td>0.931</td>
</tr>
</tbody>
</table>

9.2.7 Algorithms Fusion

Generally, an ensemble is created by combining a finite set of learning algorithms. An ensemble may contain either homogeneous or heterogeneous algorithms, which may be trained on a subset of data, and their
predictions are used to determine the outcome of ensemble [34]. In our experiment, heterogeneous algorithms are used to generate an ensemble. Some of them are learning algorithms with their own biases, and some of them are ensemble in their nature. Thus, an algorithm fusion is proposed to combine decisions from various algorithms and determine the outcome, as shown in Algorithm 2. The suggested algorithm fusion is performed in different stages as follows:

- The weight for each classifier is calculated. For calculating the weight, the probability matrix of the test data set is used. The Sequential Least Squares Programming (SLSQP) algorithm, which is a sequential quadratic programming (SQP) algorithm\(^{16}\), is used to determine the weight within a range. The value of 0.5 is given as a minimal value. The probabilities table is multiplied by the selected weight. The log loss (Please, see the equation 9.2) of the resultant table is determined. This process is repeated by changing the weight suggested by the SLSQP method until the log loss no longer decreases. The last weight that affected the log loss is considered the final weight for the classifier. This process is repeated for each classifier.

\[
\text{LogLoss} = -\frac{1}{N} \sum_{i}^{M} \sum_{j}^{M} y_{ij} \log(p_{ij})
\]  

(9.2)

- When an instance is given to the ensemble for determining its class. Each classifier provides its probabilities for each class. Further, a two-dimensional array is generated, containing all classifiers in rows, and against each classifier, its probabilities for each class are saved in columns.

- The rows are multiplied with each classifier’s weight, generating a new result matrix.

- The geometric mean of each column is calculated, resulting in a single array with ten probabilities.

\(^{16}\)http://degenerateconic.com/slsqp.html
9.2. Experimental Setup

Figure 9.2: Confusion Matrix

- The maximum geometric mean is used as the outcome for the given instance.

9.2.8 Evaluation Measures

The performance of each learning algorithm is evaluated by performing 10-fold Cross-Validation. Moreover, confusion matrices are generated by using the responses from classifiers. The four parameters to generate a confusion matrix are as follows: True Positive (TP) represents the correctly identified malware programs and their families. False Positive (FP) represents the incorrectly classified benign programs. True Negative (TN) represents the correctly identified benign programs, and False Negative (FN) represents the incorrectly identified malware programs and their families. The performance of each classifier is evaluated using Detection Rate (DR), which is the ratio of malware programs correctly identified from the total number of malware programs; False Alarm Rate
(FAR), the ratio of malware programs incorrectly identified; Precision, which is a ratio of correct positive predictions to the total number of positive predictions and measure of quality (Please, see the equation 9.3), Recall, which is a measure of quantity and is ratio of identified positive over all the positives in the data set (Please, see the equation 9.4). For aggregated indicators, Accuracy (ACC), the percentage of correctly identified programs (Please, see the equation 9.5); and F-score (F1), which is a harmonic mean of Precision and Recall (Please, see the equation 9.6), are used. The last evaluation measure is the Area Under the Receiver Operating Characteristic Curve (AUC), a single-point value derived from an ROC curve. The higher AUC of an algorithm indicates that the algorithm is more robust and better in classification. Another evaluation measure is log loss [208]. Kaggle recommends log loss for comparing models’ performance. The log loss measures the uncertainty of the predicted probabilities by comparing them with the corrected probabilities of a given model. Thus, a lower log loss indicates a better performance by the model.

\[
Precision = \frac{TP}{TP + FP} \quad (9.3)
\]

\[
Recall = \frac{TP}{TP + FN} \quad (9.4)
\]

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (9.5)
\]

\[
F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (9.6)
\]

### 9.3 Experiments

Three different experiments are performed to determine the effectiveness of the suggested approach and its viability for machines with fewer computation resources. Each experimental result provides a base for the subsequent experiment. These experiments are performed on a machine with an i7-8700 Intel processor, 1 TB, hard disk with 7200 RPM, 16-GB RAM (DDR4-266 MHz), without a graphics processing unit (GPU),
and Windows 10 installed as OS. For the first experiment, the aim is to find the validity of the extended data set. For this experiment, the binary malware detection approach is used. All malware family feature sets are combined, generating one malware class feature set. The experimental results indicated that the extended data set can be used for multi-class classification. The second experiment is a multi-class classification experiment. This experiment aims to find the combination of the best classifier and best feature representation. A sub-aim is also to determine whether a feature fusion shall be performed. Thus, before performing feature fusion, individual feature sets are generated and fed into ten classifiers. The generated matrices contained all the malware families and benign files. It is hypothesized that each classifier will obtain a different accuracy on different feature representations. Thus, it will help to select five top classifiers to make their ensemble and also indicate the best feature sets to create their fusion. As hypothesized, the experiments resulted in different accuracies for different feature representations. However, there was no significant difference in accuracies on different feature sets. Therefore, for the next experiment, all feature sets were combined into a final matrix and fed into the top five selected algorithms. The outcome of each algorithm was used as input for a customized ensemble to determine the outcome.

9.4 Results and Discussion

Malware detection and classification is typically approached as a binary classification problem. The rationale behind this idea is that many types of malware share intrinsic characteristics and boundaries, and their functionalities may be blurred or overlapped. As a result, a single file may qualify for different classes. However, the first experimental results indicate that a multi-class approach is feasible for the extended data set. The table 9.1 presents experimental results for the multi-class approach, including their training time. The best classifiers are generated with all possible parameter combinations and saved as a dictionary of accuracy with the best parameters. The dictionary is later used to get the optimal
9. Hybrid Approach

Figure 9.3: Classifiers’ Accuracy

performances. The top five classifiers are selected based on training time and accuracy, with a tradeoff between the two. For example, SVM outperforms AdaBoost, kNN, and NB on the API bi-gram feature set, but its training time is significantly higher than other algorithms. SVM has shown similar behavior in training time for all other feature sets. Regardless of its accuracy, SVM cannot be in the top five due to its training time. For the feature sets, the var-gram features have provided the best results. However, all the feature representations have provided results distinct from each other with a small margin, which indicates that the feature fusion can be performed to create the input matrix. The table 9.2 presents the last experimental results. The results suggest that classifiers can achieve optimal performance when they use multi-feature sets. The Logistic Regression algorithm has the highest ACC at 98.819, while RF has the lowest ACC at 98.59. However, the difference between both accuracies is negligible. On the other hand, Logistics Regression has a log loss of 0.043, while RF has a log loss of 0.075, indicating that LR is a better classifier than RF.

These top five classifiers are further combined as base algorithms to form a customized ensemble. Similar to any other algorithm, the confusion matrix for the ensemble is also generated, which is presented in Figure 9.2. The ensemble outperforms all its base algorithms by achieving an
9.4. Results and Discussion

ACC of 99.72 and log loss of 0.01. The experimental results are presented in table 9.3. The accuracy comparison of the ensemble and its base algorithms is presented in Figure 9.3. The diagram indicates that the ensemble outperforms its base algorithms. To validate the performance of the proposed ensemble algorithm, the confidence of the ensemble and its base algorithms is calculated and presented in Figure 9.4. The confidence of an algorithm is its probability of its prediction being correct. The confidence of the algorithms is calculated over 200 GB of the test data set, using the standard deviation to evaluate confidence. A lower standard deviation indicates a more robust classifier with little or no risk of overfitting. The ensemble’s standard deviation is 0.00001

Although the experiments have shown optimal results, there are many challenges faced during the experiments. The primary challenge is the limited computational resources, which cannot be optimized for machine learning tasks and represent a novice user machine. Another challenge is that the data set contains encrypted files that may affect the algorithms’ accuracy and confidence in finding a generalizable approach.

Moreover, to determine the validity of our approach, we have compared the suggested approach with two different approaches as follows:

Table 9.3: Model Comparison

<table>
<thead>
<tr>
<th></th>
<th>Winner’s Model</th>
<th>Proposed Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>99.50%</td>
<td>99.72%</td>
</tr>
<tr>
<td>Log loss</td>
<td>0.002</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Comparison with the Competition’s Winner  The private and public ladder boards of the competition are consulted to find the competition’s winner. According to the private ladder board, the competition winner team, i.e., "say NOOOOO to overfitting"\textsuperscript{17} has reported 0.0023 and 0.0028 multiclass log loss. The winning team’s approach can be divided

\textsuperscript{17}https://github.com/xiaozhouwang/kaggle_Microsoft_Malware
9. **Hybrid Approach**

into two distinct parts, i.e., feature engineering and modeling. The usage of the XGBoost algorithm influences both parts. For feature engineering, three types of features are extracted, i.e., OpCode $n$-gram and their count, segment line count, and ASM file pixel intensity features. From the OpCode $n$-gram, they also generated bi-gram, tri-gram, and quad-gram. However, OpCode count based on frequency, segment count, and ASM file pixel intensity are novel features and provide the best results. For the feature selection, they used information gain, random forest, and XGBoost at different stages. For the modeling, authors have used different algorithms and techniques, i.e., Random Forest, Naive Bayes, Neural Network, Gradient Boosting, and semi-supervised learning. They also used cross-validation to select their model. For the computation resources, the winning team used Google Compute Engine (instance with 104G memory with 16 CPUs) for the hardware.

Unfortunately, due to the competition’s rules, only log loss is reported by the participants. Thus, it is hard to have a thorough comparison of the approaches. However, the approach can be compared for the data set used, features used, modeling, log loss, and computation resources. The main difference in the overall approach is that the winning team used encrypted and unencrypted samples, and the third feature, i.e., ASM pixel intensity, is used because of encrypted samples. However, they have not used benign examples, which may differ in structure. Thus, they have used the given nine classes for their data set. During the experiment, they used many hard coded hyper-parameters relevant to these nine classes. In comparison, the suggested approach has used ten different classes, and no hard coded hyper-parameters are used for experimentation. Moreover, the winning team has used features that may only be valuable for the competition. Thus, the generalization of the approach is questionable. The file size may affect the approach and change the OpCode or instruction count and other features. The other problem is that their approach may not help track a characteristic of the relevant file. Another issue is that their approach only provides structural information, which may not help for further analysis. In comparison, the suggested approach uses four features, i.e., quad-gram, var-gram,
API, and DLL. These features can be tracked back to the original files in the data set to understand the functionality of the file and generalize the approach. Moreover, the variable-length-grams can provide significant information about the functionality of the file. Other features, i.e., DLL and API, can also be traced back to the original files for further analysis. Moreover, the winning team has performed incremental feature selection at an increment of 1000. However, the proposed approach combines common feature selection measures with a customized Backward feature selection using a threshold. For the modeling part of the approach, the ensemble approach is suggested, and it has shown better results than the winning team’s reported results. Moreover, the winning team has relied both on supervised and semi-supervised learning algorithms. However, the proposed approach has relied on supervised learning algorithms. For the log loss, the winning team has shown a lower log loss as presented in table 9.3. However, the winning team has performed thousands of iterations to reduce the loss of the classifiers, while the proposed method performs moderate iterations due to resource constraints. For the last part, i.e., computational resources, the proposed approach is suitable for running and updating with minimum computational resources. In conclusion, the proposed approach can be generalized, while the winning team’s approach is competition-focused and cannot be generalized.

![Figure 9.4: Classifiers’ Confidence](image-url)
Comparison with Dataset’s Owners  The proposed model is compared against another state-of-the-art study, which has similar characteristics [22]. It is worth noting that one of the authors is part of the team that released the data set. The study can be compared against the used features, classification models, and computational complexity. For the features, the authors have used both hexadecimal and ASM based features. For the hex-dump, they have used uni-gram, which may not give any valuable information for the analysis. They are also hard to track for further analysis. They also used meta-data and string length information. These features may be helpful for a particular data set and may not be used for generalizing the approach. From the disassembled files, the authors have extracted meta-data, frequencies of different features, and the ratio of instructions as features. These features also represent a similar problem, i.e., they may vary with changes in the data set and can not generalize the approach. In contrast to these features, the proposed features in our approach may be tracked for further analysis and may not vary with the data set, especially variable-length n-grams. Moreover, these features capture a function; thus, they may also indicate the function’s objective. Moreover, the features used in the selected study may take more time to extract and require a lot of read/write operations. In contrast, the proposed method extracts the basic features once, and later extracted features are used to derive the required features for input. This also helps upgrade the input matrix in less time if a model needs to be retrained with more examples.

For the feature selection, authors have used a modified forward step-wise selection, which starts with zero features and gradually augments features set with features with minimum log loss from a particular category. This study performs feature selection step by step, i.e., primary and secondary feature selection. Later, in the secondary feature selection, this study also exploits the potential of all feature selection strategies and use methods, which are often favored in literature. This increased the feature selection time; however, it ensures that only valuable features are part of the input.
9.5 Conclusion

For the classification, both methods rely on an ensemble; the article used XGBoost with bagging, and this study uses an ensemble of different classifiers. This study also shows that XGBoost can be used without generating an ensemble with the suggested features and achieve good results.

9.5 Conclusion

Automatic malware detection and classification to respective malware families may help human experts save time and launch timely responses. This paper presents an automated malware detection method that classifies the detected malware to the respective malware family. For classification, static features are extracted from disassembled binary files. A customized feature selection method is suggested to reduce feature dimensionality before the feature fusion. The reduced data set is fed into different algorithms, including a customized ensemble algorithm based on algorithm fusion. The experimental results suggest that feature fusion and algorithm fusion may help in malware detection and family classification. In the future, we plan to investigate a feature fusion between the features from the hexadecimal data set and the disassembled data set. We also plan to increase the number of families and include encrypted samples.
Chapter 10

Android Malware Detection Using Feature Fusion and Artificial Data

Raja Khurram Shahzad

Abstract

For Android malware detection/classification, the anti-malware community has relied on traditional malware detection methods as a countermeasure. However, traditional detection methods are developed for detecting computer malware, which is different from Android malware in structure and characteristics. Thus, they may not be useful for Android malware detection. Moreover, the majority of suggested detection approaches may not be generalized and are incapable of detecting zero-day malware due to different reasons such as available data set with a specific set of examples. Thus, their detection accuracy may be questionable. To address this problem, this paper presents a malware classification approach with reliable detection accuracy and evaluates the approach using artificially generated examples. The suggested approach generates the signature profiles and behavior profiles of each application in the data set, which are further used as input for the classification
task. For improving the detection accuracy, feature fusion of features from filter and wrapper methods, and algorithm fusion are investigated. Without affecting the detection accuracy, the optimal balance between real world examples and synthetic examples is also investigated. The experimental results suggest that both AUC and F1 can be obtained up to 0.94 for both known and unknown malware using original and synthetic examples.

10.1 Introduction

The malicious software (malware) authors are attracted to Android operating system (OS) based devices due to different factors such as their popularity\(^1\) and absence of review procedures for submitted applications in Google Play\(^2\) (official Android applications market) and other unofficial Android application (app) markets such as Onepf\(^3\). The malware authors either release malicious applications on unofficial app markets or infect the existing apps on Google Play [209, 210]. It is reported by Symantec Corporation\(^4\) that 27,000 new variants of Android malware are released in 2017, which is 54% increase in comparison to 2016. These malicious apps pose a threat to the privacy of users’ data by stealing personal and device information, such as IMEI number, calls log, and SMS log and may also cause financial losses [211]. To address this problem, Google has adopted reactive approaches such as relying on users’ feedback as a countermeasure and has also introduced a security service called Bouncer\(^5\) [212]. However, these solutions are incapable of detecting malware on Android devices. Moreover, due to monetary benefits for malware authors, the number of Android malware in the wild is increasing [213].

\(^1\)https://www.statista.com/statistics/216420/global-market-share-forecast-of-smartphone-operating-systems/
\(^2\)https://play.google.com/store
\(^3\)http://www.onepf.org/appstores/
\(^5\)http://blog.trendmicro.com/trendlabs-security-intelligence/a-look-at-google-bouncer/
10.1. Introduction

For android malware detection, researchers have used both static and dynamic detection methods. Static methods are used to analyze an application’s code without running it. While for the dynamic methods, the application’s code is executed in a virtual environment, i.e., an emulator, and its interaction with the system is investigated. There are some serious drawbacks with the dynamic approach, such as during the execution of the code, a malicious path may be ignored. Moreover, the malware may also detect the presence of emulation and can evade it [218]. The cost of deployment of the dynamic analysis is also high, and it is hard to automate the dynamic analysis as manual efforts are also required [219]. Thus, most researchers focused on the static methods for Android malware detection [211]. For the static approach, researchers have extracted different information from two parts of an app, i.e., AndroidManifest.xml (manifest) file and classes.dex (dex) file. The researchers have used permissions [220, 221], or intents [219], or listed hardware components [69] from manifest file. The dex file contains the executable code; thus, researchers have extracted information in the form of assembly language, i.e., SMALI [62] or decompiled the file into Java language (language for developing apps) for extracting the API calls or class names [222]. The extracted information is further used for different experimental purposes, such as to generate signature or control flow graphs. However, the majority of researchers have performed their experiments on a specific type of information either from manifest file or from dex file. Only a few researchers have investigated the combination of different information from manifest file or dex file or both. To find the valuable indicators (features) from the extracted information, researchers have investigated the usage of feature selection and feature ranking [217]. However, as per the best of our knowledge, no author has investigated feature fusion. In addition, the majority of experiments are performed on old and already available data sets or only on malware data sets. Moreover, these techniques may have a higher false detection rate for smaller files and zero day malware.

This paper presents a static Android malware detection approach based on supervised classification learning. For the experimental purpose, benign and malware data sets are collected. Further, this paper proposes automated means for extracting different information such as system’s permissions and intent filters from manifest file and mnemonics from dex file as features. The feature extraction is performed without deobfuscation and unpacking. Although it is generally considered that unpacking may provide more valuable features. However, the cost of unpacking is quite high both for known and customized packers. Further, from the extracted features, the valuable features are selected using different feature selection methods, and feature fusion is performed to generate data sets. The selected features are further used to generate signature or behavior profiles of applications. Moreover, it is practically infeasible to collect and represent all malware families examples in the wild. Thus, the Synthetic Minority Over-sampling Technique (SMOTE) [223] is used to generate artificial malware examples to represent missing families. In addition, the optimal ratio of original samples and artificially generated samples in a data set without affecting the detection accuracy is investigated. The experimental results suggest that the proposed approach is feasible for detecting known, as well as previously unseen Android malicious applications.

10.1.1 Paper Organization

The remainder of the paper is organized as follows: Section 10.2 discusses the background Section 10.3 presents theoretical background of the methodology. Section 10.4 describes the empirical procedure and Section 10.5 discusses the empirical results and also presents their analysis, and finally Section 10.6 concludes the work and describes future directions.
10.2 Background

Since 2000, the malware authors have shifted their focus from traditional malware (which appeared in the wild previously) to potentially unwanted programs (PUP), which are mainly dangerous for the privacy of the users. There is no consensus on the definition of PUP. However, a comprehensive definition of PUP is [224]:

*A piece of software that deceives the user, mainly for commercial purposes, and that may negatively affect the user’s security and/or privacy.*

Android malware contains characteristics similar to PUP as they are not destructive in their nature. Thus, due to functional similarities, Android malware complements the PUP according to the definition given above. Similar to the PUP, an Android application is downloaded as a single file, which is referred to as an Android Application Package (APK). The *apk* file contains a hierarchy for information sources\(^7\). The most important sources are as follows:

- **AndroidManifest.xml**: The android *manifest* file contains information about package name, version, referenced library files, *app* components, i.e., activities, services, content providers and broadcast receivers, and permissions. It is worth noting that the Android security model relies on the permission declaration access control mechanisms for accessing different functionalities of the device. It is difficult for a novice user to comprehend the actual intentions of the applications by looking at the system’s permissions asked from the application. Thus, it opens up possibilities of deceptive behavior.

- **Classes.dex**: A Dalvik bytecode file. Dalvik is a process virtual machine and an integral part of Android OS. The *dex* file contains all the compiled classes of a particular application.

10. Android Malware

Figure 10.1: The architecture of Android Malware Detection

- Resource files and folders that contain resources. The most common resource folders are `lib`, `assets`, and `res` folders. The resources can be libraries, graphics, etc.

### 10.3 Theoretical Explanation

#### 10.3.1 Data Set Generation

**Benign Data Set from Google Play**

The benign apps are downloaded from Google Play. However, downloading an `apk` file from Google Play contains many challenges, such as Navigation, Application Selection, Enabling Download Button, and Scrolling.

- **Navigation** The navigation in Google Play requires simulating human behavior for selecting a category, selecting an application, and downloading an application from the application’s official page.
10.3. Theoretical Explanation

- **Application Selection** Two different types of applications are available on Google Play, i.e., freeware/shareware and paid applications. The challenge is to determine the type of a particular application, i.e., freeware or paid, before downloading the application.

- **Download Button** When an application’s page is opened to download the application, the page provides a button to download and install the application. The challenge is to enable the download button and simulate human behavior to click the enabled button.

- **Scrolling** Google Play loads the top 60 applications on one page/screen. The next 60 applications are automatically displayed.loaded when a user reaches the end of the page. It is challenging to determine the end of a page.

For addressing these challenges, an application is written, which uses both Selenium⁸ and a browser add-ons APKDownloader⁹. Selenium is a web browser automation software, and APKDownloader is used to enable the download button.

<table>
<thead>
<tr>
<th>No.</th>
<th>Category</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Business</td>
<td>226</td>
</tr>
<tr>
<td>2</td>
<td>Comics</td>
<td>421</td>
</tr>
<tr>
<td>3</td>
<td>Education</td>
<td>461</td>
</tr>
<tr>
<td>4</td>
<td>Entertainment</td>
<td>408</td>
</tr>
<tr>
<td>5</td>
<td>Family</td>
<td>165</td>
</tr>
<tr>
<td>6</td>
<td>Libraries</td>
<td>882</td>
</tr>
<tr>
<td>7</td>
<td>Life Style</td>
<td>397</td>
</tr>
<tr>
<td>8</td>
<td>Medical</td>
<td>357</td>
</tr>
<tr>
<td>9</td>
<td>Music</td>
<td>206</td>
</tr>
<tr>
<td>10</td>
<td>Photography</td>
<td>151</td>
</tr>
</tbody>
</table>

⁸http://www.seleniumhq.org/
10. Android Malware

Malware Data Set

The Android apps with malware samples (malware apps) are downloaded from SandDroid\(^\text{10}\). SandDroid is an online Android applications analysis system, which regularly receives infected apps for analysis. For downloading malware samples from SandDroid, an application similar to the application for Google Play is written.

Reverse Engineering of Android Applications

For classifying an app or \textit{apk} file as either malicious or benign, the information from \textit{manifest} and \textit{dex} file is of interest. An open source tool, i.e., APKTool\(^\text{11}\) is used to extract the \textit{manifest} and \textit{dex} files from an \textit{apk} file. Further, another tool, SMALI/bakSMALI\(^\text{12}\), which is a pair of assembler and disassembler, is used to obtain SMALI file/s from a \textit{dex} file. A SMALI file contains the instructions in an assembly language, which is specifically developed for the Android OS\(^\text{13}\). Each Dalvik file contains multiple Java classes, which results in a variable number of SMALI files.

Artificial Data Generation

Synthetic Minority Over-sampling Technique (SMOTE) is used to generate the artificial malware data\(^\text{223}\). SMOTE over-samples the minority class by generating synthetic examples. Synthetic samples are introduced along the line segment of randomly selected \(k\) nearest neighbors and are added to the feature vector under consideration. The synthetic data helps the classifier grow a general decision region, which helps the classifier generalize better. By using SMOTE, over-fitting and similar problems, such as broadening the decision region of the minority class, may addressed.

\(^{10}\)http://sanddroid.xjtu.edu.cn/
\(^{11}\)https://ibotpeaches.github.io/Apktool/
\(^{12}\)https://github.com/JesusFreke/SMALI
\(^{13}\)http://bakSMALI.com
10.3.2 Machine Learning Approach

Feature Selection

Feature selection is a process of selecting a relevant subset of \( M \) features from the set of \( N \) features to optimally reduce the feature space. The feature selection can be done either by applying filter methods or by applying wrapper methods [34]. The filter methods such as Information Gain (IG) or Chi Square \( \chi^2 \) use statistical measures to evaluate the (statistical) importance of a feature in its class or to find the correlation of a feature with its class [33]. Information Gain calculates the relevance of a feature or the information provided by the feature as follows:

\[
Entropy(N) = \sum_{i=1}^{k} P_i \log_k \left( \frac{1}{P_i} \right) = - \sum_{i=1}^{k} P_i \log_k P_i \quad (10.1)
\]

\[
Entropy(D_j) = \frac{|(D_j)|}{N} \times Entropy(D_{ji}) \quad (10.2)
\]

\[
IG(D)_j = Entropy(N) - Entropy(D) \quad (10.3)
\]

The Equations (10.1) - (10.3) are steps to calculate IG. The Equation 10.1 is used to calculate the information of all classes. On the other hand, \( \chi^2 \) is applied to determine the existence of a relation between a feature and its class and can be calculated by using Equation (10.4), as follows:

\[
\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \quad (10.4)
\]

where \( O_{ij} \) is the observed frequency and \( E_{ij} \) is the expected frequency. Filter methods are considered fast and are applied to high-dimensional data. However, their results may vary due to the measured information of the features. On the contrary, the wrapper method uses a learning algorithm to select relevant features [225]. Thus, they are slow in comparison to filter methods and cannot be applied to a wide feature set. However, their results are better in terms of classification accuracy as each feature is judged with respect to the learning. However, their results may be biased by the learning algorithm.
Feature Fusion

Feature fusion is defined as the process of combining at least two feature vectors to obtain a more discriminative feature vector. For obtaining preliminary feature sets, two filter methods, i.e., IG and $\chi^2$, are applied. Later, to effectively combine the preliminary feature sets, which are obtained after applying filter methods, i.e., IG and $\chi^2$, a union (OR / $\cup$) of these feature vectors is taken. Further, a wrapper method is applied with the J48 algorithm. The best search method with forward searching is used to search the features in search space [34]. The forward search method starts with an empty set of features. The F-score, which is capable of calculating the discriminative ability of each feature, is used as the evaluation measure. The feature fusion results in a feature vector that may result in a higher malware classification accuracy.

Classification Algorithms

The learning algorithms can be categorized according to their inductive or learning biases, i.e., their assumptions for the search space and its traversing. Further, different learning biases, such as maximum conditional independence, maximum margin, and nearest neighbors can be used to identify different families of learning algorithms. tree inducers, rule set inducers, and instance based learners. To exploit the benefits of different learning biases, this study uses one algorithm from each family. Thus, to represent the decision tree family, the J48 algorithm, which is an implementation of the C4.5 algorithm, is used [111], and to represent the rule based family, the JRip algorithm is used [86]. Similarly, to represent the support vector machine (maximum margin), the SMO algorithm is used [112], and to represent the maximum conditional independence, the Bayesian theorem based algorithm Naive Bayes is used [226]. To represent the nearest neighbors concept for the classification, the IBk algorithm is used [227]. Finally, ZeroR algorithm, which predicts the majority category, is used as a baseline (base classifier) to compare the performances of all algorithms [34].
10.3.3 Evaluation Criteria

The learning algorithms are evaluated by using a 10-fold cross-validation. Subsequently, confusion matrices are generated from the responses of the algorithms. The main elements of confusion matrices are True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN). TP represents the correctly identified malware apps, FP represents the incorrectly classified benign apps, TN represents the correctly identified benign apps, and FN represents the incorrectly classified Android malware apps. Additionally, the Detection Rate (DR) was used to evaluate the performance of each classifier. Detection Rate (DR) is the percentage of correctly identified malware apps, as shown in Equation (10.5) [129]. False Negative Rate is the percentage of wrongly detected malware apps (see Equation (10.6)), and Accuracy (ACC) is the percentage of correctly classified apps (see Equation (10.7)).

\[
DetectionRate = \frac{TP}{TP + FN} \tag{10.5}
\]

\[
FalseNegativeRate = \frac{FN}{TP + FN} \tag{10.6}
\]

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10.7}
\]

ACC measure is not used as it may be an unreliable evaluation measure due to factors like class imbalance [34]. However, to complement ACC, another evaluation measure, Area Under Receiver Operating Characteristic Curve (AUC), is used, which is a single-point value derived from a ROC curve [34]. In this regard, DR is drawn on the x-axis as a function of the False Positive Rate (FPR) on the y-axis at different points. The higher AUC of an algorithm indicates its robustness and better classification ability. The other measures used are Recall (R), which is the ratio of correctly predicted malware apps to the total number of malware apps, and Precision (P), the ratio of correctly detected malware apps to the total number of apps identified as malware apps. F-Measure (F1) is the harmonic mean of the precision and the recall and is the final evaluation.
measure (see Equation (10.8)).

\[ F1 = 2 \cdot \frac{P \cdot R}{P + R} \]  

(10.8)

10.4 Experiment

The aim of experiments is to classify benign apps and malware apps by generating the signature and behavior profiles and using the artificial data set. The suggested process is divided into three distinct parts, as shown in Figure (10.1). The first part is reverse engineering, while the second part is information processing, and the third part is classification. For reverse engineering, Android applications are decompiled and disassembled. The generated output, which contains information such as permissions from manifest file and assembly instructions from dex file, are saved in a readable text format. Further, the extracted information is used to generate required features, i.e., \( n \)-grams, which is a specific size sequence of information. Further, these \( n \)-grams are used to generate both signature and behavior profiles. The generated profiles are in the text format; thus, we argue that text categorization techniques can be applied for the data pre-processing and to train a classifier. Thus, the generated profiles are used to generate Attribute-Relation File Format (ARFF) files [34]. The ARFF files are further used as input to the Waikato Environment for Knowledge Analysis (Weka) [98]. Weka is an open-source suite of machine learning algorithms and analysis tools. Weka is also used to apply different statistical measures to calculate the significance of each \( n \)-gram in a profile with respect to the classification task. Later, a reduced data set with valuable features is saved, which is further fed into learning algorithms to generate the classifiers.

10.4.1 Data Set

For generating an experimental data set, 7,000 files are used. Out of these, approximately 3,500 files are malware apps, and approximately 3,500 files are benign apps. The benign applications are scanned using
Table 10.2: Experimental Results

<table>
<thead>
<tr>
<th></th>
<th>TPR</th>
<th>FPR</th>
<th>AUC</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Manifest</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bi-gram</td>
<td>0.919(0.087)</td>
<td>0.078(0.084)</td>
<td>0.921(0.085)</td>
<td>0.927(0.079)</td>
<td>0.919(0.087)</td>
<td>0.918(0.089)</td>
</tr>
<tr>
<td>Quad-gram</td>
<td>0.926(0.077)</td>
<td>0.073(0.078)</td>
<td>0.929(0.074)</td>
<td>0.937(0.064)</td>
<td>0.926(0.077)</td>
<td>0.925(0.079)</td>
</tr>
<tr>
<td><strong>OpCode</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bi-gram</td>
<td>0.892(0.089)</td>
<td>0.115(0.112)</td>
<td>0.889(0.096)</td>
<td>0.912(0.076)</td>
<td>0.892(0.089)</td>
<td>0.890(0.091)</td>
</tr>
<tr>
<td>Quad-gram</td>
<td>0.918(0.073)</td>
<td>0.077(0.070)</td>
<td>0.921(0.071)</td>
<td>0.931(0.061)</td>
<td>0.918(0.073)</td>
<td>0.917(0.074)</td>
</tr>
<tr>
<td>Var-gram</td>
<td>0.926(0.089)</td>
<td>0.072(0.085)</td>
<td>0.927(0.087)</td>
<td>0.933(0.080)</td>
<td>0.926(0.089)</td>
<td>0.925(0.091)</td>
</tr>
<tr>
<td><strong>Hybrid</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bi-gram</td>
<td>0.934(0.085)</td>
<td>0.065(0.082)</td>
<td>0.934(0.084)</td>
<td>0.937(0.079)</td>
<td>0.934(0.085)</td>
<td>0.933(0.087)</td>
</tr>
<tr>
<td>Quad-gram</td>
<td>0.937(0.087)</td>
<td>0.082(0.089)</td>
<td>0.929(0.068)</td>
<td>0.941(0.081)</td>
<td>0.940(0.048)</td>
<td>0.938(0.051)</td>
</tr>
<tr>
<td>Var-gram</td>
<td>0.940(0.048)</td>
<td>0.062(0.084)</td>
<td>0.938(0.086)</td>
<td>0.947(0.041)</td>
<td>0.937(0.087)</td>
<td>0.937(0.089)</td>
</tr>
</tbody>
</table>
VirusTotal\textsuperscript{14} to eliminate the probability of known malware. The malware apps are also tested using VirusTotal to remove the probability of duplicate malware instances. Further, an application is written to reverse engineer the apps using the tools and procedure mentioned in Section 10.3.1.

10.4.2 Profiles Generation

From each reverse engineered application, two text files with different information are generated. Firstly, from the manifest files, the system’s permissions, intent filter actions, and intent filter categories are extracted, and the output is saved as the first text file, where each line contains one piece of information only. Secondly, from SMALI files, assembly instruction sequences are extracted, where each assembly instruction consists of mnemonic and operands. From extracted sequences, only the sequence of mnemonics may provide semantic information about the application. Thus, only mnemonic from each instruction is extracted, and the remainder of the instruction is discarded. The extracted mnemonics are saved in the order of their appearance as the second text file, where each line contains only one mnemonic. Further, a parser is written, which generates two types of profiles, i.e., signature profiles and behavior profiles, using text files. Both types of profiles can be further divided into distinct profiles on the basis of information used, i.e., from manifest file or from SMALI files or hybrid and n-gram size. These text files are further used Further, a parser is written, which generates three types of profiles using text files, i.e., manifest profiles, which contain information from manifest file only, SMALI profiles, which contain information from SMALI files, and hybrid profiles, which contain information from both manifest file and SMALI files. In the previous research for malware and PUP classification, it is indicated that bi-grams and quad-grams are suitable for generating signature profiles and achieving good results [129,197]. However, for the Android applications, it is not certain which size is more appropriate. Moreover, variable-length (var) grams are suggested for capturing the behavior of an application. Thus, the

\textsuperscript{14}https://www.virustotal.com/
detail of the generated profiles is as follows: Thus, each type of profile is further used to generate signature profiles using fixed size grams, i.e., Bi-gram, Quad-gram, and behavior profiles using variable-length grams. The detail of these sub-types is as follows:

- Signature profiles are generated using both bi-gram and quad-gram sizes. For each size, three different profiles are generated from manifest file or SMALI files or hybrid, i.e., both manifest and SMALI. The first bi-gram or quad-gram profile consists of adjacent permissions and intents. The second bi-gram or quad-gram profile contains adjacent mnemonics of respective n-gram size. The third bi-grams or quad-grams profile can be logically partitioned into two parts, i.e., the first part contains either bi-grams or quad-grams of permissions and intents, while in the second part, either bi-grams or quad-grams of mnemonics is present respectively.

- Behavior profiles are generated with variable length n-grams. It is worth noting that information from SMALI files is not converted into var-gram and is present in their original form. However, the information from SMALI files is used to generate var-gram. The generation of a var-gram starts with the start of a method or after a control transfer statement appears and stops when either the end of the method or another control transfer mnemonic appears.

It is worth noting that each profile has the same name as the application, so the file may be tracked back. Moreover, eight different profiles of each application are generated.

10.4.3 Optimizing Profiles

The generated profiles consist of millions of features (n-grams). However, all the features may not play an important role in the classification task. Thus, feature selection methods and feature fusion methods, as described in Section 10.3 are applied to select valuable features and obtain optimized profiles.
10.4.4 Algorithms Fusion

All the algorithms were used at their default configuration in Weka. Further, to take the benefit of each algorithm’s bias and improve the detection accuracy, a fusion of these classifiers with majority voting is generated. For the algorithm fusion, all the prediction results vote to generate a final prediction. It is expected that this algorithm fusion will outperform the base classifier in terms of prediction accuracy.

10.4.5 Experimental Settings

Experimental Settings I

The aim of the first experiment is three-fold. The first fold of the experiment is to determine which features The other fold is to evaluate the impact of future fusion methods, and the third fold is to evaluate the impact of algorithm fusion for Android malware detection tasks on a data set with real examples. It is worth noting that there is no class imbalance in the data set, and both classes are represented equally. The selected experimental results of the first experiment are presented in Table 10.2.

Experimental Settings II

The aim of the second experiment is to assess the impact of artificially generated data sets on the accuracy of Android malware classification. For this experiment, an imbalanced data set of all benign files and 50% randomly selected malware files is created. For this particular experiment, the feature fusion is not performed. Thus, data sets generated with $\chi^2$ and IG feature selection methods are used. For the experimental purpose, the synthetic examples are generated with the steps of 25%, 50%, 75%, and 100% increase, which have resulted in four data sets. The algorithm fusion is performed similar to Experiment I. The experimental results for Quad-gram OpCode and hybrid var-gram are presented in Table 10.3.
10.5 Results and Analysis

10.5.1 Results

The selected results of both experiments are presented in Table 10.2 and Table 10.3. In Table 10.2, each name of a data set represents the feature extraction method applied for generating that particular data set. For Table 10.3, each name of a data set represents the feature selection method applied for generating that particular data set. For example, $\chi^2$-Data set is obtained after applying $\chi^2$ feature selection method, and data sets starting with the word SMOTE represent the data sets, which are obtained after two steps. In the first step, a feature selection method is applied, and in the second step, SMOTE is applied. The numbers after the word SMOTE, i.e., 25 or 75, represent the percentage of artificial examples in the data set. Each data set is divided into the training and test sets iteratively using 10-fold cross validation. Both tables present the average results of both classes, i.e., benign and malware, for each performance evaluation criterion.

The experimental results indicate that the proposed methodology is capable of distinguishing between malware and benign files with a low
FPR for both classes. It is worth noting that TPR and FPR may indicate the cost of misclassification. However, the cost may vary according to the usage of the suggested technique. For an individual user, detecting malware is more important than misclassifying a benign application. On the other hand, for commercial purposes, the correct classification of benign applications along with minimized FPR (malware classified as benign) may be more important. On the original data sets, all the algorithms, including the ensemble, performed better than the base classifier, and the highest TPR and Precision are 0.940 and 0.947. The composite measures, i.e., AUC and F1, are 0.938 and 0.937, respectively. However, there is an imbalance of benign and malware applications in the wild. The number of benign applications available on Google Play and other Android markets may be higher than the number of malware applications. Moreover, malware can be divided into different families, and it is practically impossible to gather representative samples of all known and unknown families and perform an experiment. Thus, in the second experiment, the problem of unknown examples is addressed using SMOTE. The experimental results suggest that an appropriate ratio of synthetic examples in the data set may also help to detect unknown malware. It is observed that the performance of a classifier may not significantly improve with the increased number of synthetic examples, and a minimum addition of synthetic examples, i.e., 25%, has helped to achieve the best results. Thus, the highest TPR and Precision is achieved on the SMOTE-25 Data set for \textit{var}-gram, which is similar to TPR and Precision on the same data set with original examples, i.e., 0.940 and 0.947, respectively. However, the composite measures, i.e., AUC and F1, are 0.929 and 0.938, respectively.

The experiments are analyzed considering multiple factors such as processing overhead, dangerous features, etc. The processing overhead is highly dependent upon multiple factors, i.e., disassembly program, extraction routines, file size, and length of pair. The disassembly programs have their limitations. However, they are capable of providing the structural fingerprints of an \textit{app} in the form of OpCodes. Malware authors generally disassemble an \textit{app} and embed malicious functionality in the
Table 10.3: Experimental results with artificially generated data sets

<table>
<thead>
<tr>
<th>Quad-gram OpCode</th>
<th>TPR</th>
<th>FPR</th>
<th>AUC</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>IG</td>
<td>0.919(0.087)</td>
<td>0.078(0.084)</td>
<td>0.921(0.085)</td>
<td>0.927(0.079)</td>
<td>0.919(0.087)</td>
<td>0.918(0.089)</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>0.937(0.087)</td>
<td>0.062(0.084)</td>
<td>0.938(0.086)</td>
<td>0.941(0.081)</td>
<td>0.937(0.087)</td>
<td>0.937(0.089)</td>
</tr>
<tr>
<td>SMOTE-25</td>
<td>0.892(0.089)</td>
<td>0.115(0.112)</td>
<td>0.889(0.096)</td>
<td>0.912(0.076)</td>
<td>0.892(0.089)</td>
<td>0.890(0.091)</td>
</tr>
<tr>
<td>SMOTE-50 Data Set</td>
<td>0.912(0.072)</td>
<td>0.076(0.063)</td>
<td>0.920(0.065)</td>
<td>0.932(0.052)</td>
<td>0.912(0.072)</td>
<td>0.911(0.073)</td>
</tr>
<tr>
<td>SMOTE-75</td>
<td>0.918(0.073)</td>
<td>0.077(0.070)</td>
<td>0.921(0.071)</td>
<td>0.931(0.061)</td>
<td>0.918(0.073)</td>
<td>0.917(0.074)</td>
</tr>
<tr>
<td>SMOTE-100 Data Set</td>
<td>0.900(0.068)</td>
<td>0.097(0.066)</td>
<td>0.916(0.069)</td>
<td>0.914(0.060)</td>
<td>0.900(0.068)</td>
<td>0.899(0.069)</td>
</tr>
<tr>
<td>Var-gram Hybrid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IG</td>
<td>0.926(0.089)</td>
<td>0.072(0.085)</td>
<td>0.927(0.087)</td>
<td>0.933(0.080)</td>
<td>0.926(0.089)</td>
<td>0.925(0.091)</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>0.934(0.085)</td>
<td>0.065(0.082)</td>
<td>0.934(0.084)</td>
<td>0.937(0.079)</td>
<td>0.934(0.085)</td>
<td>0.933(0.087)</td>
</tr>
<tr>
<td>SMOTE-25</td>
<td>0.940(0.048)</td>
<td>0.082(0.089)</td>
<td>0.929(0.068)</td>
<td>0.947(0.041)</td>
<td>0.940(0.048)</td>
<td>0.938(0.051)</td>
</tr>
<tr>
<td>SMOTE-50 Data Set</td>
<td>0.895(0.075)</td>
<td>0.110(0.070)</td>
<td>0.892(0.070)</td>
<td>0.913(0.062)</td>
<td>0.895(0.075)</td>
<td>0.893(0.076)</td>
</tr>
<tr>
<td>SMOTE-75</td>
<td>0.926(0.077)</td>
<td>0.073(0.078)</td>
<td>0.929(0.074)</td>
<td>0.937(0.064)</td>
<td>0.926(0.077)</td>
<td>0.925(0.079)</td>
</tr>
<tr>
<td>SMOTE-100 Data Set</td>
<td>0.905(0.103)</td>
<td>0.093(0.100)</td>
<td>0.906(0.102)</td>
<td>0.915(0.095)</td>
<td>0.905(0.103)</td>
<td>0.904(0.105)</td>
</tr>
</tbody>
</table>
form of assembly instructions. Afterward, the app is recompiled and made available for the public. Thus, a disassembler can be used to get both the original functionality and added malicious functionality of the app. The other factor is extraction routines. The speed of extraction routines is highly dependent upon the number of classes in the dex file, which is indirectly related to the file size. If the file size is large, it is generally expected that it may have more classes. Another important factor in the experiments is the size of $n$-gram, i.e., the number of OpCodes in a $n$-gram. The extraction time and number of $n$-grams increases with the increase in the size. However, this is not the truth for the behavior profiles.

The optimized profiles, generated rule based, and tree based classifiers are analyzed to find the dangerous/valuable features. The analysis indicates that in malware, the frequency of some specific features, such as jump instruction or specific permission, is higher in terms of the frequency in comparison to benign files. Some of the selected features are discussed as follows:

- The first most frequently occurring feature is permission, i.e., READ_PHONE_STATE. This permission allows an application to read the unique ID, i.e., the IMEI number of the mobile phone. This ID is generally used by ad publishers for tracking purposes.

- Another feature is RECEIVE_BOOT_COMPLETED, which is generally used to start a particular service after the device is booted.

- The other most occurring permissions are related to installing a package and sending and receiving SMS from the device. These services can be used to send SMS to a premium number to cause financial losses to the user.

- An OpCode, i.e., packed-switch is considered a malware indication by both feature selection methods and rule based and tree based classifiers. This particular OpCode transfers the control from one place to another within the program. This may be used to execute
10.5. Results and Analysis

an embedded malicious functionality during the normal flow of an app.

For the second experiment, signature, and behavior profiles are generated with the artificial data generated by SMOTE. SMOTE is criticized by the researchers for over-fitting or over generalization. To address this, different adaptive methods have been suggested. However, for this experiment, this aspect is not addressed. For this particular experiment, malware is considered a minority class, and the five nearest neighbors are used to generate a new example. Generally, it is considered that data sets prepared with the oversampling will produce better classification results in comparison to the original data set. However, for malware detection, the artificially generated data set was unable to show a significant difference. The experimental results helped in finding the optimal balance between original malware examples and artificially generated malware examples in the data set without affecting the DR. Another problem associated with malware detection is the presence of small disjuncts, i.e., the presence of sub-families in a particular malware family. This problem increases the complexity to interpret and rely on the classification results. It is difficult to evaluate whether a particular example is representative of its entire family or a sub-family or, just a variant of a particular malware or a representative of a missing family. Few artificially generated examples were analyzed in terms of functionality, and it was concluded that they mainly represent the inter-family families, which are not present in the original data set. However, some artificial examples may also be considered variants of a particular malware or representative of a sub-family. Thus, artificially generated malware examples may help in detecting both variants of a particular malware and unseen or zero-day malware. It is worth noting that to obtain inter-family families or similar examples, pruning is not used. The pruning helps the algorithms to derive more generalized rules and may eliminate the small disjunct. While, for the malware classification, small disjuncts may represent different families; thus, rules for them are also required. Moreover, it is also observed that classification performance
is degraded as overlapping occurs between both classes. However, this problem cannot be avoided for the Android malware classification as the benign applications are generally infected by malware authors to release the malware.

10.6 Conclusion and Future Work

This paper proposes an Android malware classification method using two different approaches. For both approaches, each application’s signature and behavior profiles in the data set are produced with different configurations such as bi-gram and var-gram. For the first approach, feature fusion of features obtained from the filter and wrapper methods and algorithm fusion is investigated. For the algorithm fusion, an ensemble of five different algorithms from five different families is generated, which uses a majority vote to determine the outcome of the classification task. For the second approach, SMOTE is used to artificially generate the malicious examples. However, for the second approach, only algorithm fusion is performed. The experimental results suggest that the proposed approaches are capable of detecting both known and previously unseen Android malware while keeping false positives at a relatively low level. Another conclusion is that synthetic malware examples may also help in detecting missing examples of malware in the data set without affecting the detection accuracy. For the future, we plan to investigate the parameter tuning of algorithms to further reduce false positives. We also aim to extend our experiment for the large data set and explore other feature and algorithm fusion methods.
References


References

[Accessed: November 16, 2023].

[Accessed: May 2, 2023.]

https://www.malwarebytes.com/resources#analyst-reports
[Accessed: November 16, 2023].

https://www.sophos.com/en-us/content/security-threat-report
[Accessed: December 2, 2023.]


References


References


[93] Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, and Yuval Elovici. Unknown malware detection via text categorization


References


[176] Daniel Gibert, Carles Mateu, and Jordi Planes. The rise of machine learning for detection and classification of malware: Research de-


References


References


