Designing, Implementing, and Analyzing a System for Virus Detection

Blaine Alan Nelson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-27

http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-27.html

March 19, 2006

Acknowledgement

We thank Peter Bartlett, Michael Jordan, Christos Papadimitriou, Anil Sewani, Steve Martin, Marco Barreno, Guillaume Obozinski, Russell Sears and Brighten Godfrey for helping make this research possible. Blaine Nelson was supported by grants from Hewlett Packard, Sun Microsystems, the State of California under the MICRO program, and jointly by the US Department of Homeland Security Advanced Research Projects Agency (HSARPA) and the National Science Foundation (NSF) under grant ANI-0335241.

Designing, Implementing, and Analyzing a System for Virus Detection

by Blaine Nelson

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Anthony D. Joseph
Research Advisor

(Date)

* * * * * * *

Professor Peter L. Bartlett
Second Reader

(Date)

Designing, Implementing, and Analyzing a System for Virus Detection

Abstract

Designing, Implementing, and Analyzing a System for Virus Detection

by

Blaine Nelson
Master of Science in Computer Science

University of California at Berkeley

Professor Anthony D. Joseph, Research Advisor

In spite of advances in viral detection, the rapid proliferation of novel mass-mailing worms continues to pose a daunting threat to network administration. The crux of this problem is the slow dissemination of the up-to-date virus signatures required by traditional systems to effectively halt viral spread. Such signatures are primarily generated manually after samples of the novel worm are submitted to the anti-virus company for analysis - a process that leaves most systems open to attack for hours or days. In modern high-speed systems, this response time is becoming increasingly inadequate to prevent devastating viral epidemics that waste value network resources.

In this thesis, we present and evaluate a statistical learning system for addressing the mass-mailing worm threat. We propose a multi-tiered learning system that learns user's emailing characteristics. By monitoring the behavior of outgoing email, our system was empirically able to differentiate between normal behavior and novel worm outbreaks. In our experiments on six email-born viruses with varying characteristics, our system achieved 99% accuracy in most cases, demonstrating the effectiveness of our approach. We also look beyond the current state-of-the-art in viral techniques by developing a naïve model for analyzing the effectiveness of our statistical classifiers against threats poised by unforeseen viral adversaries determined to subvert our deterrents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Electronic mail has become one of the most ubiquitous methods of communication. In 2002, the information technology research firm IDC predicted email usage would reach 60 billion messages daily [24] while the Radicati group estimated 2004 daily levels to have already reached 76.8 billion [51]. However, the rapid proliferation of this Internet communications medium has lead to the emergence of wide-spread abusive behaviors. According to several surveys, unsolicited marketing messages, or spam, account for more than half of the total daily message traffic [24, 51]. Meanwhile, phishing attacks, all but unheard of in 2003, rapidly increased in the last half of 2004 making up 18 million of 12 billion messages scanned by MessageLabs [46].

Of these abuses, viruses, designed specifically to wreak havoc on businesses and the Internet infrastructure, remain as the largest security problem for most businesses with 70% of UK businesses citing viruses as their worst security incident [16]. A conservative measure of worm impact is given by a joint survey of 269 responding companies conducted by the Computer Security Institute (CSI) and the US Federal Bureau of Investigation (FBI) in 2004. Their survey found that viruses caused an estimated $55 million in damages, surpassing "cybertheft" as the primary cost factor in computer security incidents for the first time in 5 years [21]. Similarly in a study of 200 British business, the UK's National Hi-Tech Crime Unit (NHTCU) found that 83% of interviewed companies had been attacked by viruses, worms, or Trojans and estimated that these programs caused £70 million to small business[1] as well as £676 million to large businesses[2]. The survey found that, on average, companies were subject to virus attacks seven times a day with large businesses encountering infections once every three

[1] $100 - 1000$ employees

[2] > 1000 employees

Product	**Detected**	**Lag**	**Product**	**Detected**	**Lag**
ClamAV	22:02	00:00	AntiVir	06:09	08:07
Sophos	23:00	00:58	Ikarus	06:29	08:27
Trend Micro	00:05	02:03	Dr. Web	07:03	09:01
Fortinet	00:18	02:16	Proland	07:19	09:17
F-Prot	00:43	02:41	Panda	07:41	09:39
McAfee	00:52	02:50	RAV	07:43	09:41
eTrust-CA	01:30	03:28	BitDefender	07:56	09:54
Symantec	02:06	04:04	Norman	08:20	10:18
Command	02:44	04:42	Dr. Web	08:54	10:52
Virusbuster	02:58	04:56	AVG	10:05	12:03
Trend Micro	03:12	05:10	Kaspersky	12:18	14:16
Quickheal	04:11	06:09	F-Secure	14:46	16:44
eTrust-VET	05:31	07:29	Avast	15:23	17:21

Table 1: The amount of time, in hours, between the appearance of the MyDoom.BB worm in Asia sometime during February 16th, 2005, and its successful detection for 26 common anti-virus products. The first successful detection of the worm by any anti-virus software is assumed as its emergent time [42, 67].

days [5]. However, a third 2004 survey by the DTI of the UK provided more conservative estimates[3] that an average large business only received a virus once per week and incurred an infection once a month [37].

While most of the above studies group all self-propagating malicious code as viral and do not differentiate the methods of infection (email, network exploit, physical media, etc.), mass-mailing worms (those that spread via infected emails) are one of the more malignant variants of malicious code. At least eight out of the ten computer worms most frequently reported during 2004 to a prominent anti-virus company spread via email [70]. Moreover, in the DTI study 75% of the worst virus outbreaks were caused by Blaster, Sobig, and Bugbear; the last two being mass-mailing viruses. Thus, while viruses that exploit network vulnerabilities (*e.g.*, Code Red, Blaster and Slammer) spread faster [49], mass-mailing viruses continue to be prevalent, perhaps because they require no special exploits and still achieve success.

Protecting against mass-mailing viruses is an active research area; however, the traditional signature-based anti-virus defenses deployed in the field have not changed significantly over time. Unfortunately, while these methods generally benefit from high accuracy, recent research (and empirical observation) has demonstrated that in the one

[3]Discrepancies between these surveys may be due to the fact that the NHTCU survey only interviewed businesses with more than 100 employees while more than 50% of the DTI survey was of businesses of fewer than 100 employees [5, 37].

hour to one day it takes to receive a new virus sample, analyze its behavior, generate new signatures, and distribute updates to anti-virus scanners, widespread infection among vulnerable hosts can already occur [72]. As was shown in a study of the proliferation of the MyDoom.BB virus [44], anti-virus vendors took as long as 17 hours from the time the virus was first detected with a mean and median response time of above 7 hours [42, 67]. The results of this study are shown in Table 1. Moreover, the damages cited in the FBI survey [21] happened in spite of the fact that 99% of the total 488 companies surveyed used anti-virus software, and 98% used firewalls with similar results for both the NTHCU and DTI of the UK [5, 37].

Thus, traditional signature-based methods alone are insufficient for containing the spread of modern email worm outbreaks. As pointed out in the 2004 security report of MessageLabs, virus writers exploit the "window of vulnerability", the lag between virus release and email signature generation, in order to infect machines [46]. A key to stopping worm outbreaks is to quickly choke off avenues for infection. Furthermore, while statistical learning techniques have been applied to such problems as spam and virus detection, in the virus domain, a single false negative (*i.e.*, viral email classified as normal) could have dire consequences compared to a false negative in the spam domain. Hence, in designing a learning system for virus detection, it is not necessarily sufficient to simply reduce the number of viral emails.

We take a reactive approach to the problem; rather than detecting the incoming viral messages, we attempt to detect infected machines sending out mass-emails. In this extrusion detection paradigm, we assume that novel virus infections are an inevitability. While progress has been made for proactive novel virus detection on incoming email [45], it only takes a single false negative to cause an infection. Even if perfect virus prevention were deployed at the network level, mobile hosts could become infected outside of the network and unknowingly bring the infection inside the network defenses as has been seen with firewall defenses. Moreover, many mass-mailing viruses have multiple modes of infection so an infection can occur even while monitoring incoming email. Thus, our system aims at controlling an infection once it occurs in order to minimize the damage caused by mass-mailing activity. This methodology can be effective due to the modern strategy of worms to propagate quickly in order to leverage the window of vulnerability of traditional detection mechanisms; that is, any level of detection success would dampen a virus' chance of success. Finally, by applying a solution at the network level, quarantining the activities of an infected machine has the potential to reduce damage to mail-servers caused by an overwhelming stream of viral emails.

Our network solution to detecting viral activity in out-going email traffic uses modern statistical learning techniques to monitor for sufficient deviations from normal email behavior. This design incorporates a set of features used to represent the current state of email behavior. Using feature selection techniques, we chose a robust set of features that can accurately distinguish between normal and viral behavior. These features are used to characterize the email behavior of a machine using a classification approach that employs a coarse model of user behavior. Novelty detection is used as a filter to isolate most normal email, and a classification layer leverages past viral behavior to reduce false positives seen in traditional novelty detection. The resulting system is capable of quarantining users believed to be exhibiting viral behavior.

Finally, given the apparent arms race between spam detection and spam obfuscation, we also began to consider the long-term ramifications of deploying an automated virus detection system. While extensive work has been done to quantify the "capacity" of learning techniques [31], relatively little work has considered malicious scenarios. This is due in part to the fact that worst-case scenarios in machine learning are quite dire. However, as these techniques are deployed in security sensitive environments, it is inevitable that malicious users will attempt to subvert them. Thus, it is becoming increasingly important to isolate attack scenarios and design learning approaches that make subversion difficult. Along this line, we examines a specific attack scenario relevant to the viral detection system we implemented.

This thesis is the result of collaborative and individual efforts on the part of its author and draws from both. In particular, much of our system design and experiments for viral detection were the results of joint research efforts with Steve Martin, Anil Sewani, Karl Chen, and Dr. Anthony Joseph, which are also presented in a publication [40] and the theses of Anil and Steve [41, 68]. I also collaborated with Marco Barreno and Russell Sears in considering adversarial learning, but the analysis presented in Chapter 5 was my own contribution. In addition, the development of the learning hierarchy presented in Appendix A was largely my design and implementation.

The remainder of this thesis is organized as follows. We begin by examining the previous work that has been done in this field in Chapter 2. Then in Chapter 3 we introduce the set of features that we collected in email and motivate why these features can discriminate between user and viral behavior. Chapter 4 introduces the system itself, followed by an analysis in Chapter 5 of a simple attack scenario against the sort of statistical learning techniques utilized in our approach. Finally, Chapter 6 summarizes our work by discussing the lessons learned and how we plan to move ahead.

Chapter 2

Background

In the past decade, a variety of work has emerged on automated statistically founded approaches to networking problems such as intrusion, spam, and virus detection. These approaches have provided several successes, but have also been met with a variety of criticism. This chapter discusses some of the notable approaches that have influenced our considerations for implementing a virus detection system.

The major arena of applied statistical learning in the systems/networking domain has been in spam detection. With spam consuming more and more system resources [24, 51], there has been an increasing need for reliable spam filtering. In fact, according to Steve Ballmer of the Microsoft Corporation, the company has had to develop special technology and devote several employees to filtering mail to Bill Gates, who receives around four million unsolicited messages per day [1]. The common approach to spam filtering has been a direct application of Bayesian techniques to the corpus of words contained in email's body [22, 60]. Moreover, Bayesian and other techniques have been directly incorporated in commercial spam filters such as SpamAssassin [65], SpamBayes [47], and Mozilla's Spam Filter [71].

One interesting alternative method of examining messages is the construction of social networks [4, 52]. In these models, users within a network are considered as nodes of a graph, and communication between any two nodes is indicated via an edge between the nodes. Cliques of nodes form a social network, indicating common communication patterns. Communication that violates these behavioral patterns is considered abnormal.

Meanwhile, outside of the spam domain, there have been several notable efforts at applying similar techniques to virus and intrusion detection. However, in the spam domain, detection failures do not result in the system being compromised. Hence, statistical techniques have been faced with more criticism in security sensitive arenas. Moreover, in spam there is a clearly defined goal of advertising a product that results in highly characteristic messages. The goals in virus and intrusion detection are less well-defined although certainly characteristic.

Recently, a number of papers have examined the behavior of worms. The behavior of scanning worms such as Code-Red and Slammer were used as case studies in examining the propagation patterns of these worms [49, 50, 72]. Similarly empirical behavior of the mass-mailing worms Mydoom and SoBig was examined in the study by Wong *et. al.* [88]. From observation, virus detection systems have been built around the following characteristics: fast propagation rates, sending patterns, and the presence of malicious executables.

One effective approach to virus detection is based on the characteristic of viruses to propagate at rates only limited by the bandwidth of their host's network. Several researchers have explored email virus/worm containment strategies based on this behavior. Williamson *et. al.* present a method that throttles the frequency at which email can be sent to previously unseen recipients [85]. Similarly, Ganger *et. al.* used abnormal DNS lookups to detect behavior indicative of scanning worms [19] while Chen *et. al.* used the correlation between an increased rate of failed connections and viral behavior for this purpose [8].

Other approaches using statistical analysis of outgoing email include monitoring the frequency of communication events between clients and their mail server for anomalies [25]. The authors obtain a near 100% accuracy rate for their simulation-based experiments, but they do not discuss the consequences of their fundamental assumption that all virus traffic goes through a central SMTP server that is observable. We have found that several viruses install their own SMTP engines on infected hosts and propagate with lists of known open relays [81]. In addition we believe that there are features other than those based on frequency that indicate worm activity.

Another characteristic of viruses is that their sampling of recipients from the host's address book (or from other sources) exhibits a pattern not seen in normal email traffic. Leveraging this behavior, social network analysis detects virus emails as well as spam by creating network models where users are represented as nodes in a graph, and communication between users as edges [20, 52]. Clusters of nodes in the graph form a social network, and messages that violate these behavioral patterns are considered suspect, *e.g.*, spam or virus. While worms can easily bypass social networks by intelligently choosing recipient lists (*e.g.*, by using recent emails in the user's 'Sent Email' folder), such graphs can offer valuable insight into email behavior. Stolfo *et. al.* combine a variant of social network analysis with several features calculated on sent email (*e.g.*, frequency and quantity of emails with attachments) to detect abnormal user behavior indicative of a viral infection [74, 75]. While our classification methods do not use social networks, we do include several similar features.

The final viral characteristic is the presence of a malicious executable attached to an email message. Schultz *et. al.* use a Naïve Bayes classifier trained over consecutive byte sequences and printable strings in email attachments to detect malicious executables [63, 64]. Maloof *et. al.* extended this idea to decision trees, Support Vector Machines, k-nearest-neighbors and Term Frequency Inverse Document Frequency (TFIDF) models [35]. These attachment analysis techniques are effective, but they cannot detect email worms that spread via other exploits, such as scripts embedded in HTML or links to pages that infect vulnerable web browsers (*e.g.*, BubbleBoy and some variants of Mydoom [43, 76]). In addition, binary file analysis can be prevented by cryptographic code obfuscation techniques [18].

Novel virus detection has begun to emerge in the commercial setting although is not currently as prevalent as spam detectors. Solutions such as Panda's Truprevent© are deployed on individual machines and attempt to thwart unknown infections by analyzing behaviors using heuristics [53]. Similarly MessageLabs' Skeptic© program employs adaptive heuristics at the mail server leveraging a large corpus of viral and user behavior to prevent novel infections [45]. However, as these companies have made their methodologies proprietary, it is hard to evaluate the effectiveness of their techniques other than through empirical results.

Although attacks against learners have been not been widely reported to date, several researchers have discussed methodologies for doing so. Wittel and Wu suggest the concept of 'strong statistical attacks' against spam filters [86], in which emails are specifically designed to subvert statistical techniques. One such attack was proposed by John Graham-Cumming that used a spam filter to learn a set of words that would cause a targeted learner to misclassify spam messages [23]. While the proposed attack was restricted[1] it shows the vulnerability of learners and how attackers could use machine learning techniques against their targets.

As mentioned at the beginning of this chapter, the primary theoretical learning result for malicious situations was provided by Kearns and Li [30]. In particular, under

[1]The success of the attack depended on feedback from the filter and would only be feasible against network-wide filters

the PAC framework, a learner is characterized as 'ϵ-good' if has a false-positive and false-negative rate of less than ϵ with a probability of $1 - \delta$ for parameters ϵ and δ. Kearns and Li use this theoretical construct of PAC learning to derive bounds[2] on the maximal fraction of malicious error, which describe the degree of training pollution that learners can be expected to tolerate. The bounds are applicable in the general setting, but do not describe attacks in which the adversary is limited in choosing malicious data to insert.

Recent work has begun to address specific attack scenarios and propose countermeasures for preventing such attacks. Dalvi *et. al.* propose a game-theoretic approach to the problem of adversarial classification [12], in which they examine the effect of an attacker in a cost-sensitive learning environment. In particular, they assume that the attacker wishes to misclassify 'negative' points (*e.g.*, spam) by altering some subset of that point's features with minimal cost. The cost is some measure of the attacker's utility for that feature; *e.g.*, time required to alter the feature or decrease in value of the point due to the altered feature. The attacker's objective is quantified in terms of solving an optimality problem that leads to minimal incurred cost for maximal profit gained. Based on the attacker's optimal course of action, Dalvi *et. al.* develop a technique for detecting points likely to have been altered.

[2]The general bound derived by Kearns and Li is that the maximal malicious error for a representation class C is $E_{MAL}(C) < \frac{\epsilon}{1+\epsilon}$ and they derive more specific results for particular classes of algorithms and error [30].

Chapter 3

Email Feature Analysis

The first step in developing a system for detecting virus email is the selection of a set of features used to describe an individual email. While the term *feature* has several possible interpretations, in this document it will be used exclusively to describe a statistic that represents a measurement of some aspect of a given user's email activity or behavior. Each email is represented as a set of features, which represent various aspects of the user's behavior that produced that email. Hence, in comparing the emails by means of these feature sets, novelty detection algorithms attempt to identify sufficiently deviant behaviors.

In seeking to distinguish normal from viral email, we proposed and implemented tools to extract features thought to be relevant for this task. The features were considered based upon our observation and intuition of the differences between the two types of email behavior. Underlying these choices were the following observations of email viruses: they must have an infection medium, they often attempt to avoid detection, they often have some degree of repetition between emails, and they have traditionally sent email at extraordinary rates. Thereby, implementing features that captured these behavioral differences allowed us to detect the kinds of abnormalities indicative of viral infections.

The remainder of this chapter describes the features that were considered in our study and the process of choosing features which were most relevant to the detection of viruses. For an in-depth description of the implementation of the feature extraction architecture refer to Steve Martin's thesis [41]. The design of the remainder of the system is discussed in detail in Chapter 4.

3.1 Features Extracted from Email

We selected and implemented two dozen separate features with the underlying goal of obtaining a set of statistics that accurately distinguishes between normal and abnormal email activity. While the features are ostensibly a representation of a single email's state, this is not entirely the case; some features are derived entirely from the current email, while others are composite features calculated over a fixed-size window of the most recent emails. The per-email based statistics are straightforward descriptions of the email. However, the window-based features represent the current state of the email-flow at the point the current email is sent (these are empirical averages, variances, or frequencies). By allowing both categories of features into the email description, we blurred our definition of what a set of features represents (some strange blend of an email and an email flow) and we introduced unusual temporal dependencies between email instances. These temporal dependencies weaken the independence assumptions of our Naïve Bayes model as is discussed in Section 4.4.4. However, in analyzing the relevance of our features in Section 3.3, features from both categories were found to be highly relevant to distinguishing between viral and normal email traffic.

Our features also were classified by the domain of their values. While all were non-negative, some had support over $\mathbb{R}^{0+}$, while others were strictly integer ($\mathbb{Z}^{0+}$), and still others were restricted to a finite set of values (often $\{0, 1\}$). As an example, a frequency calculation returns a real-valued number, whereas a feature involving types of email attachments is represented as an array of bits, where each bit represents the presence of a specific type of attachment. Features with finite domains were treated as multinomials, for which we sought to calculate the probability of each of their values occurring. The remain features were treated as continuously-valued features. The continuously valued features were fit to distributions, however, often the user and viral distributions substantially differed. The distributions of the features is discussed in section 4.4.2.

The features discussed below were implemented in Java under a framework that allowed features to be extracted either from live email streams or from saved emails stored on disk. The features were implemented in an extensible fashion for ease in selecting features or adding new ones. Moreover, the architecture was designed to process email in real-time without saving the actual message but rather only the necessary state statistics. For a more thorough discussion on how this email-extraction architecture provides speed, security, and privacy, refer to the thesis of Steve Martin [41].

The following sections briefly describe our feature choices and why they are included. We revisit the question of the importance of each feature in distinguishing between separate users in Section 3.2.

3.1.1 Features based on a Single Email

The following sections describe numerical values calculated on a per-email basis.

Single Email Multinomial-Valued Features

Features in this category represent their output as one or more bits. Multinomial return values are in the form of a bit string.

Whether or not the message is a reply or forward: This feature is determined from the email text itself, and not from the subject. A truly sophisticated worm could attempt to generate replies to sent email that would fool a user, but current worms do not do this.

Presence of HTML: This feature is helpful because there are exploits resulting from buggy HTML parsing by mail user agents, such as those used by the Kak worm to propagate [77].

Presence of HTML script tags/attributes: This is split into a feature indicating the presence of script tags and a feature for indicating the presence of scripting attributes in other HTML tags. These statistics are particularly useful in detecting emails that are potential security risks from their embedded HTML code alone. As an example, the VBS.Bubbleboy virus contains an embedded VBS script in the email it sends [76].

Presence of embedded images: Embedded images are often used by spammers to verify address lists through logging requests by the mail user agent's HTML renderer for the image on the web server. Also, images could be used to exploit buggy image processing, such as the well-documented Microsoft JPEG vulnerability [48].

Presence of hyperlinks: Several worms propagate by emailing links to infected web pages that then infect the machine through browser security holes. An example is the Bubbleboy virus [76].

MIME types of file attachments: The MIME type of a file is assigned by the sending mail user agent either using magic numbers (see below), or through querying a local table with the filename extension. Each binary value represents the presence of a specific type of file. Attachments are the most prevalent infection method for email worms.

Presence of binary, text attachments: This multinomial statistic is calculated in a similar manner to the previous feature. It is particularly important in the case where an email has a binary or text file attached whose type is corrupt or unknown.

UNIX "magic number" of file attachments: Worms often assign misleading MIME types to fool virus scanners. An example of a worm that uses such a MIME exploit is the Nimda virus [78]. The magic number is an accurate method of determining the true file type, and if an attachment's magic number does not correspond to its MIME type, it is possibly malicious.

Per-Email Continuous Features

Total size of email, including attachments: Most worms send similarly sized emails due to limited polymorphism in text and identical attachments. This feature leverages these similarities.

Total size of files attached to the email: Many worms send infected messages with fixed-size attachments repeatedly, while using polymorphic subjects and bodies.

Number of files attached to the email: Most people do not attach many files to their email, whereas the majority of worms require opening an attachment to propagate.

Number of words/characters in the subject and body: These features help build a basic profile of the user's writing characteristics. Most text used by prevalent worms is randomly chosen from a fixed number of strings either contained within the worm code or from the victim's machine. In addition, spam emails have been found to share certain wording characteristics [22].

3.1.2 Features based on a Window of Emails

We now describe numerical values calculated over a window typically consisting of the user's last twenty messages. All statistics are continuous.

Number of emails sent: Worms and active spam machines tend to send emails more frequently than the average user.

Number of unique email recipients: This feature counts addresses in the To:, CC:, and BCC: (if available) headers. The frequency with which one sends mail to distinct users captures an important aspect of email behavior, *i.e.*, sending frequent emails to large numbers of distinct users might indicate worm or spam-bot activity. Another method for representing this information is social networks [4, 20, 52].

Number of unique sender addresses: Many users have multiple active accounts on the same machine. However, a single machine sending email with a large number of unique addresses in the From: header field at a high rate is indicative of obfuscation techniques used by many viruses.

Average number of words/characters per subject, body; average word length: These features capture trends in email wording that separates specific user characteristics, and likewise could also separate normal email from malicious activity.

Variance in number of words/characters per subject, body; variance in word length: These types of features have been used in previous work with some success to detect the behavior of email viruses [74, 75]. We apply then here for the same reasons.

Ratio of emails with attachments: Most users do not send many emails with attachments — *i.e.*, over all emails sent previously, the number of messages with attachments is typically low. Most worms, however, spread via attachments. This fact coupled with the typically high sending rate of viruses makes this feature extremely useful. Note that this feature does not include email agents that send HTML message portions as attachments normally, as our implementation attempts to account for this.

Ratio of emails with replies or forwards: This feature captures whether a user on average is more likely to write replies to queries or initiate original conversations. In addition, as was mentioned previously, most current worms do not try to represent infected messages as replies to emails from the user, as it could be difficult to successfully fool someone who is familiar with their email correspondents.

3.2 Feature Analysis

Our feature set is designed to capture specific elements of user email behavior that separate normal from abnormal (worm propagation) activity. To better understand the individual contributions of each feature to the overall effectiveness of our technique, we next present an analysis of the ability of each to capture information specific to individual behavior.

Moreover, examining the distributions of features for viruses and normal users provided a glimpse of the specific role that the feature plays in classification. Our analyses showed that certain features are quite discriminative between the two groups while others provide little insight to the difference between the two classes. Finally our analyses support the idea that our features provide a coarse model of email behavior.

3.2.1 Feature Distributions

In order to gain a better understanding of the contribution of our individual features to the decision of our classification system, we performed a small empirical study of the email produced by a single user (an EECS graduate student at Berkeley) and six well-known viruses. The details of the study are presented more thoroughly in Chapter 4.6 along with a more comprehensive description of the viruses in Table 3. We examined the individual features' empirical distributions and used them to estimate the distributions for our Naïve Bayes model as discussed in Chapter 4.4.2. A few of the more 'successful' features' distributions are presented in Figure 1. These histograms show the potential for separating users and viruses based on our set of features.

To get a more comprehensive understanding of the contribution of our feature set to distinguishing between users and viruses, we performed the covariance analysis discussed in Chapter 3.3.1 and used that analysis to determine which features were generally most relevant to the desired classification. The first part of this analysis focused on identifying characteristics of each virus that made it unique from other viruses as shown in Figure 2(a). As can be seen, Bubbleboy.F is by far the most distinctive virus as it has a different infection mechanism that involves embedded scripts rather than a malicious executable. It also has a fixed body and subject unlike the other polymorphic worms. As such, this virus differed in features involving scripts, word counts, and attachment presence. The remainder of the information from the first analysis yielded little insight, but the second analysis yielded far more information.

The second part of the analysis identified viral characteristics that best distinguished the virus from the user in our study as shown in Figure 2(b). Again, this figure shows that Bubbleboy.F was quite distinctive from the other viruses. Bubbleboy differed mostly in features involving its HTML and script content as well as the fact that it has an enlarged body due to its embedded script. The other viruses were clearly distinguished by features involving their deviant attachment behavior. Surprisingly though, these viruses also seemed distinctive in a number of features involving the subject of the email. Also, it was surprising that the number of emails sent in a window only seemed

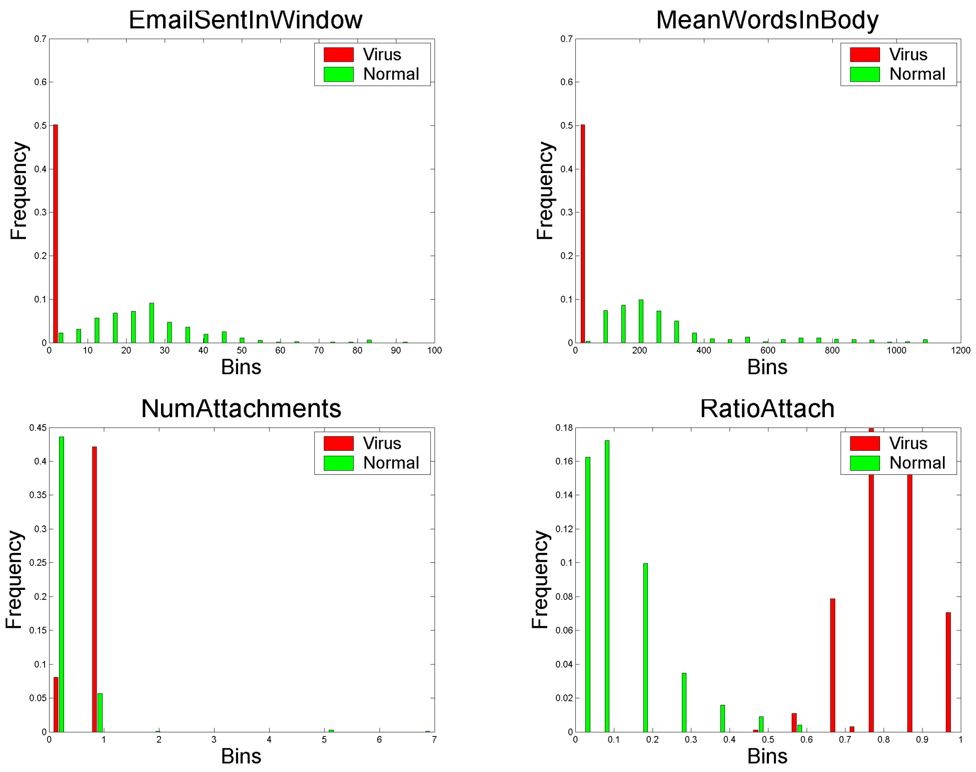


Figure 1: Diagrams of the histograms of features for a set of emails differentiating between the ‘normal’ email of an EECS grad student and the ‘virus’ email generated from the virus Bagle.F [79]. The figures show the normalized histograms (normalized to sum to one across *all* email) for the features ‘Number of Emails Sent’, ‘Mean Words in the Body’, ‘Number of Attachments’, and ‘Ratio of Attachments’. The clear separation of viral and normal traffic across these features made them good candidates for our final feature set.

to play a secondary role according to the analysis when empirical results have shown this feature to be quite distinctive. Nonetheless, the results showed the potential of the feature set to the problem and motivated the feature selection process in the following section.

3.2.2 User Models that Capture E-mail Behavior

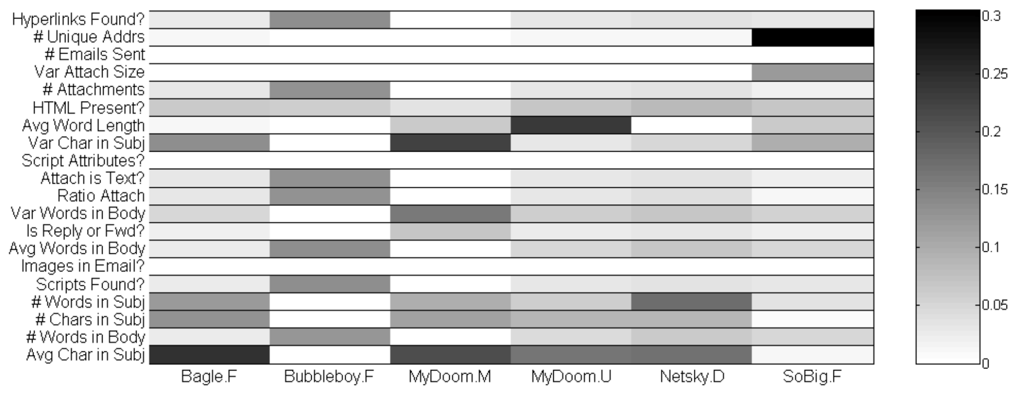
To motivate the idea of models of email behavior, we also analyzed the features across a large data corpus of the Enron dataset that has been made publicly available [33]. This corporate dataset provides insights into the email behavior at an enterprise level although the dataset lacked attachments (thus limiting the features we could examine). For the following analysis, we used data for all users in the Enron data set that had sent-mail folders. There were a total of 126,078 emails between 148 users, with each user having between 3 and 8926 emails. Most users (53.4%) had between 100 and 1000 emails with 22.3% below 100 and 24.3% above 1000.

Figure 3 demonstrates the similarities we found between users in this dataset. We performed the maximal covariance analysis to distinguish users in the Enron dataset generating a "signature" for each user based on their distinguishing features. This signature is depicted as a single column in Figure 3. By clustering the users based on these signatures we were able to identify groups of users with similar behavioral characteristics as shown in the figure. The clusters indicate that several canonical behaviors can account for the majority of individual user behaviors making the deployment of systems based on per-user models feasible.

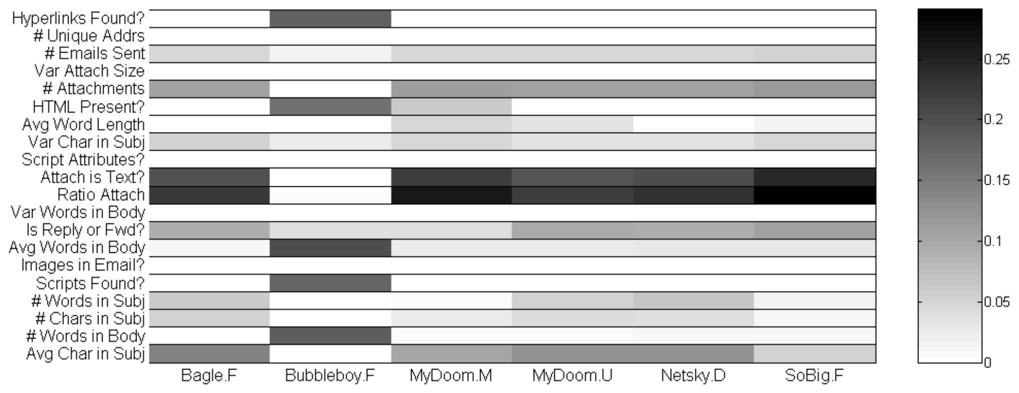
3.3 Feature Selection

The problem of optimally selecting statistical features can be categorized as *feature extraction* and *feature selection*. Feature extraction creates a smaller set of features from linear combinations of the original features, while feature selection simply chooses a subset of the original features (in essence a boolean version of feature extraction). As indicated by the title, we have concentrated on the task of feature selection since it is important to preserve the feature distributions for Naïve Bayes and selection provides insight into individual feature relevance [34, 60].

The task of feature selection, a concept that has been well studied in the statistics and machine learning literature [2, 26], is the process of choosing a subset of a feature space that best represents the problem at hand while minimizing the amount



(a) Inter-Virus Comparison



(b) Virus to User Comparison

Figure 2: Comparisons of how the viruses under study differed in their behaviors. For each virus we used the covariance analysis discussed in Chapter 3.3.1 to determine the unit direction in feature space that maximally distinguishes the virus. These figures show that direction for each virus as the squared contribution of each feature to it. The shading is indicative of the strength of the covariance between the random variable indicating the virus type and the features. Figure 2(a) is a covariance analysis comparing the viruses against each other. Thus it demonstrates the features that best distinguish each virus from the others. Figure 2(b) is a covariance analysis comparing each virus to the user we studied. This demonstrates the features most salient for the detection of each.

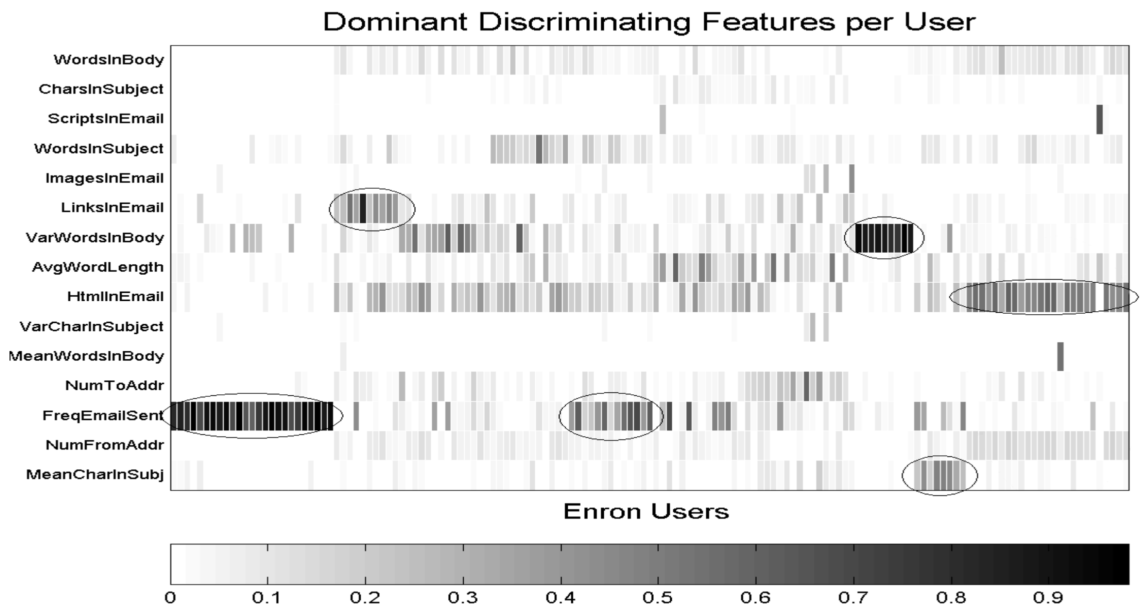


Figure 3: A plot of the direction of maximal covariance for each user in the Enron data. The users are clustered based on these directions to emphasize the similarity between users. The obvious clusters are demonstrated by circles around their most prominent features for clarity.

of lost information. Choosing the subset of features that optimally predicts the desired function is computationally infeasible - the number of subsets of a set grows exponentially with the set size. As such, greedy methods are often employed. We now present a simple method for using the results of covariance analysis to reduce the feature set by concentrating on the features that contribute the most covariance with the desired target labels Y.

The approach we chose for feature selection actually leverages methods of feature extraction to provide a 'relevance score' for each feature. The common approach uses Principal Component Analysis (PCA) to find directions in feature space that maximize variance. Classical PCA determines such directions, but fails to find individual features that maximize variance. Instead, it determines linear combinations of the feature set. Naïve greedy selection of features by choosing the dominant feature in each principal component is effective, but fails to account for redundancy in the feature set. The selectivity of PCA can be enhanced by modifying the optimality criterion to favor sparse directions of maximum variance by imposing either an $L1$ constraint on the principal components [90] or a sparsity constraint through Semi-Definite Programming [14]. These PCA-driven approaches could be incorporated in a framework for Directions of

Maximum Covariance [69] to discover directions in the data that maximize variation in labels.

3.3.1 Covariance Analysis

While PCA is useful in many settings, its choice of directions in feature space do not necessarily lead to good discrimination since class labels are not used. Instead, we apply a similar method of data analysis, referred to as *maximum covariance* by Shawe-Taylor and Cristianini [69]. This technique determines the directions in feature space that maximize the covariance between observations and their class labels (correct classifications). This is accomplished through a singular value decomposition of the covariance matrix $C_{xy} = \operatorname{cov}[X, Y]$, or the correlation matrix $\operatorname{cor}[X, Y]$ where Y is the corresponding set of labels for each observation.

To apply the directions of maximum covariance technique, consider X to be an m-dimensional random variable representing the features of a given observation and Y to be an k-dimensional random variable corresponding to the label of X. Given a set of n observations of pairs $\{(X_i, Y_i)\}_{i=1}^{n}$, the empirical covariance matrix is given by

$$\begin{aligned}\hat{C}_{xy} &= \mathbb{E}\left[(X - \mathbb{E}[X]) \cdot (Y - \mathbb{E}[Y])^{\top}\right] \\ &= \mathbb{E}\left[X \cdot Y^{\top}\right] - \mathbb{E}[X] \cdot \mathbb{E}[Y]^{\top} \\ &= \frac{1}{n}\sum_i X_i Y_i - \frac{1}{n^2}\sum_i X_i \sum_i Y_i\end{aligned}$$

The singular value decomposition decomposes C_{xy} by $C_{xy} = U\Sigma V^T$ where U is a $m \times m$ unitary matrix of x-principal components, Σ is a diagonal matrix of covariances, and V is a $k \times k$ unitary matrix of y-principal components. By computing directions of maximum covariance, the resulting principal components maximize the covariance between the random variable X and Y.

3.3.2 Feature Selection by Covariance Analysis

The simplest method to perform feature selection via covariance analysis is a greedy approach in which features are ranked according to their contribution to the first principal component of the covariance matrix. Suppose that the first principal component is given by $u_1 = \langle u_{1,1}, u_{1,2}, \ldots, u_{1,m}\rangle$. Then we simply rank the i-th feature according to its corresponding squared contribution, $u_{1,i}^2$. However, this naïve selection mechanism entirely ignores the possibility of redundancy between features. This technique is similar to a method of linear regression referred to as principal components regression in Chapter 6.7 of Shawe-Taylor and Cristianini [69]. However, in our approach we greedily select a single feature based on the weighting of the first principal component.

To remove some degree of feature redundancy, we can deflate the covariance matrix as features are chosen. In this technique, features are selected in a greedy fashion, but after each selection, the covariance matrix is deflated by the basis vector corresponding to the selected feature. The result of this operation is

$$C'_{xy} \leftarrow \left(I_m - e_i e_i^\top\right)^\top C_{xy} \left(I_k - \alpha_i \alpha_i^\top\right)$$

where e_i is the i-th basis vector, I_m is the $m \times m$ identity matrix, , I_k is the $k \times k$ identity matrix, and $\alpha_i = C_{xy} e_i / \|C_{xy} e_i\|$. This technique is equivalent to iteratively selecting the most prominent feature of the first principal component, removing the contributions of the selected feature from the covariance, and recomputing the first principal component. By recalculating the principal components from the deflated covariance matrix, covariance captured by the selected feature is removed so that subsequent selections concentrate on portions of the covariance not captured by initial choices. By deflating, redundancy can be reduced, but the selection process is still greedy so the optimal subset of features is not necessarily chosen.

The greedy approach presented here suffices for our demonstration of feature selection. In particular, in analyzing the features most relevant for distinguishing between a user and our test viruses, we found the following partial ranking:

1. Ratio of emails with attachments
2. Binary attachment
3. MIME type `application/octet-stream`
4. Magic type `application/x-ms-dos-executable`
5. Unclassified binary magic type
6. Frequency of emails in window
7. Number of attachments
8. Mean words in body
9. Mean characters in subject
10. Magic type `application/zip`

Not surprisingly, the dominant features personify the fact that viruses' behavior deviates primarily in the number and type of attachments used as well as the rate at which they send email. This list demonstrates the redundancy problem in that three of the top five features are related to whether an executable attachment is present.

In the future, more judicious pruning of redundant features is warranted to capture the essential information. Preliminary feature selection based on Fisher discriminant analysis [69] produced results similar to our method.

3.3.3 Choosing Features for Virus Detection

We next demonstrate the effectiveness of our feature selection by evaluating the performance of classification models in detecting viral traffic using artificial email traces. To this end, we captured real email worm messages from the Bagle.f, Netsky.d, Mydoom.u, Mydoom.m, and Sobig.f email worms by infecting VMWare virtual machines and using a transparent SMTP proxy setup to intercept all SMTP traffic on port 25. We chose these worms both due to their virulence and because each behaves in a slightly different manner with regards to our features. For more details on the behavior of these viruses, refer to Table 3.

Having collected samples of virus emails, we created a set of simulated email traces meant to mimick the behavior of a computer as it becomes infected. We constructed these artificial traces by combining clean and infected email data. The clean email trace was obtained from one of the investigators' 'Sent mail' folder. To simulate worm activity, we interleaved viral emails into the clean email corpus to simulate infections. The dates of the infected messages were corrected to maintain consistency, but inter-arrival times were kept the same so that frequency information was retained.

This process of infection simulation was used to create a set of training and test sets[1]. The training set consisted of 800 normal emails and 800 viral emails from 2 different viruses (400 emails from each virus). The test set consisted of 3000 normal emails and 1200 viral emails from 3 different viruses. In both training and test data sets, data was interleaved to simulate periods of normal activity followed by a burst of activity due to an infection. In the case of the training data this consisted of 400 normal emails, 400 emails for the first infection, 400 normal emails, and the 400 emails from the second infection[2]. Similarly, in the test data normal and viral emails were interleaved in periods of 1000 normal followed by 400 viral messages. Moreover, the training and test viruses were disjoint and ten such pairs of disjoint sets were created.

Having applied feature selection, we used several well-know statistical models to evaluate the performance of our feature set as the features were incrementally added

[1]These sets can be obtained from the investigators upon request.

[2]The pattern of alternating between normal and infected periods of activity is somewhat artificial, but does exhibit the fundamental behavior of bursts of viral emails after long intervals of normal traffic.

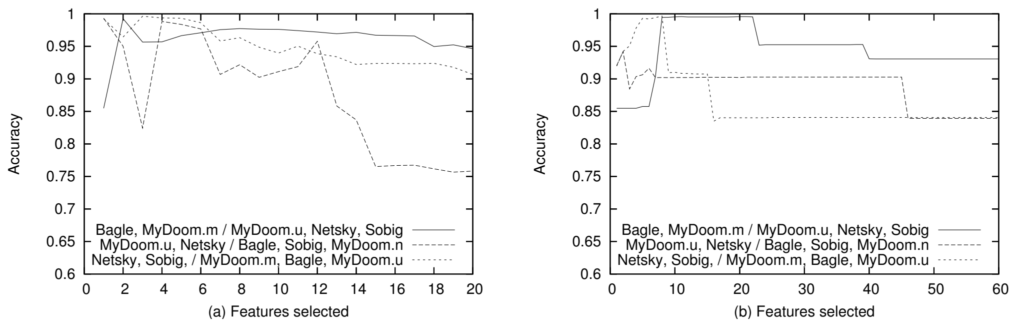


Figure 4: (a) Change in model accuracy as more features are greedily added to an SVM model. (b) Change in model accuracy as more features are greedily added to a Naïve Bayes Classifier. In the key, the first two specify the viruses trained against, and the last three the viruses tested against.

according to their ranking. In particular, we used a one-class Support Vector Machine (SVM) with a Gaussian (RBF) kernel and a Naïve Bayes model. For the SVM, the data was standardized via a PCA transformation and the one-class SVM was trained to allow only a small fraction, 0.1%, of outliers during training. Meanwhile, the Naïve Bayes model is the standard two-class probabilistic model widely used in spam detection [47, 66]. These models were selected as they are the models deployed in the actual virus detection system and are discussed in more detail in Section 4.3 and 4.4 respectively.

3.3.4 Results and Discussion

For each dataset, the models were trained on the training data. However, as discussed in Section 4.3, since the one-class SVM is an outlier detector it was only trained on the 'normal' training email. Once trained, each model was assessed on the corresponding test set. The results of testing the SVM and Naïve Bayes models with a variable number of features are shown in Figure 4 as plots of the overall classification accuracy. While not shown in the figure, the false positive rate generally started high due to the lack of generality caused by fewer features and decreased as more features were added. The false negative rates generally increased due to overfitting as extraneous features were added (although a few experiments initially had high false negative rates due to the lack of generality of a limited number of features).

It is also evident in Figure 4 that the degradation in performance of the Naïve Bayes and SVM models differ. The step-like performance of Naïve Bayes is attributable to the thresholding of the posterior probability in determining the classification of an email. Meanwhile, the more erratic behavior of the SVM is likely due to its instance-based nature. The SVM's classification boundary is supported by representative emails which may change substantially as the feature set is altered.

These experiments reflect well known guidelines; too few features are insufficient for generality while too many features cause over-fitting. In addition, the curse of dimensionality, which says that the size of the training set necessary to learn the classification function grows exponentially with respect to the dimensionality of the data, further deters a bloated feature set. However, these guidelines are not necessarily applicable to the one-class SVM, which has the ability to limit overfitting through appropriate selection of model parameters - a technique known as 'regularization'. Similarly, the amount of training data required for SVM generalization is not necessarily governed by the data's dimensionality, but rather by the 'kernel matrix'. While these topics are beyond the scope of this discussion, further information is available in Shawe-Taylor and Cristianini [69].

Chapter 4

User-based models as a Virus Detection System

Our system is designed to prevent novel email-borne worms from spreading before detection signatures are deployed by restricting propagation via extrusion detection as discussed in the introduction. The prominent statistical method for detecting novelties is aptly referred to by novelty, anomaly, or outlier detection [39]. A novelty detector learns a boundary around normal data in feature space by first being trained on known non-viral data, and then uses the boundary to classify unseen email as either normal or anomalous. Novelty detection is a well-suited classification paradigm for our problem since it is easy to observe the behavior of a single user, while inter-virus behavior may differ dramatically and future novel viruses could be designed specifically to deviate from typical viral characteristics.

However, novelty detection also has several drawbacks. Since a novelty detector is trained solely on "normal" data, standard approaches fail to exploit information gleaned from available data known to be anomalous. Moreover, implementing incremental novelty detection is problematic. The problem is that novelty detection is an unsupervised technique that assumes that anomalies are rare. In the case of virus detection, while infections may be rare, once an infection occurs, the virus will generate emails at a rate far higher than a typical user. Moreover, were the algorithm to autonomously select data to retrain on, it would bias the training distribution as is discussed in Section 5.6.1. As such, we decided to employ a multi-layered approach to virus detection that uses both novelty detection as well as two-class classification.

The overall architecture of the multi-tiered approach is motivated in Section 4.1. After discussing our design decisions, the individual components of the ar-

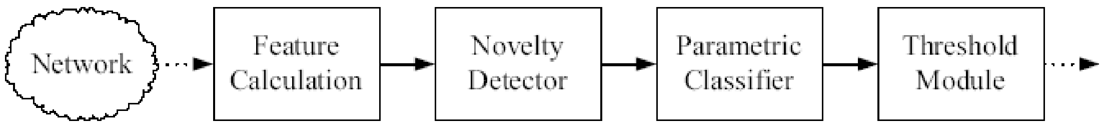


Figure 5: The classification pipeline. Calculated outgoing message features pass through a global novelty detector, and the results are filtered through a per-user parametric classifier, with any possibly infected messages reported.

chitecture are laid out sequentially: feature calculation (Section 4.2), novelty detection (Section 4.3), parametric classification (Section 4.4), and our quarantining strategy (Section 4.5). Finally, Section 4.6 presents an empirical analysis of this system followed by a discussion of its utility.

4.1 Our Multi-tiered Approach to Virus Detection

Unfortunately, a significant drawback of using novelty detection is the difficulty in selecting a sensitivity that is sufficiently high to yield a low false negative rate (*i.e.*, viral messages classified as non-viral), while also yielding a low false positive rate (*i.e.*, non-viral messages classified as viral). Too many false negatives would allow an outbreak to propagate, while too many false positives could result in alerts being ignored by human supervisors, rendering the system useless. This selection problem is a common one for adaptive intrusion detection systems [57].

In contrast, the classification framework provides the ability to incorporate both normal and viral data into the training of the decision boundary. That is, since a classifier can incorporate viral information, it can leverage knowledge from previous viruses to provide a stronger boundary. In fact, empirical studies have shown that two-class classifiers often outperform novelty detectors by creating a second 'novel' class from sampling from a reference measure [73].

To improve performance, we implemented a two-layer classification pipeline in which novelty detection plays a secondary role of isolating suspicious traffic for further inspection by the classifier as depicted in Figure 5. All outgoing email from local machines is directed through a parsing module to compute statistical features (see Section 4.2), which are then passed through a novelty detector to eliminate messages that have a high probability of being non-viral (see Section 4.3). Suspicious messages are then passed on to a secondary classifier stage to determine whether they are false positives from the novelty detector or actual infected email (see Section 4.4). Messages

Feature Description	Type		Return Value	
	Per-Email	Per-Window	Continuous	Multinomial
HTML in the Email?	✓		✓	
Number of Attachments	✓		✓	
Email has Binary Attachment	✓			✓
Ratio of Emails w/wo Attachments		✓	✓	
Email Frequency		✓	✓	
Mean Words in Body		✓	✓	
Variance of Words in Email Body		✓	✓	

Table 2: A summary of the 7 features used in the implementation of the complete email infection detection system. The first column describes the statistic being calculated, followed by whether each feature is calculated on a per-email or per-window basis, and the type of value returned.

labeled as viral by the pipeline will be used to classify individual machines as infected, at which point they will be quarantined and reported to a network administrator.

4.2 Feature Calculation

The first step in classifying an outgoing email message is computing statistical features that describe attributes of both individual messages and sender activity. Features are calculated in two ways: on a per-email basis (*e.g.*, the number of words in the body of a message), and over a window of email traffic (*e.g.*, the rate at which email is sent over a set of messages). Each feature is represented as either a continuous value or a multinomial vector: an example of a multinomial feature is the type of attachments sent with an email, where each entry indicates the presence or absence of a specific type of attachment. For a more thorough discussion of these distinctions refer to Chapter 3.1.

In order to accommodate this wide variety of features and to process emails efficiently, feature modules were implemented in Java. These modules were made to interact with live or stored streams of email traffic and extract the necessary feature in an efficient way to make real-time analysis realistic. For more details on this architecture refer to the thesis of Steve Martin [41].

In the end, we implemented 24 features calculated on outgoing email behavior. However, as discussed in Chapter 3.3.3, using too many features causes over-fitting of both the novelty detector and the parametric classifier, which reduces accuracy by increasing the false positive rate. To select the set of features used for the results in this paper, we analyzed combinations of features for their contribution to model accuracy; a process detailed in Chapter 3.3. The resulting seven statistics are described in Table 2. Each feature value is part of a data point that is passed on to the novelty detector.

4.3 Novelty Detection for Virus Detection

The fundamental process of novelty detection involves estimating a classification boundary in feature space such that data generated by the training data's distribution is likely to be classified as normal while excluding regions of feature space that are unlikely to contain data from the training distribution. For the most part, novelty detection has been accomplished through density estimation (the process of estimating the probability density function of the training distribution [27, 29]) by thresholding the estimated density function to create a boundary. In fact, in our initial project, we used a Gaussian Mixture Model to estimate the density and applied extreme value statistics to estimate the probability that a point was extreme (novel) given our model [59]. However, our current implementation uses a different approach to novelty detection.

One of the primary criticisms of using density estimation to solve novelty detection problems is that solving the more general problem goes against a simplicity principle often credited to Vapnik [61]. Instead, the technique we employed was that of the one-class support vector machine (SVM), which solves a simpler problem. Namely, the one-class SVM estimates the support of the distribution [61, 62]; that is, the smallest region whose complement has zero probability.

4.3.1 The One-Class SVM for Novelty Detection

The one-class SVM is able to create non-linear boundaries in feature space by means of kernel functions (discussed later in this section). It is one of several linear-pattern algorithms[1] that estimate a distribution's support. Several such methods attempt to bound the distribution within a hypersphere with a minimal radius [69] while one-class SVMs attempt to maximally separate the training points from the origin via a hyperplane. While the latter idea seems less like novelty detection and has been criticized for using this prior assumption of origin-separability [6], the one-class SVM has been shown to correspond to the minimally enclosing hypersphere under appropriate

[1]Linear-pattern algorithms should not be confused with algorithms that run in linear time. Rather, linear-pattern algorithms are pattern analysis algorithms that estimate a function $f(\mathbf{x})$ solely in terms of inner products: $\mathrm{f}(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$

conditions on the kernel functions [2]. Moreover, there has been extensive work done on the efficient implementation of support vector machines both in terms of their running time and the sparsity of their solution as is discussed in Appendix A.2. To the best of our knowledge, no such efficient implementation exists for the hyperspheres although their solutions are still sparse.

Essentially, the one-class SVM is a slight modification of the well-known two-class SVM that attempts to maximally separate a single 'normal' class from the origin rather than maximally separate two distinct classes. As with its counterpart, this separation is done by a hyperplane which forms a boundary; all points that fall on one-side of the boundary are classified as 'normal' while those on the other side are 'novel'. Both techniques employ a 'maximal-margin' argument that attempts to maximize the distance of training points to the separating hyperplane. Allowing some of the training data to fall on the wrong side of the hyperplane provides a more flexible approach and often provides better separating boundaries.

The dual objectives of maximizing the margin and minimizing the 'slack' (degree of misclassification) lead to a trade-off that is often personified by the 'ν-soft SVM' in which ν serves as an upper bound on the fraction of outliers [62, 69]. The ν-soft one-class SVM is represented by the following quadratic optimization [61, 62]:

$$\begin{aligned} \min_{\mathbf{w},\xi,\rho} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho \\ \text{s.t.} \quad & \forall i \quad \langle \mathbf{w}, \mathbf{x_i} \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0 \end{aligned} \tag{4.1}$$

where the training points are the set $\{\mathbf{x_i}\}$, $\mathbf{w}$ is the tangent vector of the hyperplane, ξ are the slack variables, and ρ is the displacement of the hyperplane. The above minimization is a quadratic program that has a dual formulation and a global optima that can solved for efficiently using the SMO algorithm [32, 55, 56].

A downside to the SVM approach is that SVMs are instance-based learners; that is, their boundary is a function of the training points[3], thus making it more expensive to do queries once the SVM is trained. Nonetheless, the so-called primal optimization program in Eq. (4.1) offers insight from the KKT-Conditions [54]. In this case, these conditions show that either a training point is on (or beyond) the boundary, or that training point does not contribute to the decision function used to classify new

[2]The one-class SVM is equivalent to the minimally enclosing hypersphere for a kernel κ such that $\exists c \quad \forall \mathbf{x} \in \chi \quad \kappa(\mathbf{x}, \mathbf{x}) = c$ [61]. Currently, all of the kernels considered in this work have met this condition.

[3]For a linear kernel $\kappa_{linear}(x, y) = x\dot{y}$, the decision boundary is efficiently parameterized in the original space, but this kernel restricts the SVM to linear separation.

queries. Hence, this provides sparsity in the training data such that only the so-called 'support-vectors' (data-points on or beyond the boundary) are required to compute the decision function.

Finally, the one-class SVM that has been discussed up-to this point is linear in the original space but this approach can be made non-linear by an implicit mapping into a higher dimensional 'kernel space'[4] via a special kernel function. As was seen in the primal and similarly for the dual program and the decision function, the data points only were used in inner products; essentially a measure of similarity. By generalizing this notion of dot-product, the kernel function computes the inner product in a high-dimensional space achieved by some mapping ϕ without explicitly using the mapping. In fact, given certain conditions on the kernel[5] we are guaranteed that it represents the inner product between some mapping of the points. In this way, the kernelization of the one-class SVM allows this technique to be applied in a high-dimensional kernel spaces, which corresponds to non-linear separation in our feature space.

4.3.2 Novelty Detection as a Filter

As mentioned in Section 4.1 we employed a one-class SVM to filter data points to identify messages that have a very high probability of being normal. As with the feature selection procedures, a simple Gaussian (RBF) kernel was applied to PCA-ed data. This simple approach to the kernel was done as a matter of convenience and a tailored kernel designed for email is being studied.

We fit our one-class SVM so that it is oversensitive by selecting parameters for a 25% false positive rate, which empirically results in a false negative rate under 2%. By excluding a large portion of the fringe normal email, the novelty detector can report, with high confidence, that certain messages are not anomalous. Thus, the novelty detector acts as a filter to a classifier by isolating common email behavior without creating a model for anomalous message characteristics. As shown in Anil Sewani's thesis, this technique is effective at reducing the false positive rate [68].

In our design, the one-class SVM also acts at a global level; that is, the same SVM is used for all users in order to gain a generalized view of what constitutes 'normal' message activity. By having a single SVM, there is ample data for training whereas single users may have a relatively limited outgoing mail history. However, while the

[4]While traditionally called the feature space, I'll refer to the implicit space by kernel space to avoid confusion with earlier usage of the term feature.

[5]Mercer's Theorem shows that a kernel κ is positive semi-definite if and only if there exists some mapping ϕ such that $\forall x, y \quad \kappa x, y = \langle \phi(x), \phi(y) \rangle$ [69]

SMO algorithm makes SVM learning feasible for large data sets, an enterprise system of mailboxes would make training a global SVM challenging and almost certainly an off-line task. Moreover, as discussed in Chapter 5.6.1, an improper retraining could bias the SVM. For the time being, the SVM is a static model that is only trained at inception.

4.3.3 Kernel Selection

Although we used a Gaussian (RBF) kernel for the actual system tests, the following section examines the effect of kernel choice for our data by looking at a polynomial kernel[6] of the form $\kappa_p(x, y) = (\langle x, y \rangle + R)^d$ where $d \in \mathbb{Z}^{0+}$ is the degree of the polynomial and $R \in \mathbb{R}^{0+}$ is a displacement [69]. Higher values of R decrease the relative weight on the higher degree polynomials. This kernel was chosen to present how the "learning capacity" expands as the degree of the kernel space is increased; in particular, higher order polynomial kernels have higher dimensional kernel spaces [69].

The following experiment demonstrates the performance of a one-class SVM using a polynomial kernel κ_p described above. Thus there are three variables: ν, d, and R. The experiment consisted of a training set of 400 normal emails and training sets of 1200 normal and viral emails from 6 viruses. For each set of parameters, the training and testing was performed 20 times to remove random variations. The variables were sampled over various ranges ($\nu \in \{0.05, 0.1, 0.15\}$, $d \in \{1, 2, \ldots, 19\}$, and $R \in \{0.1, 0.2, \ldots, 9.9\}$).

The results of this kernel selection experiment is presented in Figure 6 in terms of the false positive rate FP, false negative rate FN, and a composite metric $\max(FP, FN)$ which is one of several ways to represent the trade-off between the two quantities. As can be seen, lower values of R are preferred regardless of the value of ν. However, what is salient is the degree of the polynomial. For $\nu = 0.05$ and $\nu = 0.1$ low degree polynomials do not have the capacity necessary to properly separate the training data resulting in underfitting the model (a high false-negative rate). However, there is a sudden drop in the false-negative rates at about $d = 8$ and continue to drop from there. Conversely, the false positive rates steadily increase with the degree of the kernel; indicative of overfitting. This effect is not as apparent for $\nu = 0.15$ as the false negatives are much less pronounced. Nonetheless, these tests show that as the

[6]The kernel was actually normalized to $\kappa_p{}'(x, y) = \frac{\kappa_p(x,y)}{\sqrt{\kappa_p(x,x) \cdot \kappa_p(y,y)}}$.

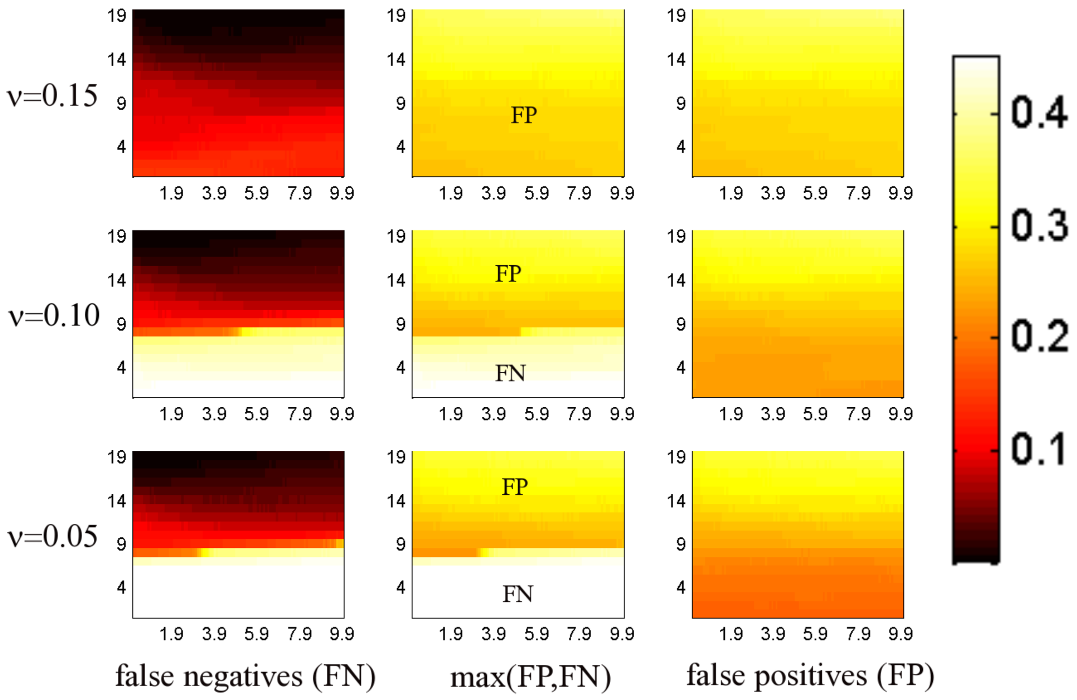


Figure 6: Plots of the false positive/false negative performance of a one-class SVM using a polynomial of the form $\kappa_p(x,y) = (\langle x,y\rangle + R)^d$ where the x and y axes are the parameters R and d respectively. The ν parameter is an argument to the SVM discussed in Section 4.3.1. The leftmost column shows the false negative rates of the kernels over 20 tests on the PCA-ed data and the rightmost column shows their false positive rates. The central column shows the maximum of the two rates; one of many possible measures. In general, increasing the degree of the polynomial gives lower false negative rates but progressively overfits yielding higher false positive rates. Thus, kernels must be judiciously selected to prevent such overfitting effects.

dimensionality of the kernel space is increased, the SVM overfits the data but a certain amount of complexity is required to capture the required separation.

This fact is reinforced by a second experiment that examines the degree exclusively in Figure 7. At one extreme, for small degrees of $d < 18$, the kernel doesn't have adequate complexity and thus underfits. At the other, large degree polynomials have progressively higher false positive rates due to overfitting. However, for degree $d > 150$ the behavior becomes erratic and eventually results in SVMs that classify all space as normal. This seems to be due to numerical precision errors that result in a saturated kernel matrix (the kernel is a constant function).

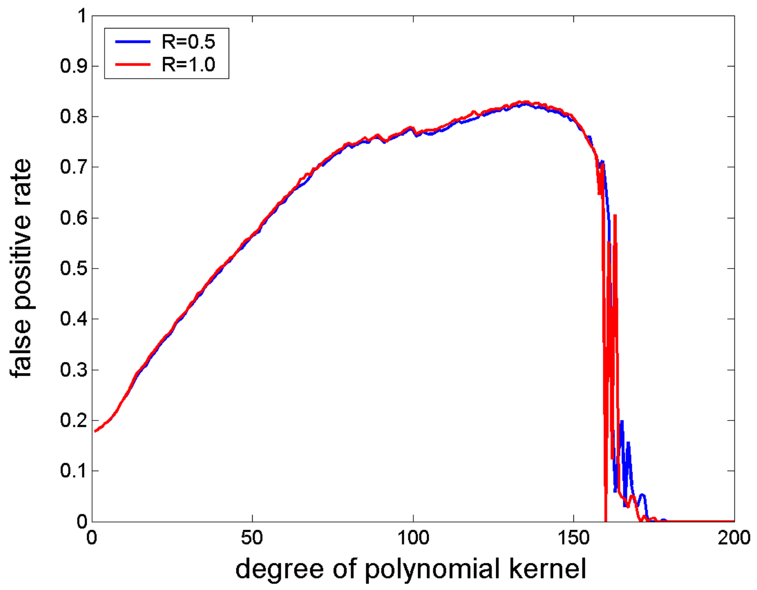
The conclusion of these tests is that the performance of our outlier detector could be improved by restricting the complexity of the simple polynomial kernels we have considered to reduce overfitting. For the Gaussian kernel used in our design, this analysis is not applicable. The dimensionality of this kernel is infinite, but many directions have minuscule weight. Nonetheless, this analysis suggests that one must be careful in kernel selection. Ideally, one should use a kernel tailored to the structure of data. Kernel selection for our email detection project is still under consideration.

4.4 Parametric Classification

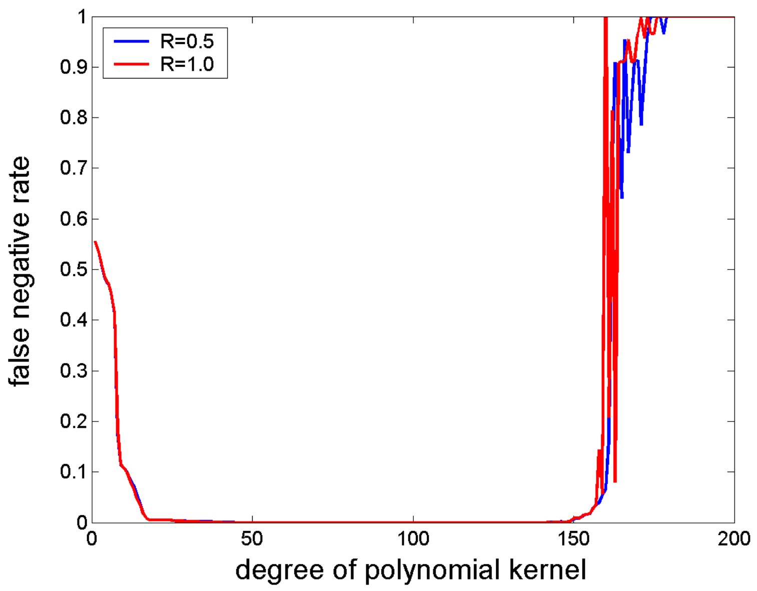
The next pipeline stage is a parametric classifier to differentiate between normal and viral traffic. While many methods could be applied to this task, we chose to implement a Naïve Bayes model that uses both labeled and unlabeled messages to approximate the joint distribution of normal and infected email.

Naïve Bayes models classify by applying Bayes rule to observed data via class-conditional distributions. The probability of the data is given by the distribution's fit to known infected data. The simplifying assumption made by Naïve Bayes models is that the features of an observation are independent given its classification. While this assumption is often violated, the model is widely used in spam detection [47, 66] and suffices for our purposes.

In contrast to the novelty detector, the classifiers are per-user models capturing the individual user's email behavior. Thus, after an email has been deemed suspicious by the novelty detector, a personalized model compares the email's characteristic to the user's previous behavior and to that of known viruses. Moreover, since Naïve Bayes



(a) False Positives



(b) False Negatives

Figure 7: The false positive and false negative for a polynomial kernel κ_p with varying degrees of freedom and two values of R chosen from the results of Figure 6. In this test, we perform 20 tests on the PCA-ed data for each set of parameters to the kernel. The classifier trained is a one-class SVM with $\nu = 0.05$.

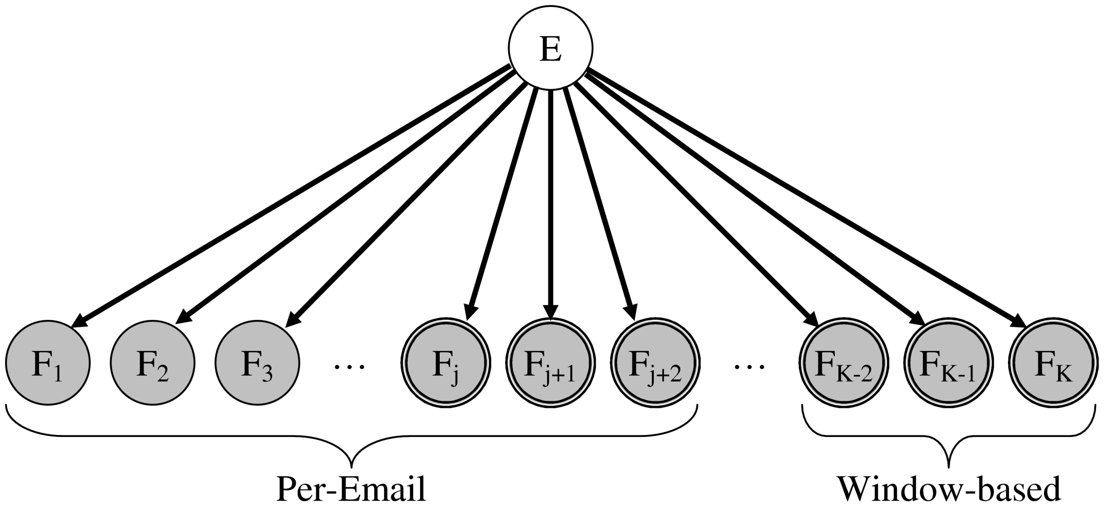


Figure 8: Our Naïve Bayes model represented as a graphical model [29]. The unobserved parent variable E is the email state – whether the email was viral or not. All child nodes F_i represent feature variables and they are observed. These feature variables are partitioned between 'Per-Mail' and 'Window-Based' features as discussed in Chapters 3.1.1 and 3.1.2 respectively. Moreover, all feature variables are over finite enumerations except for the continuously-valued features represented by nodes with double circles $\circledcirc$. The directed arrows of the graphical model represent the model's dependency structure; *e.g.*, the arc from E to F_1 represents the distribution $\Pr(F_1|E)$. Finally, the root node E has a prior distribution $Pr(E)$.

models generally require little training data[7], there is less concern about the size of individual's training set than there was for SVMs.

Finally, the Naïve Bayes learners fit in nicely to our system goals paradigm as they are light-weight (a small constant amount of state per model) and provide quick classification – both requirements for deployment on a mail server. The remainder of this section details the Naïve Bayes algorithm and how we used it in our pipeline.

4.4.1 Naïve Bayes Classification

More specifically, Naïve Bayes models use Bayes rule to calculate label probabilities, with an important simplifying assumption that the features of an observation are independent given its classification; a dependency structure that is captured by the graphical model in Figure 8. This simplification means that the joint distribution of the

[7]Naïve Bayes and other parametric probabilistic models require little data when the appropriate model is used [29].

model can be factored into individual class-conditional densities:

$$\Pr(X_1, X_2, \ldots, X_n, Y) = \Pr(Y) \prod_{i=1}^{n} \Pr(X_i|Y)$$

where X_i is the i-th feature, Y is the class label, $\Pr(Y)$ is the prior distribution of Y, and $\Pr(X_i|Y)$ is the class conditional probability of X_i given the label Y.

This joint probability is applied to classification via Bayes rule which allows us to calculate the posterior probability of the class label:

$$\begin{aligned}\Pr(Y|X_1, X_2, \ldots, X_n) &= \frac{\Pr(X_1, X_2, \ldots, X_n, Y)}{\Pr(X_1, X_2, \ldots, X_n)} \\ &\propto \Pr(X_1, X_2, \ldots, X_n, Y)\end{aligned}$$

where the proportionality constant is simply the sum of the above probabilities over all values of Y (due to the fact that the conditional probability must sum to 1). This concludes a basic overview of the Naïve Bayes classifier we used but a more in-depth explanation of the implementation is provided in Appendix A.3.

4.4.2 Feature Distributions used in the Model

By choosing a Naïve Bayes classifier we choose the generative approach to classification, *i.e.*, we explicitly modeled the probability distributions of each feature conditioned on the latent variable that indicated whether or not an email was viral. Thus, we had to describe the distributions of each of our features based on empirical evidence. If a good model of the data is available, generative models can leverage the information inherent in the condition distributions to produce accurate predictions with relatively little data. However, the difficulty lies in producing a good model of the data.

For data confined to finite outcomes, this process was relatively simple. Each feature was modeled as a multinomial distribution and each class was assigned its empirical probability observed from the training data; this process is automatically done as part of the EM algorithm for partially labeled data. Moreover, simple smoothing was used to boost the probabilities of rare events by ensuring all probabilities were at least a small constant.

The process was more difficult for features with non-finite support. Empirically, we examined the distributions of our continuously-valued email data and fit distributions to it. The goodness of the fit of a particular distribution was accessed visually by means of quantile-quantile (QQ) plots. However, rather than consider a rich set of distributions, we found that fitting the data to Gaussian or Exponential distributions produced good enough fits for our purposes. However, future work could produce better results by considering a wider spectrum of distributions.

While our distributions of choice were sufficient for the user data, they did not correspond well to the viral data at all. As seen in Figure 1, the viral data is often distributed in a spiky fashion as compared to the smooth distributions seen for user data. In part, these discrepancies are due to the fact that an individual user exhibits a broad range of behavior while a single virus has almost no variance. However, between different viruses features can differ substantially. This discrepancy was the motivation for novelty detection, but it is not necessarily a good fit for generative models. Thus, it remains to be seen if more accurate modeling of viral behavior could yield a better classifier.

4.4.3 Training User Models

We train our parametric classifier on a per-user basis so that the distributions learned for clean email are specific to each author, enabling more accurate classification of worm infection based on message sending behavior. Messages classified as worm propagation are passed on for quarantine and use in determining if the sending machine is infected.

An individual's Naïve Bayes model is constructed by training on messages extracted from the user's 'Sent-Mail' folder in order to obtain an initial sample of the user's behavior. Meanwhile, a repository of viral messages is used to construct the viral distributions. Initial training is fully supervised (all email labels are known) resulting in maximum-likelihood (ML) estimates of the model parameters. However, the system is also designed to support retraining on unlabeled email.

In a typical scenario, as a user sends email, we want to incorporate the changing behavior of that user in retraining the model, but we lack an authority to label the messages. Fortunately, the Naïve Bayes models we use can be retrained with partially labeled data by means of the famous Expectation Maximization (EM) algorithm [29]. At one extreme, if all the data were entirely unlabeled, training with EM would amount to clustering, while EM training with fully labeled data is ML-estimation. Hence, in the partially labeled case, EM is performing guided clustering (some data points are known to belong to a particular cluster) in which unlabeled points have partial membership in clusters. The unlabeled points contribute to the parameter estimation for each class according the degree of their partial membership to that class' cluster. For a more thorough discussion on the implementation details of training with EM see Appendix A.3 or Jordan's book on probabilistic graphical models [29].

Retraining with partially labeled data is effective, but we also have considered using a labeling mechanism to provide more labeled data as the system ages. In particular, we would like to leverage the information provided by traditional virus scanners. Of course, as mentioned in the introduction there is a potentially substantial lag between when a new virus appears and the release of signature for that virus. As such, we propose using a virus scanner to do delayed tagging; that is, email data remains unlabeled until a sufficient time has elapsed for the scanner to be updated and provide a more reliable label[8]. However, even virus scanners have false positives (though extraordinarily rare) and there is a potential that such mis-labeled emails could bias the distribution. For our purposes, the rarity of such events made them irrelevant for our current study.

Finally, although the EM algorithm learns the prior distribution of the class variable, in practice, this parameter was manually adjusted. This is due to the fact that in training our models, we wanted to train over a large sample of viral email from our viral repository. However, by using a large set of viruses the model would also learn that the prior probability of a virus was artificially higher than one would expect in reality.

4.4.4 Temporal Dependencies

As mentioned in the discussion of our feature set in Chapter 3.1, our features violate several of the assumptions of a Naïve Bayes model. Clearly there are correlations between a number of our features (*e.g.*, the total size of an email and the number of attachments in the email are probably correlated) that Naïve Bayes models fail to capture. However, perhaps the most notable of these violations are correlations between features and themselves over time; *e.g.*, the value of 'Average Words in Body' clearly depends on its previous values as this feature is computed over a window of email. This temporal dependency raises several questions.

One of the more obvious aspects of virus infection that our Naïve Bayes model fails to capture is temporal dependencies. Clearly, individual emails are not identically drawn from a distribution but rather are instances of a sequence of "email state". Moreover, the concept of infection is itself a temporal concept (once infected, the distributions inherently change until a machine is disinfected). This observation has lead

[8]The paranoid may also consider never allowing a scanner to label an email as clean, but rather only able to label identified viral emails as such.

us to consider sequential models that incorporate time dependency. In particular, we've been considering using a Hidden Markov Model (HMM) [29] to model the changing state over time. In such a model, the current state of an email depends on the previous email's status.

The presence of temporal dependencies has also been disquieting. As mentioned in Chapter 3.1, the identity of the state variable E in Figure 8 is unclear since some features are based on a single email while others are characteristics of the email traffic. Initially, we attempted to build a more precise interpretation through a more complex graphical tree-structured graphical model that separated flow-based and state-based features into separate components. However, this more complex model actually caused a slight degradation in performance. As such we used the Naïve Bayes model for its performance, but a lack of a clear interpretation for the state remains.

4.5 Thresholding and Quarantine

The final phase of the pipeline is the thresholding module which is designed to alleviate the potential for false positives but at the cost of introducing a few false negatives when infections do occur. Even a well-tuned detection system will have occasional false positives, so it would be overly intrusive to quarantine a system from the network on the basis of a single email classified as viral. Such a policy would interrupt users regularly thus making the system unreasonably disruptive. Moreover, when modern mass-mailing infections occur, the success of the infection depends on its ability to exploit the 'window of vulnerability' before traditional anti-virus scanners are updated [46]. Thus, we expect that infections will produce long streams of infected messages quickly making a policy of quarantining after a sufficient stream of viral-classified messages viable.

The basic quarantining strategy we have considered applies a threshold to the percentage of emails classified as infected over a sliding window of messages. Thus when that threshold is exceeded, it would be possible to report, with high confidence, a user as infected and take actions to quarantine them. We present an initial analysis of thresholding on our pipeline, which motivates its usage. Figure 9 describes how quickly our model reacts to a user infected with the Netsky worm. As an example of how such a threshold could be chosen, the horizontal dotted dashed line in the bottom graph of Figure 9 shows the point at which 30% of a window of 10 emails are actually infected, and the point at which 30% are classified as infected. Application of the 30% threshold

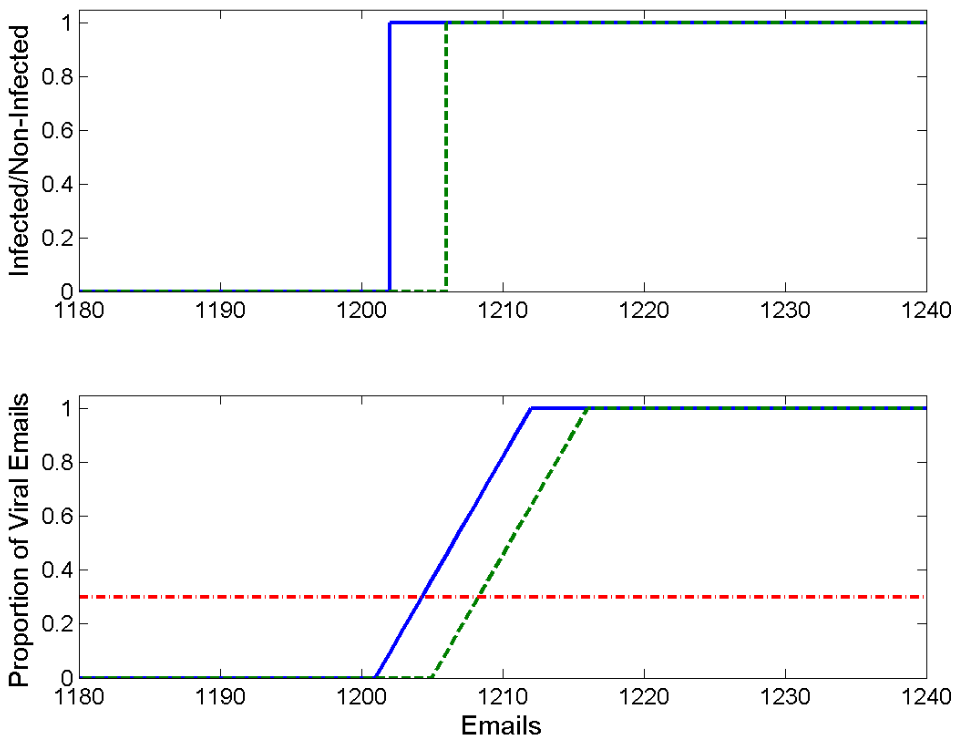


Figure 9: Reaction time in messages for our method to detect the propagation of W32.Netsky.D. The top graph shows the per-email results of our classifier with the test set, while the bottom graph shows the proportion of infected emails in a sliding queue as the infection commences, with the horizontal dotted dashed line marking the 30% mark. In both, the solid line is from the actual infection, and the dashed line from the classifications made by our method.

to our test runs shows the pipeline classifying users as infected for each worm we test on between 2 to 18 seconds from the actual time of infection. In addition, if the user is quarantined upon reaching this threshold, no worm in our evaluation set would be able to send more than 8 infected emails. Thus, our approach significantly reduces the worm's ability to propagate and in fact can reduce the propagation rate of the infection globally based on the level of deployment. For detailed analysis of the effect our system, see Anil Sewani's Thesis [68]

The other side of thresholding is the intent to reduce the possibility that false positives could cause a user to become falsely quarantined. The scheme discussed up to this point was meant to absorb small spikes of anomalous behavior by a legitimate user. However, the effectiveness of this policy depends partially on the assumption that each normal email is independently drawn from an identical distribution. However, as mentioned in Chapter 4.4.4, such an assumption is violated by the temporal nature of email. In fact, empirical examination of the false positives of our system found that false positives occur in bursts demonstrating a tendency for users to behave anomalously over a period of time. One such burst occurred when a user sent out a sequence of the same email to a number of users. Evidently, the user copied the same email multiple times and sent it quickly, causing the user to deviate over several successive emails. Moreover, bursty behavior is common among laptop users who queue up numerous outgoing messages between connections. These types of behavior, while not prevalent in our limited data, necessitate proper thresholding techniques.

The standard thresholding model discussed above can deal with bursts to some extent. However, by increasing the size of the threshold to deal with larger bursts, it also allows more viral messages to be released before an infected machine is quarantined. We are considering more advanced schemes to deal with bursts, however, we have yet to conduct an in-depth study on the size and probability of this bursty behavior to quantify the problem. For more on thresholding in viral detection and its effect on viral propagation refer to the thesis of Anil Sewani [68].

4.6 System Evaluation

To demonstrate the utility of our system for virus classification, we simulate email activity on the classification architecture. These tests demonstrate the accuracy of the predictive components of our architecture in identifying viral emails. However, these experiments do not assess the performance of the thresholding module for quarantining; its impact is discussed in Anil Sewani's Thesis [68].

4.6.1 Evaluation Methodology

For the results shown in this paper, we train and test on email traces constructed from real user activity and worm messages captured from actual infections on virtual machines. All clean email is from the outgoing mail of a single user; we used the sent mail from one of the investigators, a graduate student in the EECS department at UC Berkeley. The corpus of uninfected email spans an uninterrupted period of about two years, and is comprised of roughly 4,600 messages.

To capture worm email, we infected VMware virtual machines running Microsoft Windows 2000 and Windows 98 Second Edition with a variety of worms. Each

Characteristics	**Email Worms**					
	BubbleBoy	Bagel.F	Netsky.D	Mydoom.U	Mydoom.M	SoBig.F
Infection Techniques						
Attached .exe File		✓	✓	✓	✓	✓
Attached .zip File				✓	✓	
Script Embedded In Email	✓					
Uses Own SMTP Engine		✓	✓	✓	✓	✓
Uses Peer-to-Peer Apps		✓				
Polymorphism						
Body Variants	1	43	26	19	11	9
Subject Variants	1	26	6	31	Many	2
Attachment Name Variants	0	29	22	10	10	9
Attachment Type Variants	0	3	1	6	7	1
Anti-Scanner Mechanisms						
Uses No Attachment	✓					
Email Header Spoofing		✓	✓	✓	✓	✓
2^{nd} *Attachment Extension*					✓	
Can Send Decoy Messages				✓		
Can Encrypt Attachments		✓				
Can Compress Attachments					✓	
Virulence						
# Vulnerable Win32 OS's	3	6	5	6	7	6
Installs Backdoor		✓		✓	✓	
Distribution	Low	High	High	High	High	High
Damage	Low	Med	Low	Med	Med	Med

Table 3: A description of some of the widely varying properties of the six viruses used in our testing: VBS.Bubbleboy, W32.Bagle.F, W32.Netsky.D, W32.Mydoom.U (referred to by Symantec as W32.Mydoom.T), W32.Mydoom.M, and W32.SoBig.F. The infection techniques, degree of polymorphism, unique anti-virus scanner mechanisms, and virulence of each worm is compared. Most of this information is reported by Symantec [80].

virtual machine contained the address book used by the author of all the clean email. We trapped all outgoing email from the virtual machines, regardless of the SMTP server used to send it, by networking the host machine behind an instrumented transparent proxy. All traffic from the host machine was then dropped to ensure epidemic containment.

In this manner, we captured infected email from the *VBS.BubbleBoy*, *W32.Mydoom.M*, *W32.SoBig.F*, *W32.Netsky.D*, *W32.Mydoom.U*, and *W32.Bagle.F* mass-mailing worms. Table 3 lists several characteristics of each virus and shows that each one exhibits unique traits that test the flexibility of our techniques, such as spreading via multiple infection methods, using built-in SMTP engines, generating polymorphic emails that vary in both subject and body, and avoiding detection through strategies such as spoofing the sender's *From* address, sending out uninfected decoy messages, and encrypting or compressing attachments. As an example, unlike the other worms we captured, VBS.BubbleBoy uses Microsoft Outlook to spread as opposed to its own SMTP engine. It also infects users via a script embedded in the email that, when viewed by a buggy version of Outlook or Outlook Express, saves the worm script to the Windows startup folder.

To simulate deployment in a local area network, we construct training and test sets from clean and infected email data. Clean emails from our corpus are inserted untouched in each trace. To simulate worm activity, viral emails are then inserted into this message stream at specific "infection" points. The dates of the infected messages are corrected to maintain consistency in the trace, but time intervals between the inserted emails are kept the same so that frequency information is retained.

From the available viruses, we created six training and test set pairs using a leave-one-out validation strategy. Each training set is made up of 400 clean emails and 1000 worm emails, where the block of infected messages is made up of 200 examples from each of five different worms. The sixth virus is then included by itself in the test set, which contains 1200 clean emails and 200 infected messages. The messages from a simulated infection are placed at the end of the test set because most of the worms we use send email continuously at a high rate, and it would be unrealistic to arbitrarily cut off their activity to fit clean emails into the trace. Unfortunately, due the time-sensitive nature of many of our features, it would be inappropriate to randomly sample messages to create new training and test sets so this limited the number of tests we were able to conduct.

Experiment	‘Novel’ Email Worm Tested					
	BubbleBoy	Bagle.F	Netsky.D	Mydoom.U	Mydoom.M	Sobig.F
SVM Only						
Num. False Positives	198	219	219	215	222	222
Num. False Negatives	0	1	0	0	0	4
Num. Correctly Classified	1201	1179	1180	1184	1177	1173
% False Positives	16.50	18.25	18.25	17.92	18.50	18.50
% False Negatives	0.00	0.50	0.00	0.00	0.00	2.01
% Total Accuracy	85.85	84.27	84.35	84.63	84.13	83.85
Naïve Bayes Only						
Num. False Positives	33	17	17	17	20	17
Num. False Negatives	8	4	4	4	4	5
Num. Correctly Classified	1358	1378	1378	1378	1375	1377
% False Positives	2.75	1.42	1.42	1.42	1.67	1.42
% False Negatives	4.02	2.01	2.01	2.01	2.01	2.51
% Total Accuracy	97.07	98.50	98.50	98.50	98.28	98.43
Two-Layer Model						
Num. False Positives	9	10	10	10	12	10
Num. False Negatives	8	4	4	4	4	5
Num. Correctly Classified	1382	1385	1385	1385	1383	1384
% False Positives	0.75	0.83	0.83	0.83	1.00	0.83
% False Negatives	4.02	2.01	2.01	2.01	2.01	2.51
% Total Accuracy	98.78	99.00	99.00	99.00	99.00	98.93

Table 4: Evaluation results. See Section 4.6.1 for details on the training and test data. Each experiment was run three times: first with only the one-class SVM, then using only a Naïve Bayes parametric classifier, and finally with the two-layer system described in Section 4.1. We report the number of false positives, false negatives, and correctly classified emails. The percentage of false positives [negatives] is the percent of the ‘normal’ [viral] email misclassified. Note that W32.Mydoom.U is referred to by Symantec Inc. as W32.Mydoom.T.

4.6.2 Results and Discussion

Table 4 provides the results of our evaluation, showing the effectiveness of detecting novel worms by using the one-class SVM by itself, the Naïve Bayes classifier by itself, and our two-layer approach with both classifiers integrated into a single prediction mechanism. For the two-layer system, the false negatives are combined from the SVM and the classifier. All false positives, however, result from the classifier, as all email that is identified by the SVM as infected is further filtered through the classifier.

The one-class SVM (using a tuned RBF kernel), which is parameterized to be sensitive at detecting anomalies, captures almost all of the infected email with a very low false negative rate (0% in all but two experiments). However, this is at the cost of a high false positive rate of more than 16%. As shown in Table 4, filtering all infected emails from the SVM through the classifier eliminates most false positives at the cost of slightly increased false negatives. We observe that all false negatives from the classifier occur immediately after the user is infected due to the time needed for window-based features to reflect viral activity. For a window size of 10 emails, the classifier produces between 4 and 8 false negatives in all experiments. It should be noted that since all false negatives occur at the beginning of an infection, increasing the number of infected emails in our test sets would artificially reduce our false negative rate. Thus, it is not the percentage of false negatives we are concerned with, but rather the actual number of infected emails misclassified.

From the experiments presented here, it might seem that using the classifier alone would produce performance as good as the combined two-layer model. This is not true in general as suggested by Table 4. While it is desirable to have a large training corpus to correctly train the classifier, eliminating mundane data can produce a better classifier trained only on the "difficult" data. This technique is empirically validated in Table 4 by observing the increase in performance obtained with the two-layer model especially on the viruses *W32.Bagle.F*, *W32.Netsky.D*, and *W32.Mydoom.U*. Thus, the presence of a novelty detector that correctly filters the majority of the normal email improves performance of the whole model. However, by filtering, we alter the distributions of features, which further undermines the generative model's fit as discussed in Section 4.4.2.

Another important aspect of performance for a real network deployment is the number of emails required for training each parametric classifier before accuracy reaches acceptable levels. To evaluate this metric, we ran experiments in which the parametric classifier is trained and tested with increasing amounts of training email. After running each experiment 100 times, we found that on average no more than 50 clean emails are required to train a model that classifies messages with a false positive rate of around 0.1%. Most of the worms we use require around 20 infected emails to train a model with a false negative rate of around 2%. However, W32.Mydoom.M, requires a substantially larger training set before the model becomes accurate. As demonstrated in Figure 10, this virus required several hundred training examples before the false negative rate was acceptable - a large number of emails to collect for a single user. The results of these tests indicate that a deployed classifier generally will not require a large corpus of "clean" email to correctly classify a user. However, due to highly polymorphic worms like Mydoom.M, we envision our system being deployed with pretrained viral distributions that capture a wide variety of worms.

Overall, we believe that these findings are promising, and show the feasibility of our techniques for rapidly detecting novel worms. Our approach was able to learn characteristics from training viruses that allowed the resulting classifier to accurately predict the presence of novel test viruses. Moreover, the two-layer strategy proves to be effective at filtering out normal traffic with a novelty detection algorithm so that the classifier learners features from "difficult" examples. This results in an overall better classification accuracy than either technique produced individually.

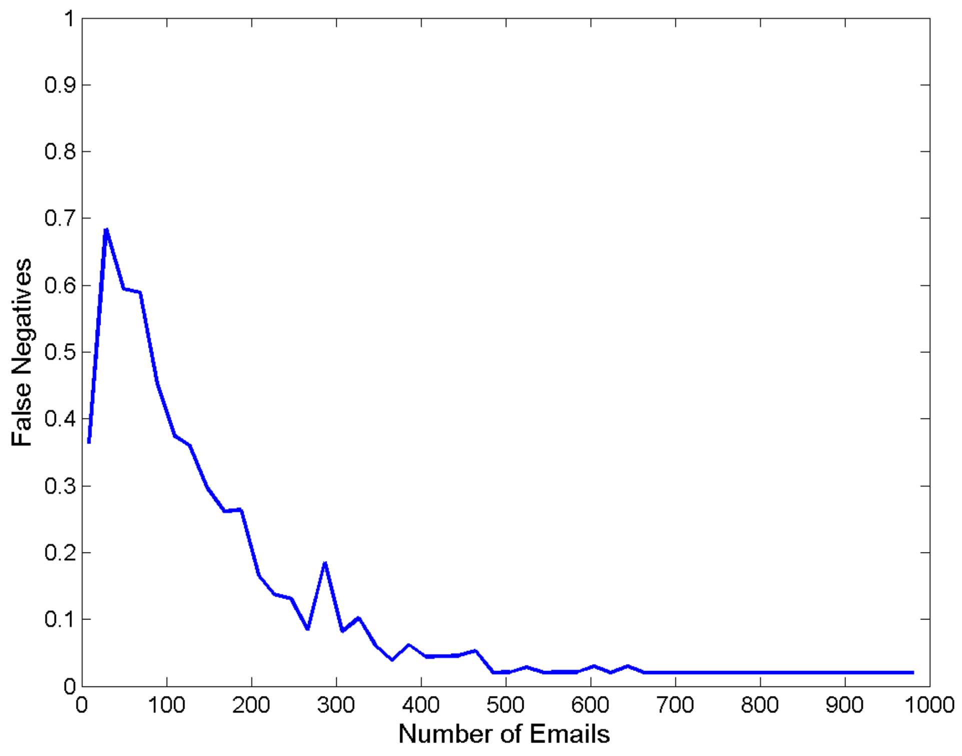


Figure 10: The false negative performance for the W32.Mydoom.M worm as the number of training emails increases.

Chapter 5

Attack Strategy Analysis

As techniques in statistical learning have matured, they have been applied toward a more and more diverse set of applications. Of particular interest to this study, statistical learning is being employed in security-sensitive applications such as spam filtering [47, 66], virus detection [74, 75], adaptive intrusion detection [38], fault detection [89], and fraud detection [17] systems. In such environments, there is a potential and very real threat of malicious users attempting to subvert these statistical learning techniques in order to compromise or disrupt the service of the system that incorporates them.

The field of learning theory is a mature research area that has led to many advanced techniques for pattern analysis. Many textbooks have been published on the subject; for example, see Jordan [29, 31, 69]. However, the vast majority of statistical learning theory is based on stationary distributions. This theory is not well-suited to situations in which an otherwise ordinary distribution is being affected by a malicious user. This issue was first addressed under the PAC[1] learning framework by Kearns and Li [30], in which they give bounds for the amount of 'malicious error[2]' that can be tolerated by learning systems. These bounds are applicable in the case where an adversary has the ability to arbitrarily manipulate data in the training set (polluting the training set) but do not apply to all attack scenarios as is discussed in the next section.

The goal of this chapter is to present the problem of learning in the face of an adversary and present an analysis of the effects of an adversary on a simple outlier detection scheme. The resulting analysis describes an optimal strategy the adversary

[1]The Probably Approximately Correct (PAC) model introduced by Valiant [83].

[2]Malicious error is a term used to describe the fraction of the training set that can be maliciously manipulated.

can use for altering the state of the learner. Moreover, it gives insight into the behavior of the naïve learning schemes discussed here.

This chapter begins with a general discussion of the idea of adversarial learning in Section 5.1 and suggests potential attacks against a naïve novelty detector in Section 5.2. A particular directed attack is formalized in Section 5.3 and a general attack strategy for the attacker is presented in Section 5.4. This attack's real-world feasibility is examined in Section 5.5. Finally, we discuss the potential for countermeasures against such attacks in Section 5.6.

5.1 Methodology for Attacking Statistical Learners

As research has only recently began to examine the effects of an adversary on learning, we have sought to categorize attacks based on the characteristics that they exhibit. There are a number of considerations that need to be made with regards to the security of a system. To begin with, it is important to quantify the knowledge the adversary has. In some cases, the adversary might only know what our system is designed to do, essentially acting like a black-box that the attacker can query. However, the attacker may know a great deal about our system – *e.g.* reverse engineering and insider attacks. Unfortunately, all too often system security of learning techniques has relied heavily on the secrecy of their classification techniques [45, 53] without providing any sort of security analysis. Historically, such obscurity can be overcome by determined attackers.

To avoid the pitfalls of obscurity, we've assumed the attacker has complete omnipotence with respect to our learning system; that is, potentially the attacker knows everything – the feature set, the training data, the learning technique, and even the learned function itself. Instead of limiting the attacker's knowledge, our approach has been to limit the attacker's potential actions. As long as such limitations are achievable and enforceable, this approach is far more practical for providing security guarantees. Dalvi *et. al.* similarly assumed that the attacker had knowledge of the classification technique and how it would behave. They initially assume the attacker doesn't realize the learner has been altered to be robust against attacks, then analyzed a repeated game of the attacker trying to fool the learner and the learner adapting to the new attack strategy.

Another aspect of attacks is the goals of the attacker. With respect to this facet, we have conceived criteria to characterize the attack in terms of a broad sense of possible attack goals. The following enumeration details the characteristics we considered[3]:

1. Does the attacker want to alter the learning process or find flaws in it?

   - **Alter the Learner (Active)**

     This category of attacks depends on the attacker being able to influence or modify the training set during initial training or subsequent retrainings. In such situations, the data distribution is at least partially malicious - a scenario that has been considered by several previous researchers. Kearns and Li [30] consider a situation where the adversary can maliciously control some percentage of the training data, E_{MAL}, and they construct a bound on this quantity based on the desired accuracy, ϵ of the learner. Meanwhile, Dalvi *et. al.* [12] consider a specific situation in which the attacker can alter data, but is constrained by the cost required to make the alterations.

   - **Find Flaws (Passive)**

     These attacks occur when a learner has been trained and the attacker wants to *probe* the learner to discover errors in the learned function that could be used to fool it. In this case it is also relevant to define what level of *feedback* is provided by the attackers queries. An example of such an attack is the spam attack by John Graham-Cumming which requires feedback as to whether or not a query spam was accepted by the target [23].

2. What class of points does the attacker want to misclassify?

   - **False Negatives**

     Attackers often want positive points,*e.g.*, spam, viruses or fraud, to be classified as negatives – false negatives. Thus the attacker wants parts of the portions of the feature space that are wrongly labeled as negatives. The attacker can either search for such points that exist in a learned function or actively attempt to cause the learner to make these mistakes.

   - **False Positives (Denial of Service)**

     Conversely, when the attacker wants normal or negative points to be tagged as positive, the attacker is causing false positives. Typically, this would be an active attack in which the attacker wants the learner to become confused to the point where normal usage can no longer function due to faulty classifications. If successful, such attacks can cause detection systems to be disregarded or discarded, thus removing the protection layer.

[3]The categorization discussed here is thought to layout the relative details of different attacks. This hierarchy was conceived by Marco Barreno, Russell Sears, and myself and is still under consideration as are the names of different classes of attacks.

- **Either/Both**

  Finally, there may be cases where the attacker is indifferent or wants the learner to make both types of mistakes. As with denial of service attacks, this category would make the learner useless for its intended purpose.

3. Does the attacker want particular points to be misclassified?

   - **Yes (Targeted Attack)**

     This case applies when the attacker has a particular goal point to be misclassified. More specifically, the goal point cannot be altered. Hence, the attacker must attempt to mislead the learner by inserting points that will cause the goal point to be mislabeled as desired. An example of such an attack would be attempting to get a specific viral message through a filter by sending out other decoys meant to confuse the learner.

   - **No**

     The attacker has a more flexible goal that allows for an entire class of points, any of which the attacker wants to be misclassified. Even in the passive case, the attacker may have a non-trivial task of determining which points are misclassified; a process that can be significantly hindered by only providing limited feedback. In the active version, we refer to this as a *numbing* attack. A common example of non-specific attacks are denial of service attacks in which the attack is only concerned with causing massive numbers of misclassifications without regard for which data items are misclassified.

Having laid out a framework for qualifying attacks, we now describe how to analyze attacks. Analyzing an attack typically attempts to quantify the effectiveness of the attack and the degree of difficulty required to achieve success. The attacker may have a variety of goals ranging from denying service to legitimate users to infiltrating a secured system, so the success of an attack must be defined with respect to a particular scenario. In terms of difficulty, one may choose to use a variety of metrics such as the amount of computational complexity, the required resources, or perhaps the degree of knowledge that must be known about the learner.

The remainder of this chapter focuses from this general setting to attacks on novelty detection. As discussed earlier in Section 4.3 we use a form of novelty detection in the pipeline of virus detection. In this chapter, we consider a far simpler approach to novelty detection involving hyperspheres that bound the normal data. These simplifications allow for an in-depth analysis of a particular attack scenario and provide a starting point for future analyses of this nature.

5.2 Attacking a Naïve Novelty Detection Algorithm

As presented throughout the earlier chapters, the overall goal of this project is viral infection detection. To this end, we have presented a pipeline for accurate viral detection utilizing a novelty detection scheme for reducing false positives and classification to accurately identify viruses from other suspicious email traffic. However, this scheme does not consider the potential for an attack, while an attacker could potentially attack any stage of our classification pipeline. We now focus on the threat of an attack at the first stage, novelty detection.

As mentioned in the previous section, in analyzing an attack, it is essential to lay out the capabilities and goals of a potential attacker. In general, a characteristic of modern mass-mailing viruses is their need to spread quickly to exploit the 'window of vulnerability' in virus scanners [46]. Secondly, a mass-mailing infection delivers the malicious code embedded in the email. Finally, the message itself is composed of some degree of social engineering designed to trick the recipient into opening the message and initiating the infection. These are the constraints we have assumed about the virus[4].

As far as resources, we must assume that the virus knows its adversary, our novelty detection system; thus, the remainder of this work proceeds assuming a worse-case scenario. Clearly, the virus can potentially do the same sort of behavioral analysis we have done of the infected user as long as the virus has access to the host's 'Sent-Mail' folder. The following analysis gives the virus the benefit of the doubt and assumes the virus knows exactly what the learner has learned and the methodology by which it adapts. Finally, we must assume the virus can potentially control all future email traffic from the infected host since the host is compromised. Moreover, since viruses can compose messages orders of magnitude faster than humans, it is safe to assume that the virus can dominate email traffic on the entire network. We now proceed to the goals of the virus.

In the virus domain, an attacker (possibly the virus itself or a separate third party) has a general goal of defeating our virus detection scheme. This could be done in a number of ways. First, the attacker could simply search for flaws in the system by means of a passive attack. Unfortunately, any statistical learning system can only learn from empirical data so its decision function is prone to flaws. However, the discussion of passive attacks is put off for future work.

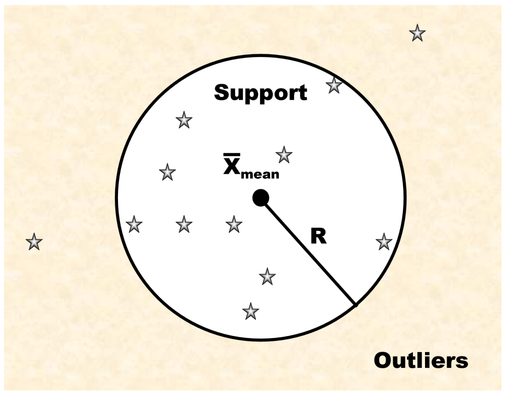
[4]To be fair, these constraints are limited as one can imagine future viruses evolving to avoid such characteristic behaviors. However, these constraints are not central to our analysis technique.

A more interesting possibility would be active attacks. Ideally, in an eventual deployed detection system, the system would have to adapt to changing behaviors over time. Thus, an attack could potentially be mounted against the retraining process by introducing points that would mislead the learner and cause it to adapt in a manner inconsistent with the actual behavior of normal and viral email. One such attack would be a denial of service attack in which the attacker would try to cause an untenable number of false positives by causing the learner to mislabel normal email as viral. Such an attack could cripple normal email system usage and thus cause administrators to disable the viral detection opening the door for an actual viral attack.

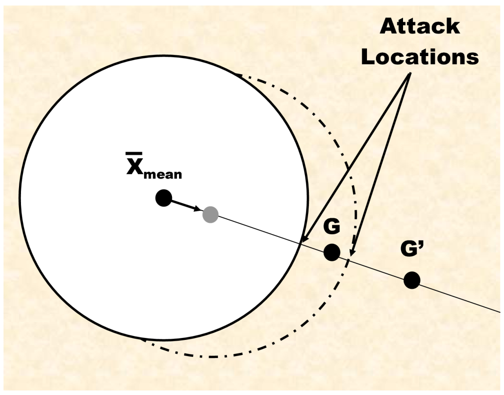
The other interesting form of an active attack would attempt to subvert the learning algorithm causing viral emails to be flagged as normal – the focus of the remainder of this study. As mentioned in Chapter 4.1, our classification pipeline begins with a novelty detection meant to identify emails that are deemed 'certainly normal'. If the attacker could mislead this stage of the pipeline into mislabeling viruses as 'certainly normal', the attack has succeeded since no further inspection of such messages takes place. The attack would work by inserting specially tailored messages into the detector's training set so that the retraining of the novelty detector would eventually mislabel viral email. As mentioned in Section 4.3 there currently is no retraining of the SVM for reasons discussed in Section 5.6.1. However, as a deployable system should incorporate retraining, we discuss the implications of it here.

As discussed in Chapter 4.3, the novelty detection algorithm deployed in our virus detection system is a one-class SVM [61]. However, this chapter focuses on a simpler scheme for novelty detection. The naïve approach depicted in Figure 11(a) uses a hypersphere whose center is the mean of the data and with a fixed radius specific to the problem – an approach that can be kernelized. While better schemes for hypersphere-based outlier detection exist, this scheme is simple to analyze and thus is the focus of this project. A goal of this project is to use the resulting analysis as a basis for more complex analyses in future work.

Currently, we focus on situations where the attacker is restricted. These restrictions are based on the idea of only retraining on points that exhibit pseudo-normal behavior. In the next section, we begin to discuss such an attack applied to novelty detection.



(a) Hypersphere Outlier Detection



(b) Attack on a Hypersphere Outlier Detector

Figure 11: Depictions of the concept of hypersphere outlier detection and the vulnerability of naïve approaches. In Figure 11(a) a bounding hypersphere centered at $\bar{X}_{mean}$ of fixed radius R is used to encapsulate the empirical support of a distribution excluding outliers. Samples from the "normal" distribution being modeled are indicated by $\star$'s with 2 outliers on the exterior of the hypersphere. Meanwhile, Figure 11(b) depicts how an attacker with knowledge about the state of the outlier detector can shift the outlier detector toward the first goal G. However, it could take several iterations of attacks to shift the hypersphere further to encompass the second goal G'.

5.3 A Directed Attack

As mentioned in the Section 4.3.1 outlier detection can be done by bounding data in a hypersphere. One naïve implementation uses a mean-centered hypersphere of fixed radius R to bound the support of the underlying distribution as depicted in Figure 11(a). Thus, one can imagine several situations in which a malicious user wants to mislead the outlier detection algorithm. If the outlier detector is only retraining on points falling within this hypersphere, this attacker could attempt to displace the bounding hypersphere by inserting points near the boundary. Moreover, if the attacker had a certain goal point G outside the hypersphere, the attacker could judiciously place these "attack points" along the line between the mean of the hypersphere and G in order to maximize the effect the "attack points" had in displacing the mean of the hypersphere in the desired direction. This situation is depicted in Figure 11(b). The following is a list of the fundamental assumptions made in this analysis as discussed throughout the remainder of this section:

1. **Novelty Detection Scheme**
   (a) The novelty detector is a naïve mean-centered hypersphere of fixed radius (possibly in a kernel-space).
   (b) The novelty detector uses a *bootstrapping* retraining policy – only add points classified as non-novel to the training set.
   (c) Points in the training set are never removed – no *aging* of data.
2. **Attacker Capabilities**
   (a) The attacker is omnipotent – he/she knows:.
      - The state of the novelty detector.
      - The policies of the novelty detector.
      - How the novelty detector will change on retraining.

   (b) The attacker controls *all* data inserted once the attack commences.
3. **Attacker Goals**
   (a) The attacker wants the novelty detector to classify a point G as normal with possible limitations on:
      M The total number of points the attacker can use in the attack.
      T The total time (number of retraining iterations) of the attack.

Under these simple assumptions, the attacker's objective is simple. Clearly the optimal strategy involves placing "attack points" at the boundary in the desired direction. Thus the problem is in fact uni-dimensional since it is assumed that the attacker has exact knowledge of the desired direction. The only complexity in the attack is choosing the number of points to place at each moment of the attack.

The naïve scheme for novelty detection uses a mean-centered hypersphere of fixed radius R to create a boundary around the normal data. The radius R is chosen to tightly bound the training data while having a low probability of false positives. Choosing the radius to meet these constraints is discussed in Chapter 5 of Shawe-Taylor and Cristianini [69], but for this work we assume the radius has been chosen *a priori*. Calculating the mean of a training set $\{\mathbf{x_i}\}_{i=1}^{n}$ in a kernel space under the mapping ϕ is given by:

$$\phi_C = \frac{1}{n}\sum_{i=1}^{n}\phi\left(\mathbf{x_i}\right)$$

To measure the distance between the center in kernel space and a query point $\mathbf{x}$ in feature space we use the following:

$$\|\phi_C - \phi(\mathbf{x})\|_2^2 = \kappa(\mathbf{x}, \mathbf{x}) + \frac{1}{n^2} \sum_{i,j=1}^{n} \kappa(\mathbf{x_i}, \mathbf{x_j}) - \frac{2}{n} \sum_{i=1}^{n} \kappa(\mathbf{x_i}, \mathbf{x})$$

where, as introduced in Section 4.3.1, κ is the kernel corresponding to the mapping ϕ: $\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$. Thus, this scheme classifies a query point $\mathbf{x}$ as an outlier if $\|\phi_C - \phi(\mathbf{x})\|_2^2 > R^2$.

We assume that the outlier detector is a mean-centered hypersphere of fixed radius R that is incrementally retrained *only on points previously classified as normal* over the course of T iterations. Once classified as "normal", points are always considered in the retraining while any point classified as "outlier" is immediately discarded and no longer used in retraining; a naïve strategy that will be called *bootstrapping* (As mentioned in the introduction, this strategy has some undesirable consequences, but is simple to analyze). As a result of retraining, the hypersphere has a sequence of means $\{\bar{X}_t\}_{t=0}^{T}$ that describe new mean of the hypersphere after the t-th retraining starting from an initial state, $\bar{X}_0$.

As for the attacker, the attacker inserts α_t points in between the $t-1$-th and t-th retraining iteration. Moreover, it is assumed in this naïve model that the attacker is the only actor generating points. Thus, the naïve attacks described above can be completely described by the sequence $A = \{\alpha_t\}_{t=1}^{T}$ where T is the duration of the attack. Thus, the *progress* of an attack at time t can be described by the current state of mean $\bar{X}_t$ due to placing a sequence of $\alpha_1, \ldots, \alpha_t$ attacks at the hypersphere's boundary. This yields the following recursion for the shift of the mean at time t:

$$\begin{aligned} \bar{X}_t &= \frac{\left(\sum_{i=1}^{t-1} \alpha_i\right) \cdot \bar{X}_{t-1} + \left(\bar{X}_{t-1} + R\right) \cdot \alpha_t}{\sum_{i=1}^{t} \alpha_i} \\ &= \bar{X}_{t-1} + R \cdot \frac{\alpha_t}{\sum_{i=1}^{t} \alpha_i} \end{aligned} \tag{5.1}$$

Fortunately, this recursion is trivially unwrapped. Moreover, by reorganizing terms we get the following:

$$\frac{\bar{X}_T - \bar{X}_0}{R} = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{\ell=1}^{t} \alpha_\ell} \tag{5.2}$$

This formulation concisely represents how far the mean is shifted relative to its radius R during the attack.

Remark 5.3.1 *A deeper insight into the nature of this problem is revealed by Eq.* (5.2)*. In particular, it shows that the change in the mean after T iterations of the attack relative to the radius of hypersphere R is a sum of "cumulatively penalized gains". That is the contribution of the t-th iteration of the attack is weighed down by the sum of the "weight" used in iterations up-to and including the current iteration.*

5.3.1 Formal Description of the Attack

Having described the problem and the assumptions made under it, a formal analysis can be conducted in order to reveal optimal attack strategies. This formal analysis begins with a formalization of the problem space and of the objective.

In order to avoid confusion, we introduce the following notation. We will denote the whole numbers as $\mathbb{Z}^{0+} = \{0, 1, 2, \ldots\}$, the natural numbers as $\mathbb{Z}^{+} = \{1, 2, \ldots\}$, the non-negative reals as $\mathbb{R}^{0+} = \{x \in \Re | x \geq 0\}$, and the positive reals as $\mathbb{R}^{+} = \{x \in \Re | x > 0\}$. Furthermore, we will often refer to the number of "attack points" used at the t-th re-training iteration by α_t and the "optimal" number of "attack points" to be used at the t-th iteration (optimal under prescribed conditions) as α_t^*. Unless specifically mentioned otherwise, $\alpha_t, \alpha_t^* \in \mathbb{Z}^{0+}$ although later in the text I'll be considering sequences in the non-negative reals, which I'll differentiate by $x_t, x_t^* \in \mathbb{R}^{0+}$, respectively.

Along with α_t and x_t, we also define the space of possible attack sequences. Formally, we define the space $\mathfrak{A}$ as any legitimate sequence of attack points; that is, $\mathfrak{A} = \left\{\alpha_t \in \mathbb{Z}^{0+}\right\}_{t=1}^{\infty}$ (where trailing terms after the attack would be 0). Similarly, several more constrained attack spaces are defined; $\mathfrak{A}^T = \left\{\alpha_t \in \mathbb{Z}^{0+}\right\}_{t=1}^{T}$, $\mathfrak{A}(M) = \left\{\alpha_t \in \mathbb{Z}^{0+}\right\}_{t=1}^{\infty}$ such that $\sum_{\ell=1}^{T} \alpha_t = M$, and $\mathfrak{A}^T(M) = \left\{\alpha_t \in \mathbb{Z}^{0+}\right\}_{t=1}^{T}$ such that $\sum_{\ell=1}^{T} \alpha_t = M$. Finally, the analogous continuous versions of these spaces are denoted χ, χ^T, $\chi(M)$, and $\chi^T(M)$ respectively with corresponding definitions to $\mathfrak{A}$ replacing $\alpha_t \in \mathbb{Z}^{0+}$ with $x_t \in \mathbb{R}^{0+}$.

Now that the attack strategy has been isolated, one can formalize the notion of optimal strategies by examining the objective of the attacker. In particular, since the attacker wishes to maximize the displacement of the hypersphere, the objective is defined in terms of Eq. (5.2). Thus, the objective function is defined with respect to an attack sequence $A \in \mathfrak{A}$,

$$\mathrm{D}(A) = \sum_{t=1}^{\infty} \frac{\alpha_t}{\sum_{\ell=1}^{t} \alpha_\ell} \tag{5.3}$$

In this objective function, we also develop a notion of the "individual contribution" of an iteration of the attack as $\delta\left(\alpha_t\right) = \frac{\alpha_t}{\sum_{\ell=1}^{t}\alpha_\ell}$.

The goal of the attacker is to maximize this objective function with respect to constraints on the size and duration of the attack.

Definition 5.3.2 Optimality *An attack sequence* $A^* \in \mathfrak{A}(M)$ *is an optimal strategy that uses a total of* M *attack points if* $\forall A \in \mathfrak{A}(M) \quad \mathrm{D}\left(A\right) \leq \mathrm{D}\left(A^*\right)$*. The optimal distance achieved by such a sequence is denoted by* $\mathrm{D}^*\left(M\right)$*. This optimality can be achieved by the attacker by solving the following program for* $A^* = \{\alpha_t^*\}$*:*

$$\begin{aligned} &\max \mathrm{D}\left(A^*\right) = \sum_{t=1}^{\infty} \frac{\alpha_t^*}{\sum_{\ell=1}^{t}\alpha_\ell^*} \\ &s.t. \sum_{t}^{\infty} \alpha_t^* = M \\ &\alpha_t^* \in \mathbb{Z}^{0+} \end{aligned} \tag{5.4}$$

5.3.2 Characteristics of the Optimal Solution

In order to properly understand the problem at hand, it is important to characterize its properties and the properties of optimal solutions to it. These observations provide the foundation for the further analysis of the problem. To begin with, in this attack formulation, there are no initial points, so no matter how many points the attacker places in the first iteration, the same displacement is achieved.

Lemma 5.3.3 *The optimal initial attack iteration is given by* $\alpha_1^* = 1$.

Proof The contribution of the first attack iteration is given by $\delta\left(\alpha_t\right) = \frac{\alpha_1}{\sum_{\ell=1}^{1}\alpha_\ell} = \frac{\alpha_1}{\alpha_1} = 1$. But since $\delta\left(\alpha_t\right) = 1$ for $\alpha_1 \in \mathbb{Z}^+$ (0 is excluded since it results in a division by zero), it is optimal to use as few points as possible so $\alpha_1^* = 1$.

QED

Next, it seems fairly obvious that the location of zero elements ($\alpha_t = 0$) is irrelevant to the sequences distance and therefore can be essentially discarded. This notion is captured by the following theorem:

Proposition 5.3.4 *Any sequence of attack points $A \in \mathfrak{A}^T$ can be rewritten as another sequence $A' = \{\alpha'_t\}_{t=1}^T$ such that $\exists \tau \in \mathbb{Z}^{0+}$ such that $\forall t \leq \tau \quad \alpha'_t > 0$ and $\forall t > \tau \quad \alpha'_t = 0$ and most importantly $\mathrm{D}(A) = \mathrm{D}(A')$. Thus, any (optimal) sequence can be rewritten as a sequence partitionable into a strictly positive subsequence followed by a subsequence of all zeros with identical distance to the original sequence.*

Proof The proof is intuitive. Consider the terms of the distance function:

$$\mathrm{D}(A) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{\ell=1}^{t} \alpha_\ell}$$

This is a sum of cumulatively penalized gains as discussed in Remark 5.3.1. Any zero-element always contributes 0 to this sum regardless of its position. Furthermore, since the denominator of each contribution is a sum of all previous elements and the current one, zero-elements before (or after) any non-zero element do not alter the denominator. Hence, only the ordering of non-zero elements determines DA; zero elements can be arbitrarily placed. Thus, there is a sequence A' in which all non-zero elements of A are before all zero elements of A' while maintaining the non-zero element's relative ordering. Moreover, this permutation of A has the same distance: $\mathrm{D}(A) = \mathrm{D}(A')$.

QED

In fact, it should be obvious that zero elements can be arbitrarily inserted into any optimal attack sequence to form a new optimal attack sequence with the same distance since zero elements neither add distance nor add "weight" to the subsequent denominators. Thus, Proposition 5.3.4 in effect suggests disregarding such elements since they do not contribute to the effectiveness of the attack. Moreover, pushing all zero elements to the end of the sequence corresponds to the notion that the attacker probably wants to minimize the time required for the attack. Finally, the fact that zero elements can be disregarded suggests the possibility of redefining $\alpha_t \in \mathbb{Z}^+$ rather than $\mathbb{Z}^{0+}$.

Having shown the irrelevance of zero elements, we now look at the properties of the salient non-zero elements. In particular, the notion of cumulatively penalized gains from Remark 5.3.1 is critical. There are two "forces" at work here. On the one hand, placing many points (large α_t) during iteration t improves the contribution of the term $\frac{\alpha_t}{\sum_{\ell=1} t\alpha_\ell}$ to the overall distance $\mathrm{D}(A)$. On the other hand, a large α_t will be detrimental to subsequent terms since it will increase the size of the denominator. This effect can be likened to having the mean of the points becoming heavier (harder to move) as more points are utilized. It seems intuitive that one doesn't want to place too much "weight" too quickly as it will cause the mean to become too "heavy" toward the end of the attack, thus making the latter efforts futile. Thus, it seems intuitive that any optimal attack sequence should be monotonically increasing. This intuition is captured in the following theorem:

Theorem 5.3.5 *Any optimal sequence of attack points, $A^* = \{\alpha_i^*\}_{i=1}^{T}$, must be monotonically increasing (though not necessarily strictly); that is, $\forall i \in \{1, \ldots, T-1\}$ $\alpha_i^* \leq \alpha_{i+1}^*$. Since the monotonicity is not strict, constant subsequences are feasible in an optimal sequence. Essentially, this result says that an optimal sequence must be sorted in ascending order.*

Proof See Appendix C.

Note that the fact that Theorem 5.3.5 does not require strict monotonicity makes it consistent with Proposition 5.3.4.

While it has been shown that any optimal attack sequence should be monotonically increasing in size, the intuition that the mean becomes "heavier" as more points are utilized seems to suggest more than just monotonicity. In fact, this notion will lead us to a unique optimal solution.

5.3.3 Unconstrained Optimal Attack: The Stacking Blocks Analogy

As it turns out, there is indeed an optimal integer strategy for the optimization in Definition 5.3.2. In fact, this optimal strategy can be derived by thinking of this problem as a center of mass problem.

Recall that in Remark 5.3.1 the distance achieved by the attack was likened to a sum of cumulatively penalized gains. We can think of this as a sequence of contributions attributable to each iteration of the attack; that is, the t-th iteration of the attack contributes $\delta(\alpha_t) = \frac{\alpha_t}{\sum_{\ell=1}^{t} \alpha_\ell}$ which is the "amount of weight" used at time t relative to the total weight used up to that time. This is analogous to a center of mass problem. In particular, if we think of the attack points α_t as units of mass that are placed at a distance of R from the current center of mass $\bar{X}_{t-1}$, $\delta(\alpha_t)$ is the amount by which the center of mass $\bar{X}_t$ is shifted relative to R. Thus, since viable attack points cannot be placed beyond distance R, this is analogous to placing a set of identical blocks below the current stack of blocks that was created at time $t-1$ such that the stack does not topple (the structure being stable corresponds to the constraint that viable attack points cannot beyond the radius R). Since the attack is constrained to only place points at the boundary, this analogy only holds when the stacking is done optimally or some of the blocks are vertically grouped (corresponding to placing several attack points in a single iteration, a notion that will be revisited in Section 5.4). Figure 12 depicts the correspondence between attacks on mean-centered hyperspheres and the stacking of blocks on the edge of a table.

Having likened the attack strategy to a high school physics demonstration, the optimal strategy emerges from latter's solution. As is shown by induction in [15], the blocks can be optimally stacked by extending the first by $\frac{1}{2}$ the second by $\frac{1}{4}$ and the t-th by $\frac{1}{2t}$. Moreover, as is mentioned in Figure 12 since the blocks are of length $2R$ we get the harmonic series exactly. Thus, the optimal integer strategy is given by placing a single point per iteration. This strategy achieves a displacement of $\mathrm{D}^* (M) = O (\ln M)$ as was predicted by the asymptotic analysis done in Appendix B.

5.4 Imposing Time Constraints on the Attack

In the previous section, optimality was achieved by simply placing a single point on the boundary at every iteration. However, while this strategy achieves maximal displacement of mean relative to the hypersphere's radius, it fails to capture the entire objective of the attacker. Namely, the goal of an attack is to achieve an objective (displace the mean a desired amount) but do so in minimal time T or with minimal effort (fewest possible points). As the analysis in the preceding section and Appendix B show, the prescribed attack achieves maximal distance of $\approx \log M$ but does so in time $T = M$ and thus, the attack's effect is only logarithmic in the time required to mount the attack. Thus, it seems advantageous to consider a more realistic version of the optimization of the attack sequence, in which the attacker achieves a goal with minimal effort.

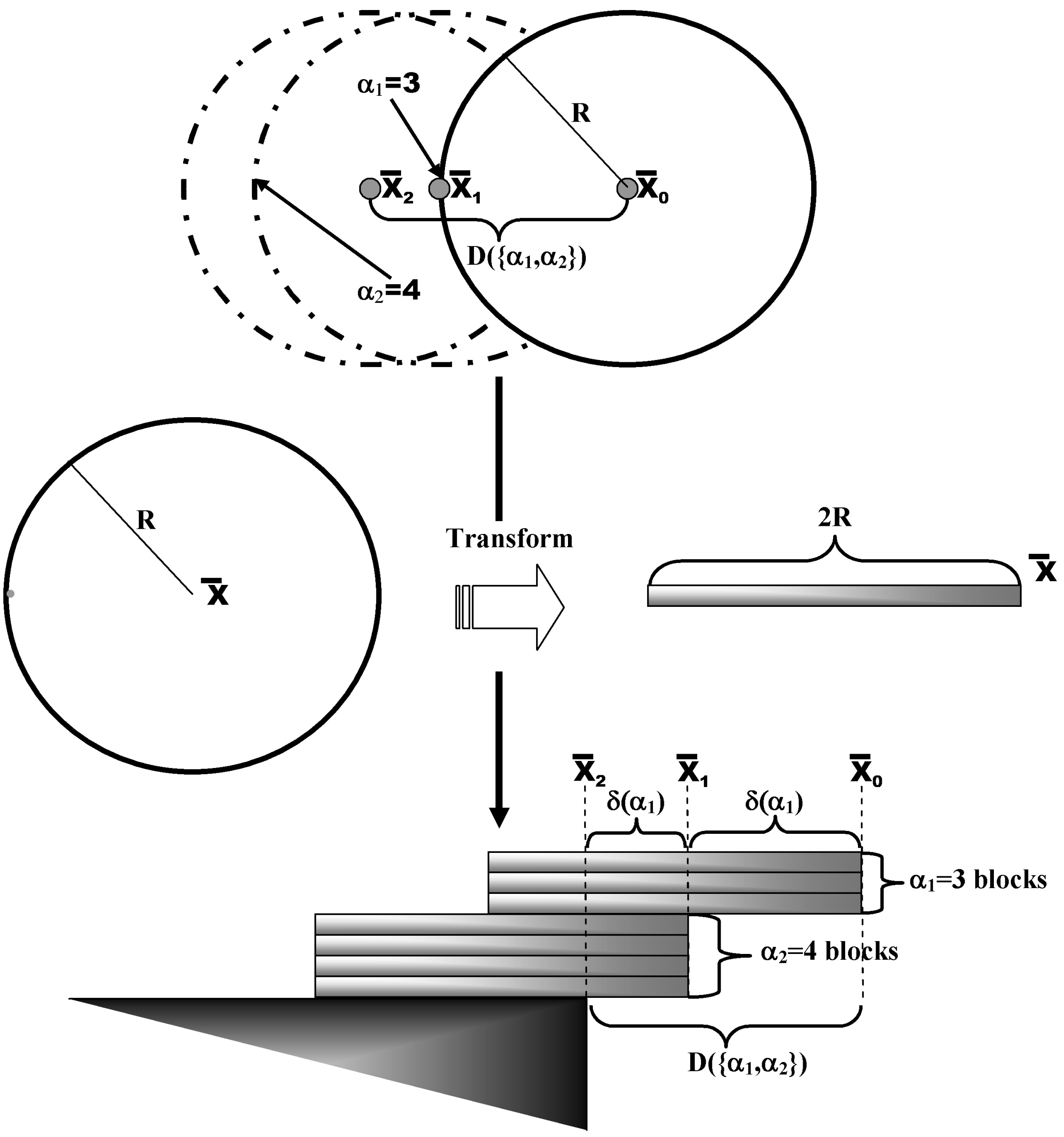


Figure 12: A figure depicting the physics analogy between the attack sequence $A = \{\alpha_1 = 3, \alpha_2 = 4\}$ and the concept of optimally stacking blocks on the edge of a table to extend beyond the edge of the table. From top to bottom, the original effect of the attack A on the naïve hypersphere outlier detector is transformed into the equivalent balancing problem. In this analogy, blocks of length $2R$ with a starting edge at $\bar{X}_t$ are equivalent to placing an attack point at the t retraining iteration of a hypersphere with mean $\bar{X}_t$ and radius R. This strange equivalence encapsulates the idea of a point of unit mass being placed at a distance R from the former mean. Vertical stacks can be interpreted as placing several points at time t and time (oddly enough) flows down the blocks to the table. Also depicted are the individual contributions $\delta(\alpha_1)$ and $\delta(\alpha_2)$ along with their overall effect $\mathrm{D}(\{\alpha_1, \alpha_2\})$.

Definition 5.4.1 Constrained Optimality *An attack sequence* $A^* \in \mathfrak{A}(M)^T$ *is considered optimal with respect to* M *total available attack points and a given duration* T *if* $\forall A \in \mathfrak{A}(M)^T \quad \mathrm{D}(A) \leq \mathrm{D}(A^*)$ *and the optimal distance achieved by such a sequence is denoted by* $\mathrm{D}^*_T(M)$*. This optimality can be achieved by the attacker by solving the following program for* $A^* = \{\alpha^*_t\}$*:*

$$\begin{aligned} \max \mathrm{D}(A^*) &= \sum_{t=1}^{T} \frac{\alpha^*_t}{\sum_{\ell=1}^{t} \alpha^*_\ell} \\ s.t. \sum_{t}^{T} \alpha^*_t &= M \\ \alpha^*_t &\in \mathbb{Z}^{0+} \end{aligned} \tag{5.5}$$

Note that an equivalent formulation of constrained optimality would take a desired distance d as an input and attempt to minimize the duration T required to achieve the desired distance with a total of M attack points. Similarly, another alternative would be to minimize M with respect to fixed d and T. However, Eq. (5.5) is a natural way to think about this optimization in terms of the attacker's goal.

As was shown in Section 5.3.3, the original problem is equivalent to the problem of optimally extending a stack of identical blocks over the edge of a table – a reduction to a solved problem. Not surprisingly, the time-limited version of the attack on the hypersphere is also analogous to a problem in physics. Naturally, it's a constrained version of the stacking blocks problem.

In this version, we have M points corresponding to M identical blocks. These points must be placed into T ordered bins. Since all points in a given bin are placed in the same (horizontal) location, this can be thought of as binding the blocks together into a vertical stack. Thus, the i-th vertical stack contains $\alpha_i \in \mathbb{Z}^{0+}$ blocks of unit weight and thus has a combined weight of α_i. Thus, we want to choose a grouping of M blocks such that the resulting T stacks can be optimally stacked beyond the edge of the table; this problem is shown in Figure 14(b). Clearly this problem is more difficult since we must not only optimally extend the stacks beyond the table's edge, but we must also choose the size of the stacks as to achieve the desired optimal.

It should be noted that in the limit when $M = T$, the constrained problem becomes the original problem since there are as many buckets as there are blocks. Moreover, following the argument in Section 5.3.3, if it were optimal to place more than one block in any of those buckets (and thus 0 in others) one could do better by simply nudging the vertically stacked blocks a bit away from the edge. However, this nudging action would correspond to moving a block to a different bucket. Thus, as expected, the optimal solution when $M = T$ is to place exactly 1 block in each stack. Furthermore, by this same argument the optimal number of buckets to maximize extension distance is $T = M$.

Despite being just a constrained form of the original physics balancing blocks analogy, having the integer buckets (stacks) makes this an integer program, for which no clear answer has emerged in the prior work or further analysis of the properties of the problem. One way to address this difficulty is to relax the constraint that the buckets contain an integer number of elements. This leads to a new formulation: we have T blocks of equal size but variable weight that we want to stack optimally. Moreover, we have M total mass in the system. This is the continuous variant of the problem.

5.4.1 Stacking Blocks of Variable Mass

By moving into the continuous realm, we have continuous sequences $X \in \chi$ where $X = \{x_1, x_2, \ldots\}$ with $x_t \in \mathbb{R}^{0+}$. Thus, for a given T and M we want to find X^* such that $\forall X \in \chi \quad \mathrm{D}(X^*) \geq \mathrm{D}(X)$. Many of the observations of the original problem carry over to the continuous realm. In particular, it is clear that the location of zero-elements is irrelevant as was shown in Proposition 5.3.4 and thus the zero-elements can be effectively discarded with respect to optimality. Moreover, the proof of Theorem 5.3.5 made no use of the fact that α_t were integers, only that $\alpha_t \geq 0$. Hence, the same line of reasoning can be applied to $x_t \in \mathbb{R}^{0+}$ and thus the optimal continuous solutions can be shown to be monotonically increasing.

Similarly, the optimization problems presented in Eq. (5.4) and (5.5) are perfectly valid shifted into the continuous context. Moreover, as with the integer variant, the variable mass blocks problem is a physics analogy for an attack sequence, in this case, where the attacker can place fractional points. However, while the original physics problem is a well know example of center of mass calculation with a well published solution, the variant of optimizing variable masses was not addressed in any of the texts examined by the authors. However, as was observed in simulations and as intuition suggests, it seems that exponentially increasing the mass with t is optimal. An argument for this intuition is made in the following subsection, however, an actual optimal solution will be also be discussed in Section 5.4.4.

While most results carry over from $\mathbb{Z}^{0+}$, it should be noted that Lemma 5.3.3 does not hold since, in real space, one can choose any $\epsilon > 0$ for $\alpha_1 = \epsilon$ and thus use fewer initial points but still achieve the optimal $\delta(\alpha_1) = 1$. This fact will provide for an interesting discussion in later sections.

5.4.2 An Intuition of Exponential Optimality

As pointed out by Brighten Godfrey, there is good reason to believe that an exponentially increasing sequence achieves optimality in the constrained optimization setting. This is due to the fact that exponential sequences make a constant contribution at each iteration thus achieving linear progress over time as is pointed out in Appendix B. Moreover, since there is M total mass to be distributed in an exponentially increasing fashion, this implies that $T = O(\log M)$. Thus, as with the optimal unconstrained solution, an exponentially increasing sequence is linear in $O(\log M)$. Moreover, an exponential sequence has the advantage that the number of points inserted at time t are a fixed multiple of all previous points. Hence, progress is constant regardless as there is a balance between not placing too many points too quickly and in making up for the previous points that are "weighting down" the mean.

Given the hypothesis that exponentially increasing attacks achieve optimality, let's analyze the strategy more closely. As shown in Table 5 the exponential strategy A_β^{exp} has an asymptotic behavior of $\mathrm{D}\left(A_\beta^{exp}\right) \sim \frac{\beta-1}{\beta} \cdot T$ in T iterations. Now, since we are trying to achieve optimality given M total mass in T iterations, we can solve for β:

$$\sum_{t=1}^{T} \beta^{t-1} = M$$

$$\frac{\beta^T - 1}{\beta - 1} = M$$

$$\beta^T - M \cdot \beta + M - 1 = 0 \tag{5.6}$$

Now by solving Eq. (5.6) for β will yield the desired value of β such that the sequence $\left\{\beta^{t-1}\right\}_{t=1}^{T}$ sums to M. Unfortunately, it's not obvious whether this T order polynomial has a form with simple solutions. However, in the case that $T = M$ the only solution is given by $\beta = 1$ which corresponds to the optimal unconstrained solution given in Section 5.3.3. In the case $T = 1$ the solution $\beta = 1$ corresponds to the fact that in the first iteration, it is optimal (in the integer scheme) to only place a single point since, in this attack formulation, all initial points achieve the same displacement as is demonstrated in Lemma 5.3.3. It is surprising that the integer constraint holds for this continuous strategy since the lemma does not hold in continuous space. Finally, in the case $T = 2$, $\beta = M - 1$ thus suggesting the sequence $\{1, M-1\}$ which is certainly optimal from Lemma 5.3.3. Thus, in these 3 cases, the exponential strategy yields the known optimal strategies!

Moreover, it should be noted that Eq. (5.6) always has 2 solutions for even T and 3 for odd T. Regardless of T's parity, one solution is always $\beta = 1$. Except in a few cases noted above, this is suboptimal, and occurs due to the division by $\beta - 1$ used in the above derivation. The other solution (positive one in odd case) is the optimal value of β. In the odd case, the 3rd solution is the (negative) pair of the optimal β. Interestingly, this negative solution yields an alternating series suggesting an unusual notion of inserting negative mass to smile about for a moment.

Having looked at the properties of exponential attack sequences, it is time to rethink the problem. It turns out that a slight paradigm shift will yield a convincing result.

5.4.3 An Alternate Formulation

In the continuous mass setting, little traction has been made in proving an optimal strategy A_T^* although the exponential sequence examined in Section 5.4.2 was promising. It turns out that it is difficult to solve for optimality in the program given in Eq. (5.5) even in the continuous case. In examining the Lagrangian [3, 54, 69], it turned out that $\frac{\partial}{\partial \alpha_\tau}\left[\mathrm{D}\left(\{\alpha_1, \alpha_2, \ldots, \alpha_T\}\right)\right] = \frac{1}{\lambda}$ for all τ where λ is the Lagrangian multiplier. However, this provided little insight since an initial examination strongly suggested that the objective function $\mathrm{D}(\cdot)$ was non-convex.

However, a slight shift in the way the problem is expressed yields significant insight that leads to conditions for optimality. This alternative formulation begins with a simplifying notation:

Definition 5.4.2 Total Mass *The total mass M_t is the sum of all mass used up to and including iteration t:*

$$M_t = \sum_{\ell=1}^{t} x_\ell$$

Originally introduced to simplify notation, the fact that $x_t = M_t - M_{t-1}$ allows us to rewrite the entire objective function in terms of M_t which results in,

$$\mathrm{D}_T\left(\{M_1, M_2, \ldots, M_T\}\right) = T - \sum_{t=2}^{T} \frac{M_{t-1}}{M_t} \tag{5.7}$$

However, unlike the original α_t-formulation, the total mass variables are interdependent. In particular $M_1 = 1 \leq M_2 \leq \ldots \leq M_T = M$ due to the fact that total mass is cumulative (hence the ordering constraints) and the final amount of mass used must be M. Moreover, we also reintroduced the constraint $M_1 = \alpha_1 = 1$ from the integer variant. Thus optimality can be achieved by the attacker by solving the following program for $M^* = \{M_t^*\}$:

$$\begin{aligned} &\max \mathrm{D}\left(M^*\right) = T - \sum_{t=2}^{T} \frac{M_{t-1}^*}{M_t^*} \\ &\text{s.t.} M_1^* = 1 \leq M_2^* \leq \ldots \leq M_T^* = M \\ &\quad M_t^* \in \mathbb{R}^+ \end{aligned} \tag{5.8}$$

5.4.4 The Optimal Relaxed Solution

Technically, reformulating the problem has not made the problem easier to solve. The total mass constraints still capture every aspect of the problem so this was not a relaxation of constraints. Nonetheless, the reformulated version of this problem captures the structure better, in the sense that it allows us to derive an optimal solution. First we see that in the program described in Eq. (5.8) the maximization is equivalent to minimizing $\sum_{t=2}^{T} \frac{M_{t-1}^*}{M_t^*}$. However, unlike the original objective function in Eq. (5.3) each of the variables M_t are involved in at most 2 terms of the summation. Thus, consider setting the derivative of the objective function with respect to M_t for $2 \leq t \leq T-1$:

$$\begin{gathered} \frac{\partial}{\partial M_\tau}\left[\sum_{t=2}^{T} \frac{M_{t-1}}{M_t}\right] = 0 \\ \frac{M_{\tau-1}}{M_\tau^2} = \frac{1}{M_{\tau+1}} \end{gathered}$$

These equations do not hold for M_1 or M_T since these two variables are only involved in a single term. However, we already have conditions for those two variables: $M_1 = 1$ and $M_T = M$. Thus, we have the following set of optimality conditions:

$$\begin{aligned} & M_1 &=& \ 1 & (5.9) \\ 1 < t < T \quad & M_t^2 &=& \ M_{t-1} \cdot M_{t+1} & (5.10) \\ & M_T &=& \ M & (5.11) \end{aligned}$$

This leads to the following lemma that generalizes for any starting value N[5]:

Lemma 5.4.3 *The general optimal strategy M^* given the constraints that $M_1^* = N$ and $M_T^* = M$ where $0 < N \leq M$ must satisfy those conditions as well as Eq.* (5.10)*.*

Proof By definition, $\forall t \quad M_t > 0$ since M_t is monotonically increasing (or constant) by definition and $M_1 = N > 0$. Thus, the optimal solution can be characterized as a positive-valued vector $M \in (0, \infty)^T$. On this domain, the objective function given in Eq. (5.7) is continuous, its first partial derivative $\frac{\mathrm{D}(M)}{M_\tau} = \frac{1}{M_{\tau+1}} - \frac{M_{\tau-1}}{M_\tau^2}$ is also continuous, and its second partial derivatives are also continuous (proofs of continuity are not shown here).

Now, since Eq. (5.10) is derived from setting the partial derivative with respect to M_t to zero, any sequence satisfying that condition along with the boundary conditions is a critical point of the objective function. Moreover, the global optima attained at M^* is either a critical point or on the boundary of the allowed space. However, in the total mass formulation, the boundary of allowed solutions is formed by the following constraints; $M_1^* = N$, $M_T^* = M$, and for all $0 < j < T$ we have $N \leq M_{j-1}^* \leq M_j^* \leq M$ since the sequence must be nondecreasing.

Thus, boundary conditions only occur when two or more consecutive elements in the total mass sequence are equal, or rather, in the original formulation, there is an element $x_j = 0$. By Proposition 5.3.4, such a boundary sequence is equivalent to a sequence of length $T - 1$. However, it can be shown that the function $\mathrm{D}_t^*(M^*)$ is increasing in t, and thus, no boundary point of the total mass formulation is an optimal sequence. Thus, the optimal sequence is a critical point described by Eq. (5.10).

QED

[5]The initial constraint is particularly important since, in the continuous domain, Lemma 5.3.3 does not hold

Now confirming our notion of the optimality of exponential strategies we have the following theorem:

Theorem 5.4.4 *The sequence of masses described by the total mass sequence* $M^* = \left\{ M_t^* = M^{\frac{t-1}{T-1}} \right\}_{t=1}^{T}$ *is the unique solution to the optimality equations in Eq.* (5.9), (5.10), *and* (5.11) *and thus is the global optima of the continuous version of the constrained optimization program in Eq.* (5.5). *Moreover, this total mass sequence results in an optimal center of mass displacement of* $\mathrm{D}_T^*(M) = T - (T-1) \cdot M^{\frac{-1}{T-1}}$. *Finally the actual optimal sequence of mass placements* $\{x_t^*\}$ *can be described by:*

$$x_t^* = \begin{cases} 1 & t = 1 \\ M^{\frac{t-2}{T-1}} \left(M^{\frac{1}{T-1}} - 1 \right) & t > 1 \end{cases} \tag{5.12}$$

Proof See Appendix D.

Also note that $\sum_{t=1}^{T} x_t = M$ as expected. Now recall that in Section 5.4.2 we hypothesized that $x_t = \beta^{t-1}$. Using the definition of $\beta = M^{\frac{1}{T-1}}$ used in the above proof, the attack strategy instead suggests placing $x_t = \beta^{t-2} \cdot (\beta - 1)$ which is close to the exponential strategy for large β. However, when $\beta = M^{\frac{1}{T-1}}$ is close to 1 (the case when $M \sim T$), the two strategies diverge. Just to be sure, we'd like to know which is better; a result hypothesized in the next proposition:

Proposition 5.4.5 *The exponential strategy* A^{exp} *of* $x_t^{exp} = \beta^{t-1}$ ($\beta > 1$) *is dominated by the optimal strategy* M^*.

Proof We assume that A^{exp} of $x_t^{exp} = \beta^{t-1}$ for $\beta > 1$ is optimal and thus, must satisfy the optimality conditions.

As stated by Lemma 5.4.3, an optimal sequence would have to meet the conditions stated in Eqs. (5.9), (5.10), and (5.11). First, we consider the sequence $\left\{\beta^{t-1}\right\}_{t=1}^{T}$, which clearly satisfies $x_1^{exp} = \beta^0 = 1$, the first condition. However, in order to satisfy Eq. (5.11), β must satisfy Eq. (5.6).

Now, given the A^{exp} strategy, the total mass variables of this strategy are given by $M_t^{exp} = \frac{\beta^t - 1}{\beta - 1}$. Using these total mass variables, we can examine the final optimality condition Eq. (5.10):

$$\begin{aligned}
(M_t^{exp})^2 &= M_{t-1}^{exp} \cdot M_{t+1}^{exp} \\
\frac{(\beta^t - 1)^2}{(\beta - 1)^2} &= \frac{(\beta^{t-1} - 1) \cdot (\beta^{t+1} - 1)}{(\beta - 1)^2} \\
\beta^{2t} - 2\beta^t + 1 &= \beta^{2t} - \beta^{t-1} - \beta^{t+1} + 1 \\
\beta^2 - 2\beta + 1 &= 0 \\
\\
\Rightarrow \qquad & \beta = 1
\end{aligned}$$

Thus, the A^{exp} strategy can only satisfy Eq. (5.10) if $\beta = 1$. However, this can only meet the mass requirement that $M_T^{exp} = M$ if $M = T$, which is not the case for the time-constrained scenario given in Definition 5.4.1. Thus, the hypothesis that A^{exp} meets the optimality conditions is a contradiction.

QED

Despite the fact that the original exponential strategies are empirically dominated by the optimal strategy M^*, exponential strategies seem to match well with integer solutions as was suggested in Section 5.4.2. The "optimal" strategy does not produce integer strategies at all except in the rare cases where $M = k^{T-1}$ for some $k \in \mathbb{Z}^{0+}$. Perhaps more disturbing, as $T \rightarrow M$, M^* is no longer monotonically increasing and thus by Theorem 5.3.5 is no longer optimal. This is due to the fact that M^* enforces a constraint that $M_1^* = 1$. However, in continuous mass space, it can clearly be beneficial to consider masses less than 1. Thus, M^* exploits this fact when there is little mass on average per block– that is when $T \rightarrow M$ – in order to optimally distribute the mass. Unfortunately in this case, the constraint $M_1^* = 1$ is artificial in the continuous case since Lemma 5.3.3 no longer holds.

In fact, in the continuous case the true constraint should be $M_1 > 0$ since any such initial mass would be valid. An initial thought is to solve the conditions of optimality in Eqs. (5.10) and (5.11) with the new initial condition, but such an analysis blows up as M_1 goes to 0. There is an intrinsic problem underlying this issue. The problem is that the initial hypersphere centered at $\bar{X}_0$ is assumed to have no initial mass. Thus, the first mass placed by the attacker on the boundary displaces the mean to $\bar{X}_1 = R$ regardless of the size of the first mass placed by the attacker. In the integer case, this assumption has little effect on the outcome due to Lemma 5.3.3, but in the continuous case, it allows for an unbounded progression of smaller masses. To counter this, we can adjust the model by adding an initial mass N as input to the problem. This leads to the "ϵ-formulation" in Definition 5.4.6.

Definition 5.4.6 The ϵ-Formulation *The ϵ-Formulation is a modification of the original attack scenario presented in Section 5.3 in which an initial mass of $x_0 = N$ or equivalently an initial total mass of $M_0 = N$ is introduced.*

As with the previous problems, the ϵ-Formulation is analogous to a block stacking of T blocks of M total mass with the modification that a (small) point mass N is placed on the outer edge of the top block where typically one would imagine $0 < N \ll M$. In fact, for the purposes of approximating the original problem we can use a small fixed $N = \epsilon > 0$ as depicted in Figure 14(d). This formulation yields the following optimal choices of total mass at iteration t:

$$M_t^* = N^{\left(\frac{T-t}{T}\right)} M^{\left(\frac{t}{T}\right)} \tag{5.13}$$

In addition the distance yielded by this formulation is,

$$\mathrm{D}_T^* \left(M, N\right) = T \left(1 - \left(\frac{N}{M}\right)^{1/T}\right) \tag{5.14}$$

where the derivations for Eqs. (5.13) and (5.14) were obtained in a manner similar to the proof of Theorem 5.4.4.

A quick observation shows that as $\epsilon \to 0$ the optimal distance achieved by Eq. (5.14) approaches $\mathrm{D}_T^* \left(M, N\right) = T$. This again shows the unrealism of the continuous approach since an optimal distance achieved in T iterations is obtained by using negligibly small (but exponentially increasing) masses for the first $T - 1$ iterations followed by using the nearly M remaining mass at the last iteration; a strategy that only bodes well in the continuous setting since its integer equivalent (either $\{0, 0, \ldots, M\}$ or $\{1, 1, \ldots, M - T + 1\}$) is suboptimal for non-trivial M, T.

In spite of being a poor approximation for integer solutions, the ϵ-formulation allows for a useful extension. By allowing N to be specified as an input parameter, we can model attack situations in which the initial state of the hypersphere was determined by N initial non-malicious points.

5.5 Implementing the Hypersphere Attack

Having examined the theoretical aspects of optimality in this problem domain, the following section details the practical implications of such attacks against the naïve novelty detection algorithm presented throughout this chapter. Until this point, we have thought of the novelty detector as an abstraction: a sphere in some feature or kernel space. Now we look at the practicality of the attack.

5.5.1 Practical Attack Strategy

To begin with, we never solved the integer version of the constrained optimization problem given in Definition 5.5. While we presented a solution to the relaxed (continuous) version of the problem in Section 5.4.4, this does not necessarily yield an integer solution. We have not been able to find an exact solution to the integer version of the constrained optimization given in Eq. (5.5). Since Eq. (5.5) is an integer program (NP-Hard), it is possible that this program is also NP-Complete, but no such proof has emerged yet. Nonetheless, the continuous solution provides a good deal of insight into the integer solution:

Remark 5.5.1 *Nonetheless, the solution to the continuous version of the program provide are an optimistic estimate of the distance achievable by the integer version. That is, the distance achieved by the optimal continuous strategy is an upper bound on the distance achievable by the optimal integer strategy.*

It should be noted that when the continuous solution does yield an integer solution, that solution is an optimal integer solution as well; a product of the fact that the continuous solution is an upper bound for the integer one. In fact, this does happen on the occasion when the following condition is met:

$$M^{\frac{1}{T-1}} \in \mathbb{Z} \tag{5.15}$$

Thus, it is feasible that the attacker could tailor their attack to meet this condition either by manipulating M or T.

However, as mentioned in Section 5.4.2, the exponential strategy given by $A_\beta^{exp} = \left\{\beta^{t-1}\right\}_{t=1}^{T}$ (where β satisfies Eq. (5.6)) seems close to optimal. Hence, it seems reasonable to use this as a guide for creating optimal strategies.

5.5.2 Empirical Study of Attack against Viral Detection

The following is an empirical study of the feasibility of the attack suggested in this chapter in the domain of virus detection. Using data gathered from the virus study discussed in Section 4.6 and the set of features laid out in Table 2, we gathered statistics for the mean-centered fixed-radius hypersphere. The study gathered the false positive and false negative rates of the classifier for a normalized polynomial kernel as discussed in Chapter 4. The degree of the polynomial was allowed to vary while the parameter R was chosen to be 1 based on the earlier results from Section 4.3.3. In addition the degree of the polynomial the radius of the hypersphere ρ was allowed to vary.

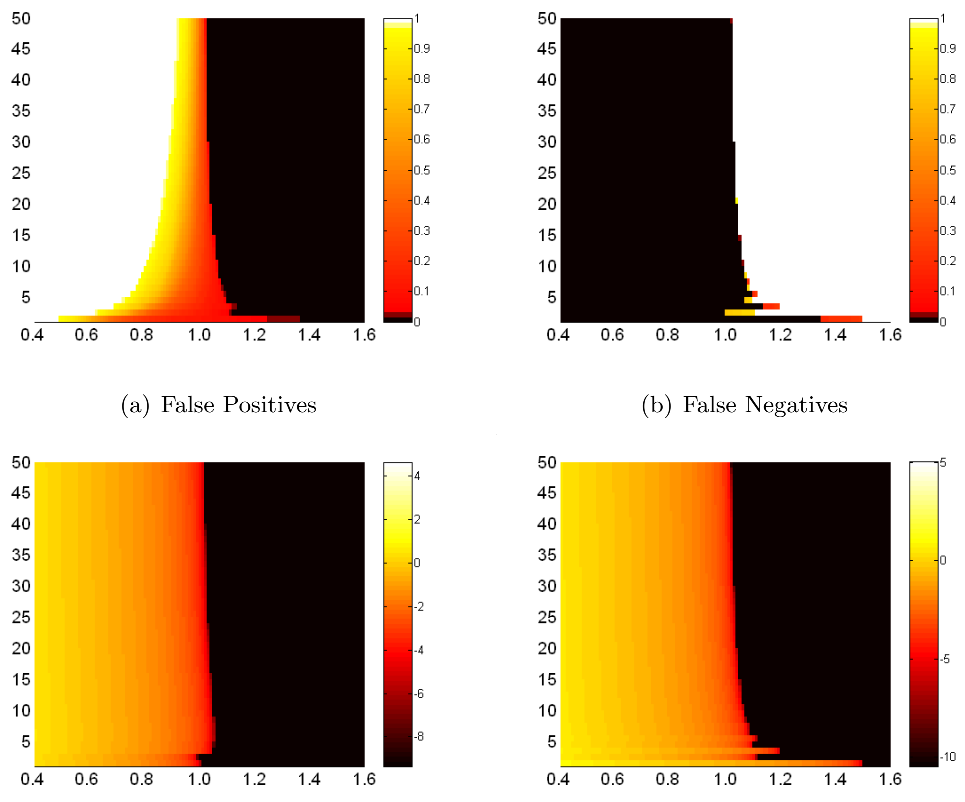
The study also collected the minimum and maximum relative distance[6] of any virus data point from the edge of the hypersphere. This gives us an idea of how much effort would be required by an attacker. If the attacker only wants to get *some* point misclassified, the minimum distance is most relevant. However, if the attacker wants *all* viruses to be misclassified, the maximal distance is the relevant metric.

The results of the study are presented in Figure 13. One thing that becomes evident in the plots is that, as the degree of the polynomial increases, all points seem to be falling within a progressively narrower band that appears to approach a distance of 1 from the mean of the hypersphere. This is consistent with the fact that in high dimensional spaces, the vast majority of the volume of a hypersphere is at its boundary. Due to this fact, it would be difficult to chose a radius ρ for the hypersphere that has a small number of false positives and is a significant distance from any viral data. Hence, in terms of security, this experiment shows that it is important to chose the kernel with the smallest kernel space still able to capture the desired classification – in this experiment a polynomial of degree greater than 3 but less than 10.

5.6 Countermeasures to Attacks on Statistical Learners

In this chapter, we have thoroughly explored how an attacker can exploit a naive hypersphere boundary. As a final thought, we discuss the inevitable future direction of this work: the countermeasures that can be employed to prevent or diminish attacks against learners. This aspect of our research is still in its preliminary phase, so this section focuses on the issues and techniques we have identified as salient directions for future exploration.

[6]relative to the radius ρ of the hypersphere



(a) False Positives

(b) False Negatives

(c) Log of Minimal Distance

(d) Log of Maximum Distance

Figure 13: An empirical study of the feasibility of the attack on naïve hyperspheres in the virus detection domain. In the above plots, the x-axis is the radius ρ of the hypersphere novelty detector and the y-axis is the degree of the polynomial kernel used in the test. Figures 13(a) and 13(b) display the false positive and false positive rates of the naïve classifier respectively. Meanwhile, Figures 13(c) and 13(d) show the logarithm of the minimal and maximal distance of viral points from the edge of the hypersphere relative to the hypersphere's radius ρ.

5.6.1 Bias of Bootstrapping

An obvious method to deter attacks on learning algorithms is to consider more sophisticated learning algorithms. Throughout this chapter, we have focused on a naïve outlier detection scheme using a mean-centered hypersphere of fixed radius. An obvious improvement would be to bound the training data in the smallest enclosing hypersphere (SEH), thus eliminating the mean-centered and fixed radius constraints (for more details see Chapter 7 of Shawe-Taylor and Cristianini [69]). In fact, for a certain class of kernels, the SEH is equivalent to the one-class SVM discussed in Section 4.3.1 and Appendix A.2.

The SEH is ideal for preventing the directed attack discussed throughout this chapter. The core idea of the directed attack was to place a large mass of points at one point on the boundary in order to shift the mean of the data thereby shifting the boundary in the desired direction. However, since the SEH is not mean-centered, but rather the best enclosing boundary, the original hypersphere boundary would also be optimal for the new attack points since they already are within the boundary. Thus, the directed attack would not shift the learned boundary toward the goal.

The astute reader should be disconcerted by an unsupervised technique that cannot be misled. The problem is our scheme for filtering new examples for retraining of the learner – our bootstrapping assumption. As indicated by the title of this section, the bootstrapping strategy is introducing a bias to the learner by artificially reinforcing its hypotheses. These biasing effects were present in the naïve hypersphere approach as well, but were masked by fixing its radius.

The bootstrapping assumption affects the learner even when it is not under attack as exemplified by the SEH detector. Consider a minimally enclosing hypersphere H_t that is 'ν-soft', which allows at most ν-percent outliers in the training set. At each time t, we retrain on the new dataset composed via bootstrapping resulting in a new hypersphere H_{t+1} that is ν-soft. Since bootstrapping only adds points that are within H_t, the percentage of data outside the hypersphere decreases with each retraining iteration. Thus, the new hypersphere H_{t+1} can only restrict the support from the previous iteration resulting in a perpetually shrinking bounded support.

5.6.2 Beyond Bootstrapping

To eliminate biasing effects, the bootstrapping policy must be replaced by a more realistic retraining policy. At one extreme, we can bootstrap the boundary and at the other we can accept all observed data for retraining. We seek to restrict the attacker without unduly damaging the function learned. To this end, we have been exploring statistical censoring and active learning, briefly explained below.

Censoring

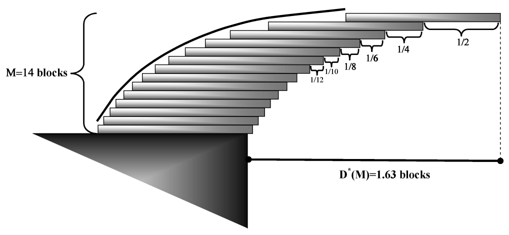
In statistics, censored data occurs in studies of limited duration and is addressed through survival analysis. More specifically, a study is conducted in which some data points exceeded the duration of the study; these are the censored data items. Thus, survival analysis seeks to exploit the censored data as well as the non-censored data in estimating statistics [11].

Censoring is being extended to the classification framework by Guillaume Obozinski as a mechanism to deal with adversarial data in classification tasks. In this case, the censoring mechanism is used to model partially labeled data (unlabeled data being censored). This framework is incorporated into a two-class SVM yielding a variation of that learner. Thus, censoring is a promising mechanism for estimating statistics when part of the data is missing. However, this approach does not suggest which data points should be labeled.

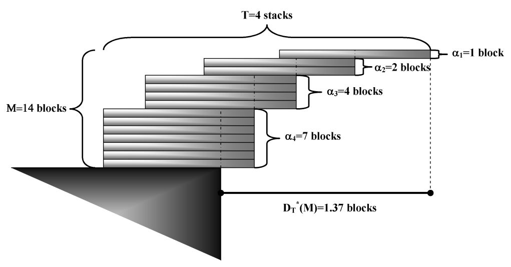
Active Learning

Active learning is the field of machine learning that addresses learners that select data to train on. As the learner builds its hypothesis, it attempts to select new training points that will provide the most information to the learner for improving the hypothesis. The goal of this selection procedure is to reduce the number of training points required to achieve a given generalization error with high probability. Work done in this area includes a number of theoretical results on these bounds [9, 13].

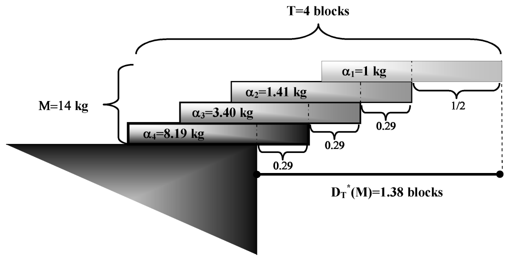
In our framework, such techniques and analysis would be useful in the context of trying to minimize the number of points that need to be labeled by the user of our system. As we have discussed, our system collects unlabeled email data, but is also vulnerable to attacks. One way we have considered to address this is by providing some amount of labeling in the training set, but each email that needs to be labeled is a burden on the user of our system. Thus, we'd like to find the smallest set of emails that can provide the most benefit from being labeled. Of particular interest to our work is the idea of pool-based learning [13] where a learner can select the data to have labeled from a pool of unlabeled data at a cost for each label made.



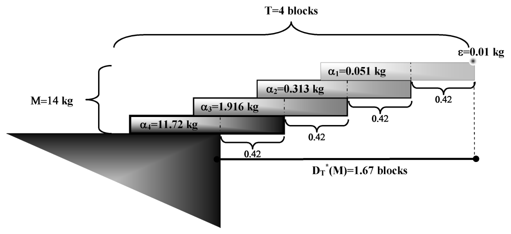
(a) Unconstrained Solution



(b) Constrained Integer Solution



(c) Continuous Solution



(d) Continuous ϵ-Solution

Figure 14: In the above diagrams are a series of solutions to variants of the block stacking problem with $T = 4$ and $M = 14$. In all the variants the objective is to maximize the stack's horizontal displacement from the edge of the table. Subfigure 14(a) depicts the globally optimal solution when no constraint is placed on the number of "vertical" stacks allowable. The displacement of each successive block follows the harmonic series thus giving a cumulative displacement following the logarithmic-like curve depicted. In subfigure 14(b), the constrained integer version of the stacking problem attempts to maximize displacement with a limit of $T = 4$ vertical stacks. This yields the depicted optimal solution. By relaxing the integer constraint, subfigure 14(c) depicts the situation where each "vertical" stack is condensed into a single block. Thus, the optimal solution partitions the overall mass into $T = 4$ chunks that maximize displacement with a constraint that the top block has mass 1kg. Finally subfigure 14(d) depicts the continuous situation where the top block is no longer constrained. Instead, it is assumed that there is a negligible initial mass $\epsilon = 0.01$kg placed at the edge of the top stack.

Game-Theoretic Approaches

Due to the fact that we are considering adversarial data, a well-tested approach is the subject of game-theory - a formalization of accessing the strategies of players in a variety of (competitive) game environments. This theory has been successful applied to machine learning in considering adversarial classification tasks [7, 12]. In general, we feel that a game-theory approach or analysis will need to be part of any countermeasures we propose in an adversarial domain, although our work to date has not incorporated it.

Chapter 6

Conclusion and Future Work

In this thesis, we demonstrated the many considerations that went into the design of a novel virus detection system as a fast-reaction system to address the threat posed by modern mass-mailing worms. In particular, we discussed in detail our feature selection process and the design decisions that lead to our multi-tiered classification pipeline of Section 4.1. This system was designed to adapt to changing email behavior to effectively detect novel worms with minimal inconvenience to the users. We also began to consider the implications of using statistical learning in security-sensitive environments by building a model for a particular adversarial attack strategy that could be deployed against our novelty detector.

The primary motivation of our system design was based on the idea of detecting infected machines by identifying deviations from normal user behavior. The first step in this process was choosing email features thought to capture viral behavior. After a process feature selection, the list of candidate features was winnowed down to those most salient to the virus detection task. As was shown in Section 3.3.4, judicious feature selection was effective at improving the performance of the novelty detection and Naïve Bayes classifier used in our system. These improvements are attributable to the fact that irrelevant features only provide extraneous information generally leading to the overfitting of the resulting classifier.

Our current work focused on statically selecting from a set of simple statistics calculated from email. Future approaches could potentially incorporate richer sets of features. Ideas such as social networks or bag-of-words could be incorporated into future models to develop a finer grained model of normal user behavior that could increase the accuracy of our system. Ideally, these features could be integrated into a combined kernel for use in the one-class SVM, thus, building a more sophisticated email kernel.

The second step was to choose a classification mechanism; we chose a multi-tiered approach in an effort to produce effective novelty detection with few false-positives. Thus, as discussed in Section 3.3.4, we motivated using a novelty detector for filtering (with high probability) normal traffic in conjunction with a Naïve Bayes classifier to produce accurate identifications. As empirically verified in Table 4, this pipeline produces significant improvements over the application of these techniques in isolation. Overall, it quickly identifies new viral threats with minimal impact on normal user activity.

While we covered classification extensively, several aspects of the overall system have been largely neglected in this thesis but covered in more detail in the theses of Steve Martin and Anil Sewani [41, 68]. The details of the implementation of our feature gathering architecture and overall system architecture are discussed in Steve Martin's thesis [41]. Steve also goes on to do a detailed benchmark of the performance of our system under various work loads in order to motivate the feasibility of wide-spread deployment. Meanwhile, the thesis of Anil Sewani [68] does an in-depth analysis of the effect our system would have on the spread of mass-mailing worms using common propagation models.

There are several future directions we feel could provide improvements to the virus detection scheme we developed. To begin with, email streams have well-defined temporal behavior; exploiting this transitory nature could prove to be more robust to error as discussed in Section 4.4.4. Another deficiency was our poor modeling of email distributions. As discussed in Jordan [29], the success of generative models depends largely on how well the data fits to the model; in our system, the data models were fit in an ad-hoc fashion as discussed in Section 4.4.2. The poor models were due, in part, to a lack of sufficient viral training data (virus emails tend to have little variance for a single virus but vary substantially between viruses). Generally, the ability to produce samples from a wide variety of viruses could be instrumental to future successes. Finally, our two-layer system was not ideal; in the future, we would like to develop a single novelty detector capable of training on the kind of semi-supervised data that is available.

Besides the potential shortcomings of our models, the system design itself has several drawbacks. First, since it is focused on outgoing email, it requires network-level deployment to be effective at limiting worm propagation. Users must still rely on established anti-virus methods to prevent external attacks from producing infections. Second, the model is built on a strictly per-user basis with no sharing of information among multiple users. We believe that for our technique to be successfully applied to a diverse population, models should share information, both to improve initial accuracy and to better identify abnormal activity.

Moreover, despite promising success on the analysis of past viruses, our current system was not designed to account for the adversarial nature of the viral domain. Our system was based largely upon the behavior of current viruses but these characteristics could change radically as viruses evolve. First, we can expect to see more sophisticated and personalized email content as the general public becomes accustomed to ignoring simple viral messages. Secondly, if the goals of the virus authors change, the behavior of the future viruses could be made radically different. Modern viruses have been primarily designed to vandalize global resources or to bring prestige to their authors; hence, we have seen fast spreading pandemics. However, it is conceivable that future virus designers could be motivated to create a “stealth worm”; *e.g.*, a virus designed for monetary gains could be clandestine. Finally, as virus detection mechanisms are deployed, virus authors will inevitably adapt their designs to avoid detection. These conjectures motivate the necessity for analyzing the success of statistical learners in adversarial environments.

The secondary focus of this thesis was based on anticipating such adversarial reaction to potentially widely deployed learning systems intended to thwart their activities. The threat of unforeseen future viral behavior motivates the need to more fully address adversarial scenarios in statistical learning. In this thesis, we developed a preliminary model of an adversarial attack against novelty detection as a starting point for further research - introduced in Chapter 5. A thorough analysis of the system presents the behavior of an attacker determined to fool a naïve novelty detection system through its retraining mechanism. This leads to our main result of revealing the optimal behavior of such an attacker.

While the results of Chapter 5 are promising, the analysis needs to be extended to more general settings. The simple model considered needs to be augmented to encompass entire classes of learning techniques and scenarios more complex than the analyzed directed attack. It remains to be seen how the categorization presented in Section 5.1 will translate into interesting attacks, and how such attacks can be realized in specific learning environments. Finally, the overall goal of developing attacker models is developing algorithms that are robust against attacks. We are currently exploring the possibility of developing these algorithms using the techniques discussed in Section 5.6.2.

Acknowledgments

We would like to thank Dr. Peter Bartlett, Dr. Michael Jordan, Dr. Christos Papadimitriou, Anil Sewani, Steve Martin, Marco Barreno, Guillaume Obozinski, Russell Sears and Brighten Godfrey for their instrumental work, discussions, and input that made this research possible. Blaine Nelson was supported in part by grants from Hewlett Packard, Sun Microsystems, the State of California under the MICRO program, and jointly by the US Department of Homeland Security Advanced Research Projects Agency (HSARPA) and the National Science Foundation (NSF) under grant ANI-0335241.

Appendix A

Implementation of Learning Models

In order to accommodate a variety of potential models and algorithms, we constructed an inheritance hierarchy of learning models in Java. While Java is not the most ideal language for many learning models' matrix-centric operations, the availability of the numerics package COLT [28] along with the implementors' experience in Java made it the language of choice. While initially made to accommodate several candidate learning algorithms with a minimal amount of rewritten code, the hierarchy also facilitated the implementation of the multi-tiered approach to virus detection by providing a single interface by which to interact with the various models.

Another core concept underlying this package of learning models was the separation of the learning model from the training algorithm. In many learning packages such as WEKA [84, 87] learning is focused around the training algorithms used to learn model parameters. However, our learning package separated the model from the algorithms used to train them in order to add flexibility and minimize the need for rewriting code. For instance, in designing the neural network implementation, there was the classic back-propagation training algorithm [58] but there was also a lesser-known algorithm called Alopex [82]. Similarly, the EM-algorithm could be potentially used for training a variety of different models. Thus, models inherited from the class *LearningModel* that encapsulated the concept of applying "learned" parameters to make a prediction for a data point. Meanwhile, the algorithms inherited from *TrainingModule*, which takes a model and a training data set as input and trains the model's parameters to implement a prediction function based on the training instances.

Also, the authors wanted to provide a learning package that was not tied to particular data types or a particular kind of learning. While most learning algorithms operate exclusively on numeric data, some algorithms allow for a more diverse range of data types. For instance, the kernel paradigm [69] implements a variety of pattern algorithms via a "kernel function". This approach allows the kernel to be constructed to compare a particular type of data independent of the pattern analysis algorithm. Initially, different data types were accommodated by using the all-encompassing class "Object" for abstract methods' argument types and casting the instances to the appropriate type in the implementing class. However, in a recent revision, the authors used Java generics to provide this flexibility (e.g. *Classifier* is parameterized by generic data $< D >$. *LogisticRegression* is a *Classifier* that requires numeric data while *TwoClassSVM* is also a *Classifier* but it can use any data type given the appropriate *Kernel* for it).

There are also many different learning tasks including density estimation, classification, novelty detection, and regression. Each of these pattern analyses are used for estimating different kinds of functions. For instance, regression attempts to map data points to a real-value while classification tries to map them to a discrete value representing the "category" to which the point belongs. As with data types, the return type of various abstract methods was also made generic to provide this flexibility.

A.1 Model Hierarchy

To implement learning models, we designed a Java type hierarchy that attempts to capture relationships between various kinds of learning models. This hierarchy is shown in Figure 15. The main connecting thread between these different models is the concept of prediction, which is realized in the common method *predict*. The *predict* method, in general, takes a generic data type D and returns a generic data type R. However, specific learning algorithms are not, in general, this flexible. Instead, most require specific data types (typically numeric) for training, which fits well into the Java Generics framework.

At the top of the inheritance hierarchy was the abstract class *LearningModel* which provides the method *predict*, but only an abstract specification of it given that prediction is done differently in each learning model. Particularly, the algorithms that extend from *NoveltyDetector* implement a method *isOutlier* that returns a boolean value. This method is used in turn to implement the *predict* method for *NoveltyDetector*, which is a class whose return value is defined as *Boolean*. Thus, the framework of novelty

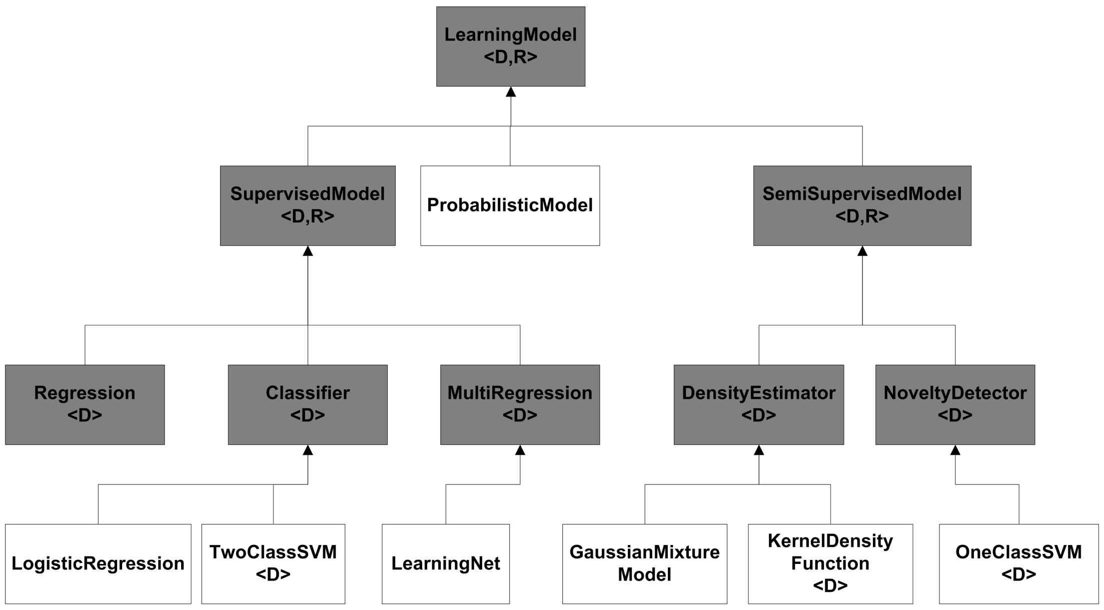


Figure 15: The class inheritance hierarchy of the learning models. As with all Java classes, this hierarchy exhibits single inheritance only which is represented by an arrow from the child to the parent class. Abstract classes are represented as grayed-in boxes while the realized models are clear boxes. The <> notation represents the generic object types used by a class; in particular a D indicates the data that can be handled is generic and a R indicates the return type of prediction is generic.

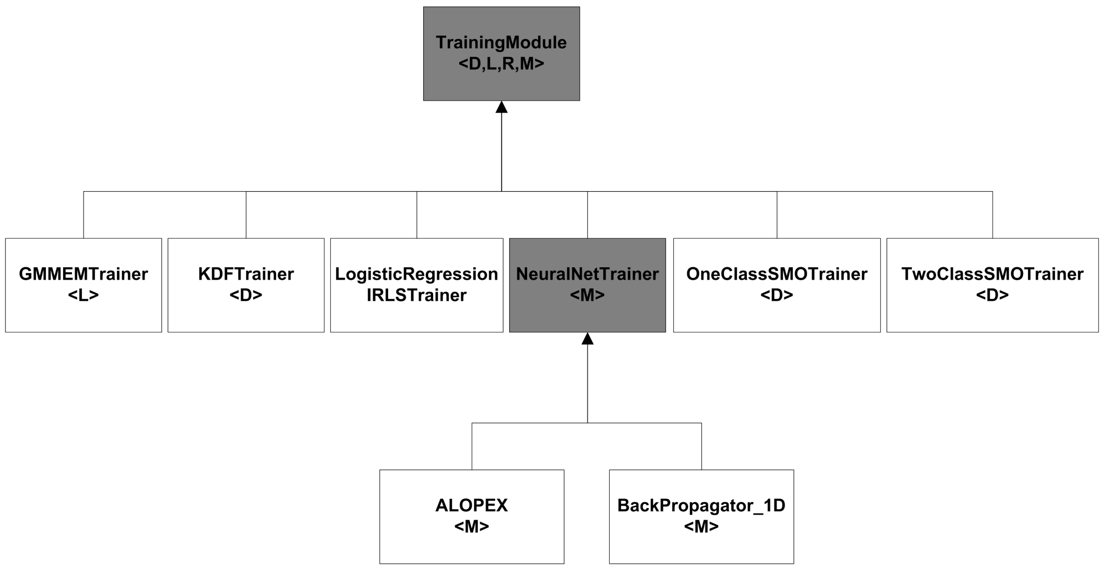


Figure 16: The class inheritance hierarchy of the training modules. As with all Java classes, this hierarchy exhibits single inheritance only which is represented by an arrow from the child to the parent class. Abstract classes are represented as grayed-in boxes while the realized models are clear boxes. The <> notation represents the generic object types used by a class; in particular, the model type trained by a trainer is specified by M, the data-type of those models is specified by D, the return type of the model's *predict* method is specified by R, and the type of the labels is specified by L.

detection is captured via the hierarchy of *LearningModel* and Java Generics. Thus, the specifics of *predict* (or more meaningfully named methods such as *isOutlier* are implemented according to the specification of a particular model, but the hierarchy is used to enforce the typing so that instances of *LearningModel* can be used somewhat interchangeability[1].

The interchangeability of our classes that inherit from *LearningModel* allows us to consider combining learning algorithms with little to no code. For instance, *boosting* (building a strong learner from a sequence of weak ones) or *voting* schemes could be implemented almost trivially. For the purposes of our project, this notion really meant that little work was required to try several different models once the models were written.

A.1.1 Training Learning Models

One should immediately notice that, while prediction was a central theme in our *LearningModel* hierarchy, there is no mention of training. In fact, training was

[1]The notion of making interchangeable learning algorithms was first implemented using Java's *Object* in place of Generics, but the latter provides stronger type enforcement. In fact, replacing *Object*s with Generics revealed a few minor bugs in our code.

implemented in a separate hierarchy as shown in Figure 16. While most learning model implementations couple training algorithms with learning models, this implementation separates the concepts for two reasons. First, occasionally there are multiple ways to train models. For instance, while most neural network training is done via back-propagation (implemented by class *Backpropagator_1D*, but the Alopex algorithm [82] is an alternative training mechanism implemented in *ALOPEX*. Secondly, there are also instances when a single training algorithm could train several learners. For instance, both one and two class SVMs use the SMO algorithm discussed in Appendix A.2, although the current code-base doesn't unify their trainers due to slight technicalities.

As with *LearningModel* the core of *TrainingModule* was a single method; in this case, the *train* method[2]. This method takes a *LearningModel* and a *DataSet* of appropriate types and returns the corresponding *LearningModel* after training is complete. In actuality, this is generally a destructive method that modifies the model it receives and returns it as a convenience.

As with the concept of training, we also implemented a separate hierarchy for "Meta-Learning" based on the idea of using multiple training/validation iterations to tune certain "meta-parameters" of a learner. However, this concept has not been emphasized in the development of our learning algorithms and remains in an experimental phase.

A.2 Implementing the One-Class SVM

The implementation of the One-Class SVM followed the adaptation of the Sequential Minimal Optimization (SMO) technique developed for Two-Class SVMs [32, 55, 56] to their single class counterparts [61, 62]. Other techniques such as the improved cache replacement policy suggested in Li *et. al.* [36] were also critical for achieving fast training on potentially large corpora. Following the algorithms presented in these works, an implementation of the SMO algorithm was crafted under the training hierarchy as the class *OneClassSMOTrainer*. The implementation proved highly effective as shown empirically in Anil Sewani's thesis [68].

Our hierarchy was tailored to incorporating kernel-based algorithms such as those suggested in Shawe-Taylor and Cristianini [69]. In particular, the specifications of our learning algorithms was specifically made to handle generic data through the recent addition of "generics" to Java 1.5 (originally the specification was done in terms

[2]This method is actually specified by the interface *Trainable* for technical reasons.

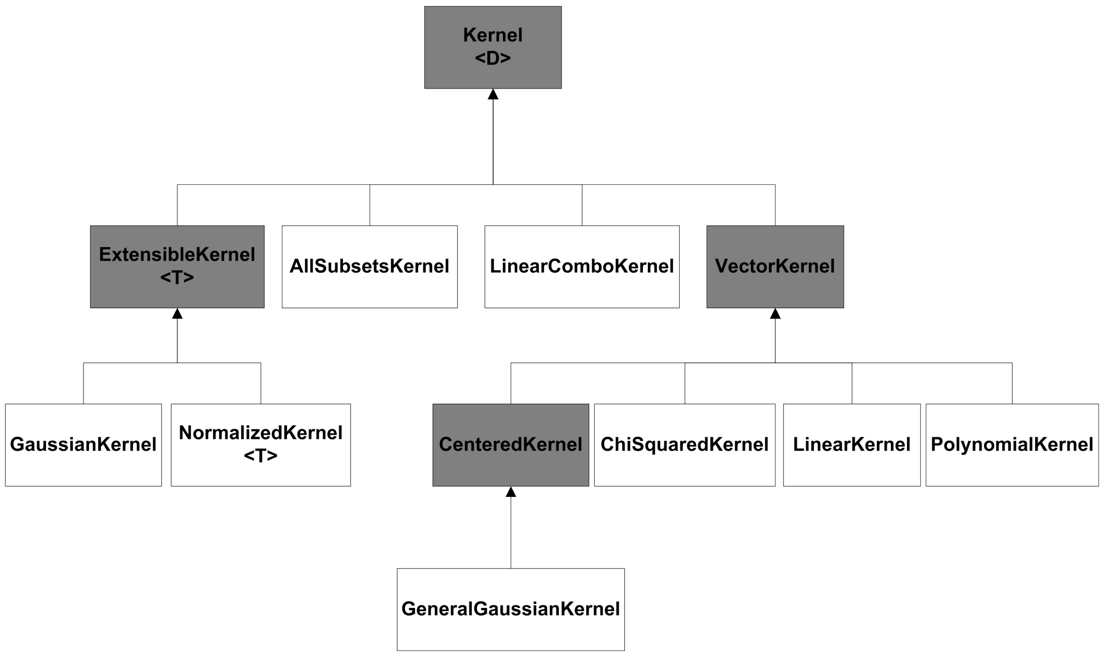


Figure 17: The class inheritance hierarchy of the kernels. As with all Java classes, this hierarchy exhibits single inheritance only which is represented by an arrow from the child to the parent class. Abstract classes are represented as grayed-in boxes while the realized models are clear boxes. The <> notation represents the generic object types used by a class; in particular, the data type handled by the *Kernel* is denoted either D or T.

of Objects, but Java generics have allowed for a higher degree of type safety within the hierarchy). In particular the approach of kernel methods suggests modular design, where the learning technique is implemented independent of the data's structure, while structure is captured by a kernel – a specially defined function that measures the similarity between two objects (the specific properties of this function are beyond the scope of this appendix; for more details see Shawe-Taylor and Cristianini [69]). Thus, the kernel methods developed under our hierarchy were generics specified by the type of input data and the kernel designed to compare objects of that type.

The kernel functions were also developed in a hierarchy rooted from the abstract object *Kernel*. Thus, implemented kernels inherited from this function and defined the specifics of the abstract function *kernel* to handle a pair of objects of the type specified by the class, and return a non-negative real value specifying the similarity between those objects (an implicit inner-product in some high-dimensional space defined by the kernel). A variety of kernels of common kernels were implemented to work with common object classes especially vectorial data [69]. A number of kernels also operated on the results of other kernels; for instance, the *NormalizedKernel* implementation takes a *Kernel* in its constructor and returns the result equivalent to the normalized inner product for the original kernel. Similarly, the *LinearComboKernel* takes a set of *Kernels* and positive weights to implement the resulting linear combination of the original kernels.

In most cases, the implementation of each kernel was straightforward requiring only a few lines of Java. However, as with most Java applications, there was quite a bit of overhead since the Java "class" is being used to provide functions that can be passed as arguments. Moreover, as with most kernel implementations, there was no attempt to verify that an implemented kernel obeyed the positive semi-definite property required of kernels [69].

A.3 Implementing the Probabilistic Model

As with the one-class SVM, Naïve Bayes was implemented in Java as a special case of the more general class of models known as probabilistic tree models. These models can be represented as tree structure (directed) graphs that represent the independence assertions about a set of random variables. The graph for Naïve Bayes is the simple tree shown in Figure 8 with a single unobserved (latent) variable that represents the label of the email as viral or non-viral. As mentioned in Section 4.4.1, this structure represents the fact that all features are independently conditioned on the latent variable - an assumption discussed in that section. To implement tree structures in general, we made a generic message-passing algorithm.

Graphical models express the independence structure between random variables, but we are interested in the probability that the random variables take on certain values given that we have evidence; a process known as inference. To infer information for a specific query from a probabilistic graphical model, we must be able to perform certain operations on the graph: summation and multiplication. This pair of operations allow us to collect information from the adjacent random variables and marginalize the effect an unknown random variable has on the query. In trees, these operations can be done directly on the original structure by means of a simple algorithm known as the "sum-product algorithm". This a two phase message-passing algorithm consisting of collecting information from children followed by disseminating the results from the root back down to the children. These operations are made efficient through matrix multiplications which implicitly performed the sums over products required to propagate information. For more detailed information on the sum-product algorithm, see Chapter 3 and 4 of Jordan [29].

The message-passing scheme was also used in training using the expectation-maximization algorithm [29]. Without going into the detail of the algorithm, the message-passing protocol allows us to estimate the parameters of the latent variables. This process was simplified by only considering internal nodes of finite support. Moreover, the message-passing protocol also made it easy to estimate the maximum a-posteriori (MAP) estimates from the model; that is, the most likely assignment of latent variables given the observed data. This simple modification was done by implementing the max-product algorithm in the message-passing protocol. For more details on the specifics of such algorithms for probabilistic tree models, refer to Jordan's textbook on the subject, "An Introduction to Probabilistic Graphical Models" [29].

Unfortunately, as Naïve Bayes was added after the original design of the project, it hasn't been fully integrated into the *LearningModel* framework. It didn't fit as cleanly into the framework as previous models due to the fact that it was implemented as an after thought with time constraints. Moreover, unlike our other models, training is integrated directly into the underlying representation of the tree-structure since message-passing is so closely knit with it. In the future, we'd like to integrate this class of models into a single unified hierarchy under the class *ProbabilisticModel*.

Appendix B

Asymptotics of Attack Strategies

Having formally defined the space of attacks on the hypersphere, a simple analysis of the objective function in Eq. (5.3) yields a good deal of insight into the problem. First, we note that the denominators of each term in the summation are summations themselves $\sum_{\ell=1}^{t} \alpha_\ell$ - summations of non-negative elements. If α_ℓ can be described by a monotonic nonnegative continuous function f such that $\alpha_\ell = f(\ell)$, then from Appendix A of "Introduction to Algorithms" [10] we have the following bound on the denominator:

$$\begin{aligned}
\forall x < y, f(x) \le f(y) \quad &\Rightarrow \quad \int_0^t f(x)\,dx \le \sum\nolimits_{\ell=1}^{t} f(\ell) \le \int_1^{t+1} f(x)\,dx \\
\forall x < y, f(x) \ge f(y) \quad &\Rightarrow \quad \int_1^{t+1} f(x)\,dx \le \sum\nolimits_{\ell=1}^{t} f(\ell) \le \int_0^{t} f(x)\,dx
\end{aligned}$$

Using these bounds, the objective function can be bounded for monotonic sequences:

$$\forall x < y, f(x) \le f(y) \quad \Rightarrow \quad \sum_{t=1}^{T} \frac{f(t)}{\int_1^{t+1} f(x)\,dx} \le D_T(A) \le \sum_{t=1}^{T} \frac{f(t)}{\int_0^{t} f(x)\,dx} \tag{B.1}$$

$$\forall x < y, f(x) \ge f(y) \quad \Rightarrow \quad \sum_{t=1}^{T} \frac{f(t)}{\int_0^{t} f(x)\,dx} \le D_T(A) \le \sum_{t=1}^{T} \frac{f(t)}{\int_1^{t+1} f(x)\,dx} \tag{B.2}$$

The bounds in Eq. (B.1) and (B.2) allow us to describe the asymptotic behavior of most simple functions. Table 5 presents estimations of the behavior of a few simple functions.

Several interesting results can be seen in Table 5. To begin with, it is interesting that there is a monotonically decreasing strategy of $f(t) = \frac{C}{t}$ that approaches strategies that still achieve unbounded progress as $T \to \infty$ even if this result cannot be reproduced with integers. Also, it is interesting that a constant number of attack points C achieves unbounded progress as $T \to \infty$. Moreover, an exponential strategy only achieves linear progress. Furthermore, polynomial strategies only seem to perform a constant factor

$f(t)$	**Lower Bound**	**Upper Bound**	**Asymptotic Behavior**
C	$\sum_{t=1}^{T}\frac{1}{t}$	$\sum_{t=1}^{T}\frac{1}{t}$	$\ln T + O(1)$
$C \cdot t$	$2\sum_{t=1}^{T}\frac{1}{t+2}$	$2\sum_{t=1}^{T}\frac{1}{t}$	$2(\ln T + O(1))$
$C \cdot t^2$	$3\sum_{t=1}^{T}\frac{t}{t^2+3t+3}$	$3\sum_{t=1}^{T}\frac{1}{t}$	$3(\ln T + O(1))$
$C \cdot t^3$	$4\sum_{t=1}^{T}\frac{t^2}{t^3+4t^2+6t+4}$	$4\sum_{t=1}^{T}\frac{1}{t}$	$4(\ln T + O(1))$
$C \cdot \beta^{t-1}$ s.t. $\beta > 1$	$\frac{\ln\beta}{\beta}\sum_{t=0}^{T-1}\frac{\beta^t}{\beta^t-1}$	$\ln\beta\sum_{t=0}^{T-1}\frac{\beta^t}{\beta^t-1}$	$\frac{\beta-1}{\beta}T$
$C \cdot \beta^{t-1}$ s.t. $0 < \beta < 1$	$\ln\beta\sum_{t=0}^{T-1}\frac{\beta^t}{\beta^t-1}$	$\frac{\ln\beta}{\beta}\sum_{t=0}^{T-1}\frac{\beta^t}{\beta^t-1}$	$\frac{1-\beta}{\beta\ln\beta}\ln\left(\frac{1-\beta}{1-\beta^{T+1}}\right)\underset{T\gg 0}{\longrightarrow}\frac{(1-\beta)\ln(1-\beta)}{\beta\ln\beta}$
$\frac{C}{t}$	$\sum_{t=1}^{T}\frac{1}{t\ln(t)+t}$	$\sum_{t=1}^{T}\frac{1}{t\ln(t+1)}$	$\ln(\ln T)$

Table 5: This table describes the behavior of several simple (monotonic) strategies of placing attack points as functions of the current iteration t where $C > 0$ and β are constants. Upper and lower bounds are based on Eq. (B.1) for increasing functions and Eq. (B.2) while asymptotic behavior was determined by more detailed analysis not described here. It should be noted that several of these strategies occur in χ and thus do not represent actual attack strategies. Some, for instance the last two, cannot be accurately reproduced as integer solutions since they are monotonically decreasing toward 0.

better than the constant strategy! In fact, this result is true of all simple monomial strategies.

Theorem B.0.1 *A simple monomial strategy* $\alpha_t = t^k$ *for* $k \in \mathbb{R}^{0+}$ *yields* $\mathrm{D}_T(\{\alpha_t\}) = O((k+1)\cdot\ln T)$.

Proof Since $k \in \mathbb{R}^{0+}$, $\alpha_t = f(t) = t^k$ is monotonically increasing (although certainly not strictly for $k = 0$) and thus Eq. (B.1) applies. Therefore,

$$\begin{aligned} D_T(A) &\leq \sum_{t=1}^{T} \frac{f(t)}{\int_0^t f(x)\,dx} \\ &= (k+1)\sum_{t=1}^{T} \frac{t^k}{x^{k+1}|_0^t} \\ &= (k+1)\sum_{t=1}^{T} \frac{1}{t} \\ &= (k+1)(\ln(T) + O(1)) \end{aligned}$$

Hence, $\mathrm{D}_T(\{\alpha_t\}) = O((k+1)\cdot\ln T)$.

QED

Now, given that we have bounds in terms of T in Table 5, it is tempting to solve for what T is in terms of M in order to rank the above strategies in terms of optimality. For $f(t) = 1$ it is clear that $T = M$ and thus, the strategy achieves $\mathrm{D}(M) \sim \ln M$. For a k-order monomial strategy, the constraint that $\sum_{t=1}^{T}\alpha_t = M$ yields $T \sim M^{1/(k+1)}$. Now given the results from Theorem B.0.1 we find that $\mathrm{D}(M) = O(\ln M)$. Similarly, for the exponential strategy with base $\beta > 1$, the time is $T = \log_\beta[M(\beta-1)+1]$ which again yields $\mathrm{D}(M) = O(\ln M)$. The above analysis was not rigorous, but certainly discouraged me from this approach; everything is $O(\ln M)$. Nonetheless, it does suggest that $\ln M$ is a reasonable upper bound on the progress.

Appendix C

Proof of Theorem 5.3.5

Proof (By Contradiction): Assume not; that is, assume the following:

$$\exists A_{opt} = \{\alpha_k^*\}_{k=1}^{T} \tag{C.1}$$

$$\text{s.t. } \forall A \in \mathfrak{A} \quad \mathrm{D}\left(A_{opt}\right) \leq \mathrm{D}\left(A\right) \tag{C.2}$$

$$\exists i \in \{1, \ldots, T-1\} | \alpha_i^* > \alpha_{i+1}^* \tag{C.3}$$

Now consider the alternative sequence that switches the i-th and $i+1$-th element of A^*: $A' = \{\beta_k\}_{k=1}^{T}$ where $\forall k \neq i, i+1 \quad \beta_k = \alpha_k^*$ and $\beta_i = \alpha_{i+1}^*$ and $\beta_{i+1} = \alpha_i^*$. Comparing the difference in the distances achieved by these two sequences we see the following:

$$\begin{aligned}
\mathrm{D}\left(A_{opt}\right) - \mathrm{D}\left(A'\right) &= \sum_{k=1}^{T} \frac{\alpha_k^*}{\sum_{\ell=1}^{k} \alpha_\ell^*} - \sum_{k=1}^{T} \frac{\beta_k}{\sum_{\ell=1}^{k} \beta_\ell} \\
&= \sum_{k=1}^{T} \left[\frac{\alpha_k^*}{\sum_{\ell=1}^{k} \alpha_\ell^*} - \frac{\beta_k}{\sum_{\ell=1}^{k} \beta_\ell} \right] \\
&= \underbrace{\sum_{k=1}^{i-1} \left[\frac{\alpha_k^*}{\sum_{\ell=1}^{k} \alpha_\ell^*} - \frac{\beta_k}{\sum_{\ell=1}^{k} \beta_\ell} \right]}_{0} \\
&\quad + \frac{\alpha_i^*}{\sum_{\ell=1}^{i} \alpha_\ell^*} - \frac{\beta_i}{\sum_{\ell=1}^{i} \beta_\ell} + \frac{\alpha_{i+1}^*}{\sum_{\ell=1}^{i+1} \alpha_\ell^*} - \frac{\beta_{i+1}}{\sum_{\ell=1}^{i+1} \beta_\ell} \\
&\quad + \underbrace{\sum_{k=i+2}^{T} \left[\frac{\alpha_k^*}{\sum_{\ell=1}^{k} \alpha_\ell^*} - \frac{\beta_k}{\sum_{\ell=1}^{k} \beta_\ell} \right]}_{0} \\
&= \frac{\alpha_i^*}{\sum_{\ell=1}^{i-1} \alpha_\ell^* + \alpha_i^*} - \frac{\alpha_{i+1}^*}{\sum_{\ell=1}^{i-1} \alpha_\ell^* + \alpha_{i+1}^*} \\
&\quad + \frac{\alpha_{i+1}^*}{\sum_{\ell=1}^{i-1} \alpha_\ell^* + \alpha_i^* + \alpha_{i+1}^*} - \frac{\alpha_i^*}{\sum_{\ell=1}^{i-1} \alpha_\ell^* + \alpha_i^* + \alpha_{i+1}^*}
\end{aligned}$$

The only obscure step is the third step where the first and last sum were declared to be 0. The first sum is 0 since $\forall k < i \quad \alpha_k^* = \beta_k$. Thus, both the numerator and denominator were identical for each pair of terms (since the denominators only sum the first k terms) and the pairs of terms canceled. Similarly, for the last sum, $\forall k > i+1 \quad \alpha_k^* = \beta_k$ made the numerators of opposing terms equal. Moreover, since by definition $\beta_i = \alpha_{i+1}^*$ and $\beta_{i+1} = \alpha_i^*$, this swapping still results in the same denominator since addition is commutative.

Now let $\Sigma = \sum_{\ell=1}^{i-1} \alpha_\ell^*$. Then we can combine terms by finding the greatest common denominator. Note, we can disregard the resulting denominator of $(\Sigma + \alpha_i^*) \cdot (\Sigma + \alpha_{i+1}^*) \cdot (\Sigma + \alpha_i^* + \alpha_{i+1}^*)$ since it is strictly positive and this proof only relies on the sign of the above. Thus, we continue examining only the denominator of the combined terms:

$$\begin{aligned}
&\propto \alpha_i^* \left(\Sigma + \alpha_{i+1}^*\right)\left(\Sigma + \alpha_i^* + \alpha_{i+1}^*\right) \\
&\quad -\alpha_{i+1}^* \left(\Sigma + \alpha_i^*\right)\left(\Sigma + \alpha_i^* + \alpha_{i+1}^*\right) \\
&\quad +\left(\alpha_{i+1}^* - \alpha_i^*\right)\left(\Sigma + \alpha_i^*\right)\left(\Sigma + \alpha_{i+1}^*\right) \\
&= \left(\alpha_{i+1}^*\right)^2 \alpha_i^* - \left(\alpha_i^*\right)^2 \alpha_{i+1}^* \\
&= \alpha_i^* \cdot \alpha_{i+1}^* \quad \underbrace{\left(\alpha_{i+1}^* - \alpha_i^*\right)}_{\alpha_{i+1}^* < \alpha_i^* \text{by Eq. (C.3)}} \quad < 0
\end{aligned}$$

Note that I spared the reader the expansion and cancellation of terms from the original numerator. Also, it is implicit in this proof that all terms α_i are non-negative as defined in the definition of an attack sequence. This fact allowed us to discard several terms. Thus, we have

$$\mathrm{D}\left(A_{opt}\right) < \mathrm{D}\left(A'\right) \tag{C.4}$$

But Eq. (C.4) contradicts our assumption Eq. (C.2) thus proving the hypothesis. Hence, any optimal attack sequence must be monotonically increasing.

QED

Appendix D

Proof of Theorem 5.4.4

Proof Suppose that an arbitrary but particular total mass sequence, M, is an optimal solution to the program in Eq. (5.8). Thus, according to Lemma 5.4.3, the Eqs. (5.9), (5.10), and (5.11) must be satisfied for M. To reiterate, this means that the total mass sequence $M = \{M_t\}$ must have the following conditions:

$$\begin{array}{rrcl} & M_1 & = & 1 \\ 1 < t < T & M_t^2 & = & M_{t-1} \cdot M_{t+1} \\ & M_T & = & M \end{array}$$

corresponding to the Eqs. (5.9), (5.10), and (5.11). However, according to the program in Eq. (5.8), we also must have $M_t^* \in \mathbb{R}^+$, which means we can safely take the *logarithm* of these variables. Thus, the above equations are equivalent to the following conditions:

$$\ell M_1 = 0 \tag{D.1}$$

$$1 < t < T \quad 2 \cdot \ell M_t = \ell M_{t-1} + \ell M_{t+1} \tag{D.2}$$

$$\ell M_T = \ell M \tag{D.3}$$

where $\ell M_t = \log M_t$ and $\ell M = \log M$.

Thus, we have a set of linear equations[1] which can be written as,

$$\begin{bmatrix} 1 & 0 & & 0 & 0 & & \\ -1 & 2 & -1 & 0 & 0 & & \cdots \\ 0 & -1 & 2 & -1 & 0 & & \\ 0 & 0 & -1 & 2 & -1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 2 & -1 & 0 \\ & & \vdots & & -1 & 2 & -1 \\ & & & & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \ell M_1 \\ \ell M_2 \\ \ell M_3 \\ \ell M_4 \\ \vdots \\ \ell M_{T-2} \\ \ell M_{T-1} \\ \ell M_T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \ell M \end{bmatrix}$$

Due to the repetition along the diagonal, this system of equations can be solved generically for T by Gaussian elimination and yields the solution $\ell M_t = \frac{t-1}{T-1} \cdot \ell M$; the unique solution to this linear system. This corresponds to the solution $M_t = M^{\frac{t-1}{T-1}}$. Thus, for a sequence M to satisfy Eqs. (5.9), (5.10), and (5.11), $M_t = M^{\frac{t-1}{T-1}}$ which is equivalent the the sequence M^*. Thus, M^* is the unique sequence that satisfies those conditions. Moreover, since we showed in Lemma 5.4.3 that any optimal solution must satisfy Eqs. (5.9), (5.10), and (5.11), M^* is the unique optimal solution.

Having established optimality, it is easy to derive the optimal distance achieved from the alternate objective function Eq. (5.7):

$$\begin{aligned} \mathrm{D}_T^* (M) &= T - \sum_{t=2}^{T} \frac{M_{t-1}^*}{M_t^*} = T - \sum_{t=2}^{T} M^{\frac{t-2}{T-1}} M^{\frac{1-t}{T-1}} \\ &= T - M^{\frac{1}{T-1}} \sum_{t=2}^{T} 1 = T - (T-1) \cdot M^{\frac{1}{T-1}} \end{aligned}$$

as was to be shown.

Finally, using the definition of total mass in Definition 5.4.2, we see that for $t > 1$ we have:

$$\begin{aligned} x_t^* &= M_t^* - M_{t-1}^* = M^{\frac{t-1}{T-1}} - M^{\frac{t-2}{T-1}} \\ &= M^{\frac{t-2}{T-1}} \left(M^{\frac{1}{T-1}} - 1 \right) \end{aligned}$$

This completes the proof.

QED

[1] Another approach to solving this problem would be to solve it as a difference equation.

Bibliography

[1] ABC News and The Associated Press. Bill Gates gets 4 million e-mails a day. Online, 2005. `http://abcnews.go.com/US/wireStory?id=262372`.

[2] Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] P. Oscar Boykin and Vwani P. Roychowdhury. Leveraging social networks to fight spam. *IEEE COMPUTER*, 38(4):61–68, 2005.

[5] NOP World Business and Technology. Hi-tech crime: The impact on UK business 2005. Technical report, UK National Hi-Tech Crime Unit, 2005. `http://www.nhtcu.org/media/documents/publications/8817_Survey.pdf`.

[6] Colin Campbell and Kristin P. Bennett. A linear programming approach to novelty detection. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 395–401. MIT Press, 2000.

[7] Nicolò Cesa-Bianchi and Gábor Lugosi. Potential-based algorithms in on-line prediction and game theory. *Machine Learning*, 51(3):239–261, 2003.

[8] Shigang Chen and Yong Tang. Slowing down internet worms. In *Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS)*, Tokyo, Japan, March 2004.

[9] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. In Gerry Tesauro, David Touretzsky, and Todd Leen, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 7, pages 705–712. The MIT Press, 1995.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. McGraw Hill Book Company, 2^{nd} edition, 2001.

[11] D. R. Cox and D. Oakes. *Analysis of Survival Data*. CRC Press, 1984.

[12] Nilesh N. Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 99–108. ACM Press, 2004.

[13] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 17, pages 337–344. The MIT Press, 2005.

[14] Alexandre d'Aspremont, Laurent El Ghaoui, Michael I. Jordan, and Gert R. G Lanckriet. A direct formulation for sparse PCA using semidefinite programming. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 17, 2005.

[15] Srini Devadas and Eric Lehman. Sums, approximations, and asymptotics II. Internet, Oct 2004. `http://theory.lcs.mit.edu/classes/6.042/spring05/sums2.pdf`.

[16] Ted Humphreys et al. Assessing and managing digital risks. Technical report, UK Department of Trade and Industry and The International Underwriting Association of London, 2004. `http://www.dti.gov.uk/industry_files/pdf/Assessing_and_Managing_Digital_Risk.pdf`.

[17] Tom Fawcett and Foster J. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3):291–316, 1997.

[18] Eric Filiol. Strong cryptography armoured computer viruses forbidding code analysis: the bradley virus. In *Proceedings of the Fourteenth European Institute for Computer Anti-Virus Research Conference (EICAR)*, pages 201–217, 2005.

[19] Gregory R. Ganger, Gregg Economou, and Stanley M. Bielski. Self-securing network interfaces: What, why and how. Technical Report CMU-CS-02-144, Carnegie Mellon University, August 2002.

[20] Jennifer Golbeck and James Hendler. Reputation network analysis for email filtering. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2004.

[21] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn, and Robert Richardson. 2004 CSI/FBI computer crime and security survey. Technical report, Computer Security Institute and U.S. F.B.I., 2004.

[22] Paul Graham. A plan for spam. Online, 2002. `http://www.paulgraham.com/spam.html`.

[23] John Graham-Cumming. How to beat an adaptive spam filter. In *MIT Spam Conference*, 2004.

[24] International Data Group. Worldwide email usage 2002 - 2006: Know what's coming your way, 2002.

[25] Ajay Gupta and R. Sekar. An approach for detecting self-propagating email using anomaly detection. In *Recent Advances in Intrusion Detection (RAID)*, pages 55–72, 2003.

[26] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[27] Wolfgang Härdle, Marlene Müller, Stefan Sperlich, and Axel Werwatz. *Nonparametric and Semiparametric Models*. Springer-Verlag, 2004.

[28] Wolfgang Hoschek. The Colt distribution: Open source libraries and high performance scientific and technical computing in Java. Internet, Nov 2002. `http://hoschek.home.cern.ch/hoschek/colt/`.

[29] Michael I. Jordan. An introduction to probabilistic graphical models. MIT Press, 2002.

[30] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[31] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

[32] S. Sathiya Keerthi and Elmer G. Gilbert. Convergence of a generalized SMO algorithm for SVM classifier design. *Machine Learning*, 46(1-3):351–360, 2002.

[33] Bryan Klimt and Yiming Yang. Introducing the Enron corpus. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2004.

[34] Aleksander Kolcz, Abdur Chowdhury, and Joshua Alspector. The impact of feature selection on signature-driven spam detection. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2004.

[35] Jeremy Z. Kolter and Marcus A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 470–478, New York, NY, 2004. ACM Press.

[36] Jianmin Li, Bo Zhang, and Fuzong Lin. A new cache replacement algorithm in SMO. In *Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines (SVM)*, pages 342–353, London, UK, 2002. Springer-Verlag.

[37] PricewaterhouseCoopers LLP. Information security breaches survey 2004. Technical report, UK Department of Trade and Industry. `http://www.infosec.co.uk/files/DTI_Survey_Report.pdf`.

[38] Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 376–385, New York, NY, USA, 2002. ACM Press.

[39] Markos Markou and Sameer Singh. Novelty detection: A review - part 1: Statistical approaches. *Signal Processing*, 83(12):2481–2497, 2003.

[40] Steve Martin, Anil Sewani, Blaine Nelson, Karl Chen, and Anthony D. Joseph. Analyzing behaviorial features for email classification. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2005. `http://www.ceas.cc/papers-2005/123.pdf`.

[41] Steven L. Martin. Learning on email behavior to detect novel worm infections. Master's thesis, University of California at Berkeley, 2005.

[42] Andreas Marx. Outbreak response times: Putting AV to the test. *Virus Bulletin*, February 2004. `http://www.av-test.org/down/papers/2004-02_vb_outbreak.pdf`.

[43] McAfee Corporation. W32/Mydoom.ah@MM. Online, 2004. `http://vil.nai.com/vil/content/v_129631.htm`.

[44] McAfee Corporation. W32/Mydoom.bb@MM. Online, 2005. `http://vil.nai.com/vil/content/v_131856.htm`.

[45] MessageLabs. The skeptic factor. Online, 2002. `http://www.hotspace.com.au/library/pdf/MessageLabs-TheSkepticFactor.pdf`.

[46] MessageLabs. MessageLabs intelligence annual email security report. Online, 2004. `http://www.messagelabs.com/Threat_Watch/Intelligence_Reports/Archive`.

[47] Tony A. Meyer and Brendon Whateley. SpamBayes: Effective open-source, Bayesian based, email classification system. In *Proceeding of the Conference on Email and Anti-Spam (CEAS)*, 2004.

[48] Microsoft Corporation. Microsoft security bulletin MS04-028. Online, 2004. `http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx`.

[49] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.

[50] David Moore, Colleen Shannon, and Jeffery Brown. Code-Red: a case study on the spread and victims of an internet worm. In *Proceedings of the Second ACM SIGCOMM Workshop on Internet measurment (IMW)*, pages 273–284, New York, NY, USA, 2002. ACM Press.

[51] MXLogic. Email defense industry statistics. Online, 2005. `http://www.mxlogic.com/PDFs/Industry_Stats_02_03_05.pdf`.

[52] Mark E. J. Newman, Stephanie Forrest, and Justin Balthrop. Email networks and the spread of computer viruses. *Physical Review*, E 66(035101), 2002.

[53] Panda Software. Products - TruPrevent technologies. Online, 2005. `http://www.pandasoftware.com/products/truprevent_tec/`.

[54] Anthony L. Peressini, Francis E. Sullivan, and Jr. J.J. Uhl. *The Mathematics of Nonlinear Programming*. Springer-Verlag, 1988.

[55] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, Redmond, Washington, April 1998.

[56] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[57] Marcus J. Ranum. False positives: A user's guide to making sense of IDS alarms. Online, 2003. `http://www.icsalabs.com/html/communities/ids/whitepaper/FalsePositives.pdf`.

[58] Russell D. Reed and Robert J. Marks II. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. The MIT Press, 1999.

[59] Stephen J. Roberts. Novelty detection using extreme value statistics. *IEEE Proceedings on Vision, Image and Signal Processing*, 146(3):124–129, 1999.

[60] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI TR WS-98-05.

[61] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. Technical Report 87, Microsoft Research, 1999.

[62] Bernhard Schölkopf, Robert C. Williamson, Alex J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 12, pages 582–588. MIT Press, 2000.

[63] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy (IEEE S&P)*, May 2001.

[64] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. MEF: Malicious email filter, a UNIX mail filter that detects malicious windows executables. In *USENIX Annual Technical Conference - FREENIX Track*, June 2001.

[65] Alan Schwartz. *SpamAssassin*. O'Reilly Media, Inc, 2004.

[66] Richard Segal, Jason Crawford, Jeff Kephart, and Barry Leiba. SpamGuru: An enterprise anti-spam filtering system. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2004.

[67] Larry J. Seltzer. Security watch: Mydoom reappears. *Security Watch Newsletter*, February 2005. `http://www.pcmag.com/article2/0,1759,1767805,00.asp`.

[68] Anil A. Sewani. A system for novel email virus and worm detection. Master's thesis, University of California at Berkeley, 2005.

[69] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[70] Sophos Corporation. War of the worms: Netsky-P tops list of year's worst virus outbreaks. Online, 2004. `http://www.sophos.com/pressoffice/pressrel/uk/20041208yeartopten.html`.

[71] Seth Spitzer. Mozilla spam / junk mail architecture. Online, 2003. `http://www.mozilla.org/mailnews/arch/spam/index.html`.

[72] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *USENIX Security Symposium*, pages 149–167, Berkeley, CA, USA, 2002. USENIX Association.

[73] Ingo Steinwart, Don Hush, and Clint Scovel. A classification framework for anomaly detection. *Journal of Machine Learning Research*, 6:211–232, 2005.

[74] Salvatore J. Stolfo, Shlomo Hershkop, Ke Wang, Olivier Nimeskern, and Chia-Wei Hu. A behavior-based approach to securing email systems. In *Mathematical Methods, Models and Architectures for Computer Networks Security*, 2003.

[75] Salvatore J. Stolfo, Wei-Jen Li, Shlomo Hershkop, Ke Wang, Chia-Wei Hu, and Olivier Nimeskern. Detecting viral propagations using email behavior profiles. In *ACM Transactions on Internet Technology (TOIT)*, 2004.

[76] Symantec Corporation. VBS.BubbleBoy. Online, 1999. `http://www.symantec.com/avcenter/venc/data/vbs.bubbleboy.html`.

[77] Symantec Corporation. Wscript.KakWorm. Online, 1999. `http://www.symantec.com/avcenter/venc/data/wscript.kakworm.html`.

[78] Symantec Corporation. W32.Nimda.A@mm, 2001. `http://www.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html`.

[79] Symantec Corporation. W32.Beagle.F. Online, 2004. `http://securityresponse.symantec.com/avcenter/venc/data/w32.beagle.f@mm.html`.

[80] Symantec Corporation. Symantec Security Response, 2005. `http://www.symantec.com/avcenter/`.

[81] Symantec Corporation. W32.Beagle.BG. Online, 2005. `http://www.sarc.com/avcenter/venc/data/w32.beagle.bg@mm.html`.

[82] K. P. Unnikrishnan and K. P. Venugopal. Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, 6(3):469–490, 1994.

[83] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[84] WEKA. Machine learning project. Internet. `http://www.cs.waikato.ac.nz/~ml/index.html`.

[85] Matthew M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. Technical Report HPL2002 -172, HP Laboratories Bristol, 2002.

[86] Gregory L. Wittel and S. Felix Wu. On attacking statistical spam filters. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2004.

[87] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2^{nd} edition, 2005.

[88] Cynthia Wong, Stan Bielski, Jonathan M. McCune, and Chenxi Wang. A study of mass-mailing worms. In *Proceedings of the 2004 ACM workshop on Rapid malcode (WORM)*, pages 1–10, New York, NY, USA, 2004. ACM Press.

[89] Wei Xu, Peter Bodík, and David Patterson. A flexible architecture for statistical learning and data mining from system log streams. In *Temporal Data Mining: Algorithms, Theory and Applications*, Brighton, UK, November 2004. The Fourth IEEE International Conference on Data Mining.

[90] Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. Technical report, Stanford University, 2004.