

Semi-Supervised Learning on Email Characteristics for Novel Worm Detection

Steve Martin and Anil Sewani
{steve0, anil}@cs.berkeley.edu
University of California, Berkeley

A major drawback of unsupervised learning for worm detection is the possibility of false negatives. Previous work copes with this problem by increasing the sensitivity of the unsupervised classification algorithms. This, in turn, creates many more false positives. Our focus is narrowed to worms propagating through email.

We present the following contributions. First, we examine a wide range of features calculated on email traffic to determine indicators that discriminate between infected from normal email behavior. Using these features, we next present a new method that uses semi-supervised learning for adaptive virus detection that leverages system administrator feedback to improve classification. Our approach combines the strengths of sensitive novelty detection with a parametric classifier to drastically reduce the false positives.

1 Introduction

One of the most prevalent security problems in computing today is the rampant proliferation of malicious, self-propagating computer viruses known as worms. As networks become increasingly ubiquitous, these programs can infect more machines than ever before, with each new outbreak causing staggering amounts of damage. As an example, in 2001, two worms named Nimda and Code Red spread rapidly throughout the internet, with cost estimates for lost productivity in the billions of dollars [3]. Last year, the MyDoom and MS Blaster worms combined for estimated damages in the tens of billions of dollars in the United States alone [2]. These recent events serve to highlight a disturbing trend: as industry, governments, and militaries grow more reliant on computer networks, the potential for a truly crippling virus attack is becoming immense.

While protection against worms continues to be an area of intense research, traditional antivirus defenses deployed in the field have not changed significantly for many years. Recent worms such as MS Blaster and Sapphire have demonstrated that in the time it takes for an administrator to reconfigure a firewall and update antivirus scanners, widespread infec-

tion among vulnerable hosts can already occur. To make matters worse, contemporary worms spread at extremely fast rates; it has been shown that it is reasonably trivial to create self-propagating internet viruses that can infect up to a million hosts in about thirty seconds [19].

It is clear that the key to stopping a novel worm from becoming widespread is to choke off its avenues for infection as quickly as possible. One general approach is to place specific restrictions on the network activity a machine can conduct over a given length of time. This idea, called a *virus throttle*, works well in certain scenarios, but is also overly restrictive and inflexible to changes in network utilization [5, 8].

An alternate idea is to learn on network transmissions to attempt to discover and isolate infected machines before an epidemic can set in. Previous work has used unsupervised learning on network behavior to attempt to detect worm propagation. However, because false negatives are highly undesirable in virus detection, these systems are in general purposefully configured to be overly sensitive. This trades off additional false negatives for excessive amounts of false positives.

To this end, we present the following contributions. First, we examine a wide range of features calculated on email traffic to determine what indicators best separate virus from normal email behavior. Using these features, we next present a new method for doing *semi-supervised* learning that leverages system administrator involvement to improve classification. This is done via an approach that combines the strengths of sensitive novelty detection with a parametric classifier that drastically reduces the false positives.

This paper presents the results of our work. Section 2 gives a brief overview of previous work related to this paper. Next we discuss the overall structure of our application in section 3, followed by a description of the numerical features we calculate on email traffic in section 4. Section 5 describes the parametric classifier. Section 6 gives some preliminary results and discussion, and we close with some conclusions and ideas for future research.

2 Previous Work

Current work on containing worms (including email worms) has largely been based on automatic detection of abnormal network behavior. The La Brea project, for example, attempts to slow the growth of TCP based worms by slowing probes to unallocated addresses [21]. This is accomplished by blocking the thread making the probe. This technique however can be easily circumvented by running the virus asynchronously.

Williamson et al proposed per-host TCP throttling by restricting the rate of new connections a host makes in a given time [5]. A similar approach has been proposed for constraining email virus propagation by restricting the frequency at which email can be sent to new email recipients [8]. Apart from requiring universal deployment to be effective, this technique involves the difficult task of determining the threshold used to consider a TCP connection or an email to be malicious. A fixed setting of the threshold is also likely to delay legitimate traffic occasionally.

Several previously published methods of network intrusion detection involve variants of signature detection [12]. Signature detection uses libraries of past attacks and rule-learning techniques to identify attacks on the network. Vern Paxson's work on Bro provides a strongly-typed language for implementing policies to deal with abnormal events detected on network streams at a TCP/UDP level [15]. However, we believe this technique might be inadequate because modern virus attacks can overwhelm network resources before rules can be manually updated to provide protection against new threats.

Singh et al. recently proposed a mechanism for automated fingerprinting of worms [20]. The authors implemented an efficient distributed system that detects invariant substrings in the network traffic to fingerprint novel worms. The system aggregates this information across different source and destination IP addresses to determine if a worm attack is underway. Even though the system is shown to work well for new and known worms, there are some serious limitations on its effectiveness. Polymorphic worms (e.g. those that can make use of per-session encryption) can evade such a system completely by changing the payload for every infection. The attacker can also exploit the limited state information maintained by the system to perform efficiently. For example, as mentioned in the paper, worms seen less frequently than 60 seconds are never registered by the system.

Early attempts at using machine learning models for detection of viruses used Artificial Neural Networks for detecting boot sector viruses [4]. More recently, Schultz et al used a naïve Bayes classifier

trained over consecutive byte sequences in email attachments to detect malicious executables [9]. They later extended their work to include printable strings in executables as another feature to train their models [10]. Maloof et al have further explored this area of work by using models other than naïve Bayes (including decision trees, Support Vector Machines, k-nearest-neighbors and Term Frequency Inverse Document Frequency (TFIDF) models) [6]. In general, tracking malicious email by exclusively analyzing attachments tends to ignore a certain type of email worms, such as those that spread via HTML embedded scripts or links in emails. A glaring example of such a worm is MyDoom. The techniques mentioned above can also be easily fooled by using standard code obfuscation techniques.

Social network analysis of email has been used for detecting spam as well as virus emails [13, 7]. 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. Clusters of nodes linked to one another form a social network, indicating common communication patterns among users. Communication that violates these behavioral patterns is considered suspect of spam or virus email. To circumvent this technique, an email virus can trick the model by intelligently choosing its recipient list (e.g. by using recipient information from recent emails in the 'Sent Email' folder). Hence, such techniques alone can not provide a good defense against email worms.

Stolfo et al created an email data mining system that used social network analysis along with other features indicative of user behavior to identify viral propagations [16, 17]. The authors maintained 'user cliques' for every user in the system. User cliques are basically clusters of email accounts that participate with each other in common email communication. Other features considered by the authors included variance in number of distinct recipients, sending rate of email and the number of emails with attachments over a sliding window of emails. The social network analysis done here is vulnerable to the type of attacks mentioned earlier. The classification mechanism we describe in this paper includes all other features used by the authors, along with several others that we believe are useful in discriminating normal user behavior from viral behavior.

3 System Architecture

The overall architecture for our system consists of a modular pipeline. Each segment encapsulates a separate stage in the process of classifying email behavior

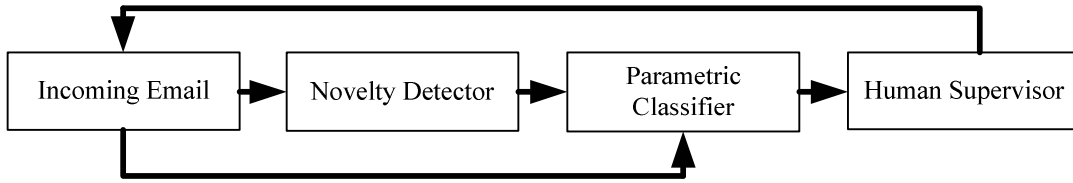


Figure 1. The basic architecture for the classification system presented in this paper.

as infected or normal.

Figure 1 gives an overview of our basic architecture. In the first module of our system, email is gathered into a data corpus on an ongoing basis. As messages arrive, statistical features are calculated on both the messages themselves and on the sender over a sliding window of emails. To do this, we implemented an online data-collection framework using Sendmail, a well-known SMTP server, and MIMEDefang, a mail filter using Sendmail’s “Milter” API. Further discussion of this framework, what email features we used, how they are calculated, and their distributions is given in section 4.

Once features have been calculated, the data points are then passed into the novelty detector. This module could contain a variety of methods for unsupervised learning. We have previously implemented mixture models and support vector machines. For the purpose of this paper, however, the novelty detector is unimportant; to eliminate variables and allow us to make statements about feature distributions and our parametric filtering layer, we model this module as a static probability of false positives. To avoid any assumptions about the effectiveness of novelty detection algorithms on our data, we set this value to be extremely high.

The next stage in the pipeline is the parametric classifier. As the second part of our multi-layer classification method, this module takes the results of the novelty detector and leverages previous human interaction with the network to filter out false positives. Although many methods of learning could be employed here, we have built a parametric classifier using a three-layer naive Bayes graphical model. The classifier improves estimates via semi-supervised learning on the corpus of email data and human actions resulting from previous classifications. A detailed explanation of this process is given in section 5.

Finally, data points that have been determined as abnormal by the novelty detector and as infected by the parametric classifier are passed on for inspection. A human supervisor will then react to the warnings by either deleting them as false positives or taking further action to decontaminate a machine. The reaction to each point is then used to update the data corpus with

labels, allowing the parametric classifier to be re-trained in a semi-supervised manner for increased accuracy. In our planned deployable system, a machine would be automatically quarantined to prevent propagation while awaiting inspection.

Our current prototype is implemented using Matlab 6.5 for our machine learning algorithms, Perl 5 for data collection and feature calculation on messages, and takes advantage of Sendmail and MIMEDefang to gather data on outgoing email.

4 Feature Analysis

Our system uses a large set of features designed to exploit differences between worms and normal email. For our approach to be successful, we require at least a subset of the features we model to differ statistically between normal and infected email behavior.

The following sections will provide details on the feature set. Section 4.1 gives a more detailed overview of our data collection framework. Section 4.2 describes the actual features themselves and gives some motivation for each of them.

4.1 Data Collection

To analyze feature distributions on normal and infected emails, we implemented an online data-collection framework. The framework is capable of processing a large amount of email traffic in real-time and saving the anonymous data points to disk. This system allows us to gather data, which is then used to determine the distributions of each of the features for normal and infected email traffic.

Our framework is built on the Sendmail 8.13.1 SMTP server running on Redhat Linux using the 2.4.18-14 kernel. We chose Sendmail because the program provides the ‘Milter’ API that allows for callbacks when various parts of a message are processed; these callbacks are often used by spam filters and virus scanners to quickly examine email. To interface Sendmail with Perl, we use MIMEDefang 2.45. MIMEDefang implements a Perl wrapper around the Sendmail Milter API, allowing us to quickly build a

Perl layer around Sendmail to manipulate email data and send it on to the feature calculation system. [18, 11]

Actual feature calculation is done by a separate server application that continually listens for TCP connections from MIMEDefang. Once email data is received over the network, the server calculates all of the statistical features and saves the results to log files. This program is also written in Perl.

To analyze our feature set with real data, we enlisted twenty volunteers within Berkley EECS to send their email using our authenticated Sendmail server. Our framework calculated features on these outgoing messages.

To gather the same type of data on email worms, we used our data collection framework in conjunction with Microsoft Virtual PC 2004. We loaded the guest OS on the virtual machine with one of the author’s address books containing three hundred and twenty addresses. We then saved the machine state and infected the virtual machine with several worms individually, restoring after each infection. In gathering worm data, the Windows 98 and XP operating systems were used.

While virtual machines provide a practical method of gathering data on email worm propagation, it should be noted that effects of virtualization on worms are unknown. Furthermore, we often had to ‘cheat’ by changing the host files in the guest OS in order to get worms with their own SMTP servers to send email to through our data collection framework. Therefore while our virus data provides significant insight into feature distributions on real email worms, we do not regard it as a true representative sample of real worm traffic.

4.2 Feature Descriptions

We implemented twenty six features initially from empirical observations of email behavior and previous work. Through further analysis, we then culled this list to the eighteen that were used to generate the results presented in this paper. There are several additional features we plan to implement; however, we learned much from our first set of features.

Our set of features can be divided into two distinct classes: *per-email* and *per-user* features. Per-email features are numerical values calculated on a single email, which can be thought of as a single point in the ongoing email activity that comprises a user’s behavior. Per-user features describe a particular facet of a user’s email behavior over a fixed sending window; an example of a per-user feature would be the mean number of characters in email sent over the window.

Table 1 gives a summary of the distribution type we assigned each feature. These decisions were made based on observation alone; in most cases, there was an obvious best fit. Example histogram plots of feature data are given in the Appendix.

The following sections discuss each feature briefly. Section 4.1.1 will explain the per-email features, while section 4.1.2 will give an overview of the per-user features.

4.1.1 Per-Email Features

We now describe the numerical values calculated on a per-email basis, along with the rationale behind why we consider each important. Referenced figures are included in the Appendix.

Whether or not there is HTML embedded in the email. Many viruses send emails with HTML in them, while several users, including the authors, pre-

Table 1. The distribution assigned as the best fit to per-email and per-user feature data for.

		<u>Distribution Type</u>		
		Gaussian	Exponential	Binary
Per-Email Features	<ul style="list-style-type: none"> - Average word length - Number of words in subject - Number of characters in subject 	<ul style="list-style-type: none"> - Number of attached files - Number of words in body 	<ul style="list-style-type: none"> - HTML in email - Script tags in email - Attribute Scripts in email - Images in email - Links in email - Attachment MIME type - UNIX Magic number type 	
Per-User Features	<ul style="list-style-type: none"> - Number of addresses sent to in window - Email frequency in window - Average characters in subject - Average words in body 	<ul style="list-style-type: none"> - Ratio of emails with attachments to emails without 		

fer plaintext messages. Examples are the Klez worm, and most macro viruses that use Microsoft Outlook.

Whether or not there are scripts in the email HTML. This is divided into two values: whether or not there are explicit script tags in the HTML of the email, and whether or not there are scripts in the attributes of tags in the email. There have been exploits in the past for Outlook and Eudora that involve scripts in html email. These exploits have been used in several worms, most notably Bubbleboy and Kak.worm.

The number of characters in the subject, the number of words in the subject, and the number of words in the body of the email. These three features leverage the fact that many email worms spread by sending replicates of a single email. The infamous LoveLetter.C macro virus is an example. In addition, several polymorphic worms draw subjects from a limited set of candidates. A distribution comparison between normal behavior and the Bagle.A email worm for the number of characters in the subject is given in figure 3 in the Appendix. A further example comparing the distribution of the number of words in the subject against the Loveletter.C worm is shown in figure 4.

Whether or not there are images embedded in the email. There are known buffer-overflow exploits in several email applications regarding image rendering. In addition, image tags in an email are used by malicious spammers to contact remote servers. While the authors are not presently aware of any worms that specifically take advantage of these bugs, it is possible that future viruses could.

Whether or not there are links in the email. Links in email could trick users into infecting themselves by visiting a website that exploits web browser vulnerabilities. While many users send links in perfectly legit emails, worms exploit this method of propagation as well. As an example, this is how MyDoom spread itself.

The number of files attached to the email. Most email viruses spread via at least one attachment. However, we found that most normal email does not include attachments. A comparison of the distribution of normal email activity versus the activity of the Klez email worm for this feature is given in figure 5 in the Appendix.

The MIME type of files attached to the email, and whether or not the attachment is a binary. Many viruses send executables through email to propagate, while a normal user would have very little reason to do so. These two features seek to leverage this trend.

The UNIX ‘magic number’ type of the files attached to the email. Often, files have been renamed by the virus to mask their MIME type when passing through filtering at the endpoint. This feature attempts to discern the true file type by taking advan-

tage of UNIX file-handling semantics. Again, a normal user would have very little reason to try and mix file extensions; an example of a virus that does this is the Anna Kournikova worm.

4.1.2 Per-User Features

We next describe the numerical values calculated on a per-user basis, along with a brief justification for each. These features were calculated over a per-user sending window of email, typically consisting of the user’s last twenty messages. Referenced figures are included in the Appendix.

The number of different email to addresses sent to over the sending window. Most worms seek to spread themselves as quickly as possible, leading to email sent to a large set of unique addresses within a short amount of time. A comparison of this feature between the LoveLetter.C worm and normal user behavior is given in figure 6 in the Appendix.

The frequency with which emails were sent within the sending window. Again, many worms seek to propagate as quickly as possible, often using their own SMTP engines. MyDoom, Nimbda, and Klez are examples of recent viruses that exhibit this behavior. A comparison of Klez worm activity with normal behavior is given in figure 7.

The average number of words per email and the average word length among emails over the sending window. Although many recent email worms are polymorphic, we found these features to be relatively similar among the worms we analyzed. An example distribution comparison between normal behavior and the Bagle.A email worm for the average number of words per email is given in figure 8 in the Appendix. A further example comparing the distribution of the average word length against the Klez worm is shown in figure 9.

The average number of characters per subject among emails in the sending window. Again, subjects and bodies of emails sent by worms—even polymorphic worms—tend to have certain similarities that differ over time from normal email behavior. A comparison of this feature between the Bagle.A worm and normal user behavior is given in figure 10.

The ratio of emails with attachments to those without over the sending window. Most worms propagate via attachments, whereas we found it to be statistically unlikely that normal users send large amounts of attachments at once. The differences in the distribution between the LoveLetter.C worm and normal behavior for this feature can be seen in feature 11 in the Appendix.

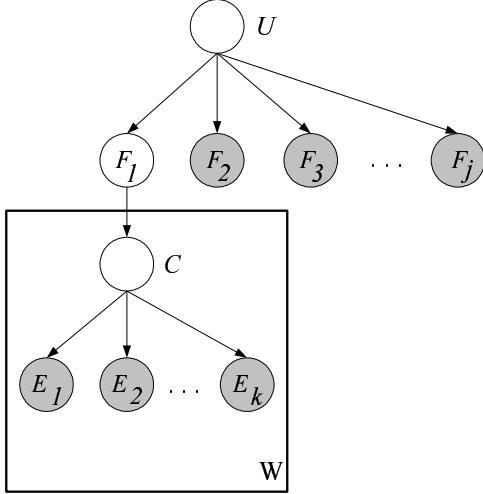


Figure 2. The graphical model representing our parametric classifier.

5 The Parametric Classifier

Our parametric classifier filters the unsupervised classifications from the novelty detector using a technique similar to naïve Bayes. As explained in the previous section, there is significant disparity between feature distributions of normal and infected email traffic. Our model exploits this disparity by maintaining separate parameter sets for feature distributions on virus and non-virus classes.

Figure 2 shows a generative graphical model for the parametric classifier. Each node in the graph corresponds to a random variable. Nodes E_1, \dots, E_k represent feature nodes extracted from individual emails, whereas nodes F_1, \dots, F_j represent features evaluated over flows of emails with a fixed window size W . Shaded nodes symbolize observed values in the dataset, whereas partly shaded nodes symbolize partially observed values. Nodes U and C are latent indicator nodes for user and an email being infected respectively. The values of these nodes are deduced by the classifier based on its training. These nodes are partially observed (as opposed to unobserved) since the classifier gets periodically retrained on the corrective actions of a supervisor. Since user behavior is expected to differ among users, every user has a separate instance of the model.

The absence of an arrow between two nodes in the graphical model is indicative of independence between those nodes. Figure 2 shows dependence of node C on the per-email-based features and node U on the user-based based features. The graphical model for the classifier is essentially a result of combining two sub-models that are based on inherently different features. The per-user features are temporally ordered

since they are based on email flows over time. Per-email features on the other hand possess no such ordering. Feature F_j shown in the model above glues the two sub-models together into a collective model. This is done by aggregating classification results from the per-email based sub-model into a user based feature F_j . This feature is then used in the user-based sub-model for user classification. The aggregation of classification results from the per-email sub-model is done by taking an average over the past W email classifications. Alternatively, an exponentially decaying average can be taken to give more weight to current classification than prior classifications. Since email worms tend to send streams of email viruses continuously, this feature subtly captures the dependency between the current and preceding classifications.

5.2 Classifier Operation

The classifier is trained initially on bootstrap virus and non-virus data which is completely labeled. After initial training, the classifier categorizes emails filtered through the novelty detector as viral or non-viral. As the supervisor takes corrective actions on misclassified emails, the email corpus is updated with these labels. This partially labeled data is used to retrain the model periodically. Constant retraining of the classifier improves accuracy over time.

The next three sections describe this process in detail in a manner similar to the presentation in [22].

5.2.1 Initial Training

Training the model involves a given amount of bootstrap data on both normal email and worm activity. Our experiments presented in section 6 use records of sent email with artificially injected streams of known worm messages to train the classifier on virus and non-virus email.

We begin by considering the training of the per-email sub-model. Let the indicator random variable C in the graphical model be associated with a Bernoulli distribution with parameter π . Let $E = \{E_1, \dots, E_k\}$ be the set of email-based features. When classifying an email with a given set of feature values, the classifier needs to calculate the posterior probability of that email being a virus given its feature values. Using Bayes rule, we can express this probability as:

$$\frac{P(C=1 | E, \theta_1) \times P(C=1)}{P(E | \theta)} \quad (1.1)$$

where θ_1 is the set of parameter values for all distributions of per-email feature components for virus email.

Since we assume features to be independent given the class to which the email belongs, we can rewrite the above equation as

$$\begin{aligned}
P(C = 1 | E, \theta_1) &= \\
\frac{\prod_{i=1}^k P(E_i | C = 1, \theta_1) \times P(C = 1)}{P(E | \theta)} &= \\
\frac{\pi \prod_{i=1}^k P(E_i | C = 1, \theta_1)}{(1 - \pi) \prod_{i=1}^k P(E_i | C = 0, \theta_0) + \pi \prod_{i=1}^k P(E_i | C = 1, \theta_1)} & \quad (1.2)
\end{aligned}$$

The posterior probability for the non-virus class can be calculated similarly. To be able to calculate the posterior probabilities for new data points in the classification stage, the set for parameter values θ for virus and non-virus feature distributions need to be calculated in the training phase. We use Maximum Likelihood Estimation (MLE) on the training data to infer these parameters and the prior probability π .

Assuming a total of N training points and representing observations of random variables with lowercase letters, the maximum likelihood estimate for the prior is the sample proportion, given by:

$$\hat{\pi}_{ML} = \frac{\sum_{n=1}^N c_n}{N} \quad (1.3)$$

For normally distributed features, the maximum likelihood estimate for the mean and variance parameters is the sample mean and sample variance respectively. For the j^{th} feature component, the sample mean for the virus class is given by:

$$\hat{\mu}_{j1,ML} = \frac{\sum_{n=1}^N c_n x_{j,n}}{\sum_{n=1}^N c_n} \quad (1.4)$$

and the sample variance is given by:

$$\hat{\sigma}_{j1,ML} = \frac{\sum_{n=1}^N c_n (x_{j,n} - \hat{\mu}_{j1,ML})^2}{\sum_{n=1}^N c_n} \quad (1.5)$$

where $x_{j,n}$ is the n^{th} observation of the of the j^{th} feature component.

For binary values features, the probability of the j^{th} Bernoulli feature being set for the virus class is a weighted sample proportion given by:

$$\hat{\eta}_{j1,ML} = \frac{\sum_{n=1}^N c_n x_{j,n}}{\sum_{n=1}^N c_n} \quad (1.6)$$

For exponentially distributed features, the maximum likelihood estimate of the scale parameter is the inverse of the sample mean.

Once all the per-email parameters for both classes have been estimated, the per-email-based training classifications are aggregated into a single training feature for the user-based model. The parameters for the user-based model can then be calculated in precisely the same manner as done for the per-email-based sub-model.

5.2.2 Classification

Once maximum likelihood parameters have been computed in the training phase, a new data point during classification is assigned to the class for which the posterior probability is maximized. The per-email feature values are first used to compute classification results for the per-email model. The classification results, together with the past W email classifications for the user are aggregated to get the value of the latent feature F_j . This value, along with values for other user-based features are then used to compute posterior probabilities for the user-based model. Finally, the user is assigned to the class for which the posterior is maximized.

5.2.3 Semi-Supervised Retraining

After a fixed number of emails have been classified, any corrections from the supervisor are incorporated in the training email corpus, and the model is retrained on partially labeled data. In the current implementation of the model, we retrain over all emails seen in the past for a particular user. In a deployable version of this system, however, this data would be limited by the buffering capabilities of the system.

We use the Expectation Maximization (EM) algorithm for updating our model parameters on partially labeled data. Learning on parameters during retraining is semi-supervised since only some of the training points are labeled by the supervisor. Each iteration of EM involves an expectation step and a corresponding maximization step. In the expectation step (E-Step), the class expectations conditioned on the training data are calculated for each training point based on the current estimate of θ . The maximization step (M-Step) maximizes the likelihood of the parameter set θ based on the expected class probabilities in the E-Step. Initializing θ with the results of the previous training, this process is repeated until convergence.

Specifically, the conditional expectation for the i^{th} training point being a virus in the i^{th} iteration of the E-Step is given by:

$$\tau_i^{(t)} = P(C=1|x_i, \theta_1) \quad (1.7)$$

which can be computed using equation (1.2).

Using the conditional expectations for all data points, the update equations for parameter estimates for the virus class in the M-Step are given by the following equations:

$$\pi^{(t+1)} = \frac{\sum_{n=1}^N \tau_n^{(t)}}{N} \quad (1.8)$$

$$\mu_j^{(t+1)} = \frac{\sum_{n=1}^N \tau_n^{(t)} x_{j,n}}{\sum_{n=1}^N \tau_n^{(t)}} \quad (1.9)$$

$$\sigma_j^{(t+1)} = \frac{\sum_{n=1}^N \tau_n^{(t)} (x_{j,n} - \mu_j^{(t+1)})^2}{\sum_{n=1}^N \tau_n^{(t)}} \quad (1.10)$$

$$\eta_j^{(t+1)} = \frac{\sum_{n=1}^N \tau_n^{(t)} x_{j,n}}{\sum_{n=1}^N \tau_n^{(t)}} \quad (1.11)$$

5.3 Model Selection

Having explained the generative model used for classification, we discuss the rationale behind the choice of this model for our problem. As explained in section 4, there is significant disparity between feature distributions of virus and non-virus email data. Hence, it would make sense to use a model that would use this divergence property inherent in the data to distinguish between virus and non-virus data. Generative models explicitly exploit this disparity by maintaining separate parameter sets for feature distributions of virus and non-virus classes. These features are then used in the calculation of the posterior probability for the classes, which determine the final classification decision.

Discriminative models offer an alternative to generative models. With discriminative models, the form of the posterior probability is assumed at the outset, and parameters for the classifier are computed directly. Such models however, are known to approach their asymptotic error much slower than generative models, and hence need significantly more training data [1]. Major concern for space efficiency in our overall architecture makes discriminative models a less attractive candidate than generative models.

6 Results and Discussion

We tested our prototype implementation using virus email samples from three different viruses – Bagle.A, Klez and LoveLetter.C. LoveLetter.C is the only non-polymorphic worm among the three. In addition, LoveLetter.C propagates via Microsoft Outlook, whereas Bagle.A and Klez contain their own SMTP engines. The virus emails used were extracted from real infections as explained in section 4.1. We used data from our live user study for an initial estimate of whether each feature best fit a normal, exponential, or binomial distribution. However, to date we have not been able to gather enough live data for extensive online testing. Instead, for these preliminary numbers we used sent email traces from several people.

The training set was constructed by injecting randomly sampled worm emails into a trace of normal email activity assuming a fixed propagation rate. Two of the three worms were placed into the training set, while the third was inserted into the similarly constructed test set. Three training and test sets were created for each of the three viruses. A sending frequency of 4 emails per minute was assumed for LoveLetter.C, and 8 emails per minute was assumed for Klez and Bagle.A. The sending frequency in reality would depend on factors like network connection and firewall settings at the host being infected. We assumed our sending frequency keeping in mind the time it takes for a machine to make an SMTP connection and send an email on high speed network.

Our prototype assumes that the novelty detector manages to detect all viruses at the cost of having a 50% false positive rate. As a result, all virus emails along with 50% randomly sampled from normal emails are assumed to filter into the parametric classifier. Our current implementation of the classifier does not make use of how the novelty detector labels the incoming emails.

The classifier is currently retrained after every batch of 10 classifications in our prototype. All emails that filter through the classifier are assumed to be labeled by the supervisor in the subsequent retraining. The classifier retrains on all the emails, including those that do not filter through the classifier, in the course of performing semi-supervised learning.

6.1 Discussion of Results

Table 2 shows results of three tests performed on our prototype. The first column shows the name of the virus present in the test set. As demonstrated in the table, the classifier predicted emails with a high accuracy of around 98% in all tests. The false positives stayed low around 1% in each case. For Klez, the false negative rate was around 5%, which is significantly higher than around 2% for LoveLetter.C and Bagel.

Table 2. Parametric filter classification results using semi-supervised learning.

Worm Name	Total Emails	Num. Worm Emails	Num. Clean Emails	False Positives	False Negatives	Correctly Classified
Bagle.A	1090	789	301	6 (0.76%)	6 (1.99%)	1078 (98.90%)
Klez	1090	789	301	4 (0.50%)	15 (4.98%)	1071 (98.26%)
LoveLetter.C	1090	787	303	9 (1.14%)	5 (1.65%)	1076 (98.72%)

This difference is plausible, given that Klez was highly polymorphic in comparison to LoveLetter.C and Bagle.A.

The preliminary results shown in table 1 are quite encouraging. However, there are several assumptions in our prototype and test methodology. These assumptions need to be addressed in order to support the validity of our results.

As was mentioned earlier, the virus samples used for training and testing our model might not be representative of true worm traffic. Samples from several diverse viruses would need to be considered to form a population representative of true worm traffic.

In our prototype, we assume a fixed false positive rate of 50% for the novelty detector. It is a generally accepted fact that constraining false negatives in any novelty detection system results in an increased false positive rate. This has been true in our experience with building a novelty detector using Gaussian Mixture Models. However, the assumption of a fixed 50% false positive rate and 0% false negative rate is simply a threshold we chose for the experiments. Perhaps modeling the novelty detector probabilistically as a part of the graphical model would be more realistic.

We performed several additional experiments to validate the idea of doing semi-supervised learning instead of training strictly on the data labeled by human supervisors. Although we do not have enough data to report results on this aspect of our experiments, initial trials show that doing EM over unlabeled as well as learning on labeled data does improve our results. Intuitively, this makes sense; in our deployable system, we would expect the amount of unlabeled data to vastly outnumber the labeled data available.

We chose to retrain our model after every 10 classifications in our experiments. In reality, retraining would likely be done in a completely online fashion. Intuitively, this should have a positive impact on our overall results.

7 Conclusions

This paper presents two contributions. First, we discuss a wide range of features calculated on email traffic that could provide the statistical insight necessary to detect novel worms. In addition, sample distribution plots from real worm data are provided to demonstrate the utility of such features. To the best of the authors’ knowledge, there is no previous work of this type specific to email worm propagation.

Second, using these features, we next present a new method for doing *semi-supervised* learning that leverages system administrator involvement to improve classification. This is done via a two-layer approach that combines the strengths of sensitive novelty detection with a parametric classifier that drastically reduces the false positives. While there has been much previous work in the area of adaptive virus detection, the Achilles heel of such systems has been higher false positive rates resulting from the need to attain extremely low numbers of false negatives. Our solution to this problem uses a generative model that leverages past knowledge to filter the results of any generic novelty detection.

We believe our initial results, while not completely conclusive, show the promise of our approach. It should also be noted that a successful attack against our method could be carried out by a novel virus that is able to propagate in a manner that precisely mimics normal email behavior. However, while our classifier might be ‘fooled’, at the same time the overall goal of our system will be at least partially achieved—namely, the infection capability of the worm will be throttled.

8 Future Work

There are several areas where the methods presented in this paper could be improved. The following section presents a few of our thoughts on how to extend this work.

The graphical model presented in section 5 assumes independence among feature components within virus and non-virus classes. This might not necessarily be true. For example, consider the feature

‘HTML in email’ and ‘Scripts in email’. Presence of HTML in an email probably makes it more likely for Javascript or other scripts to be present in the email. Our model however assumes no such correlation. This problem can be addressed by enhancing the model through additional dependencies among features. Alternatively, some form of discriminative model (e.g. logistic regression) could be used for classification since these models assume no prior distributions or dependencies among features.

In addition to independence among features, our model also assumes independence among users by modeling each user separately. In practice, however, infection of a given user can increase the chances of other users within the same network being infected. We believe the collective accuracy of the model could improve if virus data is shared among individual user models.

All user-based features in the user model are evaluated using a fixed window size in the current implementation. For better results, window sizes can be adaptively adjusted so that user based features optimally reflect user behavior.

There are several features we did not consider in our model that can further improve the accuracy of our model. Specifically, these are the variance of words in the body of emails, the variance of characters in the subject, and the variance in attachment sizes. Variance-based features have been seen in previous literature. Since a worm sends out virus email based on a fixed subset of samples, we can expect these variance based features to differ from normal email traffic. Furthermore, we have implemented features that use ideas from Natural Language Processing (e.g. a TFIDF based feature to classify emails based on word frequencies). Such features have proved to be very successful in spam filtering systems, but have yet to be incorporated into our system.

There is a wide variety of models that can be used in place of the relatively simple generative model described in this paper. As mentioned before, discriminative models are a possible alternative. Non-parametric models like decision trees can be used as well. A comparative study based on these models would be a significant step towards designing a better virus classification system.

Finally, more work needs to be done to justify our distribution assignments for each feature. Although our initial guesses based on observations appear to work well, it would be nice to make a more thorough study.

9 References

- [1] Andrew Y. Ng, Michael I. Jordan. On Discriminative vs. Generative Classifiers: A comparison of logistic regression and Naive Bayes. In *NIPS 14*, 2002.
- [2] Antone, Gonsalves. Mydoom Shows Vulnerability of The Web. TechWeb, <http://www.techweb.com/wire/story/TWB20040202S0011>
- [3] Daniel F. DeLong. Code Red Virus ‘Most Expensive in History of Internet.’ NewsFactor, <http://www.newsfactor.com/perl/story/12668.html>
- [4] G. Tesauro, J. Kephart, G. Sorkin. Neural Networks for Computer Virus Detection. *IEEE Expert*, 11(4) pgs 5-6, 1996.
- [5] Jamie Twycross and Matthew M Williamson. Implementing and testing a virus throttle. In *Proceedings of the 12th USENIX Security Symposium*, pages 285--294. USENIX, August 2003.
- [6] Kolter, J.Z., & Maloof, M.A. (2004). Learning to detect malicious executables in the wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 470-478. New York, NY: ACM Press. (Best Application Paper)
- [7] M. Newman, S. Forrest and J. Balthrop. Email Networks and the Spread of Computer Viruses. *Physical Review E* 66, 035101. (2002)
- [8] M. Williamson. Design, Implementation and Test of an Email Virus Throttle. *HP Labs Tech Report HPL-2003-118 20030626*, 2003.
- [9] Matthew G. Schultz, Eleazar Eskin, Salvatore J. Stolfo. MEF: Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables. To appear in *USENIX Annual Technical Conference - FREENIX Track*, Boston, MA, June 2001
- [10] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, Salvatore J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. To appear in *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001
- [11] MIMEDefang Homepage. <http://www.mimedefang.org/>
- [12] Snort Homepage. www.snort.org.
- [13] P. Oscar Boykin, Vwani Roychowdhury. Per-

sonal Email Networks: An Effective Anti-Spam Tool. <http://arXiv.org/abs/cond-mat/0402143>

[15] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998*.

[16] Salvatore J. Stolfo, Wei-Jen Li, Shlomo Hershkop, Ke Wang, Chia-Wei Hu, Olivier Nimeskern. Detecting Viral Propagations Using Email Behavior Profiles. *ACM TOIT 2004*.

[17] Salvatore J. Stolfo, Shlomo Hershkop, Ke Wang, Olivier Nimeskern and Chia-Wei Hu. A Behavior-based Approach to Securing Email Systems. *"Mathematical Methods, Models and Architectures for Computer Networks Security"*, Proceedings published by Springer Verlag, Sept. 2003.

[18] Sendmail Home Page. <http://www.sendmail.org/>

[19] S. Staniford, D. Moore, V. Paxson and N. Weaver, The Top Speed of Flash Worms, Proc. ACM CCS WORM, October 2004.

[20] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage, Automated Worm Fingerprinting, *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, December 2004.

[21] T. Liston. Welcome to My Tarpit: The Tactical and Stategic Use of LaBrea. *Dshield.org WhitePaper, 2001*.

[22] Jordan, M. An Introduction to Probabilistic Graphical Models. *Unpublished, 2004*.

Appendix A: Feature Distributions

All histogram plots in this section were generated using random samples of roughly five hundred worm emails and complete traces of a single user's normal email activity. The red bars indicate the worm data, while the green indicates normal email.

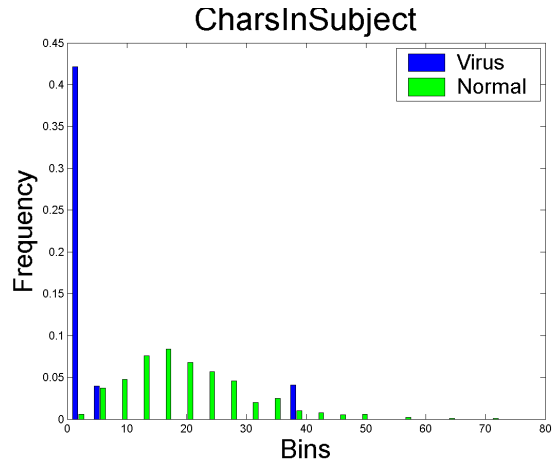


Figure 3. Distributions of the number of characters in the subject of Bagle.A worm and normal email activity.

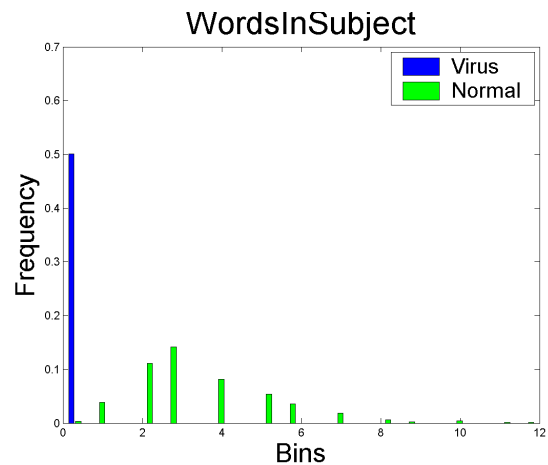


Figure 4. Distributions of the number of words in the subject of Loveletter.C worm and normal email activity.

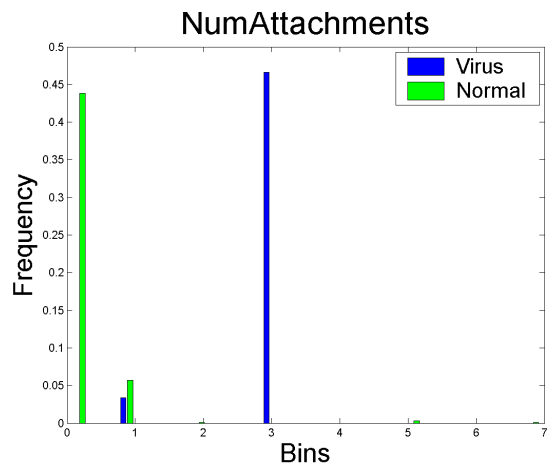


Figure 5. Distributions of the number of attachments in Klez worm and normal email activity.

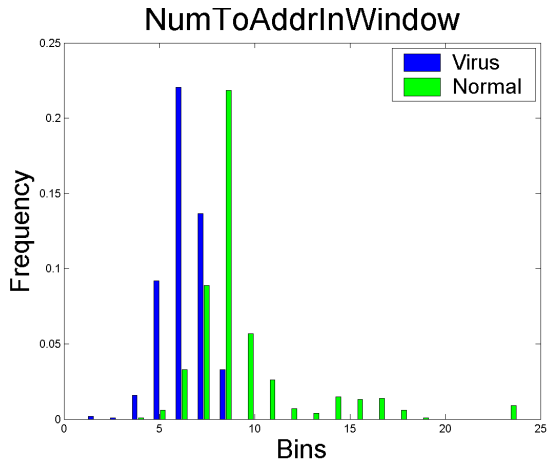


Figure 6. Distributions of the number of address sent to over time in Loveletter.C worm and normal email activity.

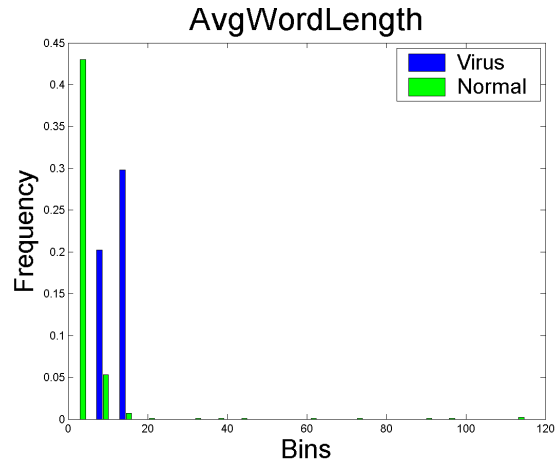


Figure 9. Distributions of the average word length per email in Klez worm and normal email activity.

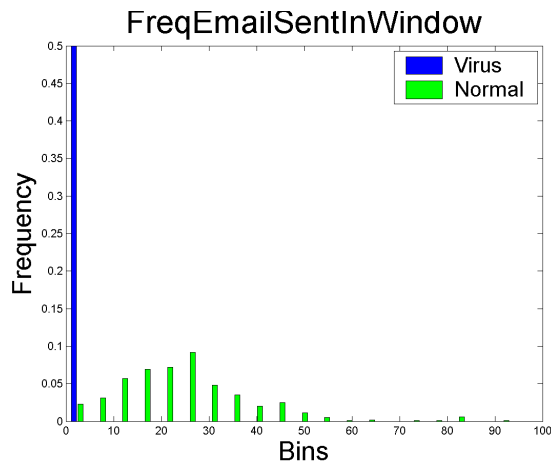


Figure 7. Distributions of the amount of time between sent emails in Klez worm and normal email activity.

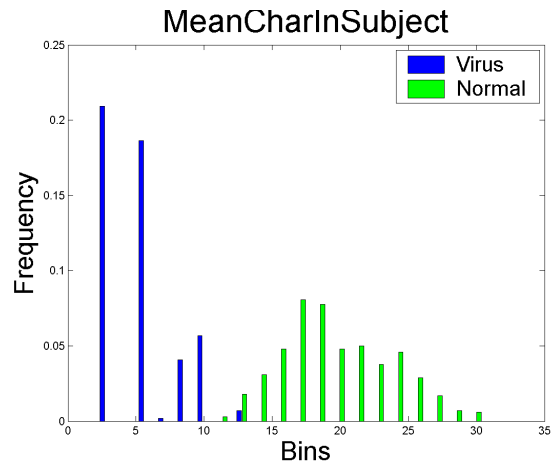


Figure 10. Distributions of the average number of characters in an email subject in Bagle.A worm vs. normal email activity.

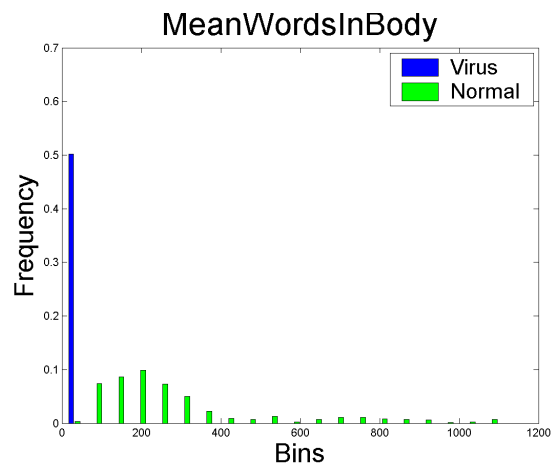


Figure 8. Distributions of the average number of words per email in Bagle.A worm vs. normal email activity.

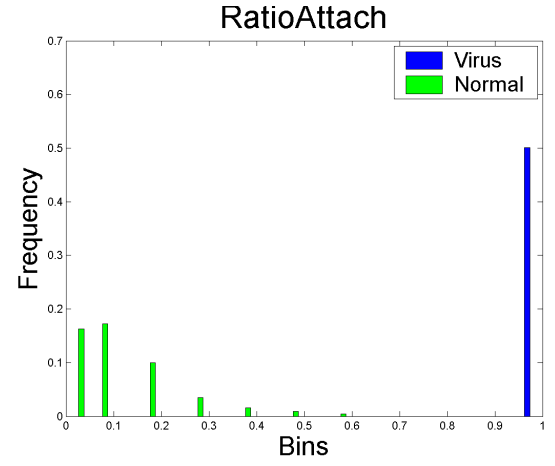


Figure 11. Distributions of the ratio of attachments between Loveletter.C worm and normal email activity.