uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Olivier Henchiri

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S. (Master of Computer Science)

GRADE / DEGREE

School of Information Technology

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Feature Selection and Evaluation Scheme for Computer Virus Detection

TITRE DE LA THÈSE / TITLE OF THESIS

Nathalie Japkowicz

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Stan Matwin

Anil Somayaji

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

A Feature Selection and Evaluation Scheme for Computer Virus Detection

OLIVIER HENCHIRI

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the Master's Degree in
Computer Science
Thesis Supervisor: **Dr. Nathalie Japkowicz**

Ottawa-Carleton Institute for Computer Science
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-18423-3
Our file Notre référence
ISBN: 978-0-494-18423-3

Abstract

Computer viruses have existed since the early days of personal computers, and have since become a ubiquitous problem in the world of computing. The rate at which new viruses are created today and the age of networking have contributed to making anti-virus software a crucial aspect of just about every computer system. But while current virus detection methods provide good protection against known viruses, they remain primarily reactive to outbreaks, always one step behind the latest virus.

In our work, we implement a virus detection scheme that addresses this issue and focuses on improving the predictive power of a virus classifier on new viruses. We propose a hierarchical process for feature extraction that allows for an exhaustive feature search and, in contrast with current signature detection methods, obviates over-fitting.

We also introduce an evaluation scheme that relies on sub-classes, using our understanding of virus taxonomies and a priori knowledge of our dataset. This more rigorous evaluation ensures that the classifier does not rely on a collection of over-fitted features and measures the predictive power of the classifier more accurately and in a way that is relevant to the domain of computer viruses. We show that our model performs better than the traditional approach on a test set containing viruses unrelated to any training example, and that it can be expected to perform as well in the real world.

Acknowledgments

I thank my thesis supervisor, Dr. Nathalie Japkowicz, for providing a direction for my thesis and for her guidance throughout my research. I also thank her for providing me opportunities to present my ongoing research in class, at research group meetings and to third parties in the industry. I am also grateful for her patience and support during my graduate studies.

I would like to thank Dr. Japkowicz's research group for providing opportunities to communicate and discuss my research, and particularly to Reuben Smith for his insightful comments.

Special thanks go to Matthew G. Schultz for so generously sharing his dataset of computer viruses with us and for his encouragement.

Thank you also to the system staff at the School of Information Technology and Engineering for their help and for increasing my disk quotas during the initial stages of my experimentation.

Finally, I thank my family for their tremendous support and encouragement during my studies at the University of Ottawa. Special thanks to my mom for proofreading my thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer viruses are programs that are designed to spread through self-replication. When activated, viruses often perform undesired actions, often with malicious intent and without the knowledge of computer users. Viruses have been a long-standing problem in the world of computers and have become a critical concern to organisations and computer users in the last decade because of two main reasons. Firstly, greater technological expertise and access to programming tools and resources have contributed to an increase in the rate at which computer viruses are written. Secondly, the age of networking and the ubiquity of the Internet have increased the risks of infection by providing viruses with new and expeditious means of spreading.

Anti-virus systems help to mitigate the problem by providing a line of defence against the threat of viruses. These systems are carefully programmed to recognize known viruses and prevent them from causing harm. However most virus detectors protect almost exclusively against existing viruses, leaving the host machine vulnerable to outbreaks of new viruses, until they are updated to include these new viruses. Because of their reactive nature, virus detection systems must constantly keep up with the latest virus, a challenge

that has become increasingly difficult with the ever-increasing rate at which new viruses are being written.

A better solution to computer virus detection would be to extend the scope of the protection to new and future viruses, by making speculations based on previous data. By adding a predictive component to an anti-virus system, we allow it to build defences against viruses in advance, thereby putting it a step ahead of the threat.

In our research, we are concerned primarily with training a machine learning agent to detect unseen viruses, based on prior experience with a set of completely unrelated instances. In particular, we preoccupy ourselves with the challenge of selecting features representative of viral properties that would enable a classifier to identify new viruses. Prevalent anti-virus techniques rely principally on signature detection, which can only detect previously encountered viruses. While this is a recognized lacuna in the industry, little research has been done in the field.

1.1 Goal of Research

In this thesis, we are interested in applying machine learning methods to the problem of virus detection. Traditional detection methods rely on experts studying each new virus to find characteristics that would enable automatic scanners to detect it in the future. Machine learning methods can generate sets of rules in order to determine classifications of new data automatically. We wish to investigate whether or not machine learning could be used to reduce the amount of human supervision required to keep up to date with latest threats. Supervised learning algorithms are used in situations where data previously classified by experts is available. Thanks to the fact that there exists an abundance of

viruses that have been identified to-date, this problem is well suited to supervised learning.

We particularly wish to focus on the problem of feature selection that exists in the computer virus detection domain. The task of analyzing virus code by hand in order to unearth features and rules is time consuming and likely not exhaustive. Viruses targeting the same platforms, such as the Windows operating systems, use similar system resources in their execution, and therefore have similar traits. These general traits are used in heuristic methods, which attempt to detect a wider range of viruses than that covered by signatures, including new viruses, but these methods have been found less than adequate and require significant human supervision. Efforts to automate the search for more general features are faced with the challenge of high computational and memory costs – making them impracticable. Machine learning can be used to discover features in large sets of data and generate rules useful for classification. Such a method could also uncover hidden features and correlations that are beyond the reach of a human expert, either because of the high complexity of the task or because of the sheer quantity of data to analyze.

Finally, we are interested in maximizing the predictive power of virus classifiers. As machine learning strives to allow a system's performance to improve with experience, we wish to take advantage of it in classifying new virus instances. Detecting previously encountered viruses can be done effectively without the use of machine learning, but detecting new viruses with high accuracy represents a more difficult task. Therefore, we focus our efforts on ensuring that our system is specifically trained to detect completely new virus instances. After examining virus types and taxonomies, and looking at the typical ways that they strike in the real world, we endeavour to formulate a rigorous testing methodology. Then, through proper evaluation, we attempt to generate a model

that will be proactive in its classification of viruses, and yield better results than that of currently available virus detection systems.

1.2 Overview of the System

The system we propose is comprised of three main stages: the feature selection step, the training step and the evaluation step, as shown in Figure 1. These three steps are repeated in wrapper-style iterations, in order to find the best possible set of features, based on the performance of the classifier. We explore the hypothesis space using different feature sets by varying feature search parameters and using a learning algorithm as a black box. We scan through a range of parameter combinations in an attempt to select the most advantageous features. Our system thus produces a series of concepts from a training set of viruses, and it evaluates each one on a separate uncorrelated test set. We then examine the performance of our virus classifiers and identify the feature sets that yield the best performances.

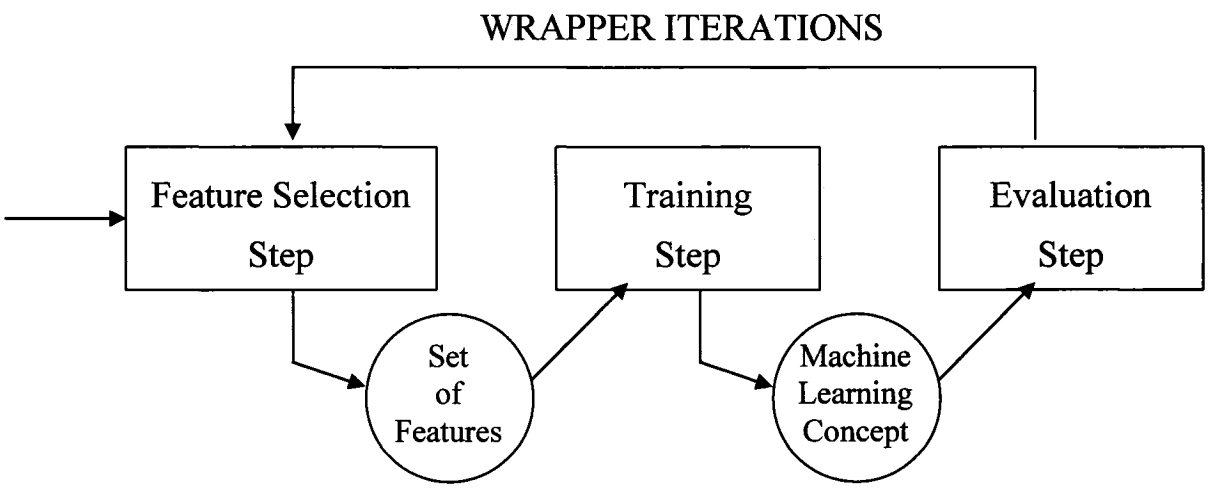

Figure 1: Overview of the system

In the feature selection step, we focus on finding features that are representative of general viral properties. To achieve this goal, we propose a pre-processing of the dataset based on a priori knowledge, aimed at preventing any unwanted correlation between the training set and the test set. We then proceed with a hierarchical search through the resulting training set. We discuss our method in detail in Section 3.1.

For the training and evaluation steps, we first convert the examples in the training set and those in the test set to their new representation in the established feature space. The classifier is then trained on the training set in order to generate classification rules using the chosen set of features. For the evaluation, we use a family-based leave-one-batch-out (LOBO) scheme where each of the five batches consists of a section of the pre-processed dataset. The feature selection, the training and the evaluation steps are all repeated on the data for each of the five LOBO validation rounds. The classification performance for a given set of features is taken as the average of the results of the virus classifier in each of the five rounds. We discuss our evaluation method in Section 3.2.

Using a wrapper approach, we construct a wide range of feature sets in a brute force manner and we generate a series of classifiers using these feature sets. At each iteration, we modify certain parameters, such as the feature length and the minimum support threshold, to consider different features. By carefully ensuring that the batches stay identical in every iteration – i.e. the same data make up corresponding batches in different iterations – we achieve a neutral basis for comparison of the classifiers, and thus of the effectiveness of the feature sets.

The performance of our learner is also compared with that of a classifier based on recent machine learning research in this domain. In Chapter 4, we present our experimental results and, based on them, we validate our feature selection philosophy and our evaluation methodology.

1.3 List of Contributions

In this thesis, we devise a feature selection method that is automatic and searches exhaustively through our dataset. Our method exploits the close similarities of highly related viruses – viruses that belong to the same family – and, unlike existing feature extraction schemes, it strives to identify features that are not exclusive to any particular virus or family. As a result, it also ensures that features are not overfitted to the training set – i.e. previously seen viruses – but that they have a larger scope of usefulness that extends to unseen examples. We compare sets of features of different lengths, from traditional 16-byte long features to very short 3-byte long sequences, and we show that shorter sequences are necessary to avoid overfitting the data, and to utilize heuristic rules on unseen examples successfully.

We present an evaluation model that is more rigorous than traditional evaluation practices. Our model, used in concert with our feature selection method, isolates highly related viruses together, preventing accidental correlations between the training and test sets. We show that traditional evaluation schemes give an unfair advantage to classifiers, which makes them unlikely to perform as well when deployed in a real world setting. Our model prevents the classifier from having that advantage.

We show that current machine learning methods do not fare well in our proposed evaluation method. In our experimentation, we test a method based on the traditional feature selection process, found to yield 96.88% overall accuracy in previous literature. While this approach deteriorates to 63.52% when tested using our evaluation method, we show that a classifier using our feature selection method maintains a 93.29% overall accuracy. We also evaluate a number of computer virus classifiers, each exploiting

different feature spaces. Our experiments yield great results, with detection rates of new instances reaching as high as 98% with a false positive rate under 1%. We analyze our results and show that our classifiers can be expected to perform similarly well when deployed in the real world.

Chapter 2

Background

Computer viruses have been a serious security threat in the world of computers for some time. The scientific community anticipated as early as 1984 the scope of the menace that viruses could pose [10]. Its surmise that they would one day have the potential to wreak havoc on modern civilisation, once a fashionable science-fiction theme, is now echoed by experts as more than mere speculation. Indeed, before the turn of the millennium viruses were being created at the rate of eight to ten per day [44] and their economic impact has been felt globally. Today, the number of computer viruses in existence is growing faster than ever, with as many as 50 new variants surfacing every day [40]. As a result, computer viruses are a liability no organisation or individual can ignore. In Section 2.1, we provide an overview of computer viruses, of their characteristics and modes of transmission, and then we discuss the notion of false negatives, which is a key issue in virus detection research.

In the past decade especially, and at present, with computers so readily interconnected and with new malicious programs being created at the rate of thousands every year, computer virus detection systems have become an essential tool in guarding the uninterrupted operation of modern-day computing devices. Organisations and computer

users alike have come to rely on security systems to safeguard them from these threats. In Section 2.2, we review standard commercial approaches to virus detection systems.

But virus detection systems face a bigger challenge than ever: with new viruses constantly being written every day and a time-to-exposure reduced to a matter of hours, they must be able not only to detect known viruses but also to predict future outbreaks. Without this ability, anti-virus systems are unable to defend reasonably well against new threats, leaving users vulnerable to serious security risks. We introduce machine learning concepts in Section 2.3.1 and discuss their relevance to the problem of virus detection. Finally, in Section 2.3.2, we report on the state of current research in virus detection using machine learning.

2.1 Overview of Virus Detection

In Section 2.1.1, we first provide a definition and characterization of viruses and then, in Section 2.1.2, we expose the problem of false positives – an important hurdle facing any detection method intended to provide dependable and viable protection against viruses.

2.1.1 Characteristics of Computer Viruses

Malicious executable programs, or malware, have been developed both for in-laboratory study and for public dissemination. Although arguments have been made for the possible utility of some viruses, challenging the notion that viruses are inherently bad, it is generally agreed that viruses cause "unexpected and undesired behavior on the infected computer or on the network to which the computer is attached" [30] and compromise the security of computer networks.

CERT, a coordination centre of Internet security expertise operated by Carnegie Mellon University, defines security incidents as follows [7]:

1. attempts (either failed or successful) to gain unauthorized access to a system or its data,
2. unwanted disruption or denial of service,
3. the unauthorized use of a system for the process or storage of data, and
4. changes to system hardware, firmware, or software characteristics without the owner's knowledge, instruction, or consent.

The perniciousness of computer viruses stems from their unbound capacity to proliferate through self-reproduction. This distinctive property was recognized early on by Cohen, in the first and possibly most widely cited paper about computer viruses, in his 1984 observation that "the virus is interesting because of its ability to attach itself to other programs and cause them to become viruses as well" [10].

A variety of programs can be considered malicious, but there exist three major types of malicious executables that meet the self-reproducing property: computer viruses, Trojans horses and worms. Put generally, a computer virus is a "sequence of instructions that copies itself into other programs in such a way that executing the program also executes that sequence of instructions" [2]. While the terms 'computer virus' are commonly used to describe Trojans and worms, the latter differ by their mode of transmission. Trojans are related to viruses except that they cannot operate or replicate autonomously. They disguise themselves as legitimate programs and depend on victims running their code in order to carry out their objective. Worms are similar to viruses "in their ability to self-replicate but do not need to attach parasitically to other programs" [8].

Many viruses now take advantage of network services and interconnectivity between computers. The expansion of the Internet has revolutionized the way that we use computers and communicate, but along with that transformation came about a widespread propagation of viruses. Today's viruses can propagate via a myriad of vehicles: email, file servers, newsgroups, Internet Relay Chat, P2P file sharing, instant messaging, and through the multitude of programs and freeware readily available on the World Wide Web. This has the effect of blurring the line between viruses and worms. Similarly, Trojans horses, once limited to floppy disk circulation, now employ automatic distribution techniques using email applications and the Internet, further confusing the distinctions between viruses, worms and Trojans. For example, in 2000, the LoveLetter worm, which spread worldwide in a matter of hours, causing as much as $10 billion in damages worldwide [6], relied on its victims executing its script – mainly out of curiosity. Technically a macro, the worm employed a VBScript mechanism to automatically disseminate itself to all addresses in the victim's contact list. It also replicated itself countless times on the victim's computer, destroying and replacing legitimate files.

Computer viruses have long since been classified in groups. Many viruses are variants of each other, on account of similarities in structure, code or method of infection that they share [3]. Virus writers often write altered versions of their viruses, so as to obfuscate detection systems, or worse, they employ methods designed to conceal their malicious code or mutate viruses automatically when they replicate during infection and propagate from one victim to the next. We briefly review some of these documented methods [41], as they are relevant to our motivation for the detection techniques presented in this thesis.

Encryption

Encryption is one the easiest ways for virus writers to hide the functionality of their code. Typically a constant decryptor routine is placed at the beginning of the virus' code, and is then followed by the encrypted virus body. This does not render typical scanners ineffective, but it prevents virus analysts from developing detection methods based on a virus's code as it plainly relates to a set of logical instructions.

Oligomorphism

Oligomorphic viruses have the ability to alternate their decryption routines between different versions from a finite set of alternatives. Virus writers achieve this by programming the virus to automatically modify its decryptor patterns from one generation to the next; while the encryption scheme stays identical, the decryption routines vary slightly in their implementation. This has the advantage of not exhibiting the same obvious encryption signature, which could be recognized by pattern recognition software, and thus reveal the potential presence of the virus.

Polymorphism

Polymorphic viruses can take on an unlimited number of different encryptors. However, unlike oligomorphic viruses, they are capable of using different encryption schemes. This results in viruses that have the same code base but that are expressed in dissimilar encrypted binary sections in each variant's body – and often under multiple layers of encryption – from one generation to the next. This has the effect of making the task of analyzing either the virus body or the encryption engine more arduous.

Metamorphism

Metamorphic viruses are programmed to modify their code base before encryption. This can be done by inserting and deleting ineffectual "junk" code, using different registers or

exchanging register usage, and using code permutation techniques. This makes it very difficult for virus scanners to find constant patterns anywhere in the virus.

Each of these methods creates virus variants that are related through core sections of common code or structural similarities. As such, these viruses are considered to belong to the same **virus families**.

Virus outbreaks typically consist of an initial incidence and its related morphed varieties, if applicable. Progressively, as experts become aware of this and develop a detection technique, anti-virus systems are updated to protect against this new virus, though, in most cases, not without involving casualties before the update is made available and distributed. Usually, however, new variants are created as a direct response to this newfound immunity. The new versions can appear months after the original instance of the virus, as new ways of circumventing detection techniques are found, and, as such, they may require additional detection techniques more specific to them. On the WildList [32], a widely respected source of information on active viruses, for nearly all virus families identified, the initial reported virus is followed by an ensuing number of variants in the following months, sometimes up to a couple of years ahead. However, the first occurrence of a new virus is typically the most devastating [29], as virus detection systems are often completely unequipped to react to new families of viruses and thus incapable of providing any protection against them.

2.1.2 False Positives

The term **False Positive,** or false alert, is used when anti-virus software misclassifies a benign file as a virus. This incorrect detection can happen as a result of heuristic rules or by a too general virus signature. The false positives exist in contrast with three other

classification outcomes. Table 1 lists the four possible outcomes of the decision of a virus scanner versus the actual nature of the file being checked.

	Actual Virus	**Benign File**
Generate Alert	True Positive	False Positive
Do Not Generate Alert	False Negative	True Negative

Table 1: Categories of decisions of a virus scanner versus actual nature of a file.

Classification of an Actual Virus

• If a real virus is reported by the virus scanner, this is called a **true positive**. The true positive rate represents the ability of the scanner to detect viruses, and is often boasted by anti-virus software vendors as the "detection accuracy" of their product. The true positive rate should be as close to 100% as possible.

• If a real virus is not reported by the virus scanner, this is called a **false negative**. The false negative rate is the ratio of viruses that the scanner is unable to detect; therefore a low false negative rate is desirable. When evaluating a virus classifier, the false negative and true positive rates represent the classification and misclassification rates of positive examples, and their sum equals one.

Classification of a Benign File

• If a benign file is correctly identified as a non-virus by the virus scanner, this is called a **true negative**. The true negative rate represents the relative amount of non-viruses that pass through the virus scanner without any problem. The true negative rate should be as close to 100% as possible.

• If a benign file is incorrectly identified as a virus by the virus scanner, this is called a **false positive**. The false positive rate is the proportion of benign files that the scanner incorrectly classifies as viruses. The false positive rate must remain as low as possible. When evaluating a virus classifier, the false positive and true negative rates represent the classification and misclassification rates of negative examples, and their sum equals one.

The false positive rate – the measure of the degree to which a virus scanner makes such misclassifications – has always been a major concern for virus scanners, and, lately, has been the object of increasing interest. The anti-virus industry has begun to conduct more thorough evaluations of virus detection systems, including an estimation of false positive rates [45, 12]. The false positive rate is defined as follows:

$$\text{False positive rate} = \frac{\text{Number of false positives}}{\text{Total number of negative examples}}$$

Because false alarms result in a considerable waste of time and resources looking for a non-existing problem, it is generally agreed that the cost of false positives is as high as that of a genuine infection, if not higher [14, 4, 37] and can have legal consequences for anti-virus companies [5]. Whereas the false negative rate is proportional to the effectiveness of a virus classifier at protecting against viral threats, the false positive rate is important in "determining the usability of the malware detector: if it incorrectly flags

too many benign programs as being infected, the user may lose faith in the malware detector and stop using it altogether" [9].

Unfortunately, a tradeoff must be made between false positives and false negatives [1]. The more generic a detection scheme is, the more viruses it will be able to detect, but the more likely it is to misclassify benign files as viruses. On the other hand, if it is more specific to known viruses, the false positive will be lower, but some new viruses or variants may pass through undetected.

Commercial approaches have dealt with this problem by using conservative methods for detecting computer viruses. The next section exposes the two main techniques used to that effect.

2.2 Commercial Approaches

Traditional anti-virus systems are predominantly signatures-based, in that they use specific features extracted from viruses in order to detect those same instances in the future. White et al. [45] noted in 1998:

> "Over the past ten years, a single method of detecting computer viruses has nearly eclipsed all others: scanning for known viruses. Originally, a string of bytes was selected from some known virus, and the virus scanner looked for that string in files as a way of determining if that file was infected with that virus. Later, more complex techniques were developed which involved looking for various substrings in various parts of the file. But all of these techniques have one thing in common: they look for static characteristics of viruses that are already known."

While these methods yield excellent detection rates for existing and previously encountered viruses, they lack the capacity to efficiently detect new unseen instances or variants. More recently, heuristics methods have been developed with the aim of compensating for this lack. We first look at the signature method and examine its weak points in Section 2.2.1. Then, in Section 2.2.2, we review heuristic analysis methods, developed in the last decade, and we discuss the limitations of these techniques.

2.2.1 The Signature Method

The signature method is the most widely used technique in commercial virus detection systems, and it includes no machine learning component. Each time a new virus is detected, distinctive characteristics are extracted from it and added to a database. These characteristics are generally byte sequences, but they can also include recognizable system calls, filenames, symptoms of infection, or anything peculiar about a specific virus. Sometimes, several characteristics are concatenated together to form a signature. Signatures can be narrowly defined, so that they would be extremely unlikely to be found in any file other than the malicious executable from which it was extracted, or they can be defined more broadly, so as to be more resilient to rudimentary code obfuscation techniques. However broadly defined signatures introduce the problem of false positives and, in any case, would not be able to detect new threats [8]. Thus, signatures are generally defined conservatively, and it is reported that commercial scanners may even use the whole virus body as signature in some cases [9]. An occurrence of that signature would then unmistakably reveal the presence of that virus. The signature method is widely used because it provides dependable protection against known viruses and generally yields a false positive rate close to zero.

Traditionally, experts were required to scrutinize each new virus and extract from it a signature, a practice that is both time-consuming and error prone. To expedite and

facilitate the task, some efficient virus signatures extraction methods have been developed. Kephart et al. [19] elaborated a popular technique for automatic extraction of virus signatures. In their experiments, 12 to 36-byte long strings were extracted after a large number of files had been infected with the same virus, and constant regions were identified. Then, from the considerable number of signatures collected, the ones with lowest predicted false-positive rates were selected. For each individual virus, this method proved capable of finding features symptomatic of that specific virus. While this method permits to extract signatures quickly and without the help of an expert, the features found in each case are meant to be used to detect the same viruses used in the infection stage, and are by no means expected to appear in other viruses. Indeed, the authors concede that the algorithm fails for viruses that are even moderately polymorphic.

Modern viruses employ advanced strategies such as polymorphism and metamorphism [41], through which parts of the virus change in a sometimes random or unpredictable way each time it replicates. As well, virus writers, who have access to commercially available anti-virus software, can direct their efforts toward outwitting the scanners, by writing or modifying their code so that it passes undetected. This can be accomplished by strategically modifying the virus such that the characteristics that comprise the virus signature be changed. Because virus detection software predominantly relies on virus signatures, it provides little protection against these tactics. In each case, the signatures used by virus scanners are so specific that they are unlikely to spot the new viruses, or their variants, therefore new signatures must be found in case.

Signature detection is an exclusively reactive, rather than proactive, detection method: when a new virus is encountered, it must be analyzed and its signature must be added to a repository so that it can be detected in the future; before this update is completed, the virus detection system leaves a window of vulnerability open. Established practices attempt to mitigate the problem by providing signature updates as frequently and as

quickly as possible. However, as computers can exchange data extremely quickly, this approach is bound to be broken. Recent global outbreaks, such as the Melissa worm, made it clear that even daily updates of signature files are insufficient to prevent the pandemic spread of viruses [12].

To compensate for these lacunas, anti-virus systems have incorporated, during the last several years, heuristics rules, whereby they try to extend the scope of their detection abilities to new viruses. Heuristic analysis has become an indispensable part of modern virus scanners, and most now complement their signature-based detection with heuristics rules [39].

2.2.2 Heuristic Analysis

Heuristic scanners attempt to compensate for the limitations of signature detection by using more general features from viral code, such as structural or behavioral patterns [14]. Heuristic classifiers are modeled on the way that experts can ascertain whether or not a computer program is malicious, in the absence of visible symptoms. When an executable does not exhibit large familiar viral code sections, security experts must resort to one of two means to identify it as a virus or a non-virus: to run them and classify their behaviour as malicious or benign, or to examine their code meticulously and predict their behaviour. The latter requires expert knowledge of computer viruses and a time-consuming in-depth study and classification of each individual case. Heuristics automate part of this process by making the case-by-case classification decision based on a set of general features and rules as determined by human experts.

Experts can identify a malicious program by its very nature, as opposed to its undesired effects, by examining the way that its algorithm is written. Through reverse engineering –

disassembling the machine code back into more readable logical instructions, one can decipher behavioural patterns in the code, i.e. what the executable is programmed to do and how it does it. With that information, it is then possible for an expert to determine whether or not an executable has malicious intentions.

A heuristic scanner automates part of this process and makes its decision based on these algorithmic patterns that match a set of rules as determined by a human expert. With these more general characteristics, a scanner can be programmed to examine an executable file and make an automatic judgment, thus eliminating the need for a human to examine the code and understand its algorithms.

Some detection methods utilize a variety of features, such as Win32 dll file calls, printable character strings and byte sequences contained in the binary files. In an early heuristic approach [20], features such as duplicated UNIX system calls and files targeted by the program for writing purposes were used to estimate the likeliness of an executable of having malicious intent. In [41], the author reports that manual analysis of polymorphic viruses can uncover algorithmic patterns that persist through polymorphism. Wildcard strings – mapping logical functions while allowing for junk code insertions – can, in some cases, detect viruses that are using obscuring methods such as junk code insertions. Gryaznov examines the case of a simple parasitic file infector and provides an example set of heuristic rules as follows [14]:

1. The program immediately passes control close to the end of itself;
2. it modifies some bytes at the beginning of its copy in memory;
3. then it starts looking for executable files on a disk;
4. when found, a file is opened;
5. some data are read from the file;
6. some data are written to the end of the file.

Each of these rules is represented by a particular sequence in the binary machine code or assembly language code, as shown in Algorithm 1 (as presented in [14]).

Algorithm 1 FILE_INFECTOR

```
START:                          ; Start of the infected program
        JMP  VIRUSCODE          ; Rule 1: the control is passed to the virus body

<victim's code>

VIRUS:                          ; Virus body starts here
SAVED:                          ; Saved original bytes of the victim's code
MASK: DB `*.COM' , 0            ; Search mask
VIRUSCODE:                      ; Start of the virus code
        MOV  DI,OFFSET START    ; Rule 2: the virus restores
        MOV  SI,OFFSET SAVED    ; victim's code
        MOVSW                   ; in memory
        MOVSB

        MOV  DX,OFFSET MASK     ; Rule 3: the virus
        MOV  AH,4EH             ; looks for other
        INT  21H                ; programs to infect

        MOV  AX,3D02H           ; Rule 4: the virus opens a file
        INT  21H

        MOV  DX,OFFSET SAVED    ; Rule 5: first bytes of a file
        MOV  AH,3FH             ; are read to the virus
        INT  21H                ; body

        MOV  DX,OFFSET VIRUS    ; Rule 6: the virus writes itself
        MOV  AH,40H             ; to the file
        INT  21H
```

A heuristic program would then check whether or not an executable contains pieces of code corresponding to each of the six rules, not necessarily consecutively or in the order listed. In a simple case such as this one, this can be done by matching a wildcard string. If the executable satisfies rules 1 to 6, it would then be considered a virus.

These heuristic rules, relating to sets of instructions of a particular kind, are more generalized than binary signatures, and they are more likely to occur in different viruses that use the same techniques. However because benign executables may also follow similar instruction patterns - albeit not with malicious intent - they are faced with the classic dilemma of striking a balance between a high incidence of false positives and poor accuracy. By its very nature, heuristic analysis is more prone to false alarms than the signature method, so a low false positive rate often comes at the price of a higher false negative rate [14, 42, 17, 38]. Some scanners delegate this responsibility to the users by letting them choose a mode of operation – from low sensitivity to high sensitivity – and thus determining the false positive rate and level of accuracy [14].

Hybrid virus detection systems utilize traditional signature databases and a set of heuristic rules as determined by experts in the field. But heuristic scanners still require human intervention, as they require virus experts to formulate sets of rules that they judge distinctive of malicious intent, a task that is time-consuming. Arnold and Tesauro report as late as 2000 that "heuristic classifiers [...] have usually been constructed by hand" [1]. As well, heuristic rules, much like signatures, must be updated as new viruses are encountered and new infection or obfuscation techniques are employed. These updates constitute their sole means of improvement; that is, they have no innate learning capability.

In the same way that detection by signature can be evaded, heuristics can also be circumvented [17]. Because heuristic rules are static, it is possible to confuse a scanner by deliberately altering a virus in strategic places. Virus writers have access to virus scanners and can focus their efforts on figuring out the rules that apply to their viruses and modifying their code so that it meets them. Effective anti-heuristic infection techniques appeared soon after the introduction of heuristic scanning, forcing the

development of newer and better heuristics [39]. Metamorphism techniques can be so complex, as in the case of W32/Simile described in [31], that they can avoid triggering recognition heuristics, as well as make obfuscated code extremely challenging for virus researchers to analyse.

Heuristics have become an indispensable part of modern virus scanners, but they often come with a higher risk of false positives, they require meticulous and time-consuming analysis, and they involve an in-depth understanding of different virus types. The resulting models fall short of yielding both good detection rates for new unseen viruses and low false positives. As recently as 2002, the accuracy of detection of new viruses by heuristic methods was estimated between 15% and 55% for most scanners [26], and many experts have deemed heuristics unreliable [5, 12].

2.3 Machine Learning Approaches

Current anti-virus technology is largely reactive, relying on finding and analyzing a particular virus before being able to deal with it well. Anti-virus systems must be able to provide protection against both known and previously unknown viruses, and many have become hybrid systems using both specific and generic detection strategies, both of which typically require frequent updates. However, modern programming and networking environments can give rise to viruses that spread increasingly rapidly, and for which a reactive approach becomes ever more difficult [45]. Methods such as signatures detection and heuristic rules, where human intervention is still required, will always be faced with the challenge of providing a timely immunity to a host of users worldwide [12].

A handful of machine learning methods have strived to address these flaws. Strictly speaking, heuristic detection of computer viruses encompasses both signatures and so called heuristic rules, for the only certain way to identify a virus is to detect an exact copy of a known virus, or to witness its effect by executing or disassembling it. In both the signature and heuristic rule cases, the presence of a given feature is used to infer with a certain probability that an executable file is a virus, but with no theoretical guarantee. In the case of signatures, the features are long bytes sequences, typically over 16 bytes long and as long as 32 bytes [20, 18]. As seen in Section 2.2.1, signatures are very long fragments of viruses and are chosen such that they be extremely unlikely to be found in files other than the viral instances from which they were extracted. Because signatures are features that are essentially designed to match a specific virus exclusively, a signature may not match even close variants of a virus from which it was extracted. On the other hand, heuristics are based on more general features, such as algorithmic patterns, structural characteristics of the code, or operating system resources used, as well as bytes sequences shorter than traditional signatures. These heuristic rules are typically designed to match a number of virus variants or groups of related viruses, and they have been used to provide more consistent detection of polymorphic viruses, as they mutate from one version to another. But, whereas automatic signature extraction techniques do exist [19], heuristic rules still require human supervision [1]. However, much like virus scanners automate the application of heuristic rules, by matching wildcard string or other features, machine learning algorithms can automate and supervise the search for these features and rules. Moreover, automatic extraction methods deployed in the field have shown that a search algorithm's ability to select good signatures can be better than that of human experts [18]. Current research applying data mining to virus detection strives to discover useful features, train a classifier on a set of viruses and evaluate its accuracy on a set of unseen virus examples. This process has been tackled from two different angles: extracting optimal signatures from a dataset of viruses, and discovering more general features for use in an advanced classification scheme.

First we introduce machine learning in Section 2.3.1, and discuss how it relates to the field of virus detection, and then, in Section 2.3.2, we review existing computer virus research using machine learning.

2.3.1 Machine Learning

Machine learning seeks to make it possible for computer programs to learn from data and "improve automatically with experience" [27]. Machine learning uses inductive reasoning to infer a target concept from multiple observations. This concept can then be applied to classify new data in the future. Machine learning has been applied to a wide spectrum of domains, including text classification, medical diagnosis, speech recognition, detecting credit card fraud, computer vision, Internet search engines, and intrusion detection systems.

In supervised learning, a learner is given a training set of already classified data and tries to infer rules as to how to correctly classify new unseen objects of that kind. These rules are based on correlations between certain features of the objects and the classes to which they belong. Thus a learner establishes a mapping of input values (the features) to output values (the class) based on a set of features.

Advances in information technology have afforded us the capacity to collect and store large amounts of data. As a result, feature selection has become key to machine learning applications and an increasing focus of interest. The objective of feature selection is three-fold: it serves to improve "the prediction performance of the predictors", provide "faster and more cost-effective predictors", and provide "a better understanding of the underlying process that generated the data". Depending on its intended application,

feature selection may focus on addressing the overabundance of features, avoiding the curse of dimensionality, reducing training times, or other problems [15].

Feature selection is commonly conducted using one of three ways: wrappers, filters, and embedded methods [15]. In the **wrapper approach**, "the induction algorithm is used as a black box, [and] is repeatedly run on the dataset using various feature subsets" [21]. The feature subsets are appraised according to their predictive power, and the subset scoring the highest is chosen as the final set. **Filter methods** use a pre-processing phase to select subsets of variables before the training step and without regard to the chosen predictor. **Embedded methods** perform feature selection within the training step of the machine learning algorithm and specific classifiers.

Features are key to the classifiers predicting correct output values, and learning algorithms are often faced with "the problem of selecting a relevant subset of features" [22]. Depending on the domain of application, they can be visible discrete quantitative measurements or qualitative traits. Other times, they can be points on a spectrum of continuous values or properties hidden in the complex makeup of the data. In real-world applications, where objects do not come in specifications convenient for computer processing, it is oftentimes necessary to pre-process the data in order to define what constitutes a feature and obtain measurements conducive to machine learning. In such cases, a feature search step must precede the learning step, in order to provide it with input values. When a feature set is found, the data are converted to a representation using these features. In this new representation, the data are given to the classifier, where a mapping of inputs to outputs can then be learned.

2.3.2 Machine Learning-Based Systems

A novel virus detection system was developed in 1996 by Tesauro et al. [42], where a neural network was used for generic detection of a particular class of computer viruses – boot sector viruses – which infect the boot sector of floppy disks and hard drives. In this system, short byte strings (specifically in this case trigrams) that appeared frequently in infected boot sectors but infrequently in legitimate ones were used as features. After an elimination step to reduce the number of features, a new data representation was produced and every instance was then converted to an input pattern indicating the presence or absence of the trigram in its binary code. A single-layer neural network was then trained using about 100 viral examples and 50 nonviral examples. The authors reported a detection accuracy of 80-85% on the evaluation set, with a false positive rate lower than the resolution of their validation set (about 1%). The commercial product incorporating this method was reported to yield a detection accuracy of 75% for new boot sector viruses.

Using byte sequences as short as trigrams was an approach differing from the signature method, and did provide their classifier the means to detect new instances successfully. However, because the training set was very small and the set of features greatly reduced during the elimination step, the remaining features were highly correlated to the viral boot sectors. This had the effect that no legitimate boot sectors in the training scored for any of the features. This led to an incorrect generalization, which would have produced a false positive for any legitimate boot sector that scored for any of the features. To correct this problem, the authors generated a large amount of artificial data, which they used to relax the classification model.

While the results were encouraging, the authors conceded that the available training data was extremely limited. As well, because boot sectors represented only about 5% of all

known viruses at the time, and significantly less so today, and because they did not use obfuscation techniques such as polymorphism or metamorphism, this method has a limited application today.

Arnold et al. extended the scope of this technique in second experiment involving Win32 viruses [1]. Here again, short byte sequences were extracted from constant binary regions of viruses, this time of lengths 3 and 4 were chosen, but eight separate feature sets were generated using complete n-gram 1-covers of the training set. A complete 1-cover is a set of features such that at least one feature is present in each virus in the training set. Eight of such covers were generated and used with 8 separate single-layer neural networks, each cover representing the input data differently to each classifier. Voting was used to combine the outputs from each network into a single classification result, and different voting thresholds were used in experiments. Results on evaluation sets displayed a best-case detection accuracy of 97% with a minimal false positive rate for byte sequences of length 4, while they showed a best-case accuracy of 56% with a false positive rate under 0.09% for sequences of length 3.

In this case as well, the results are encouraging but questionable, due to the fact that the available training data was extremely scarce. Furthermore, this method would be impracticable on a large dataset. Generating 1-covers of a large training set, using a then larger number of candidate n-grams, would require significant computational work and a considerable amount of memory.

More recently, data mining methods for detection of new malicious executables were developed by Schultz et al. [34] and incorporated into an email filter [35]. Features consisted of a variety of properties: Header information from Common Object File Format files, Win32 dll file calls, printable character strings contained in the binary files, and byte sequences.

The search for byte sequences was done by successively examining 16-byte segments in the binary code. Using a conversion tool, they transformed binary files into hexadecimal files, as shown in Figure 2, where each line represents a sequence of machine code instruction. Each line displayed was considered a feature and, from that point on, was treated as a text.

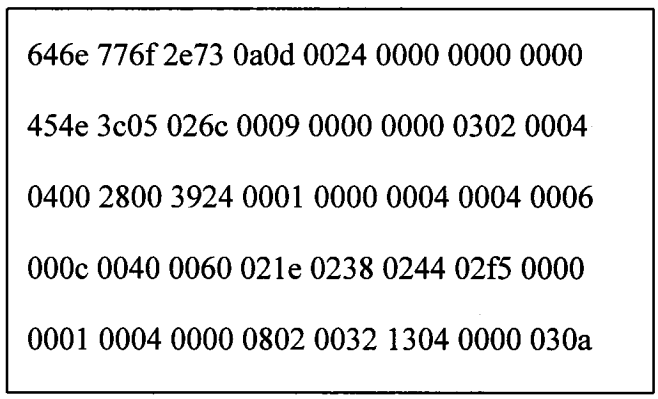

Figure 2: Example set of byte sequences features

The choice of longer 16-byte long features contrasts with previous experimentations [1, 42], where short and therefore more generic sequences were selected. We note that this method could miss useful features if these are split between two lines of code. Figure 3, for example, shows 32-byte sections of code from two different files, where a 16-byte long sequence appearing in both files would remain undetected because it does not line up in both sections.

We also note that, because each and every hexadecimal line is used as a feature, a very large number of features were elected. Many of these features were likely not very discriminating of viral code, as many must not have had much support, if any, outside of the particular instance they were taken from. Also, because of this, and despite the byte

sequence having a fixed length of 16, the feature space was very large and had very high memory requirements, to such an extent that their dataset had to be split into partitions and separate classifiers had to be trained on each partition. Classification of test examples was then done by having each classifier give its vote on the output value. We believe that features with very low support in past examples – i.e. features that have a very low rate of incidence in the set of known viruses – are less likely to be useful in classifying new instances. In Section 3.2, we examine how we can eliminate useless features in order to generate a feature space of reasonable size, and insure that all retained features have a minimum support in the general corpus of infected files.

File # 1:

[line 33]: ...
[line 34]: 2828 2828 **730a 0d65 5d89 1a32 7c55 0145**
[line 35]: **9e72 61bb** 3500 3500 3500 3500 3500 3500
[line 36]: ...

File # 2:

[line 10]: ...
[line 11]: 0000 0071 4811 642e **730a 0d65 5d89 1a32**
[line 12]: **7c55 0145 9e72 61bb** a101 6f95 0000 0000
[line 13]: ...

Figure 3: Undetected byte sequence.

In classification experiments [34], printable character strings and bytes sequences yielded the best results. The method using printable character strings used a Naïve Bayes classifier and yielded a detection rate of 97.43% with a false positive rate of 3.80%. The method using byte sequences employed a Multi-Naïve Bayes system where the outputs of 6 separate classifiers, each using a different partition of the feature space, were combined. A detection rate of 97.76% and a false positive rate of 6.01% were achieved.

While the method using printable character strings did show a high detection rate, its classification is based on actual data contained in the viruses, as opposed to executable code. Data such as strings of characters can easily be modified by virus writers to evade detection. In Section 3.1.3, we show examples of such characteristics that are present in viruses but that are not useful in trying to classify new virus instances.

Previous research has presented novel methods and developed virus classifiers that show promising results. However, none has paid attention to existing correlations of viruses in the training and evaluation sets. In Chapter 3, we study the importance of evaluating a virus classifier on truly independent datasets, and we show how failing to partition a virus dataset can lead to biased results. We then propose a feature search method and evaluation methodology that addresses this problem. In Chapter 4, we conduct experiments using viruses used in [34], and support our concept with experimental data.

Chapter 3

Our Model

In this section, we present a data mining system that conducts an exhaustive feature search on a set of computer viruses and strives to obviate over-fitting, thereby increasing the predictive power of the classifier. Our features cover the virus examples better than signature-like features, and are shown, in our experimental results, to yield high detection accuracies while maintaining low false positive rates. We begin by introducing, in Section 3.1, our concept of feature space and the relationships that exist between computer viruses, then, in Section 3.2, we present our feature search method based on these concepts. Finally, in Section 3.3, we explore ways to train and evaluate a classifier in a manner that takes into account these relationships.

3.1 Properties of Viruses

Most viruses share certain characteristics, not only in maliciousness of intent but also in the means used to accomplish their goal. Viruses will often delete or modify files, add or change registry entries in the Windows operating system, open ports and use the TCP/IP stack in order to send information or to carry out a Denial of Service attack. A key part of a virus lies in its ability to reproduce either locally, by infecting other programs, or

remotely, by sending itself through email or another means of dissemination. Finally, viruses commonly employ concealment techniques such as compression and encryption, to avoid being detected by existing virus scanners.

In the next two sections, we will see how these characteristics express themselves in executable files, and how they can be discovered. Firstly, we define the format and range of features that we will consider and the way that we will scan the data in search of these features. Secondly, we will examine relationships that exist between viruses in the wild, and how we plan to use these associations to our benefit during the feature search process.

3.1.1 Feature Space

Signature-based methods [19] as well as previous research [1, 20, 34, 42, 35] use a variety of properties as features, or characteristics to look for. These features may include:

- Header information from Common Object File Format files, including the list of DLLs used by the library, the list of function calls made by the executable and the number of different function calls within each DLL.
- Printable character strings found in the binary code.
- Bytes sequences from the binary code.
- Operating system calls
- Files targeted for writing purposes

In one machine-learning approach [34], a separate classifier was used for each feature type above. A classifier using all of the features types would avail itself of a richer feature set, which may result in improved accuracy. As all of these properties are found in some form within the binary makeup of the executable file, they must be represented by a byte sequence. For example, an alphanumeric string present in the file would be represented, in its ASCII form, by a certain number of bytes in the binary code. It follows that all feature types could be replaced by binary sequences, without loss, provided the search process were able to find non-binary features in their binary form. Furthermore, as malicious executables are often encrypted, the system calls and data sections would only be found in their original form in active memory, upon running the executable. Scrutinizing all byte sequences would therefore detect strategic sequences that might be masked behind a compression algorithm.

However, this represents a difficulty: it is easier to instruct a scanner to search for strings of printable characters or a set of known operating system calls than it is to identify byte sequences common to several files, with neither a predetermined set of sequences nor any restriction as to their constitution. We will see in Section 3.2 how our approach solves this complexity issue.

In a previous research using machine learning methods [34], binary code was output as hexadecimal data in lines representing 16-byte long sequences, whereas classic signature extraction methods use strings as long as 32 bytes [18, 20]. A 16-byte string offers 1.8 x 10^{19} different possible sequences, and could potentially represent as many as 8 basic assembly language instructions, such as **"mov al, 61h"**. A code segment of that length represents an identifiable set of instructions reminiscent of a signature. Shorter sequences would permit more flexibility in the matching of certain sets of instruction or data segments, and would permit a classifier to use more of a heuristic approach. The presence of a combination of basic instructions in an executable, not necessarily appearing

consecutively, could be a sign of a viral property. For example, in the context of a high-level language, let us imagine that three particular instructions represent the actions of reading, writing and deleting, respectively. It is not uncommon for a program to open files in order to access data, whether to gather user settings, to find the location of system resources or simply to display some information. Other programs frequently write data to the disk in order to save information, while benign files may very well delete temporary files without any malicious intent. However the presence of all three in the same file could indicate that it is more likely to be a virus, as it would allow it to read a file, delete it and write it back to the disk in a corrupted state. Thus, sequences representing basic instructions could permit a classifier to exploit these hidden feature correlations.

3.1.2 Grouping Viruses into Families

Generally when an effective virus is created, many variants are then written in a short time after its first public appearance. These variants are often created in order to evade typical virus detection systems, or to find new ways to fool reckless yet alerted computer users once again. At the onset of an outbreak, virus experts will extract a signature from the initial virus and that signature will be added to databases around the world. However, it is not an impossible task for virus writers, who have access to commercially available anti-virus systems, to modify these new viruses in order to render them undetectable; altering a section that is part of signature will often suffice. Polymorphic and metamorphic viruses, designed to mutate when they replicate, also create slightly different versions of the same virus.

Viruses from the same family share similar traits, and many practices in the industry demonstrate the effectiveness of using these similarities to one's advantage. In a method for automatically extracting signatures [19] as presented in Section 2.2.1, a number of programs were intentionally infected with the same virus. By so doing, the virus injected

part of its code into the target programs. Armed with this code, these new carriers inherited the ability to infect other files, thereby becoming viruses themselves. Within this new collection of viruses, the authors were able to detect constant sections of code. These common sections were then retained as signatures, by which any subsequent instances of the virus could most probably be identified. In [3], Bontchev mentions that files infected by the same virus variant should be classified within the same virus family. However as soon as two viruses differ by one bit in their code or constant data parts, they should be categorized separately.

While this property of viruses has not been utilized in machine learning methods for virus detection, it has been applied successfully to other domains. In text classification [33, 43, 28], exploiting category hierarchy has been proved to be helpful, where a top-down approach was taken to discover broad classes before classifying more specific sub-classes. In [11], it was noted that "many potentially good features are not useful discriminators in non-hierarchical representations", and demonstrated how category features could help improve the accuracy of a classifier. In authorship verification [24], creative techniques have leveraged the close similarities that exist between two works of a same author to distinguish a given text from works of other authors from the same period or genre.

In these text classification problems, as well as the authorship verification, features consisted of the characteristic bag-of-words, or simple word counts. Our feature space consists of byte sequences and can be likened to words, which are character sequences. We therefore expect that careful discernment between general similarities of viruses and the close resemblances of same-subclass instances will lead to a more successful approach to virus detection.

Upon examination of our dataset of viruses, it is possible to identify virus families i.e. groups of similar exemplars. Table 2 shows a group of viruses with common properties: they both share similar names and comparable file sizes.

ARCV.1060.Dropper.com	ARCV.1060__1_.com	ARCV.773.com
ARCV.Anna.745.com	ARCV.Benoit_Gen1_2.com	ARCV.Evul.805_drp_.exe
ARCV.Ice-9.com	ARCV.Jo.com	ARCV.Lurve.718_G1_2_.com
ARCV.Made.334.com	ARCV.Made.334.exe	ARCV.New_Year.com
ARCV.Reaper.com	ARCV.Sandwich.com	ARCV.Scroll_Gen1_2_.com
ARCV.Scythe.com	ARCV.X-2_Gen1_2_.com	ARCV.Xmas.670_G1_2_.com
ARCV.Zaphod_Gen1_.com	ARCV.com	ARCV.more_Gen1_2_.com

Table 2: Viruses belonging to the ARCV family

The viruses in our dataset were classified by human experts and were assigned names; similar viruses were given the same prefix, and individual examples within the groups are distinguished by a suffix. Naming conventions practices for computer viruses began as early as in 1991, when the Computer Antivirus Research Organization (CARO) drafted a standard virus naming scheme [36]. Currently used naming conventions prescribe the inclusion of information such as family name, platform, infective length, subvariant, packer or encryption engine identification. For example, the virus family "ARCV" contains 21 separate instances, as shown in Table 2. It is therefore easy to distinguish virus families in our dataset. We use this partitioning of our dataset based on this for both the feature search and evaluation method, as described in Section 3.2 and 3.3 respectively.

3.1.3 Family-Specific Features

For most viruses, there exist many variants that are only slightly different, but usually sufficiently different to modify a section that is part of a signature, and thus fool commercially available signature-based virus detectors. Nevertheless, different varieties of viruses in the same family group will tend to possess similar features such as strings in the file header as well as binary sequences.

Table 3 shows some of the results of a feature search executed on the virus family group labeled "ARCV" of 21 examples, shown above. Binary sequences of length 15, 13 and 10 found with at least 20% support in the ARCV set (which amounts to occurring in at least 5 instances) are listed. One 15-byte long sequence appears with high support, three that are 13-byte long and many more that are 10-byte long. Each of the sequences of shorter length is a sub-sequence of a larger sequence, up to the longest sequence of 15 bytes. As expected, the shorter features appear with higher support than longer ones.

However, an important fact is that the ASCII representation of that sequence translates to "Made in England". This is an example of a sequence that is not representative of the viral features of a virus. The presence of the string in the code of a virus is not indicative of malicious intent any more than it would in the code of a benign file. The correlation in this case is based on the idiosyncratic nature of the viruses from this family, and is unlikely to extend to viruses from different families. Other examples of retained features that were not representative of viral properties include padding bytes – sequences of several repeated bytes used as buffer, such as the string "0000000000000000". These sequences are part of data sections of the executables, and it would be very easy for a programmer to modify that part of the virus and thus avoid detection.

Binary Sequence	Sequence Length	Support
77 97 100 101 32 105 110 32 69 110 103 108 97 110 100	15	20%
77 97 100 101 32 105 110 32 69 110 103 108 97	13	20%
97 100 101 32 105 110 32 69 110 103 108 97 110	13	20%
100 101 32 105 110 32 69 110 103 108 97 110 100	13	20%
77 97 100 101 32 105 110 32 69 110	10	25%
97 100 101 32 105 110 32 69 110 103	10	25%
100 101 32 105 110 32 69 110 103 108	10	25%
101 32 105 110 32 69 110 103 108 97	10	25%
Etc...		

Table 3: Binary Sequence Features and their support in virus family ARCV

Utilizing the sequence above in the classification of viruses will therefore have the following three results:

1 – The sequence is unlikely to be found in uninfected cases. A classifier using it as a feature is therefore unlikely to produce false negatives.

2 – The sequence will be found in the training examples from which it was extracted, and might be found in variants of these training examples. Therefore the feature may help a learning agent classify members of this virus family.

3 – However the sequence is unlikely to be found in unrelated viruses, which in all probability would not exhibit the same idiosyncrasy. The feature's usefulness is therefore limited; more features are needed to correctly classify viruses from different families.

Figure 4 depicts the standard way to search for features, wherein sequences are scanned from each virus in the training set and their frequency is recorded. Only features with a minimum support in the training set are retained. In experiments where we harvested features using this method, without regard to virus families, we noticed that some of these idiosyncratic properties appeared with higher than average support within the corpus of viruses. Because they were present in several viruses from the same family or few families, such long sequences appeared to be a striking abnormality, but these occurrences did not account for significant overall support in the general corpus of viruses.

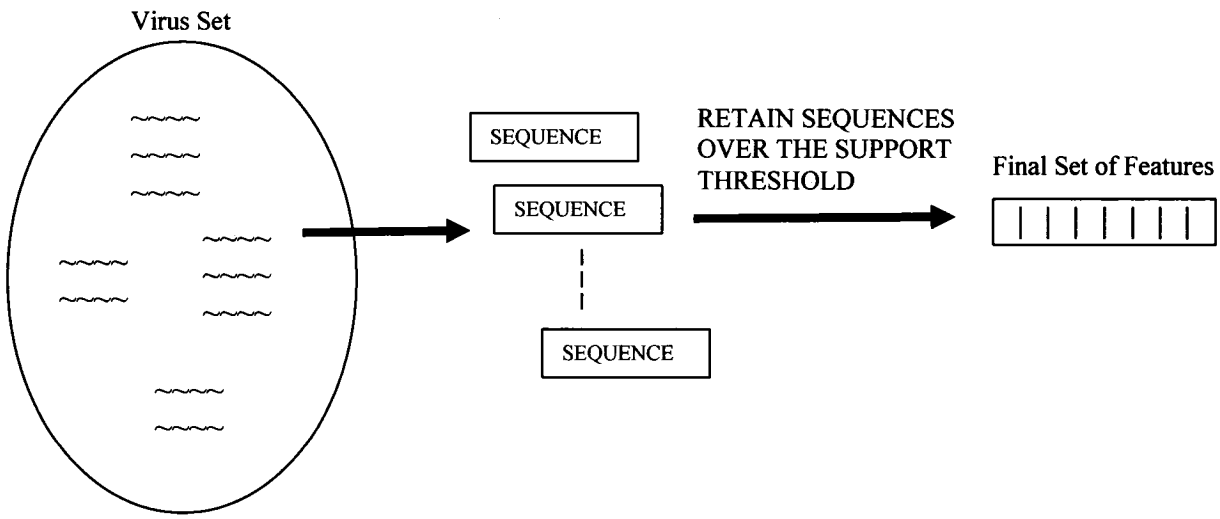

Figure 4: Traditional feature selection process

In previous research [34] machine code was output as hexadecimal data in lines of 16-byte-long strings. Each line displayed was then considered a feature representative of a particular code segment, and from that point on was treated as a text. This generated many features, many of which without much support, if any, in the corpus of infected files. Those with support must have come from large segments of code common to a number of viruses. Table 4 shows the number of such 16-byte-long sequences found, using this method, for a given overall support threshold in a dataset of 1338 viruses. The results show that long sequences are unlikely to have high support.

Minimum Support	**40%**	**10%**	**7%**	**5%**	**2%**	**1%**
Number of Features	1	2	2	82	99	378

Table 4: Number of 16-Byte Long Sequences for a Given Overall Support

It should be noted that the feature appearing with 40% overall support is:

"0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0"

Likewise, the second feature appearing with 10% overall support is:

"144 144 144 144 144 144 144 144 144 144 144 144 144 144 144 144"

Finally, several other features with higher support consist of short sequences appended to sections of the previous two. Such features may be common to several related viruses from the same family, but they would not be of much help to classify viruses in the real world, where new types of viruses represent the most dangerous threat. Similarly, a 5% support consists of a minimum of 67 viruses, which can represent as few as 2 or 3 families. Thus to provide a classifier with a large enough feature set to learn an effective concept, the minimum support would need to be very low. If a scanner were to inspect every possible sequence in a large dataset as described in Section 3.1.1, this would entail

tremendous computational work, together with very large memory requirements in order to record and keep count of every encountered sequence.

We have also found in experiments that, even with the above set of 378 features, most positive examples score very sparsely over the feature set, often only scoring for one or two features, apart from the "0" and "144" padding sequences. Several exceptional examples would score profusely, thus revealing the fact that they share a large common code base. Furthermore, the negative class, for the most part, scored for none of the features, save for some padding sequences. Thus, a classifier's role would be mostly limited to rudimentary pattern matching, with hardly any machine learning component.

This clearly indicates that this feature set is not conducive to machine learning, and does not provide a classifier the tools necessary to learn an advanced concept. The next section explains how our feature selection process considers virus taxonomies when selecting features: it produces a set of features that permit the use of machine learning algorithms while maintaining a reasonable size.

3.2 Feature Search

As an executable is comprised of a long continuous sequence of bytes, finding useful features without the intervention of an expert poses a particular challenge. Our feature extraction method conducts an exhaustive search for sequences in a hierarchical manner on a large dataset, and yields a final set of features with maximized usefulness. This system makes it possible to harvest features from a large dataset without having to partition it. It also allows the user to adjust different levels of laxity for the selection criteria, and therefore yield a reasonable number of features.

3.2.1 Sequence Scanning

We propose an exhaustive search for n-grams, short sequences of n bytes, where n-grams starting at every byte of the files' machine code are recorded. In our experiments, we varied sequence lengths between 3 and 8 bytes long, which represents less than half that of previous research [34]. Figure 5 illustrates the search process, which can be compared to a scanning window moving across the binary code and examining every 6-byte long sequence.

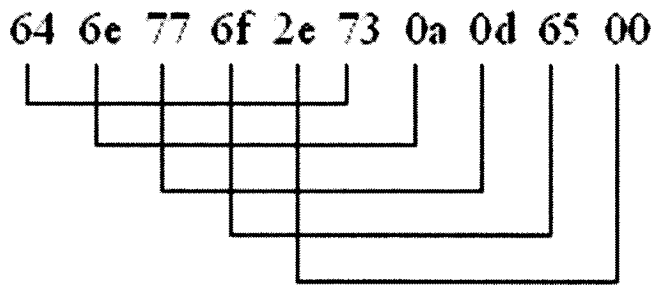

Figure 5: All 6-grams from a short byte sequence.

Sequences are scanned from each virus in the training set and their frequency is recorded. Features with a minimum support in the training set are retained. This scanning method has the capacity to detect recurring sequences that may or may not occur within large blocks of code, and it will therefore detect a considerable number of sequences. To reduce the complexity of the task, sequences are recorded in a dynamic tree structure where each internal node represents a byte value, and each leaf contains the frequency count of the sequence characterized by the values of its ascendants. Thus all recorded sequences can be stored in memory, and can be accessed for count updates through a binary search.

This scanning method is more flexible in detecting sequences that may not occur within large blocks of code found in several viruses. However, as it will detect a much larger number of sequences, it will be more difficult for the classifier to handle the resulting feature set – a task that, even without this extensive search, already had large memory requirements and that necessitated partitioning the feature space into subsets for use with separate learners [34]. The next section explains the feature selection process used to reduce the feature set, and that makes it possible for the classifier to work without an extraordinary amount of memory or having to partition the feature set.

3.2.2 Selecting Cross-Family Features

In this feature extraction scheme, we are interested in extracting features that represent viral properties of the positive examples – as opposed to idiosyncrasies of specific virus groups. Because viruses in the same family often share significant portions of code, long sequences may appear to have high support within a given dataset and lead to believe that it represents a viral property. However we wish for genuine viral properties to carry over to viruses from different families. We strive to obtain a reasonably large number of short generic features that permit our classifier to achieve superior results when classifying test cases of any virus family, even in the absence in the training set of any other instance from that family.

Our method conducts the search in a hierarchical fashion. In our experiments, it consists of three steps as shown in Figure 6. In the first step, we scan all instances in each sub-class of viruses in our dataset. In the second step, we build lists of features for each family, and in the third step, we reduce the feature set by eliminating weak features that are only relevant to their respective family. Our method makes it possible for the classifier to work with our complete dataset without requiring an extraordinary amount of

memory or the need to partition the feature set in the learning stage. The method is scalable to datasets of any size.

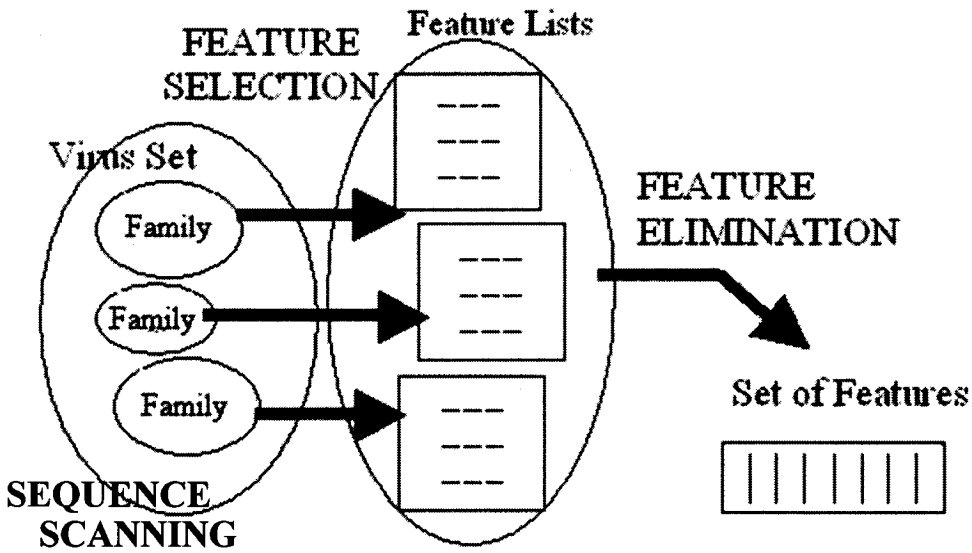

Figure 6: Hierarchical feature selection process.

The search, as shown in Figure 6, involves three main steps, according to the following scheme:

1. **Sequence Scanning:** Scan all viruses within a given virus family, as described in 3.2.1, and record their frequency within that family.

2. **Feature Selection:** For each virus family, construct a list of all features that meet a given support threshold within that family.

3. **Feature Elimination:** Consolidate family feature lists, and retain the features that meet a given support threshold among the feature lists.

We found this method to be scalable to large datasets for two reasons. Firstly, sequence scanning is conducted independently on smaller subsets. As storing records of all sequences in a large collection of viruses can require more memory space than the size of the collection itself, compiling feature lists for each virus family one at a time allows the system to reuse memory space and reduces its overall requirements. Secondly, because for each family a shorter list of selected feature is compiled, the feature elimination step is left with fewer sequences to match and count, which greatly reduces the complexity of the task.

The FEATURE_SEARCH algorithm, shown below as Algorithm 2, makes it possible to adjust different settings according to how general the features are to be, and how rigorous the feature elimination. Parameters can be given to our feature search function as follows:

1. Sequence Length Setting:

In the sequence-scanning step, the ***sequence length*** can be specified. The shorter the length, the more likely the feature is to have general relevance in the dataset. But a short length will yield a larger number of features in each virus family. The length of the sequence is specified as a number of bytes.

2. Intra-Family Support Settings:

In the feature selection step, the ***intra-family support*** threshold of features within each virus family can be specified. Sequences occurring at or above a specified frequency are retained as candidate features. This puts a constraint on the number of sequences obtained for each virus family. Because there are a variable number of instances in each virus family, the intra-family support is specified as a percentage.

In addition, a maximum number of features per virus family can be specified. The reason behind this is the fact that, in some families, many viruses can be almost identical. A moderately low support threshold could then produce an overabundance of features for that family. As well, in families consisting of a small number of viruses, a few very similar variants could cause a very large number of features to pass the *intra-family support* threshold.

We therefore specify a number of features as ***intra-family limit***, which is applicable to all virus families regardless of their size. If the feature selection yields an amount of features that exceeds this limit, then the *intra-family support* is increased by one instance, and the feature list is rebuilt. To achieve this, the *intra-family support* (initially a percentage) is first converted to a discrete quantity and then incremented.

3. Inter-Family Support Setting:

In the feature elimination step, the ***inter-family support*** threshold of features within the general corpus of viruses can be set to a desired value. This ensures that only those features that appear with a high enough inter-family support, as specified, are retained. This step discards unwanted features that are found to occur frequently in one virus family but that are exclusive to that family. This parameter is specified as a number of virus families.

Algorithm 2 FEATURE_SEARCH (S, len, intraSup, intraLim, interSup)

Input: A non-empty set of viruses, S, classified in virus families.

Input: A non-zero sequence length, *len*.

Input: The intra-family support, *intraSup*, as a percentage.

Input: The intra-family limit, *intraLim*, as a number of features.

Input: The inter-family support, *interSup*, as a number of virus families.

Output: A set features, F, representing common characteristics of the viruses in S.

1: **for** each virus family S_i in S **do**
2: **for** each virus V_{ij} in family S_i **do**
3: record all sequences of length *len* found in V_{ij} (without repeats)
4: compute minimum incidence $intraMin_i$ = *intraSup* x number of viruses in S_i
5: build the list L_i of M_i sequences with support of at least $intraMin_i$ in S_i
6: **if** $M_i > intraLim$
7: increment $intraMin_i$ and go to 4:
8: build the set of features F of sequences with support of at least *interSup* in S
9: return F

When recording the frequency of occurrences of sequences within virus families, multiple occurrences in the same virus instance is counted as one, such that the count of a sequence represents the number of distinct files in which it appears. The total number of features in the final list can easily be controlled by varying the *inter-family support* threshold. Table 5 shows how increasing that threshold can dramatically decrease the number of candidate features. Experiments were done on a dataset of 7 families, comprising a total of 108 files. A description of this dataset, listing the virus families it contains, is given in Appendix A.1. The number of features selected in each virus family is displayed for *sequence lengths* of 4, 5 and 6. The *intra-family support* threshold was

set to 40%, with an *intra-family limit* of 500 features. During the consolidation phase, features present in at least 2 or 3 families were retained, and are shown in Table 5.

Sequence Length	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Final Features (support = 2)**	**Final Features (support = 3)**
6	182	458	329	42	246	303	310	317	3
5	200	442	356	60	264	314	327	342	14
4	231	416	378	103	278	320	331	360	35

Table 5: Binary Sequence Features and their support in virus family ARCV

Unlike the traditional method shown in Table 4, the final features generated in this experiment do not correspond to idiosyncratic properties, such as padding (« 00000000 ») or labels (« Made in England »). The results show that:

1 – While many features – often hundreds – are found in abundance in each virus families, only a few represent viral properties common to more than one family.

2 – The longer the features, the less likely they are to be common to more virus families, indicating that they over-fit the examples.

In feature search experiments, we found that the longer the feature, the lower the *inter-family support* threshold needed to be, due to the fact that long sequences occur less frequently in the general corpus. With a low threshold and long sequence length, it is possible to generate a set of features sufficiently big to cover the training set completely. However at a certain length, the support of long features becomes constant, indicating

that the remaining features are part of even longer signature-like features present in the same set of cases.

The benefits of a hierarchical feature selection are twofold. Scanning features within each family group offer a divide and conquer solution to the high memory constraints of window scanning. It is now possible to select every possible feature in the binary code of executable and write to file those meeting the given support threshold. Finally, the consolidation stage ensures that only those features that have a high enough inter-family support are retained. This step discards unwanted features that are found to occur frequently in any single virus family but that are exclusive to that family.

This method ensures that features represent viral properties that are common to many types of viruses. Because we conduct an exhaustive search for all n-grams of a short constant length, we can expect all useful features to be included in our final feature set. In the next section, we verify the true usefulness of these features by testing our classifier in real world-like conditions.

3.2.3 Negative Class

In previous research [1, 42], where a large feature set was constructed from the constant regions of infected files, features were eliminated using the negative set. With a sizeable negative set of thousands of files on hand, all features appearing in more negative examples than a chosen threshold value were pruned out of the candidate set, leaving a much smaller number of features to be used by the classifier. We opted against using negative examples to downsize our feature set, as all of our final features are guaranteed to represent characteristics common to several strains of viruses. As explained in Section 3.1.1, these characteristics, such as the actions of reading, writing and deleting, may not

be cause for concern if encountered on their own, but may be indicative of malicious instructions if found concurrently in the same file. In the related field of text categorization, Koppel et al. [23] investigate search methods for finding corpus-independent features and warn against reducing a feature set to only "those that, individually, discriminate well on the training corpus". They explain that, while the number of potential features may be overwhelmingly large, eliminating a vast majority of features through direct feature selection methods can "sometimes do harm by pre-empting the learning algorithms they are meant to serve: the learning algorithms themselves, by taking into account dependencies among features, eliminate useless features in more subtle ways".

We also refrain from searching the negative set for features that could help identify non-viruses. Gryaznov [14] makes use of negative heuristics, a set of rules that are true for non-virus programs. But he concedes that virus writers easily fool these scanners by injecting static pieces of code with the only purpose of triggering negative heuristics. Another problem with this approach is that it is impossible to get a set of negative features that would be representative of even a fraction of all non-viruses in existence, as the number and variety of non-viral programs are astronomical. The number of normal programs greatly outnumbers the number of viruses ever written, and so it would be impracticable to attempt to distinguish the latter using features representative of the former. Hence, in classifying viruses, we opt to use only viral features.

3.2.4 Data Representation

Once a set of sequences has been produced by our feature selection, the instances in the training set, as well as those in the test set, are represented using those features, where each is a discrete attribute taking a value of 1 or 0, respectively for whether or not the sequence is present in the example. Table 6 shows examples represented in the feature

space, where the last attribute is the class, and can take one of two possible values (*pos* or *neg*) for when the example is positive or negative, respectively.

1,0,1,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,0,1,1,0,0,1,0,0,0,pos
1,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,1,1,0,0,0,0,0,0,1,0,0,0,0,1,0,0,neg

Table 6: Representation of examples using a feature set of 38 sequences.

In experiments where we focused on finding longer sequences by searching for sequences of variable lengths and discarding shorter ones for longer ones, many long sequences were found to have adequate support. However the retained sequences were highly correlated with a specific family of viruses. This resulted in these undesirable effects:

1 – Test cases from a virus family that was used during feature extraction would predominantly score for features found in that family, but not for other features.

2 – Test cases from a virus family not used during feature extraction would not score well for any features, causing the classifier to perform poorly in this particular context.

Under their new representation, not only were viruses clearly demarcated from the negative class, but also it was possible to visually identify viruses belonging to the same family, as shown in Table 7. The use of over fitted features made classification a near-trivial task, but most importantly, a classifier would not perform well with test examples from a virus group that was not used during training.

1,0,0,0,1,1,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,pos
1,0,0,0,1,1,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,pos
1,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,pos
1,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,pos
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,neg
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,neg
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,neg

Table 7: Example of clearly demarcated viruses from the same family and negative examples.

3.3 Evaluation Method

While assessing the effectiveness of anti-virus systems has been a topic of study, it has been noted that "the evaluation of anti-virus software is not adequately covered by any existing criteria based on formal methods" [13]. For a long time, commercial anti-virus systems have been tested on custom laboratory sets of known viruses, using a variety of tools and testing methodologies, which can lead to misleading or damaging information. Some researchers stress the importance of using the WildList when selecting a sample set [13, 25, 16], as viruses appearing on the WildList are a more recent and more significant threat than older viruses that are hardly ever encountered anymore. However, it has been argued that, as the WildList constitutes only 1 to 2% of all know viruses, scanners detecting a larger share of the whole set of known viruses "are more likely to catch something new than scanners only focusing on the 'WildList' viruses" [29]. Other

methods – retrospective or proactive testing [25, 26], obfuscation-based techniques [9] – have sought to more accurately evaluate how well different scanners detect new threats.

Machine learning focuses specifically on the classification of unseen examples, but the evaluation method is key to accurately measuring the effectiveness of a classifier at detecting completely new viruses. Existing machine learning research has relied on evaluation methods that overlook an important characteristic of viruses: that many viruses are inter-related. Two distinct virus examples may be variants, morphed replicas; they may contain different information in their data sections, or may reside in different benign host files. Consequently, the evaluation of heuristics rules can be skewed if variants related to known viruses are used in testing as a measure of detection accuracy.

We consider this fact in our evaluation method and introduce a leave-one-batch-out (LOBO) scheme that ensures that our training and test sets are sufficiently independent, and that tests our classifier by simulating real-world conditions of new virus outbreaks.

3.3.1 Family-Based Leave-One-Batch-Out Methodology

As seen in Section 3.1.2, viruses belong to family groups and we showed in Section 3.2.2 that this taxonomy could be used to our advantage in feature selection. Likewise, to evaluate how well our classifier is able to classify new unseen viruses, we ensure that cases used in the test set come from a family group that was not used in either feature extraction or training of the classifier. We describe this method as follows:

Given a set of viruses categorized into N families, we partition the dataset into k family sets, each consisting of N/k families. These family sets are labeled

$$S_1 , S_2 , S_3 , \ldots , S_k$$

Prior to conducting experiments, virus family groups are selected at random and added to each family set. Feature selection is first carried out on subsamples of the virus sets only, then the corpus of benign programs is partitioned into ***k*** groups of equal size and distributed over the virus family sets for use in the evaluation step.

Once set partitioning is complete, each validation batch contains the same number of family groups (modulo k) – with the remainder number of groups, ***N mod k***, distributed to the first few sets. The **k** sets of benign programs are then distributed evenly to the **k** validation batches.

Experiments are then carried out on the dataset using a LOBO evaluation scheme with k batches in the following way:

1) For each sets $\mathbf{S_i}$ (i = 1 to k):

   Scan all viruses and generate feature lists of all byte sequences occurring with a frequency that meets the **intra-family support** threshold.

2) For ***(j = 1 to k)*** do:

   a) Consolidate feature lists for set $S_{train} = \{ \mathbf{S_i} \mid _{i \neq j} \}$ keeping only those meeting a given **inter-family support** threshold

   b) Train classifier on set $\mathbf{S_{train}} = \{ \mathbf{S_i} \mid _{i \neq j} \}$

   c) Validate classifier on test set $\mathbf{S_{test}} = \mathbf{S_j}$

Figure 7 gives an overview of our system using a family-based LOBO evaluation scheme. Each of the five batches uses family-independent sets, on which feature selection can be performed as described in Section 3.2. Feature selection is repeated on the set of viruses in the training subsamples for each LOBO round.

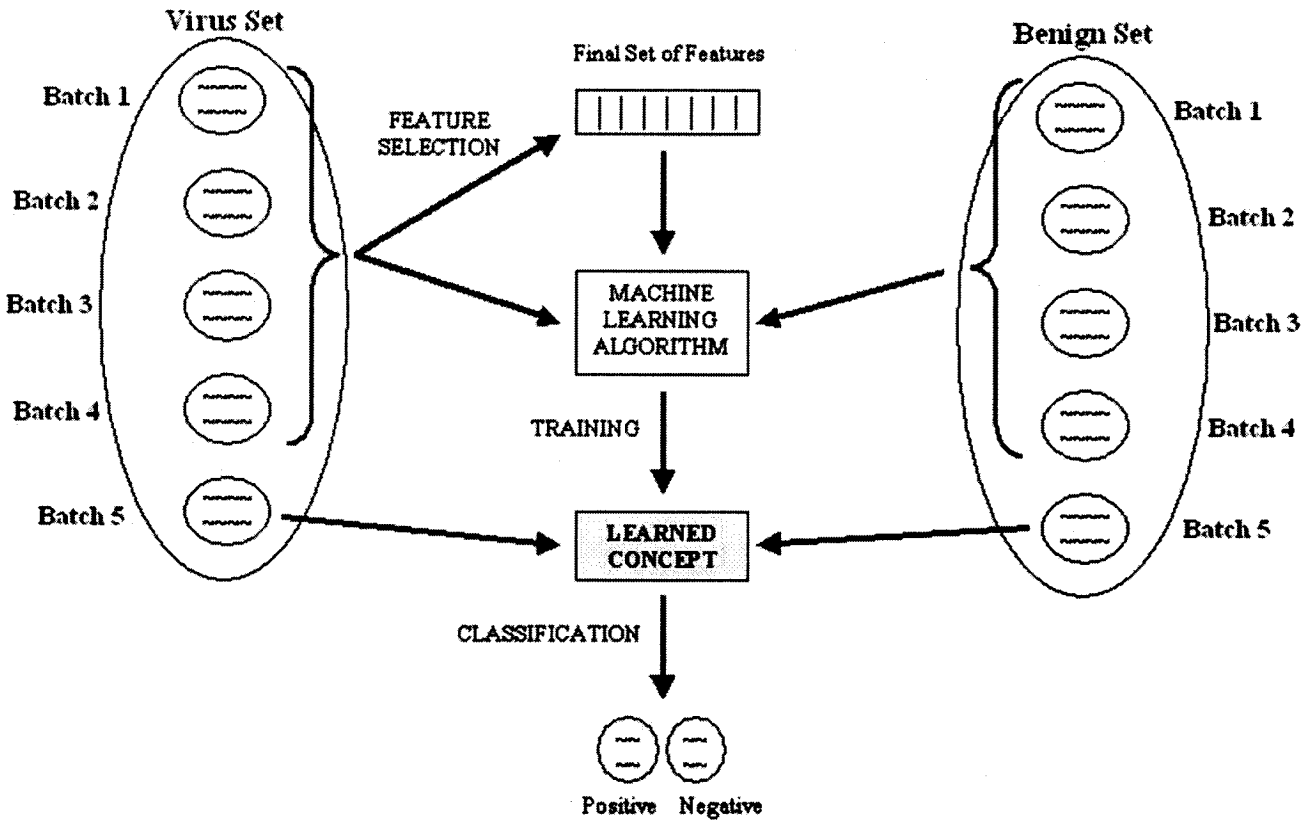

Figure 7: Feature selection, classifier training and evaluation using family-based LOBO validation (example using batch 5 as test set).

This evaluation scheme simulates an environment where a virus detection system is faced with the outbreak of a new unseen virus. Critics of traditional testing methods argue that "only completely new worms cause global outbreaks" [29], and stress that standard

random selection of test sets does not provide a reliable assessment of the proactive abilities of virus scanners. Our testing methodology evaluates the performance of our classifier in stringent conditions: by separating all virus families, it provides a more accurate measure of its proactive abilities and, in particular, its capacity to detect completely new instances.

Chapter 4

Experimental Results

In this section, we report the results of a number of experiments intended to demonstrate the advantages of our feature selection model and the validity of our testing methodology, as presented in Chapter 3. We compare our feature selection method, as described in Section 3.2, with a benchmark technique. In Section 4.1, we evaluate the performance of a classifier using a set of features selected according to our method and then using a similar set selected in the traditional way, and we show the clear advantage that our selection method has, as is made evident when tested by our rigorous evaluation scheme. In Section 4.2, we search for optimal feature criteria through a wrapper approach. Finally in Section 4.3, we demonstrate the effectiveness of our technique in real-world conditions, by showing evidence of its resilience and flexibility.

4.1 Comparison Against a Benchmark Model

Our experiments were carried out on a dataset of 3000 examples consisting of 1512 viruses and 1488 small benign executables. The viruses were taken from the collection used in previous research [34] and were classified into 110 family groups, based on their

filename, as they were previously labeled by a commercial virus scanner. We also further substantiated this classification by verifying that the viruses had comparable file sizes in each of the family groups. A description of this dataset, listing the virus families it contains, is given in Appendix A.2. The 1488 benign executables are small programs – less than 40 KB in size – that were gathered from several desktops using various versions of the Windows operating system, i.e. Window 95/98, Windows NT, Windows 2000 and Windows XP.

We compare our method with a benchmark model where feature selection is done according to a traditional criterion of the general support in the training set. This method mirrors the model used in previous research [34], where each 16-byte line of code is considered a feature. Hence we examine each consecutive 16-byte sequence in every virus in the training set, and retain those appearing with sufficient support. We chose a support threshold of 1% of the training set, as we found that it yielded a reasonable amount of 16-byte sequences. However, LOBO validation was done in the same way as in our hierarchical selection scheme – that is using, for each validation batch, the same training and test sets. Thus the classifier was evaluated on the same test sets of new unseen families as our model was.

We used a family-based LOBO validation scheme, where each of the five batches consists of a viral set of 22 virus families – one fifth of the total number of families – and one fifth of the files in our benign set – i.e. 297 or 298 executables. Every experiment comprises of five separate trials: during each trial, feature selection is conducted on the training set, a final set of features is generated and then both the training and test sets are represented in the batch-specific feature space. Finally the feature list, and training and test sets are given, in this form, to a learning agent for classification.

For our hierarchical model, we chose a *sequence length* of 8, which represents half that of the traditional method. We set the *intra-family support* threshold to 40% with an *intra-family limit* of 500, and the *inter-family support* threshold to 3. These settings yielded a final set of features of reasonable and comparable size to our benchmark test.

Table 8 shows the number of features, averaged over the five LOBO validation batches, of both our hierarchical model and the traditional model. Although virus families do not contain exactly the same number of individual viruses – and thus each batch consists of a slightly different number of cases – the minimum support by and large within the whole dataset in our hierarchical feature selection model can be estimated to the product of the two support parameters (intra-family support of 40% and inter-family support of 3 families out of 110), whereas the minimum general support in the benchmark tests were set to 1%. The overall feature support in our hierarchical model and the traditional model, averaged over the five LOBO validation batches, are comparable, as shown in Table 8.

Model	**Average Number of Features**	**Minimum Expected Overall Support**
Hierarchical	428	0.4 x (3 / 110) = 1.09%
Traditional	377	1%

Table 8: Feature number and support, for the hierarchical (Sequence length of 8 and intra-family support threshold of 3) and traditional models

4.1.1 Detection Accuracy

Since the features used in both models have comparable minimum overall support, our experiments provide a reliable comparison of the feature selection methods, based on the usefulness of each set of features in classifying new unseen viruses. Once the final feature set was obtained, we represented our positive and negative data in the feature space, using 1's and 0's to indicate whether or not a given example contained each feature. For classification, we chose decision tree learning. These types of classifiers work well with disjunctive attribute expressions and discrete output values [27]. This kind of classification function also mirrors most of the mainstream detection methods, such as signature matching. Its very intuitive decision making process will make it possible to study certain aspects the classification, such as decision complexity using the number of leaf nodes in the tree, in Section 4.2. Our experiments were done using WEKA's ID3 tree learner and yielded the results displayed in Table 9. We show the overall classification accuracy, the false positive rate and the detection accuracy (or true positive rate). More experiments, using different types of classifiers, were conducted and are shown in Appendix B.1.

Model	**Overall Accuracy**	**False Positive Rate**	**Detection Accuracy**
Hierarchical	93.29 %	4.16 %	90.32 %
Traditional	63.52 %	14.02 %	47.83 %

Table 9: Experimental results of the ID3 classifier using family-based LOBO validation, for the hierarchical feature search (sequence length of 8 and intra-family support threshold of 3) and the tradition search method.

Our experimental results indicate that a virus classifier can be made more accurate by using features that are representative of general viral properties. Whereas traditional anti-virus detection methods and most of the existing machine learning research rely on signature detection, or heuristics loosely based on signatures, our system focuses on short sequences with demonstrated usefulness in more than one virus family. With 90.32% detection accuracy, our system outperforms sub-50% rates of traditional detection methods and achieves better results than leading research in the field [34], which performs at a detection rate of 47.83% when tested under our more stringent evaluation method – significantly less than its reported best detection rate of 97.76% with the traditional cross-validation evaluation. However, the same classifier, using the set of features generated by our feature selection process, was able to achieve a high overall accuracy, while maintaining a lower false positive rate.

Experiments using different classifiers, which are presented in Appendix B.1, show results that are consistent with the above. For more information on these experiments, please refer to the results and discussion in Appendix B.1.

4.1.2 Feature Utility

Our evaluation method averts the fortuitous benefits that could result from a classifier inadvertently using signature-like features. By granting access to the dataset in a way that is characteristic of real world conditions, we ensured that our system is expected to achieve the same detection rate in the real world as in testing. Our results show that our system using family non-specific features performs better than one using features based on general support within the dataset, despite a very liberal support threshold. This shows that systems do not perform as well with new unseen virus family as with new unseen instances when not discriminating between families. However a family-aware extraction

scheme does detect viral features across different family groups and can be expected to provide better protection against outbreaks of new viruses.

Table 8 shows that the number and minimum overall support of the features used in both experiments are comparable, whereas Table 9 reveals a significant performance discrepancy between the two classifiers. This demonstrates that the set of shorter features found by our selection scheme is better than the 16-byte long sequences found by the traditional method. To demonstrate how the individual features from our set are more general than the overfitted 16-byte long sequence, we show in Figure 8 the typical scope of 16-byte features over the training set, and in Figures 9 and 10, examples of the scope of two of the shorter features that we generated hierarchically at a sequence length of 8 and inter-family support threshold of 3.

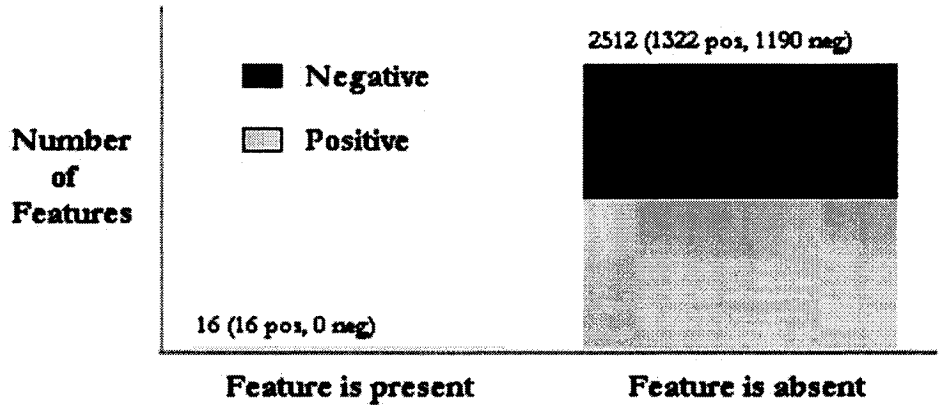

Figure 8: Typical example of the scope of a 16-byte long sequence

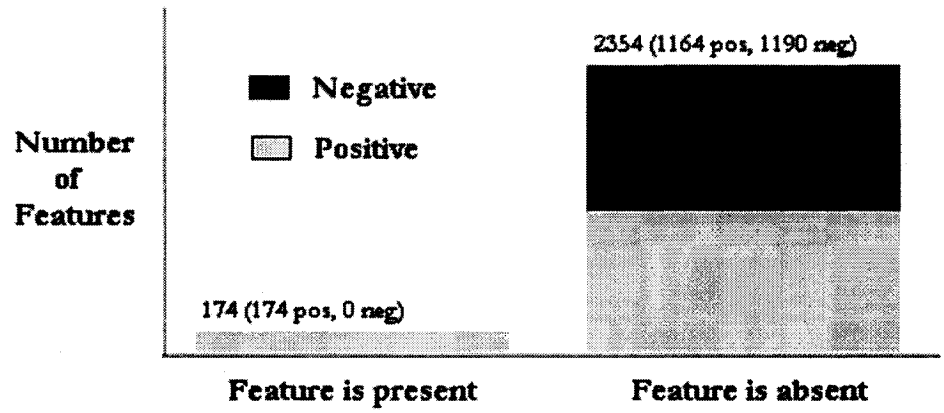

Figure 9: Example of the scope of an 8-byte long sequence generated hierarchically with inter-family support of 3 (example 1)

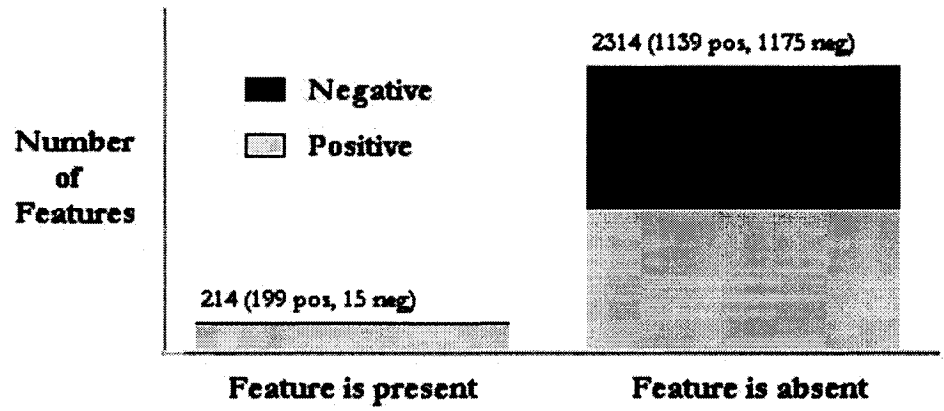

Figure 10: Example of the scope of an 8-byte long sequence generated hierarchically with inter-family support of 3 (example 2)

A feature which partitions the examples as seen in Figure 10, despite having impurities, is more useful than the feature shown in Figure 8. The utility of a feature can be assessed in proportion to the Information Gain that it provides. Information Gain is the expected reduction of entropy of the dataset caused by partitioning it using an attribute [27]. The attribute in Figure 10 provides an information gain of 0.05133 while the attribute in Figure 8 achieves only 0.00307.

The features generated by our model have a larger scope, as they occur in more training examples. Such is the case with the feature shown in Figure 9, as compared with the feature in Figure 8. Many 16-byte sequences can appear in as few as 16 viruses, making them pass the 1% overall support threshold. However, these features are without a doubt overfitted to these few training examples. Table 10 shows the average frequency of features in the positive and negative examples in the training set used in a batch during the experiment described in Section 4.1. The features also occur more frequently in the negative examples, as can be seen in Table 10. While this is intuitively not a quality for features, it shows evidence that our features do not overfit the data to the extent that traditional features do.

Model	**Average Positive Frequency**	**Average Negative Frequency**	**Negative/Positive Frequency Ratio**
Hierarchical	4.38 %	1.01 %	23.1 %
Traditional	1.95 %	0.22 %	11.3 %

Table 10: Average frequency of features in the positive and negative examples of the training set

4.2 Optimal Feature Selection Criteria

In this series of experiments, we wish to discover the optimal parameters for our feature search process presented in Section 3.2. Our family-based hierarchical feature extraction makes it possible to select features that meet a desired level of versatility, regardless of the size of the dataset. In this section, we use a wrapper to search through a wide range of search parameters in an attempt to optimize our feature selection. We decrease the sequence length from 8 to 3 bytes, while limiting the number of candidate features per virus family to a maximum of 500. To achieve this, our initial intra-family support threshold of 40% is automatically incremented until a tolerable number of features are preserved, as we described in Section 3.2.2. We also investigate different inter-family thresholds, as they directly affect the generality and final number of features. We explore a range of threshold values from 3 to 6. We report the results of our experiments in Table 11 and 12.

In both tables, the results represent the average performance of the classifier over five LOBO validation batches, using the family-based evaluation method described in Section 3.3. Table 11 lists the overall classification accuracy of the ID3 classifier for the range of feature search parameters described above, while Table 12 shows the virus detection accuracy, or true positive rate, as well as the false positive rate. More experiments, using different types of classifiers, were conducted and are shown in Appendix B.2.

Sequence length / Inter-family support	8	7	6	5	4	3
3	93.29%	95.66%	96.56%	97.63%	96.91%	99.89%
4	90.07%	94.15%	95.23%	97.36%	97.24%	96.66%
5	85.99%	93.11%	94.99%	95.78%	96.51%	95.58%
6	80.69%	93.03%	95.11%	95.96%	96.53%	96.85%

Table 11: Classification accuracy, in relation to different feature selection parameters (using the ID3 decision tree learner)

Sequence length / Inter-family support	8	7	6	5	4	3
3	90.31% 4.17%	93.92% 2.55%	94.69% 1.68%	96.11% 0.81%	95.85% 1.41%	99.77% 0.34%
4	84.28% 4.77%	92.71% 4.50%	93.86% 3.50%	96.44% 1.68%	94.45% 1.14%	94.58% 1.28%
5	80.47% 5.98%	92.40% 6.45%	93.54% 3.63%	93.85% 2.22%	94.73% 1.68%	92.26% 1.21%
6	64.42% 4.03%	92.28% 6.45%	94.88% 4.62%	94.61% 2.76%	94.51% 1.48%	95.12% 1.41%

Table 12: Virus detection accuracy (top) and false positive rates (bottom), in relation to different feature selection parameters (using the ID3 decision tree)

The results indicate that the classifier achieves better overall performance with shorter sequences steadily as the length decreases from 8 to 5. This is also true for the most part for the detection accuracy and false positive rate. This is indicative that shorter sequences are more versatile and are more valuable to the classifier. At length 5, performance seems to reach a peak, as the results at lengths 3 and 4 are not consistently better or worse, but they remain comparable to those at length 5 (save for the result in the top right corner, which we will discuss later).

The inter-family support generally yields better results when low. This is especially true for longer sequences – those of length 6 to 8 – where we observe a more rapid drop in accuracy as the support threshold decreases. In the case of longer sequences, this deterioration is likely due to the dwindling number of features, as we will examine later. The average number of features for each inter-family support and sequence length pair is shown in Table 13. At low support thresholds, searches for sequences of lengths 6, 7 and 8 generate sets of less than 100 features, and as low as 16. In the case of shorter sequences, the performance remains very good, thanks to the fact that a sufficient number of features are available to the classifier. However, we notice that performance is hindered when the classifier is working with a set smaller than 200 features. Experiments using different types of classifiers, which are presented in Appendix B.1, show results that are consistent with those of the ID3 decision tree. For more information on these experiments, please refer to the results and discussion in Appendix B.2.

Table 13 shows the average number of features in the set used by the classifiers at each of the five LOBO validation batches for each combination of feature selection parameters. The results highlight how the choice of feature has an influence on the classification accuracy. With the results reported in Table 11, we see that our classifiers generally perform better with a larger set of features, but we also notice that certain feature sets comparable in number lead to different performances depending on the parameters with

which the feature selection was carried out. Some sets yield better accuracy than similar sets equal or bigger in size. In particular, the sets diagonally adjacent – down and to the right – to all feature sets of lengths between 6 to 8, are most often significantly smaller than their diagonal neighbour, or otherwise comparable in size, and yet they, all but for one, lead to better performance. The exception is minor: it is the average set of 248.4 features (for sequences of length 6 and support threshold of 4 virus families), which leads to an accuracy of 95.23% compared with an accuracy of 95.66% for the set of 532 features (for sequences of length 7 and support threshold of 3 virus families).

Sequence length **Inter-family support**	**8**	**7**	**6**	**5**	**4**	**3**
3	427	531	690.4	879	1150	1633.6
4	108	166.4	247.4	365.8	531.6	956.4
5	35.2	65.8	111	186.6	333.6	654
6	16.6	33	59.2	111.6	223.6	479

Table 13: Number of features, averaged over the 5 LOBO validation batches, in relation to different feature selection parameters

To judge how useful each of the feature sets are to the classifier, we look at how the features are used in classification. Because we used the Id3 decision tree classifier, it is easy and intuitive to understand the way that features are used. We are interested in seeing how the qualitative aspects of features – sequences length and inter-family support – come into play in comparison to the quantitative aspect – the size of the feature set. To systematically evaluate the extent to which the learned concept uses the feature set, we

counted the number of leaf nodes in each decision tree. Table 14 presents, for each pair of sequence length and inter-family support parameters, the average number of leaf nodes over the five LOBO validation batches.

Sequence length **Inter-family support**	**8**	**7**	**6**	**5**	**4**	**3**
3	166	129.2	94.2	73.4	49.4	4.8
4	101.2	133	119	77.2	62.4	38.4
5	73	125.2	107.2	91	78.6	39.8
6	43.4	96.8	96	103	*79.5*	42.2

Table 14: Number of leaf nodes in the decision tree, averaged over the 5 LOBO validation batches, in relation to different feature selection parameters

The leaf node count reveals a peculiarity with the concept found in the top right corner – using the set of features of length 3 and inter-family support of 3. Despite a high level of accuracy – the highest found – we regard that classification as precarious. The concepts generated at these settings are surprisingly simple: while representing a training set of, on average, 1200 viruses and 1200 non-viruses, they consist of an average of 4.8 leaf nodes using an average of 3.8 features. We see that the large feature set of close to 2000 features had the effect of providing the learner with sequences that have a strong correlation to the viruses in the training set. While this correlation does persist in the test set, this undersized concept may not provide a reliable means of classifying viruses in general. Some leaves in the decision trees attribute the positive class and others attribute

the negative class. But in this case, the class is attributed based on the occurrence of few features, and sometimes as few as 2 or 3. While viruses and benign files may have broad similarities, based on similar code assembly and compilation processes, the presence of encryption algorithms, or similar trademark data sections, relying on so few short features may be an unsound strategy. Three-byte long sequences allow for 16 777 216 possible alternatives, making two given sequences unlikely to match out of pure coincidence, but the viruses and benign files in our dataset averaged several kilobytes in size, with some amounting up to 40 KB. While the number of possibilities for a 3-byte long sequence is still far greater than then number of sequences present in one file, it is clear that, with a large negative set, the probability of occurrence of a given string increases. The number of files handled by an active server or present on a computer (where executable files can also be very large in size), is considerable, and random occurrences of any given 3-byte sequence in these circumstances would be more frequent. Similarly, when the classification of a virus is based on a simple match of 2 or 3 sequences, it becomes easy to decipher and understand the concept it is based on. A virus writer could then simply remove the few flagged sequences from his code. Or worse, he could find out which sequences are used to classify benign files, and then insert these is any virus he creates, thus giving it "benign characteristics". We therefore favour more complex concepts, which combine the presence and absence of many features, as they would be more resistant to random sequence occurrences and intentional insertions.

We also notice that for longer sequences, the number of leaf nodes decreases as the inter-family support increases. The higher number of leaf nodes at low support values is indicative of a more signature-like quality of the features. As longer features fit more specifically certain types of training examples – or, in our case, virus families – more decisions are needed to match virus families to features. These features are also more specific to particular cases; therefore in the absence of some of them, the classifier would not be able to include some virus families in the concept. This can be observed in experiments with short sequences of length 7 and 8, where as the inter-family support

decreases, and along with it the number of features in the final set, the classifier's performance drops significantly compared with experiments with shorter features.

Experiments with longer features of length 3 to 6 do not experience this trend, as the performance remains comparable or decreases only slightly – apart from the exceptional result with 3-byte long sequences with support of 3 families. This demonstrates that the feature sets, while decreasing in size, contain general enough features that are capable of covering the majority of the training and test sets. With more versatile features and a more adaptable feature set, the classifiers are therefore able to generate a high-quality concept substituting some features for others that might not be present in feature sets at higher support thresholds. We observe that for the 3 shortest sequence lengths, in fact, as the support threshold increases and the feature sets are reduced, the number of leaf nodes increases. This confirms that features with higher entropy are replaced with other features, or combination of features, with lower entropy from smaller feature sets at higher inter-family support thresholds – likely causing the classifier to partition the training set in a less broad and direct fashion. We consider this phenomenon to be a desired effect at the shortest sequence lengths, as a too large and general feature set can lead to a poor concept such as the one at length and support of 3, which yields the best overall results.

4.3 Classification Robustness

We illustrate the advantage of classifiers using the sets of features as described in Section 3.1 and produced by the method we presented in Section 3.2. We saw in Section 4.1.1 that, when evaluated on a test set of new virus families, our set of features lead to a much higher classification accuracy than that using a set of 16-byte long sequences generated by the traditional method. To better examine the effects of sequence length in classification, we generated a set of 16-byte long sequences using our hierarchical model,

at different inter-family support levels, in order to compare it with the feature set of 5-byte long sequences, which yielded the best results overall in our experiments. Table 15 shows the overall accuracy as well as the false positive and false negative rate of the resulting classifiers, using the ID3 decision tree.

Sequence length **Inter-family support**	**16**	**5**
3	88.55% FP = 6.6% FN = 33.3%	97.63% FP = 0.8% FN = 3.9%
4	79.12% FP = 2.1% FN = 39.9%	97.36% FP = 1.68% FN = 3.58%
5	76.93% FP = 1.5% FN = 44.8%	95.78% FP = 2.2% FN = 6.2%
6	76.93% FP = 1.5% FN = 44.8%	95.96% FP = 2.8% FN = 5.4%

Table 15: Performance of two ID3 decision trees, averaged over the 5 LOBO validation batches, for 16-byte and 5-byte long features.
(Overall Accuracy; FP = False Positive; FN = False Negative)

Firstly, the classifier using the feature set of 16-byte long sequences produced by our hierarchical search has much higher overall accuracy than that of the classifier using the feature set generated the traditional way, as reported in Section 4.1.1. Secondly, while the false positive rates remain low for both classifiers, we observe that the false negative rates are very high for the classifier using 16-byte long sequences. In fact, the majority of the overall inaccuracy is due to the classifier's inability to detect viruses. Not surprisingly, this deficiency is exacerbated as the feature sets decrease in size, and thus fewer features are available to the classifiers. However the classifier using sets of short 5-byte long sequences is still able to achieve a high detection rate – with a false negative rate barely exceeding 6%, while maintaining a low false positive rate of under 3%.

A large problem in virus detection is the fact that constant sections of code do not provide a reliable means of detecting viruses. We saw in Section 2.1 that these sections are often tampered with in an attempt to fool detection systems: variants created by virus writers specifically intended to break existing commercial anti-virus systems, polymorphic and metamorphic viruses automatically alter their structure or code, making them much harder to detect, first outbreaks of previously unseen viruses challenge anti-virus systems with viruses for which no constant sections have been identified yet, etc...

We saw in 4.2 that performance is much less affected by a reduction of features sets when shorter features are used, and the results in Table 15 clearly demonstrate the advantage that using short and more general features proffers to a virus classifier. The advantage, as made evident by the reported higher overall accuracies, provides improved resistance to code alterations. A virus, of which constant code sections were modified, would then score fewer matches with features in the set used by the classifier. This would have a more acute effect on classifiers using long features, as they rely more exclusively on particular features, and it would render them incapable of detecting variants or brand new viruses written for that very purpose. Sets of shorter features, however, permit the

classifier to use a different combination of features to cover new viruses. We observe this phenomenon in Table 15 – as well as in Section 4.2 – where reduced feature sets do not consistently or considerably deteriorate the accuracy of the classifier.

Our results show that, when not discriminating between families, virus classifiers do not perform as well with new unseen viruses that have a larger degree of independence from instances present in the training set. However a family-aware extraction scheme does detect viral features across different family groups and can be expected to provide better protection against the first few viruses of a new outbreak, before signatures are discovered. In addition, as seen in Section 4.2, sets of shorter features generally lead to more accurate detection rates than feature sets of equivalent size but that use longer features. Table 15 shows that this performance is also achieved while maintaining a low false positive rate, on par with that of classifiers using very long, signature-like features. Through varying our feature selection criteria, we have obtained results that outperform previous models. In [34], the authors report a best-case false positive rate of 3.80% while achieving a detection rate of 97.43%. Our model, even through family-independent evaluation, can achieve a detection rate as high as 98.81% while maintaining a false positive rate under 0.87%, as shown in Table 24.

Our results also indicate that our set of features is more versatile in covering viruses in which constant code sections may have been intentionally altered to avoid detection. The absence of a particular feature is less likely to undermine a classifier if others can be used in its place. Our classifiers also possess a considerable advantage over signature-based methods, in that they use a combination of features to identify viruses, as verified by the number of leaf nodes reported in Section 4.2. This, as opposed to matching a single signature as sole indicator of an executable's viral properties, confers robustness and flexibility to our system.

When our system encounters new viruses, its adaptable detection strategy is more apt at defending against them, but it can also add these new instances to its knowledge bank and, with them, enhance its classification model. Thus it can keep up with new infection and obfuscation techniques and preparing as best it can for the next new outbreak. In our experiments, we found that the task of scanning virus examples to construct the feature lists takes an execution time much greater than both the tasks of converting the data to its new representation and training the classifier. However, our model can be updated without having to scan every example in the dataset over again. When adding new virus families, or when adding new instances to an existing virus family, only the feature list from those families needs to be constructed. Then the final feature list can be recompiled using the existing lists, saved from a previous iteration, and the newly constructed ones. The data, under their new representation, can at this point be given to a machine learning algorithm to build a classification model.

Classification could, in that way, be constructed locally by providing users with feature lists and letting them train their own custom classifier. As we described in Chapter 2, virus writers strive to write code that evades anti-virus systems. In this endeavor, virus writers have an advantage over anti-virus research, as they have access to anti-virus software. Thus, they can focus the efforts on breaking a specific anti-virus product. Because a great number of users worldwide use the same software, using the same detection algorithms, a virus that goes undetected by this software would have a chance to spread far and wide. This could be prevented by letting individual users build their own classification models, such that virus writers could not certain to defeat all models in one single sweep. Users could build classification models based on different feature lists (representing different feature support values) and could set different parameters for the training step. Thus, code that would go undetected by one software installation would not necessarily bypass all others, as different models would exist with different customizable specifications.

Chapter 5

Conclusion

We introduced a feature search method that focuses on selecting generic features that are applicable to different families of viruses. This ensures that our classifier is genuinely heuristic and does not rely on signatures, or a collection of overfitted features. In experiments testing our method against and another existing model, ours performed better. Our search method also took place without large memory requirements and was carried out in reasonable execution time, as listed in Appendix B.3. In both models tested, the traditional feature search and our family-based hierarchical search, the features selected and used by the classifier had comparable overall support within the dataset, indicating that our feature search method produces features that are more useful in detecting new unseen viruses.

We also presented a novel evaluation method for virus classifiers that tests more convincingly its ability to detect new viruses. Our method does not allow classifiers to use examples in training that are variants of viruses present in the test set. This denies them an unfair advantage that they would not have in real world conditions. Through

experimental results we showed that a classifier based on standard detection techniques from previous research for detecting previously undetected malicious executables performs significantly more poorly under our evaluation method. In contrast, classifiers using our feature search method achieved detection rates of new instances as high as 98% with a false positive rate under 1%.

An enduring problem faced by traditional anti-virus detection systems is the incessant outbreaks of new strains of viruses, against which they are unprepared to protect. To deal with this problem, virus scanners are updated regularly, sometimes daily, with the latest signatures. This, however, leaves a window of vulnerability, during which a computer is at risk of infection. And, as virtually all devices are interconnected through the Internet, and viruses spread at extraordinary speeds, any window of vulnerability constitutes a substantial risk. Our method addresses this problem in a proactive way by learning from past outbreaks and preparing for future ones. While traditional signature-based methods can protect reliably against known viruses, our method could be used to complement a signature-based system by extending its detection to new unseen viruses. Our experiments yielded results as good as 98% detection accuracy with false positives well under 1%, suggesting that our classifiers could offer very high new virus detection rates at the expense of low false positives. Using a cost matrix in the learning step, and imposing a higher penalty for misclassifying negative examples could likely further reduce our false positive rate. Using receiver operating characteristic (ROC) analysis with the results of our classification models, as shown in Tables 11 and 12, it would be possible for a user to easily select a particular classifier based on a desired trade-off between detection accuracy and false positives.

In a real word situation, benign files received by a computer or server would be expected to vastly outnumber malicious viruses, therefore maintaining a low false positive rate

when deployed would be essential for a detection system to be usable. But because our system does not use the negative class in the search for features, our false positive percentages are not expected to change due to a class imbalance in a deployment situation, which would be comparable to imbalanced test sets – with a larger number of negative examples than positive. Benign files of larger size could be more likely to contain more features, having possibly large portions of encrypted data. However, executable files that are widely distributed through Internet traffic or email, and thus efficient carriers of viruses, are typically of a small size. Therefore, the detection and false positive rates of our classifiers could be expected to be comparable to the experimental results found in Chapter 4.

5.1 Future Work

In future work we propose focusing on reducing the false positive rate. To do so, our experiments could include a larger number of negative examples. Past research has used a significantly greater number of benign files, either to select safe signatures or to train a virus classifier [1, 19]. This mimics real life usage, where, in most cases, considerably more uninfected files are exchanged than actual viruses. In our experiments, this would allow us to evaluate the false positive rates more accurately, and could help us pick the classifier with the lowest rate. Another way we could reduce the false positive rate is by training a classifier using a cost matrix and by setting a higher cost to misclassifying negative examples.

In the feature search, we would also like to explore a multi-level hierarchical approach. Virus taxonomies [3], as well as virus naming conventions [36], identify hierarchies of viruses, based on their construction or infection techniques. They show that some viruses inherit properties or evolve from others. Viruses are classified in a hierarchical way, with

low-level families belonging to parent families, and in turn to grandparent families, and so on. Two groups of viruses could then be classified as being two separate families, but they could belong to a common parent family, or grandparent family. A multi-level hierarchical approach could then focus on finding features focusing on a particular family, then generalize to the parent family, grandparent family, and so on, as desired. A multi-level hierarchical approach could also prove especially useful in reducing the complexity of the feature selection when using larger training sets. Clustering could also be done to validate the virus taxonomy, to insure that no bias exists, or to automate the classification of viruses into family groups.

We would like to conduct a feature search that considers multi-length sequences for use in the same classification model. This would entail additional operations to prevent redundant features, when shorter byte sequences happen to be sub-strings of longer sequences.

Some similarities may exist in fields where mining data containing long sequences poses a challenge. In fields such as bioinformatics and spam, or junk email, detection, techniques are used to analyze nucleotide sequences and long strings of alphanumeric characters. In spam detection, heuristics rules using words, common strings or text patterns, are often used to assign a score to email messages, and estimate their likelihood of being junk emails. Our feature search method may help automatically find features to be used with spam heuristics. Junk email is also commonly catalogued into different "spam categories" in a way similar to how computer viruses are classified into families. Research could be done to explore any possible application of this research to other fields.

Finally we would like to explore retrospective, or chronological, testing. As we introduced in Section 3.3, retrospective testing is a recent methodology for realistic

evaluation of virus detection systems [25, 26]. This would involve using a set of older viruses in the training set and a set of more recent ones in the test set. A number of experiments could be done by merging the training and test sets, and by constructing new test sets with successively more recent viruses. We could also optimize the evaluation method by increasing the number of validation batches. In fact, a "leave-one-out" validation scheme could be used, where each batch would consist of one virus family. This would involve much work in evaluating our classifiers, as the feature list consolidation part of the feature search, the conversion of data to their new representation, as well as training and evaluation, would need to be conducted for each batch. In an ideal evaluation scheme, we would conduct successive retrospective evaluations using test sets consisting of one virus family. This would require a meticulous organization of the data but would mimic a real-world situation of virus outbreak as closely as one could conceive of.

Bibliography

[1] Arnold, W. and Tesauro, G. "Automatically Generated Win32 Heuristic Virus Detection". Proceedings of the 2000 International Virus Bulletin Conference, 2000.

[2] Bishop, M. "An Overview of Computer Viruses in a Research Environment", Department of Mathematics and Computer Science, Dartmouth College, 1992

[3] Bontchev, B. "Analysis and Maintenance of a Clean Virus Library". Proceedings of the 3rd International Virus Bulleting Conference, 1993.

[4] Bontchev, V. "Are 'Good' Computer Viruses Still a Bad Idea?". Proceedings of EICAR Conference, pages 25-47, 1994.

[5] Bontchev, V. "Future Trends in Virus Writing". Proceedings of the 4th International Virus Bulletin Conference, pages 65-82, 1994.

[6] Briney, A. "Information Security Industry Survey 2000". Information Security Magazine.

[7] CERT Coordination Center. "Incident Reporting Guidelines". http://www.cert.org/tech_tips/incident_reporting.html

[8] Chen, T. M., "Intrusion Detection for Viruses and Worms". IEC Annual Review of Communications, vol. 57, Fall 2004.

[9] Christodorescu, M. and Jha, S. "Testing Malware Detectors". Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Boston, MA, USA, pages 34–44, July 2004.

[10] Cohen, F. "Computer Viruses - Theory and Experiments", IFIP TC-11 Conference, Toronto, 1984

[11] Dumais, S. and Chen, H. "Hierarchical Classification of Web Content". Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 256-263, Athens, 2000.

[12] Gordon, S. and Howard, F. "Antivirus Software Testing for the New Millenium". 23rd National Information Systems Security Conference, Baltimore, MD, USA, pages 126-140, October 2000.

[13] Gordon, S. and Ford, R. "Real World Anti-Virus Product Reviews and Evaluation – The Current State Of Affairs". 19th National Information Systems Security Conference. Baltimore Maryland, 1996.

[14] Gryaznov, D. "Scanners of the Year 2000: Heuristics". Proceedings of the 5th International Virus Bulletin Conference, 1999.

[15] Guyon, I. and Elisseeff, A. "An Introduction to Variable and Feature Selection". Journal of Machine Learning Research 3, 2003, pages 1157-1182.

[16] Helenius, M. "Antivirus scanner analysis based on Joe Well's list of PC viruses in the wild 7/1997". Virus Research Unit, University of Tampere, Finland. http://www.uta.fi/laitokset/virus/test1997.html

[17] Helenius, M. "A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities", Department of Computer and Information Sciences, University of Tampere, 2002. http://acta.uta.fi/pdf/951-44-5394-8.pdf

[18] Kephart, J. O. "A Biologically Inspired Immune System for Computers", Proc. of the 4th International Workshop on the Synthesis and Simulation, of Living Systems, pages 130-139. MIT Press 1994.

[19] Kephart, J.O. and Arnold, W. C. "Automatic Extraction of Computer Virus Signatures". 4th Virus Bulletin International Conference, pages 178-184, 1994

[20] Kerchen, P., Lo, R., Crossley, J., Elkinbard, G., and Olsson, R. "Static Analysis Virus Detection Tools for Unix Systems". 13th National Computer Security Conference, 1990.

[21] Kohavi, R. and John, G. "The Wrapper Approach". Feature Selection for Knowledge Discovery and Data Mining, H. Liu & H. Motoda (eds.), Kluwer Academic Publishers, pages 33-50, 1998.

[22] Kohavi, R. and John, G. "Wrappers for Feature Subset Selection". Artificial Intelligence, vol. 97, pages 273-324, 1997.

[23] Koppel, M., Akiva, N. and Dagan, I. "A Corpus-Independent Feature Set for Style-Based Text Categorization". Proceedings of IJCAI'03 Workshop on Computational Approaches to Style Analysis and Synthesis, Acapulco, Mexico, 2003.

[24] Koppel, M. and Schler, J. "Authorship Verification as a One-Class Classification Problem". Proceedings of 21st International Conference on Machine Learning, Banff, Canada, pages 489-495, July 2004.

[25] Lee, A. "Testing Heuristic Detection in a Real-World Scenario". Virus Bulletin, October 2004 Edition.

[26] Marx, A. "Retrospective testing – how good heuristics really work", Proceedings of the 2002 Virus Bulletin Conference (VB2002), New Orleans, LA, USA, Sept. 2002.

[27] Mitchell, T. "Machine Learning", McGraw-Hill, 1997.

[28] Mladenic D, Grobelnik M. "Feature Selection for Classification Based on Text Hierarchy". http://citeseer.ist.psu.edu/10425.html.

[29] Muttik, I. "Comparing the Comparatives", Virus Bulletin Conference, September 2001, pages 45-56.

[30] Neubauer, B. J. and Harris, J. D. "Protection of computer systems from computer viruses: ethical and practical issues." Journal for Computing Sciences in Colleges 18, no. 1, pages 270-279. 2002.

[31] Perriot, F., Ferrie, P. and Ször, P. "Striking Similarities". Virus Bulletin, May 2002, page 4-6.

[32] Real Time WildList, www.wildlist.org

[33] Ruiz, M. E. and Srinivasan, P. "Hierarchical Text Categorization Using Neural Networks", Information Retrieval, 5(1), pages 87-118, 2002.

[34] Schultz, M. G., Eskin, E., Zadok, E., and Stolfo, S. J. "Data Mining Methods for Detection of New Malicious Executables". IEEE Symposium on security and privacy 2001.

[35] Schultz, M. G., Eskin, E, Zadok, E., Bhattacharyya, M, and Stolfo, S. J. "MEF: Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables". USENIX Annual Technical Conference - FREENIX Track, 2001.

[36] Skulason, F., Solomon, A. and Bontchev, V. "A New Virus Naming Convention". Computer Antivirus Research Organization, 1991. http://www.caro.org/tiki-read_article.php?articleId=1

[37] Solomon, A. "A Guide to Evaluating Anti-Virus Software", 2000. http://vx.netlux.org/lib/aas04.html

[38] Ször, P. "Attacks on Win32". Virus Bulletin Conference, October 1998, Munich, Germany, pages 57-84.

[39] Ször, P. "Attacks on Win32 - Part II". Virus Bulletin Conference September 2000, Orlando, FL, 2000.

[40] Ször, P. "EPOCalypse NOW! (The Explosion of Win32 Worm Threats)". Virus Bulletin, July 2004, page 2.

[41] Ször, P., and Ferrie, P. "Hunting for Metamorphic". Virus Bulletin Conference September 2001, pages 123-144.

[42] Tesauro, G., Kephart, J. O., and Sorkin, G. B. "Neural Networks for Computer Virus Recognition". IEEE Expert, 11(4):5–6. IEEE Computer Society, August, 1996.

[43] Weigend, A. S., Wiener, E. D. and Pedersen, J. O. "Exploiting Hierarchy in Text Categorization". Information Retrieval, vol. 1, no. 3, pages 193--216, 1999.

[44] White, S. R., Swimmer, M., Pring, E.J., Arnold, W.C., Chess, D., and Morar, J.F. "Anatomy of a Commercial-Grade Immune System". IBM Research White Paper, 1999.
http://www.research.ibm.com/antivirus/SciPapers/White/Anatomy/anatomy.html

[45] White, S. R. "Open Problems in Computer Virus Research". Virus Bulletin Conference, 1998

Appendix A

Description of Data Sets

A.1 Partial Dataset

The experiment in Chapter 3, Section 3.2.2, was conducted on a dataset of 7 virus families, comprising a total of 108 files. The 7 families contained in the dataset are listed in Table 16.

ARCV.*	b-560.*	Burger.*
Ash.*	bammpc.*	
Australian.*	Black_Jec.*	

Table 16: Virus families in the dataset used for the experiment in Section 3.2.2.

A.2 Full Dataset

The experiments in Chapter 4 were conducted on a dataset comprised of 1512 viruses and 1488 benign executables, comprising a total of 3000 executable files. The 1512 viruses were classified into 110 family groups as listed in Table 17.

ac.*	bljec.*	Father_Mac.*	J-*.*	Predator.*
Akuku.*	Burger.*	Flip.*	jd.*	Proto.*
Alabama.*	BW.*	fri-*.*	Jeru.*	PS.*
Albania.*	c-*.*	Frodo.*	Jerusalem.*	PS-0.*
Am.*	ca170*.*	gc*.*	Keypress.*	PS-MPC.*
Amazon.*	Cheeba.*	Genvir.*	Leprosy.*	Rape.*
amst.*	civil.*	Gergana.*	Liberty.*	Rubbit.*
an*.*	CloneWar.*	Gotcha.*	Mayberry.*	Scream.*
AntiCAD.*	CX.*	Happy_New_Year.*	MG.*	Sentinel.*
ap*.*	cyb-*.*	Helloween.*	MPS-OPC.*	Silly.*
apl-*.*	d-*.*	HLL.*	MTE.*	SVC.*
April.*	Da.*	HLLC.*	Murphy.*	Sylvia.*
ARCV.*	Danish_Tiny.*	HLLO.*	Naziphobia.*	Tiny.*
Ash.*	Darth_Vader.*	Horse.*	Npox.*	Trivial.*
AT.*	DataCrime.*	Hydra.*	Number_of_Beast.*	Vacsina.*
ATAS.*	Dead.*	Hymn.*	Nympho.*	VCL.*
Aus.*	Deicide.*	ice.*	Old_Yankee.*	Vien.*
Australian.*	Diamond.*	Icelandic.*	Ontario.*	Voronezh.*
b-560.*	Doom.*	IMI.*	PCBB.*	WordSwap.*
Bad_Boy.*	DOS_x.*	Infector.*	Phalcon.*	Yankee.*
bammpc.*	Dutch_Tiny.*	Intruder.*	Phoenix.*	YB.*
Black_Jec.*	Ear.*	IV*.*	Pixel.*	Zherkov.*

Table 17: Virus families in the dataset used for the experiments in Chapter 4.

Appendix B

Additional Experiments

B.1 Additional Comparisons Against the Benchmark

The experiment in Chapter 4, Section 4.1.1, was repeated with different classifiers: the J48 decision tree, a Naïve Bayes classifier and the Sequential Minimal Optimization (SMO) algorithm for support vector machines. We compared the accuracies and false positive rate of each classifier, using our hierarchical search for features of length 8 and intra-family support threshold of 3, and using the traditional search for features of length 16 with general support of at least 1% in the training set.

The results, presented in Table 18, show that the classifiers using our feature search method perform consistently better than when using the generic search method. They outperform their counterpart in all aspects: overall accuracy, false positive rate and detection accuracy. We notice that the Naïve Bayes classifier is significantly more conservative compared with the other two classifiers, as well as compared with the ID3 decision tree used in Section 4.1.1, reporting an extremely low false positive rate, though at the expense of the detection accuracy. Nonetheless, here as well, our feature selection method yields better results than the traditional 16-byte feature selection.

Classifier	Overall Accuracy	False Positive Rate	Detection Accuracy
Hierarchical Feature Search			
J48 Decision tree	93.65%	5.24%	92.56%
Naïve Bayes	69.51%	0.13%	37.17%
SMO Algorithm	93.39%	5.71%	92.26%
Traditional Feature Search			
J48 Decision tree	64.69%	13.18%	46.72%
Naïve Bayes	60.69%	0.16%	18.66%
SMO Algorithm	65.04%	13.36%	47.52%

Table 18: Experimental results of different classifiers using family-based LOBO validation, for the hierarchical feature search (sequence length of 8 and intra-family support threshold of 3) and the tradition search method.

B.2 Additional Experiments Using Different Classifiers

The experiment in Chapter 4, Section 4.2, evaluated the ID3 classifier using feature sets constructed with different feature search parameters. Here, we repeat this experiment with different classifiers: the J48 decision tree, a Naïve Bayes classifier and the Sequential Minimal Optimization (SMO) algorithm for support vector machines. We report, in Tables 19 to 24, the overall classification accuracies, the virus detection accuracies and the false positive rates.

Sequence length / Inter-family support	8	7	6	5	4	3
3	93.65%	95.41%	95.56%	96.65%	97.00%	99.49%
4	89.53%	94.29%	94.71%	96.14%	96.30%	97.25%
5	87.50%	93.38%	94.60%	96.36%	96.11%	96.18%
6	81.13%	92.92%	94.25%	95.01%	96.21%	96.24%

Table 19: Classification accuracy, in relation to different feature selection parameters (using the J48 decision tree learner)

Sequence length / Inter-family support	8	7	6	5	4	3
3	92.56% 5.24%	94.82% 4.10%	94.61% 3.43%	96.01% 2.75%	96.42% 2.29%	98.85% 0.67%
4	83.15% 4.64%	94.28% 5.71%	93.69% 4.30%	95.62% 3.16%	95.10% 2.49%	96.80% 2.35%
5	80.89% 6.52%	93.56% 6.99%	93.24% 4.10%	95.57% 2.76%	94.36% 2.08%	94.33% 2.15%
6	65.76% 4.44%	93.45% 7.73%	93.62% 5.38%	94.37% 4.37%	95.26% 2.76%	94.54% 2.55%

Table 20: Virus detection accuracy (top) and false positive rates (bottom), in relation to different feature selection parameters (using the J48 decision tree)

Sequence length / Inter-family support	8	7	6	5	4	3
3	69.51%	77.83%	86.21%	93.32%	96.05%	99.33%
4	79.61%	80.82%	84.86%	89.51%	94.66%	95.00%
5	84.97%	82.91%	88.05%	89.74%	93.03%	94.22%
6	80.55%	89.50%	90.12%	88.43%	92.00%	93.59%

Table 21: Classification accuracy, in relation to different feature selection parameters (using a Naïve Bayes learner)

Sequence length / Inter-family support	8	7	6	5	4	3
3	37.17% 0.13%	50.31% 0.27%	71.57% 0.47%	86.95% 1.75%	98.00% 5.85%	99.84% 1.14%
4	57.92% 1.41%	60.73% 1.01%	69.70% 1.28%	79.78% 1.81%	95.82% 6.39%	98.61% 8.40%
5	72.51% 3.76%	67.54% 2.69%	77.84% 2.55%	81.79% 3.16%	92.14% 5.58%	98.11% 9.68%
6	63.56% 3.29%	82.52% 3.97%	82.45% 2.94%	79.91% 3.70%	90.02% 5.91%	97.39% 10.15%

Table 22: Virus detection accuracy (top) and false positive rates (bottom), in relation to different feature selection parameters (using a Naïve Bayes learner)

Sequence length / Inter-family support	8	7	6	5	4	3
3	93.39%	96.12%	97.59%	97.98%	98.92%	100%
4	88.72%	94.08%	95.05%	97.20%	98.55%	99.33%
5	86.50%	93.57%	94.88%	96.91%	97.66%	98.72%
6	83.08%	91.77%	94.60%	95.40%	97.43%	99.03%

Table 23: Classification accuracy, in relation to different feature selection parameters (using the SMO algorithm for support vector machines)

Sequence length / Inter-family support	8	7	6	5	4	3
3	92.26% 5.71%	95.34% 3.23%	97.54% 2.35%	97.24% 1.48%	98.81% 0.87%	100% 0%
4	81.96% 4.91%	93.22% 5.78%	94.48% 4.23%	96.39% 2.02%	98.07% 1.01%	99.11% 0.34%
5	76.94% 4.71%	93.54% 6.59%	94.54% 4.77%	96.04% 2.42%	96.91% 1.55%	97.98% 0.74%
6	63.76% 2.49%	91.65% 8.07%	95.49% 6.30%	94.30% 3.49%	96.76% 1.95%	98.58% 0.60%

Table 24: Virus detection accuracy (top) and false positive rates (bottom), in relation to different feature selection parameters (using the SMO algorithm)

B.3 Execution Times

The most time-consuming steps of the family-based feature search are the sequence scanning for every virus and the feature selection within each virus family. The approximate execution times of these two steps for sequence lengths 3 to 8 are listed in Table 25. Because the feature elimination step uses lists of features selected for each virus family, that step requires very little execution time – less than 1 minute.

Sequence length	8	7	6	5	4	3
	60 min	60 min	55 min	55 min	60 min	65 min

Table 25: Execution times for the family-based feature search