One Class Model Training for Zero-Day Virus Detection

Wei-Jen Li, Salvatore J. Stolfo
Computer Science Department, Columbia University
{ weijen, sal }@cs.columbia.edu

Abstract

The big ideas to write about in this introductory section is a) this is an improvement over prior MEF version that was based upon training models to detect new viruses using examples of known viruses and known benign attachments. This is hard to do in practice since one needs good examples of benign attachments that may be hard to sample, and also expensive in practice and b) this version trains models ONLY using known viral attachments.
The methodology is to sample known viruses provided by volunteers (who have captured many examples of viruses over the last year), training in the order in which these viruses were captured using a one-class learning method, and then testing each model on later viruses to determine their accuracy. As a new virus is detected, it is added to the training corpus to determine the accuracy of the updated model on future unknown viruses. This fits the method that the proposed system would be used in practice. Furthermore, two modeling approaches are evaluated, one based upon naïve bayes classifier applied to features extracted from the contents of the attachments, and the other based upon Mahalanobis distance applied to the distribution of features extracted from the contents of the attachments. The results indicate surprisingly accurate detection performance, by both modeling methods, with Naïve Bayes classifier performing slightly better.

1. Introduction

A virus is defined to be a self propagating program that performs an unauthorized malicious act compromising a system's security, damaging a system or obtaining sensitive information without the users' permission. Recently, there have been many new viruses that propagated via email, such as Sobig.F and Bugbear. These malicious attachments caused significant damage in a short time.

The standard approach to detecting viruses is to use *signature based* scanners that are frequently upgraded with new signatures. For each known malicious attachment, the scanner contains a unique byte sequence that identifies the characteristic strings or byte sequences of each known virus. Such byte sequences serve as the virus signature. This kind of approach can detect viruses with one hundred percent accuracy. However, any unknown malicious binary will be undetected. Naturally, it is the new viral attachments frequently released in the wild that has damaged so many systems. Zero-day virus detection remains the holy grail.

In this paper we extend our prior work on virus detection [] by considering new methods to automatically learn the characteristic features that distinguish malicious from benign email attachments. The *Malicious Email Filter* (MEF) first reported in [] was designed to detect new viral attachments not detectable by up to date signature detectors, nor by detecting propagating emails. Rather, MEF was designed to classify attachments by analyzing their content. This analysis consists of training a classifier on known viruses, in the hope of computing a classifier that generalizes to new never before seen viral attachments, thus capable of detecting zero day viruses. The supervised method trained classifiers on examples of known viruses and known benign attachments.

In our subsequent work we developed the Email Mining Toolkit (EMT) that provides a large array of behavior-analyses applied to email logs. EMT indeed has been designed for a variety of forensic analysis and detection tasks. The core idea of EMT is to detect errant misuses of email (such as a viral propagation) by detecting abnormal behavior such as changing email flows that are not characteristic of typical user email behavior. In the case of viruses, EMT can effectively detect viral propagations (see [] for a full treatment of this topic). Unfortunately, this is detected after the virus has successfully penetrated a victim and launched its payload and propagation strategy. In our work on MEF, we intend to detect the first appearance of the viral attachment prior to its execution.

MEF was designed as a component of EMT. It focuses on training and testing the binary of email attachments. It extracts content features of a set of known attachments. The features are then used to compose a set of training data for a supervised learning program that computes a classifier. Each newly identified attachment flowing into an email account would first be tested by a previously

learned classifier, and if the likelihood of "malicious" were deemed high enough, the attachment would be so labeled. Instead of searching for the signature, we used two mathematical approaches to cluster email attachments. The first one uses Naïve Bayes to compute the likelihood between normal and malicious byte sequences. On the other hand, the second one uses Mahalanobis Distance to compare the "distance" between two classes of files. Hence, the detections are flexible and they are possible to detect similar malicious binary.

The rest of this paper is organized as follows. Section 2 discusses related work in network intrusion detection. Section 3 describes the methodology we used to the detecting task. Section 4 demonstrates the experiments. Section 5 introduces another approach and it's experiment, while section 6 concluded the paper.

2. Related Work

Intrusion detection can be classified into two approaches: misuse detection and anomaly detection []. Misuse detection uses known patterns to build signatures or rules and to identify intrusions. In anomaly detection, the intrusion is assumed to be unknown, but the intrusion behavior will be different from normal systems. The signature based detection always has low false positive. However, the drawback is that it can't detect or predict new attack. On the other hand, anomaly detection is more flexible to catch similar attack, but usually suffers high false positive rates.

Anomaly detection systems were first proposed by Denning [] for intrusion detection, and later implemented in NIDES [] to model normal network behavior in order to detect deviant behavior that may correspond to an attack against a network computer system. W. Lee et al. describe a framework and system for auditing and data mining and feature selection for intrusion detection. This framework consists of classification, link analysis and sequence analysis for constructing intrusion detection models. []

A variety of other work has appeared in the literature detailing alternative algorithms to establish normal profiles, applied to a variety of different audit sources, some specific to user commands for masquerade detection [], others specific to network protocols and LAN traffic for detecting denial of service attacks [] or Trojan execution, or application or system call-level data for malware detection [], to name a few. A variety of different modeling approaches have been described in the literature to compute baseline profiles. These include probabilistic or statistical distributions over temporal data [], supervised machine learning [] and unsupervised clusterbased algorithms []. Some approaches consider the correlation of multiple models [].

In this paper, we focus on the anomaly detection, which analyzes the binary content of email attachments. This work is an extension of our earlier work on the Malicious Email filter (MEF) []. It classified normal and malicious binary using Naïve Bayes algorithm. It achieved a good result – high detection rate and low false positive.

However, the previous experiment had three problems. First it used two classes analysis. MEF trained both normal and malicious files to two classes, tested the unknown file against both classes and clustered it to the closer one. The problem was that there was no way to gather and train all of the normal files and the model would be huge. Second, the previous MEF didn't consider the order of time. In real world, it is meaningless if we already know a virus and test the data before it. The whole testing data should appear after the training data. The last problem is that a virus is a tiny piece of binary and is always hidden in the normal files. The previous experiment only trained and tested the viral binary, not the entire virus file. Most of the binary of a viral file is not "viral". To test the viruses that include fake part is much harder than to test only the viral binary.

Our current research was to correct these three problems. First, we developed a one-class detecting algorithm that we only needed to train malicious files. Second, our tests were "time sensitive", which means we selected a specific date, trained viruses before that date, and tested both viruses and normal files after it. Third, instead of the specific signature, we computed all of the byte sequences in a file, which include both normal and viral part. Finally, we used two mathematical approaches to calculate the difference and similarity between normal and malicious files. We first tested some real viruses against normal files to verify the method and evaluate the parameters. Then, we tested the real email data using both methods.

3. Methodology

A viral file is a small piece of code that hides in a normal or faked executable file. Such code is the real body of a virus. A file is a long sequence of bytes and most of them are not viral. The general viral detection looks for some special short byte sequences in a file, such sequences are called signature. However, a signature based detection need to find the signature manually. It takes time, and is impossible to detect an unknown virus. We know that different types of files have different styles of byte distribution. Viruses have the same attribute. They must have something in common and different to other kinds of files. Our learning task is to build a flexible classifier that not only can find specific viruses easily, but also has the ability to find similar viral files. In this

section, we used a machine learning approach, which computed the probability of the byte sequences, to classify normal and anomaly files.

3.1. Feature Extraction

We statically extracted byte sequences from each file. These byte sequences were the features of the learning algorithm. We used multiple byte sequences instead of a single byte since the single bye sequence might not represent the feature well. We extracted the byte sequences using n-gram analysis, where an n-gram is the window size that is also the length of the bytes of a feature. A sliding window with width n is passed over the whole file and the occurrence of each feature is counted. In the experiment section, we tested varies window sizes to find out the best result.

The great majority of the binary of a viral file is harmless, faked and useless. Virus makers made up them to let the files look like normal. Sometimes, a viral file is simpler than a normal executable file. However, some viruses have byte distributions that look like normal files. We can't figure them out easily.

Figure 1 and 2 display the samples of the distribution and the number of occurrence of byte sequences. In these two figures, the window size is 6, which means each column represents the combination of 6 bytes. Note that the *y*-axes in the two figures are logarithmic. Figure 1 demonstrates the distribution of a MS Windows operating system file. In the lower part of this figure, we can see that only a few columns are counted a lot, more than hundreds or thousands of times. More than 90% of bytes occur only once or twice. We can see a long tail. Figure 2 is extracted from a virus and it's similar to figure 1. The tail is longer, and the curve looks like an *L* shape, which is called the Zipf-like distribution.

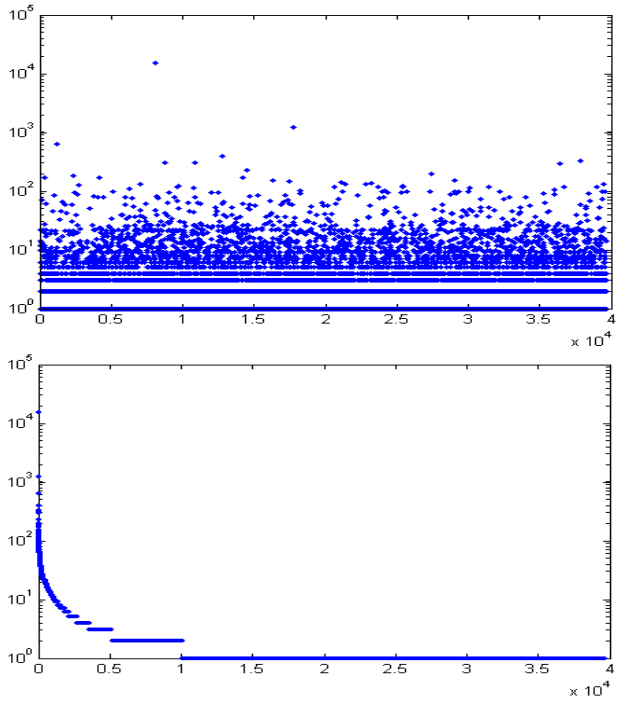

Figure 1: The byte distribution and the number of occurrence of a normal windows executable file (Xconfig32.exe). The lower one is the sorted array.

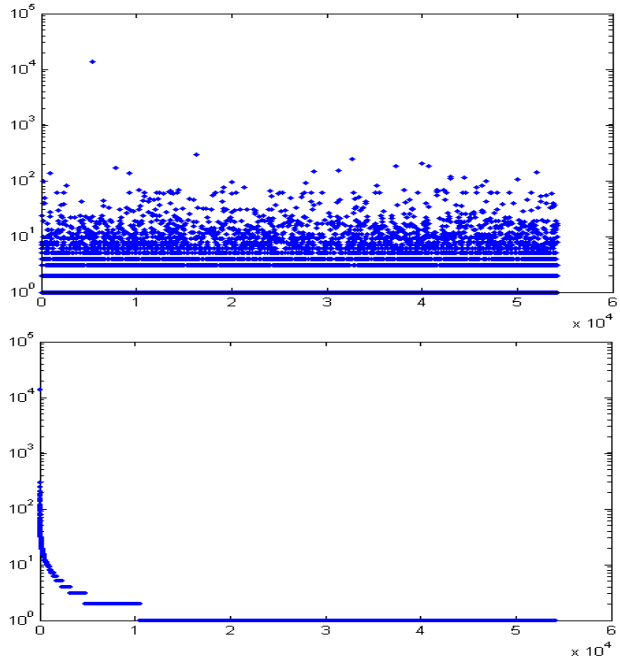

Figure 2: The byte distribution and the number of occurrence of a virus (Swen.A). The lower one is the sorted array.

These byte sequences that occur a lot are, for example, usually the combinations of "0" or "-1", or some other specific bytes. Such kind of sequences usually represents

empty spaces, zeros, lines, etc. They are apparently not the important part of a file. The signature of a file should be the in long tail. However, these non-important byte sequences are the majority part of the likelihood, which make the files hard to be classified (i.e. the likelihoods of normal and malicious files are very close or have overlap).

To increase the detecting accuracy, we manually remove this kind of special situations. For example, if a feature, which is a combination of several bytes, contains only "0" and "-1" (i.e. -1 -1 0 0 0 -1), we skip it.

3.2. One Class Naïve Bayes Classifier

We first collected the probability of all the features in a file, and then, used Naïve Bayes classifier to compute the likelihood that indicated whether a file was more like a malicious one or not.

The features are very sparse in the model. There are 2^{8*w} possible features, where w is the window size, but more than half don't really exist. We have to use a smoothing function to deal with the unseen feature. Which means if there is a feature x_i from a testing file that is not in the training model, we don't have the probability $p(x_i)$, but can't ignore it. We set $p(x_i) = \frac{1}{2N}$, where N is the total number of distinct features in the training model.

The general Naïve Bayes classifier is for multiple classes. A set of data is computed against each class and we say that it belongs to the class that the likelihood is closer. However, in our case, we had only one class, which was the malicious class. The likelihood is normalized, and it is a relative value, not absolute. If there are more features, the sum of the likelihood will be smaller. We can't say which class is closer if there is only one class. We can't set an absolute threshold, either. Instead, we can compute how close a data set to a class. The method we used is simple. First, we trained some viruses to model M, computed the same viruses in M using Naïve Bayes classifier and get a set of likelihood L_{train}. Test an unknown file to M and get another likelihood L_{test}. If L_{test} is close to L_{train} ($threshold \leq |L_{train} - L_{test}|$), this unknown file is a virus. We will discuss more details in next section. Actually, it would be more reasonable to set the threshold be a ratio to the likelihood. We simply set a fixed value here, and it worked well in our experiments.

3.3. Cluster normal and anomaly

Viruses are not always the same, or even similar. If we put all the trained viruses in one single model, the model would grow big and wild. See figure 3 (a), not all the viruses are in one group. Some normal files may sit between viruses. If we built all the viruses in one group, we would have more overlap between two classes. In figure 3 (a), the big circle groups the viruses, but two normal files are inside the group, too. In figure 3 (b), if we built two smaller models for the viruses, we could eliminate the false positive.

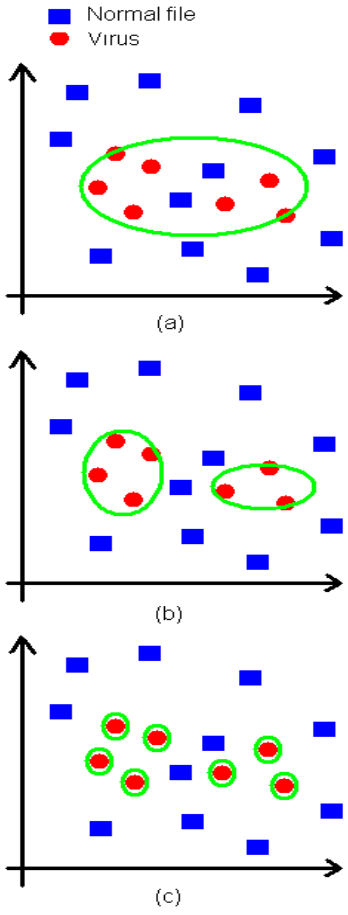

Figure 3: The sample normal and malicious files distribution. Note that the axes are meaningless, only for easier understanding.

Of course, to achieve zero false positive, we can build an individual model for each virus, see figure 3 (c). However, we need more space for the models and it can't detect (predict) unknown viruses, because the threshold is too strict. This way is almost like a signature-based detection.

3.4. Detecting Algorithm

Note that because we used one class classifier and the likelihood is normalized, it is a relative value, not absolute. We had to do the "self-testing" on each model to generate the standard likelihood of each of them. The testing data should be computed using each model and compared to each of their standard likelihood. Having the

previous ideas in mind, we can establish our detecting algorithm. There are three functions – training, testing and self-testing.

Dataset:

Training viruses $V = \{V_1, V_2, ...V_n\}$

Trained models $M = \{M_1, M_2, ...M_m\}$

Testing unknown attachments $U = \{U_1, U_2, ...U_x\}$

Training:

To make sure that the model won't grow too big, we won't add a virus that already exists in a model, and we have to set the constraint of the range of likelihood (maximum and minimum). *maximum - minimum < c * threshold,* where *c* is a constant.

for each training virus V_i
  if size(*M*) = 0
    new_model(V_i)
  for each trained model M_j
    max_like , min_like = self_testing(M_j)
    testing_likelihood = compute_naïve_bayes(V_i, M_j)
    if *testing_likelihood > min_like – threshold*
      and *testing_likelihood < max_like + threshold*
      if V_i not exist in M_j
        and *maximum - minimum < c * threshold*
        append_model(V_i, M_j)//add V_i to model M_j
      else if *maximum - minimum < c * threshold*
        //create a new model that contains V_i
        new_model(V_i)
      else
        //create a new model that contains V_i
        new_model(V_i)
  end
end

Testing:

The threshold here can be different from the threshold in training function. In our experiment, we simply used the same threshold.

for each unknown attachment U_i
  for each trained model M_j
    max_like , *min_like* = self_testing(M_j)
    testing_likelihood = compute_naïve_bayes(U_i, M_j)
    if *testing_likelihood > min_like – threshold*
      and *testing_likelihood < max_like + threshold*
      report_alert(U_i, M_j)
  end
end

Self-testing:

for each exist sample S_i in model *M*
  likelihood = compute_naïve_bayes(S_i, *M*)
  if *likelihood < minimum*
    minimum = likelihood
  if *likelihood > maximum*
    maximum = likelihood
end
return *maximum* and *minimum*

4. Experiment

We did two different tests for the virus clustering. The first test demonstrated the overlap between normal and malicious files. In the second test, we ran the classifier to real email attachments. We trained emails before a specific date and tested emails after it. We used it as a virus filter, which could detect incoming emails and label the anomaly ones.

There were two important parameters - the window size and threshold. First, the window size decided the accuracy of detecting, the size of models and the running time. For example, if the window size was 6, there were 2^{48} possible of features. Although the data were sparse in the experiment, the model size and running time caused a serious problem.

Second, as we discussed in section 3.3, the threshold was the minimum distance between viral likelihood and normal likelihood. It not only decided whether a testing file is normal or malicious, but also decided how we clustered the training models.

We will demonstrate the test of different parameters in the rest of this section.

4.1. Test viruses against normal Microsoft Windows files

In this test, we trained 39 viral files such as Sobig, Bugbear, Mydoom and others, and tested 3738 normal Microsoft Windows files. These normal files included types such as .exe, .pif, .zip, .pdf, .doc, .jpg, .gif and .bmp, and they are gathered from the Microsoft operating systems and public web sites.

This test was to demonstrate the overlap between normal and malicious classes using different parameters. The results are in figure 4. The false positives are the overlap, which means the likelihoods of some normal files are close to or higher than the likelihoods of the trained viruses.

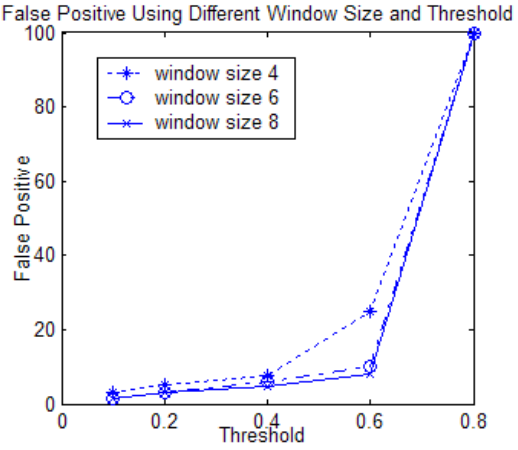

Figure 4: Train virus and test against normal files.

The false positive grew when thresholds grew and window size dropped. A model with bigger window size had more distinct features. It increased the detecting accuracy but not much. In this figure, we can find out that 0.4 is a good threshold, which is currently used in EMT.

4.2. Test viruses against email attachments

The dataset of this test included 13240 emails totally and 340 emails with viral attachment. The emails were collected from April 2003 to October 2003. We trained the viral emails before July 31, 2003, and tested emails after August 1, 2003. Figure 5 and 6 report the detection rate and false positive.

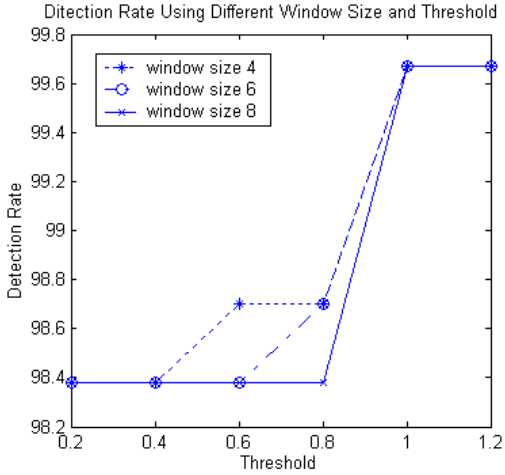

Figure 5: Test of real email attachment. (Detection Rate %)

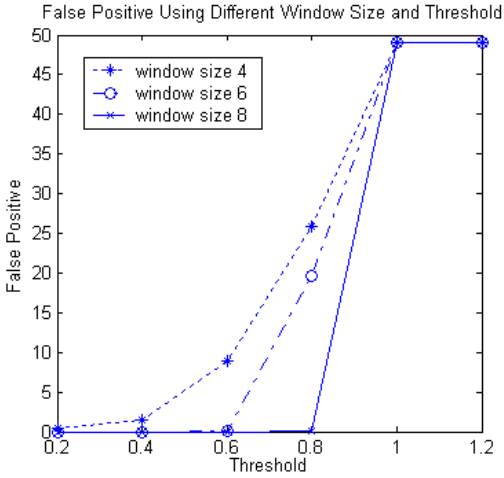

Figure 6: Test of real email attachment. (False Positive %)

Both detection rate and false positive grew with bigger threshold. The idea is straight forward. A bigger threshold represented a bigger tolerance of the "similarity".

On the other hand, the detecting accuracy was better when the window size was bigger. However, as we discussed in previous sections, the model size and running time grew exponentially.

Basically, the results were good. People rarely receive executable files. They usually exchange document, pictures, html files or videos using emails. This is why we could cluster normal and malicious emails easily.

On the other hand, in real world, the same or similar viruses come again and again. In the 340 malicious emails, there were only 22 distinct viral files. Some viruses, such as Sobig.F and Bugbear, propagated themselves a lot in a short period. Although, for example, Sobig.F had many different attachments, they were all similar. This is the reason that we had good detection rate.

This experiment proved that, in real email system, we could build such kind of virus filter. When we see an inbound email virus, we add it to the training models for future detection. If the virus or a similar one appears again, we can detect it without searching for signature.

5. Mahalanobis Distance Analysis

In this section, we tried another method to detect malicious email attachments. The formula we used was a simplified version of Mahalanobis Distance. We used the same method as section 3.1 to extract the features. Instead of considering only the existing features, we considered all of the possible features. We simply marked the unseen feature to 0. Hence, for example, if the window size is 1,

the length of the array is 255. If the window size is 2, the length of the array is 65536.

5.1. Formula

Mahalanobis Distance is a standard distance metric to compare two statistical distributions. It is a very useful way to measure the similarity between two matrices. Here we compute the distance between the byte distributions of the inbound unknown email attachments against the pre-built models. The higher the distance score, the more likely this attachment is normal. The formula for the Mahalanobis Distance is:

$$d^2(x, y) = (x - y)C^{-1}(x - y)$$

where x and y are two feature vectors, and each element of the vector is a variable. x is the feature vector of the new observation, and y is the averaged feature vector computed from the training examples, each of which is a vector (i.e. 1*N matrix). And C^{-1} is the inverse covariance matrix as $C_{ij} = Cov(y_i, y_j)$. y_i and y_j are the *i*th and *j*th elements of the training vector.

The advantage of Mahalanobis Distance is that it takes into account not only the average value but also its variance and the covariance of the variables measured. Instead of simply computing the distance from the mean values, it weights each variable by its standard deviation and covariance, so the computed value gives a statistical measure of how well the new example matches (or is consistent with) the training samples.

In our problem, we use the "naïve" assumption that the bytes are statistically independent. Thus, the covariance matrix C becomes diagonal and the elements along the diagonal are just the variance of each byte. Therefore, we derive the *simplified Mahalanobis Distance*:

$$d(x, y) = \sum_{i=0}^{n-1} (|x_i - y_i| / \sigma_i)$$

For the simplified Mahalanobis Distance, there is the possibility that the standard deviation σ_i equals zero and the distance will become infinite. This will happen when a character or byte value never appears in the training samples or, oddly enough, it appears with exactly the same frequency in each sample. To avoid this situation, we give a smoothing factor α to the standard deviation similar to the prior observation:

$$d(x, y) = \sum_{i=0}^{n-1} (|x_i - y_i| / (\sigma_i + \alpha))$$

The *smoothing factor* α reflects the statistical confidence of the sampled training data. The larger the value of α, the less the confidence the samples are truly representative of the actual distribution, and thus the byte distribution can be more variable. Over time, as more samples are observed in training, α may be decremented automatically.

5.2. Experiment of Mahalanobis Distance

We used the same dataset and the training/testing period in section 4.2. Thus, we could compare and evaluate these two methods easily. We only tested 1-gram for Mahalanobis Distance because it needed a much bigger space for the matrices than the Naïve Bayes approach.

In figure 7, the result was close to Naïve Bayes analysis. Since the window size is 1, the running time was fast and the model was small. However, the best threshold was not a constant. It depended on the email behavior. In other words, the same best threshold for user A was not the best for user B.

Both Naïve Bayes and Mahalanobis Distance analysis had good detection rate, but it's useless to combine them. They had same false negative, which means they miss the same viruses. Both of them analyzed the binary content of files. Hence, if a virus has a totally new style that is not seen before, they'll miss classify.

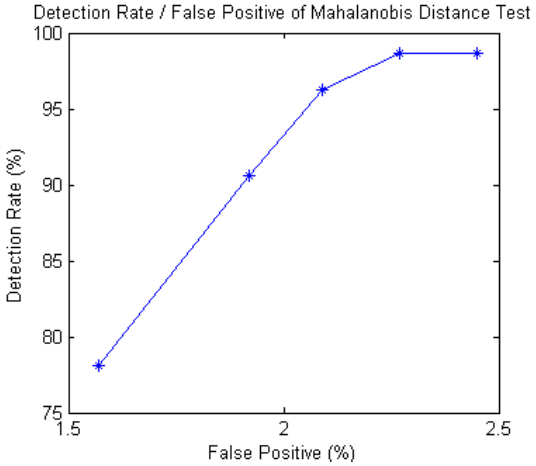

Figure 7: Mahalanobis Distance test of real email data.

6. Conclusion

In this paper we present two approaches to detect malicious email attachment. Both of them have good results. Naïve Bayes approach can achieve a 0% false positive and more than 98% detection rate. Mahalanobis Distance analysis is a little bit worse because we didn't try a bigger window size for it. Using the same window size, Naïve Bayes is faster and smaller than Mahalanobis Distance analysis.

MEF works as a virus filter system. It trains old viruses for future detection. Since we didn't have enough virus samples and enough test data, we couldn't prove that

MEF can perfectly predict new viruses, and we couldn't prove that the false positive is always perfect, either. However, it did detect similar ones. For example, it could catch Sobig.F by using Sobig.B model and vise versa. MEF also demonstrated that it didn't need to train all of the possible normal files. It worked with one class model very well.

In our future work, we will try to develop a faster algorithm, setup the best parameters and add it to EMT. Furthermore, we will also try to combine these two analyses with other models that we have built in EMT, take the advantage of each model and achieve the best result.

References:

[1] Gersho A. and Gray R.M. "Vector quantization and signal compression", Boston, Kluwer 1991

[2] Matthew G. Schultz, Eleazar Eskin, and Salvatore J. Stolfo. "Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables." Proceedings of USENIX Annual Technical Conference - FREENIX Track. Boston, MA: June 2001.

[3] Salvatore J. Stolfo, Wei-Jen Li, Shlomo Hershkop, Ke Wang, Chia-Wei Hu, Olivier Nimeskern. "Detecting Viral Propagations Using Email Behavior Profiles". CU Tech Report 2003.

[4] 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.

[5] Salvatore J. Stolfo, Chia-Wai Hu, Ke Wang, Wei-Jen Li, Shlomo Hershkop, and Olivier Nimeskern. "Combining Behavior Models to Secure Email Systems" CU Tech Report April 2003.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friendman. "The Elements of Statistical Learning, Data Mining Inference, and prediction."

[7] Wei-Jen Li, "MEAF: Malicious Email Attachments Filter, Clustering of Normal and Anomalous Email Attachments", not published, yet.