

# 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.

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. 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 thirty seconds. [6]

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. 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 purposefully configured to be overly sensitive, thus generating few false negatives, but also excessive amounts of false positives.

To this end, we present the following contributions.

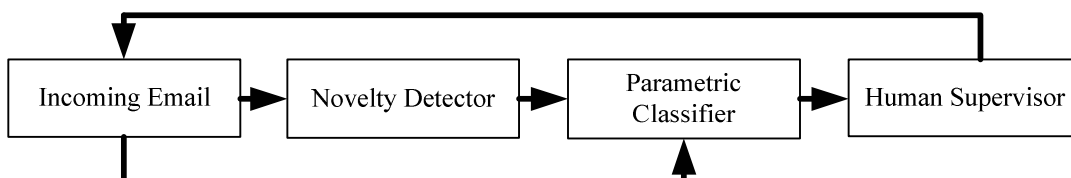
First, we examine a wide range of features calculated on email traffic to determine what indicators best separate virus from normal email behavior. Using these features, we next present a new method for doing *semi-supervised* learning that leverages system administrator feedback to improve classification for the elimination of excess false positives.

This paper presents the results of our work. Section 2 gives some previous research, 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

Early attempts at using machine learning models for detection of viruses used Artificial Neural Networks for detecting boot sector viruses [2]. More recently, Schultz et al used a naïve Bayes classifier trained over consecutive byte sequences in email attachments to detect malicious executables. [4] They later extended their work to include printable strings in executables as another feature to train their models [5]. 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) [3]. The techniques mentioned above do not consider emails that propagate via HTML, such as MyDoom, and can also be easily fooled by using standard code obfuscation techniques.

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 [10, 11]. The authors used social network analysis and considered features such as variance in number of distinct recipients, sending rate of email and the number of emails with attachments over a window of emails. The work done here is vulnerable to several attacks, some which are described in [8,9].



### 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 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 time.

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. For the purpose of this paper, however, the novelty detector is unimportant; we model this module as a high static probability of false positives.

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 true false positives. The model is retrained on batches of new data on an ongoing basis for increasing accuracy among changing network conditions.

Finally, email behavior data points that have been determined as abnormal by the novelty detector and as infected by the parametric classifier are passed on for human inspection. The reaction to each point is then used to update the data corpus with labels, allowing the parametric classifier to be retrained in a semi-supervised manner for increased accuracy.

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 uses Sendmail to interface with 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.

To analyze feature distributions on normal and infected emails, we enlisted twenty volunteers within the department to send their email using a feature collection framework we implemented. Real virus data was collected using this framework, Microsoft Virtual PC 2004, and one of the author's address books containing three hundred and twenty addresses. In gathering worm data, the Windows 98 and XP operating systems were used.

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.

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

sending window.

The following sections discuss each feature briefly. Section 4.1 will explain the per-email features, while section 4.2 will give an overview of the per-user features. Referenced figures are included in the Appendix.

#### 4.1 Per-Email Features

*Whether or not there is HTML embedded in the email.* Many viruses send emails with HTML in them, while several users, including the authors, prefer plaintext messages. Examples are the Klez worm, and most macro viruses that use Microsoft Outlook. Figure 3 in the Appendix shows an example distribution plot comparing a single user's activity versus the Klez email worm.

*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 attribute scripts in other 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 4 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 5.

*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 a user 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, while we found that most legitimate email does not include attachments. A comparison of the distribution of normal email activity versus the activity of the Klez email work for this feature is given in figure 6 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

by taking advantage 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.

#### 4.2 Per-User Features

These features were calculated over a per-user sending window of email, typically consisting of the user’s last twenty messages.

*The number of different email addresses mail was 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, Nimda, 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.

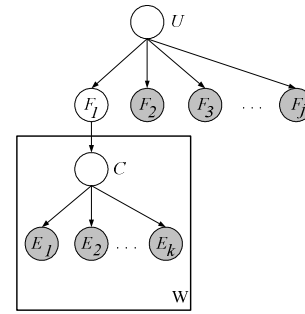
*The average number of words per email and the average word length among emails over the sending window.* Although many recent email worms are polymorphic, we found in that number of words and the relative grammatical complexity of the bodies of email sent by worms we analyzed to be relatively similar. These features seek to leverage this fact. 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.

*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—tended to have certain similarities that differed over time from normal email behavior. A comparison of this feature between the Bagle.A worm and normal user behavior is given in figure 9.

*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 10 in the Appendix.

### 5 The Parametric Classifier

Our parametric classifier filters the unsupervised classifications from the novelty detector using a two-layer naïve Bayes generative model. 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 features nodes extracted from individual emails, whereas nodes  $F_1, \dots, F_j$  represent features evaluated over flows of emails with a fixed window size  $W$ . 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 not



**Figure 2.** The graphical model representing our parametric classifier.

tially 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.

Figure 2 shows dependence of node  $C$  on the per-email-based features and node  $U$  on the flow-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-flow 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  $F_1$  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 flow based feature  $F_1$ . This feature is then used in the flow-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. 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.

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.

The following description is presented in a manner similar to the material in [7]. We begin by considering the training of the per-email sub-model. Let  $C \sim \text{Bernoulli}(\pi)$  and  $E = \{E_1, \dots, E_k\}$  be the set of email-based features. The classifier needs to calculate the posterior probability of an email being a virus given its feature values. Using Bayes rule, we can express this probability as:

$$P(C = 1 | E, \theta_1) = \frac{P(E | C = 1, \theta_1) \times P(C = 1)}{P(E | C = 1, \theta_1) \times P(C = 1) + P(E | C = 0, \theta_0) \times P(C = 0)} \quad (1)$$

where  $\theta_l$  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. We use Maximum Likelihood Estimation (MLE) on the training data to infer  $\theta$  and the prior  $\pi$ .

Assuming a total of  $N$  training points and representing observations of random variables with lower-case 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  $j^{\text{th}}$  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  $n^{\text{th}}$  observation of the of the  $j^{\text{th}}$  feature component.

For binary values features, the probability of the  $j^{\text{th}}$  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 flow-based model. The parameters for the flow-based model can then be calculated in precisely the same manner as done for

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 flow-based features are then used to compute posterior probabilities for the flow-based model. Finally, the user is assigned to the class for which the posterior is maximized.

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. 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  $\theta$ . The maximization step (M-Step) maximizes the likelihood of the parameter set  $\theta$  based on the expected class probabilities in the E-Step. Initializing  $\theta$  with the results of the previous training, this process is repeated until convergence.

Specifically, the conditional expectation for the  $i^{\text{th}}$  training point being a virus in the  $t^{\text{th}}$  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 signifi-

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 previously. We used data from our live user study for an initial estimate of whether each real-valued feature best fits a normal or exponential 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.

Table 1 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. 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.

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.

**Table 1.** 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	5	1076

## 7 Conclusions

This paper presents two contributions. First, we discuss a wide range of features calculated on email traffic that could provide the statistical granularity necessary to detect novel worms. In addition, sample distribution plots from real worm data are provided to demonstrate the discriminatory effectiveness 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.

We believe our initial results, while not completely conclusive, show the promise of our approach.

## 8 Future Work

There are several areas where the methods presented in this paper could be improved.

- Gather much more live data, and run additional tests.
- Eliminate unrealistic independence assumptions among users and features to increase model accuracy.
- Use an adaptive window size for behavioral features.
- Additional features, including ones leveraging NLP techniques.
- Experiment with other non-parametric models in our filter layer.

## 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] G. Tesauro, J. Kephart, G. Sorkin. Neural Networks for Computer Virus Detection. *IEEE Expert*, 11(4) pgs 5-6, 1996.
- [3] Koller, 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)
- [4] 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
- [5] 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
- [6] S. Staniford, D. Moore, V. Paxson and N. Weaver, The Top Speed of Flash Worms, Proc. ACM CCS WORM, October 2004.
- [7] Jordan, M. An Introduction to Probabilistic Graphical Models. *Unpublished*, 2004.
- [8] P. Oscar Boykin, Vwani Roychowdhury. Personal Email Networks: An

[9] M. Newman, S. Forrest and J. Balthrop. Email Networks and the Spread of Computer Viruses. *Physical Review E* 66, 035101. (2002)

[10] 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*.

[11] 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.

## 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.

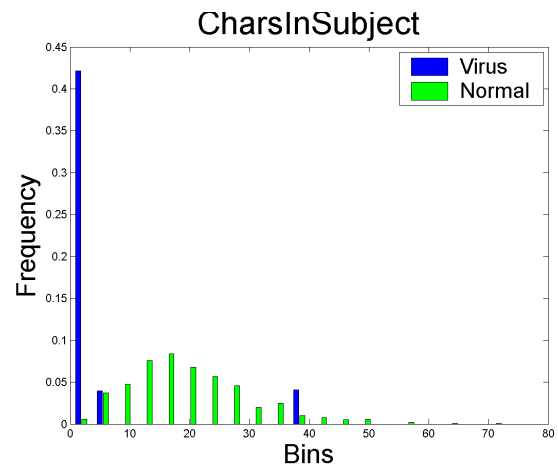


Figure 3. Distributions of the number of characters in the subject of Bagle.A worm and normal email activity.

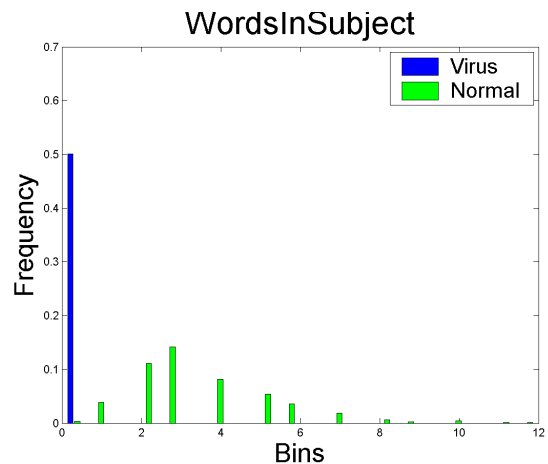
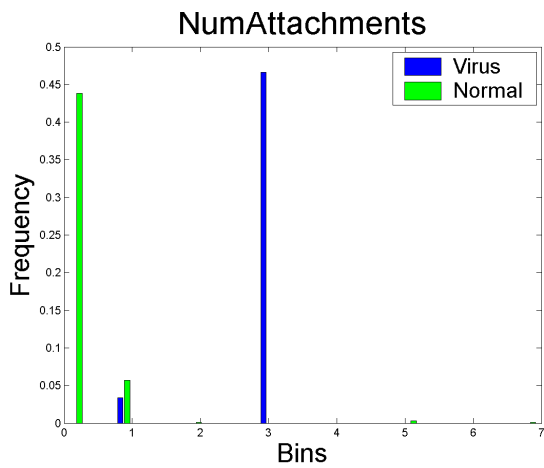
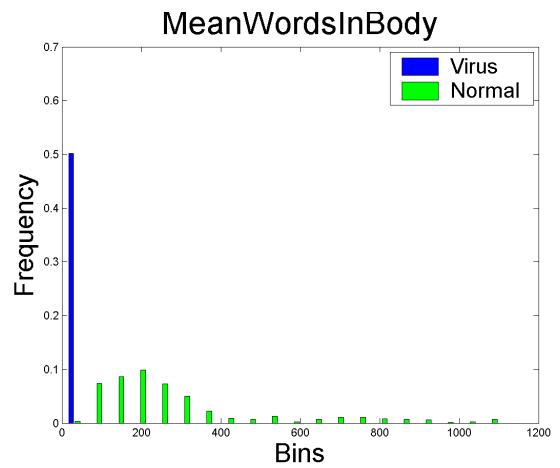


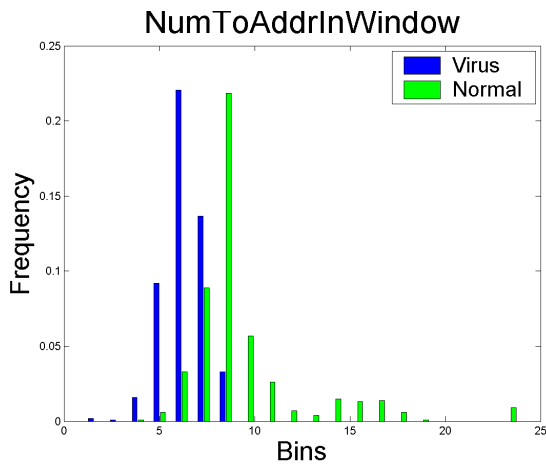
Figure 4. Distributions of the number of words in the subject of Loveletter.C worm and normal email activity.



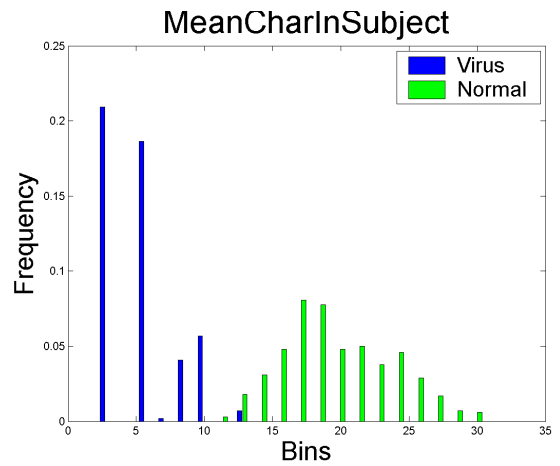
**Figure 5.** Distributions of the number of attachments in Klez worm and normal email activity.



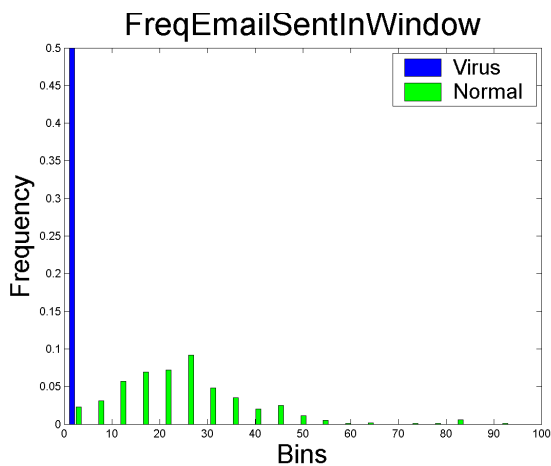
**Figure 8.** Distributions of the average number of words per email in Bagle.A worm vs. normal email activity.



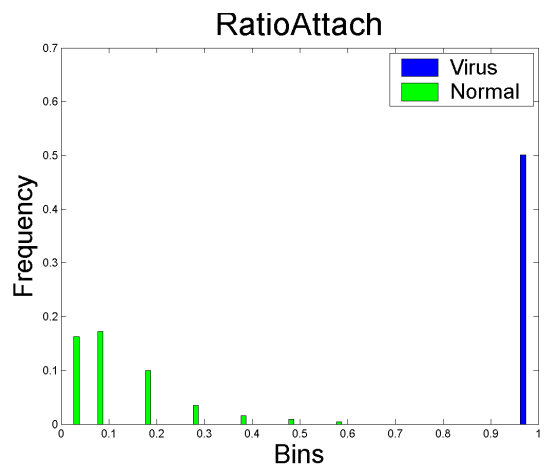
**Figure 6.** Distributions of the number of address sent to over time in Loveletter.C worm and normal email activity.



**Figure 9.** Distributions of the average number of characters in an email subject in Bagle.A worm vs. 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 ratio of attachments between Loveletter.C worm and normal email activity.