Available online at www.sciencedirect.com
ScienceDirect
Computational Statistics & Data Analysis 51 (2007) 3156–3172
COMPUTATIONAL STATISTICS & DATA ANALYSIS
www.elsevier.com/locate/csda

Comparison of feature selection and classification algorithms in identifying malicious executables

D. Michael Cai[a,*], Maya Gokhale[b], James Theiler[c]

[a]*Space Data Systems Group, Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
[b]*Advanced Computing Laboratory, Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
[c]*Space and Remote Sensing Sciences Group, Los Alamos National laboratory, Los Alamos, NM 87545, USA*

Received 21 October 2005; received in revised form 12 September 2006; accepted 13 September 2006
Available online 9 October 2006

Abstract

Malicious executables, often spread as email attachments, impose serious security threats to computer systems and associated networks. We investigated the use of byte sequence frequencies as a way to automatically distinguish malicious from benign executables without actually executing them. In a series of experiments, we compared classification accuracies over seven feature selection methods, four classification algorithms, and variable byte sequence lengths. We found that single-byte patterns provided surprisingly reliable features to separate malicious executables from benign. Between classifiers and feature selection methods, the overall performance of the models depended more on the choice of classifier than the method of feature selection. Support vector machine (SVM) classifiers were found to be superior in terms of prediction accuracy, training time, and aversion to overfitting.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Feature selection; Support vector machine; Naïve Bayes classifier; Malicious executable file; Email filtering

1. Introduction

Exponential growth and development of the Internet has created unprecedented opportunities to access and share information. From confidential business operations to chocolate-chip cookie recipes, information exchange on the Internet is carried out continuously and ubiquitously. While originally conceived as a convenient tool for text messages, email has since evolved into the backbone of the Internet, and has become the primary medium not only for communicating ideas, opinions, and appointments, but also for unauthorized accesses and malicious attacks. For instance, a malicious executable program attached to an apparently benign email can easily be sent to thousands of recipients. With just a couple of mouse clicks, it can (and has) caused damage to computer systems and associated networks. These unfriendly activities include improperly gaining access privilege (*Trapdoor*), disclosing sensitive information (*Covert Channel*), exhausting system resources (*Worm*), and infecting normal programs (*Virus*). Some programs possess all the aforementioned malicious actions (*Trojan Horse*, *Time/Logic Bomb*, etc.).

There are several ways to determine whether or not a program might perform malicious functions. A program being screened can be compared with a known "clean" copy of the program (Cohen, 1987). Known malicious codes can be

* Corresponding author. Tel.: +1 505 6671559; fax: +1 505 6654197.
E-mail addresses: dmc@lanl.gov (D.M. Cai), maya@lanl.gov (M. Gokhale), jt@lanl.gov (J. Theiler).

0167-9473/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2006.09.005

detected by virus scanners or compared against a set of verification rules serving as malicious code filters (Crocker and Pozzo, 1989). Dynamic analysis combines the concept of testing and debugging to detect malicious activities by running a program in a clean-room environment (Lo et al., 1991; Crawford et al., 1991).

To identify signatures of a malicious code, a computer security expert can screen and analyze a suspicious code with respect to its functions, data flow, and variable usages in order to find those signatures that indicate malicious intentions. Although a human can reason about a program in detail, it becomes difficult to examine code and data when the malicious components are embedded in different sections, their activities are triggered only when certain conditions are satisfied, or a large number of programs need to be handled at one time. Lo and his colleagues investigated and developed a malicious code filter based on the use of "*tell-tale signs*" and program slicing verification (Lo et al., 1995). A "tell-tale sign" is a program signature that determines whether the program is malicious or benign. For instance, copying a viral code to the end of the text segment would be considered a malicious signature. The program "slicing" produces a subset of the original program behaving the same with respect to the realization of a specified property. The purpose of this method is to determine whether a program is malicious or benign by examining characteristics of a small subset of a large program.

However, a malicious program may elude detection because it does not match any known signature. Its signature may deviate from known signatures, or the program may contain new signatures that have not been previously seen. As virus-writers continually create and modify malicious programs, it is essential not only to identify those malicious programs that exactly match known signatures, but also to detect new ones with *similar* features.

In an attempt to address the problem, Kephart and his colleagues at the high integrity computing laboratory of IBM used statistical methods to automatically extract computer virus signatures (Kephart and Arnold, 1994). Their idea was to identify the presence of a virus in an executable file, a boot record, or memory using short identifiers or signatures, which consist of sequences of bytes in the machine code. A good signature is one that is found in every object infected by the virus, but is unlikely to be found if the virus is not present. Later, researchers from the same group successfully developed a neural network based anti-virus scanner to detect the boot sector virus (Tesauro et al., 1996). Due to system limitations at that time, it was difficult to extend the neural network classifier to detect types of virus other than the boot sector virus.

In several later studies, statistical analyses and pattern recognition techniques have been applied to identify the malicious codes. The researchers (Schultz et al., 2001a,b) at Columbia University applied data mining techniques (Lee et al., 1999) to analyze a large set of malicious executables instead of only boot-sector (Tesauro et al., 1996), or only Win32 binaries (Arnold and Tesauro, 2000). Schultz and his colleagues used system resource information, embedded text strings and byte sequences as features extracted from executables. Three learning algorithms were used to classify the data: (1) an inductive rule-based learner that generates Boolean rules based on feature attributes; (2) a Naïve Bayes classifier that estimates posterior probabilities that a testing file is in a class given a set of features, and (3) a multi-classifier system that combines the outputs from several Naïve Bayes classifiers to generate an overall prediction. Their results show two classifiers, the Naïve Bayes classifier (using text strings as features) and multi-Naïve Bayes classifiers (with only byte sequences), outperform all other methods in terms of overall performance measured by detection rate and false positive rate.

More recently, using the same data set created by the researchers at Columbia University, Wang and colleagues tried the Naïve Bayes classifier and compared its performance with a Decision Tree (Wang et al., 2003).The motivation was, instead of using binary sequence information, to use the feature information of the assembly language level or even at high language levels. To do so, a reverse-engineering step is needed to convert the machine codes to their original computer languages if possible. Interestingly, the overall accuracy of the Naïve Bayes classifier was significantly lower than that reported in Schultz's paper (Schultz et al., 2001a). It is not clear whether the discrepancy originated from the different approaches for feature selection. In a pilot study, Kolter and Maloof also tried to use different classifications algorithms to detect malicious executables using byte sequence patterns. Of the four evaluated classifiers, boosted decision trees outperformed other methods when each feature variable consists of 4 bytes (4-g) (Kolter and Maloof, 2004).

Detecting malicious executables using text strings or byte sequences carries some similarities to text categorization, which uses words and/or phrases to assign documents to pre-defined categories. In the text categorization problem, the native feature space consists of the unique terms (words or phrases) that occur in documents, which can be tens or hundreds of thousands of terms for even a moderate-sized text collection. To effectively pursue dimensionality reduction, Yang and Pedersen did a comprehensive study of five feature selection methods in statistical learning of

text categorization (Yang and Pedersen, 1997). The feature selection method based on an information gain metric yielded improved performance by removing up to 98% of the terms. In addition to the fact that the prediction accuracy depends on data and feature selection methods, the right choice of classifiers also plays a critical role. Besides the successful applications of the Naïve Bayes classifiers in text categorization (Lewis and Ringuette, 1994; McCallum and Nigam, 1998), there have been reported a significant improvement made by the implementation of SVMs as classifiers (Joachimes, 1998b; Dumais et al., 1998; Yang and Liu, 1999). As SVMs have gradually gained a reputation as "must-try" classifiers, more comparative studies have been made on other aspects of applications of machine leaning for text classification, such as feature selection. For example, Forman presented an extensive empirical study of 12 feature selection metrics evaluated on a benchmark of 229 text classification problems. The results revealed a new feature selection metric called 'bi-normal separation' (BNS), which outperformed the others by a substantial margin in a range of situations (Forman, 2003).

In this study, we first compared the performance of seven different feature selection methods against three types of classifiers in distinguishing benign and malicious binaries. Then, we examined whether extending the features to multiple-byte sequences could improve the classification accuracy. Finally, SVM classifiers with different kernel functions were applied and their performance was compared. The rationale for choosing byte sequences as candidate features is that those byte patterns are the most accessible, and at the same time, the most direct information about the machine code in an executable. Using embedded text strings as features, such as head information, program names, authors' names, or comments, is not robust since they can be easily changed. Some malicious executables intentionally camouflage these signatures by randomly generating these fields to deceive virus scanners (Kephart and Arnold, 1994; Schultz et al., 2001a).

2. Methods and experimental settings

2.1. Data description and preparation

Our experimental data were downloaded from the intrusion detection system site at Columbia University: http://www.cs.columbia.edu/ids/mef/. It consists of 4754 total files with 1074 benign and 3680 malicious. There are no duplicate programs in the data set, and each file is labeled as either benign or malicious. A more detailed description of the dataset can be found in (Schultz et al., 2001a,b).

The downloaded files have been already transformed from their binary form into hexadecimal text format. An example hexadecimal code extracted from the notorious mail executable "CHRISTMA" is shown in Fig. 1.

A byte sequence pattern consists of n sequential bytes; for $n = 1$, the pattern consists of a single byte (e.g., `07`, `eb`, `0a`, etc.), for $n = 2$, two bytes (e.g., `07eb`, `eb0a`, `0a56`, etc.), and so on, up to 10 consecutive bytes. For each file, a vector of features is created, with each component corresponding to the frequency of occurrence of a given byte pattern in the file. Following the notation in (Schultz et al., 2001a), we use F to represent a byte pattern and $P(F)$ to represent its frequency in a given file. The expression $P(F|B)$ corresponds to the average frequency of the pattern F over all benign files and $P(F|M)$ is the average frequency over all malicious files. Fig. 2 shows these average frequencies for benign (top panel) and malicious (middle panel) files for the single-byte features.

The histograms illustrate that the byte patterns of both benign and malicious files are not uniformly distributed. Frequencies of certain bytes are much higher than the others. For example, of all training benign files, the most frequent pattern `00` has a frequency of 13.3%, and the rarest pattern is the hexadecimal `97` with 0.1% frequency. Note that the y scale is truncated at 2% on the top and middle panels, to better show the variability among the rarer patterns.

2.2. Feature selection methods

Feature selection is a process for finding the most informative characteristics (or features) that describe the data samples. In this case, the samples are the individual files, and the candidate features are frequencies of the byte patterns. Even for single byte patterns, but even more so for multiple byte patterns, the number of possible features is large, and the goal of feature selection is to choose from among these variables to yield the most satisfactory segmentation between malicious and benign executables. Seven feature selection methods were investigated: (1) maximal difference; (2) maximal normalized difference; (3) maximal ratio; (4) maximal weighted ratio; (5) maximal Kullback–Leibler distance; (6) bi-normal separation; and (7) forward stepwise selection and backward stepwise selection.

```
07eb 0a56 5903 1a00 0e00 00e8 fa00 ec8b
3258 89c0 0246 4681 2800 b900 05ce 2cb0
4688 8bff 007e 4e88 8afe ff4e 0d00 00eb
8a47 fe4e c4e2 88cf a103 daf5 0727 6294
74ac d8eb e4f8 5424 ebfa c3d8 9c60 ac62
```

Fig. 1. An example of a binary executable in its hexadecimal text format was excerpted from a malicious file. To extract byte sequence patterns, each file is processed sequentially from beginning to end, using the pre-determined fixed size windows (1-, 2-, and so on, up to 10-bytes).

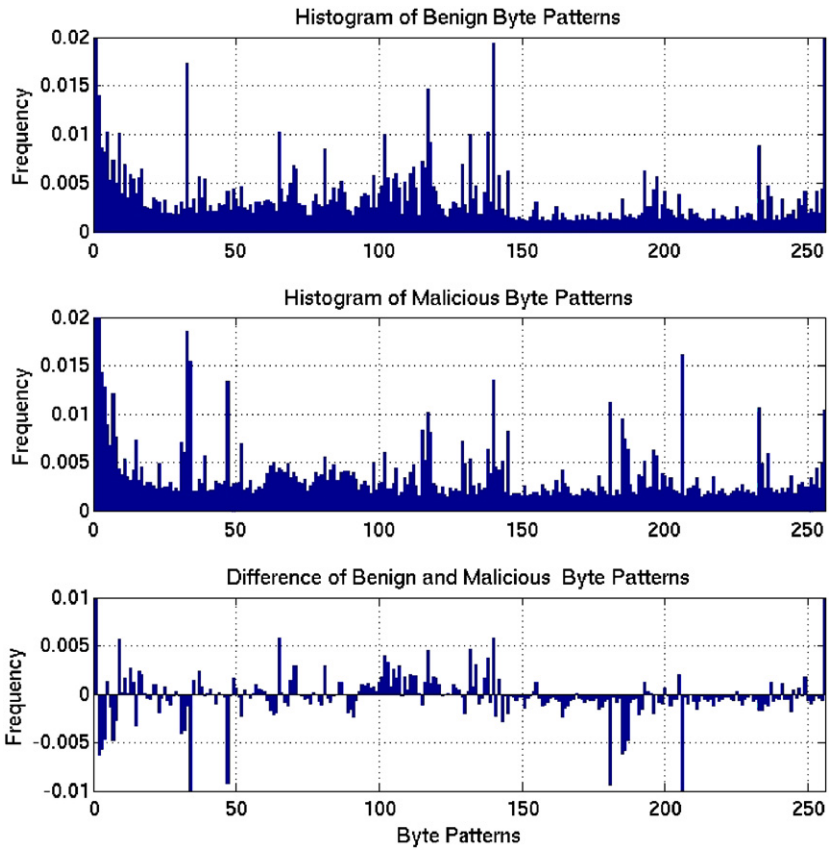

Fig. 2. The histogram of benign (the top panel), malicious executables (the middle panel) and the corresponding difference (the bottom panel) from the training set. The x-axis is the single byte pattern in hexadecimal from "`00`" (0 in decimal) to "`ff`" (256 in decimal). The y-axis shows the occurring frequency. To facilitate the difference, the y plotting scale is set between 0 to 0.02 on the top and middle panel, and from -0.01 to 0.01 for the bottom panel.

2.2.1. *Maximal difference (MD)*

We compute the difference between the average frequency of the pattern F in the benign files and the average frequency in the malicious files. This difference is computed for each feature F

$$P(F|B) - P(F|M) \tag{1}$$

and the differences are sorted in ascending order. The patterns on the top of the list occur more often in malicious than in benign executables, while patterns at the bottom of the list occur more often in benign files than in malicious. To choose n features, we take the $n/2$ top features in the list and the $n/2$ bottom features.

In the bottom panel of Fig. 2, we plot $P(F|B) - P(F|M)$ for the single-byte patterns. The most positive differences (most frequent in benign files) occur for bytes `00`, `ff`, and `40`; the most negative (most frequent in malicious files) occur for bytes `cd`, `21`, and `b4`.

2.2.2. Maximal normalized difference (MND)

This method is like the MD method, but it takes into account the file-to-file variability in the pattern frequencies. In particular, write $\sigma(F|B)$ as the standard deviation of the frequencies $P(F)$ over all benign files, and similarly $\sigma(F|M)$ for the malicious files. The pooled standard deviation is given by

$$\sigma^2(F) = \frac{N_B\sigma^2(F|B) + N_M\sigma^2(F|M)}{N_B + N_M}. \tag{2}$$

The normalized difference is then given by

$$\frac{P(F|B) - P(F|M)}{\sigma(F)}. \tag{3}$$

2.2.3. Maximal ratio (MR)

Instead of using differences, this feature selection method uses the ratio of frequencies,

$$\frac{P(F|B)}{P(F|M)} \tag{4}$$

or, equivalently, the difference of the logarithms:

$$\log P(F|B) - \log P(F|M). \tag{5}$$

As with the direct differences, the list is sorted in an ascending order, and the $n/2$ largest (most positive) and the $n/2$ smallest (most negative) features are retained.

2.2.4. Maximal weighted ratio (MWR)

We also considered a heuristic feature selector which can be seen as a modified maximal ratio that gives extra weight to those features for which $P(F)$ tends to be large.

$$(P(F|B) + P(F|M))(\log P(F|B) - \log P(F|M)). \tag{6}$$

2.2.5. Maximal Kullback–Leibler distance (MKLD)

The Kullback–Leibler distance or relative entropy between two probability distributions $p(x)$ and $q(x)$ is (MacKay, 2003)

$$D_{\mathrm{KL}}(p\|q) = \sum_x p(x)\log p(x)/q(x). \tag{7}$$

Note that since in general $D_{\mathrm{KL}}(p\|q) \neq D_{\mathrm{KL}}(q\|p)$, this measure is not strictly a 'distance.' Thus, we use a symmetrized variant

$$D^*(p\|q) = D_{\mathrm{KL}}(p\|q) + D_{\mathrm{KL}}(q\|p) \tag{8}$$

which is also known as the 'discrimination'.

Applying KL distance for feature selection, we seek features that contribute maximally to the sum in the KL distance formula. These are the features F for which

$$(P(F|B) - P(F|M))(\log P(F|B) - \log P(F|M)) \tag{9}$$

is large. Note that this expression is always positive, and so we use features with the largest values.

2.2.6. Bi-normal separation (BNS)

The BNS method is a new feature selection metric that was shown to yield the best performance with a range of data sets in text classification (Forman, 2003). BNS is defined as $N^{-1}(tpr) - N^{-1}(fpr)$, where N^{-1} is the standard normal distribution's inverse cumulative probability function. Here, *tpr* and *fpr* represent true positive rate and false positive rate, respectively, and $N^{-1}(tpr) - N^{-1}(fpr)$ is assumed to be greater or equal to zero. In this study, we used the Naïve

Bayes classifiers to screen all 256 variables. For each byte pattern F, we extracted the largest separation $D_{\max}(F)$ of $N^{-1}(tpr) - N^{-1}(fpr)$ in performing the classification of malicious and benign executables. The features of all 256 byte patterns were selected by their values $D_{\max}(\boldsymbol{F})$ in descending order.

2.2.7. Forward stepwise selection (FSS) and backward stepwise selection (BSS)

The previous six methods are "filters" for feature selection; stepwise selection is a "wrapper" method (Kohavi and John, 1997). The previous methods involve computing an independent numerical value for each feature, and then selecting features on the basis of a sorted list of these values. In forward stepwise selection, features are chosen one at a time. In the first stage, each feature is used to form a one-feature predictive model. The feature that gives the best performance is retained for the next stage. The remaining features are then paired with the selected feature for the second stage, which evaluates all of the two-feature models. The process continues until the desired number of features is chosen or the performance no longer improves as more features are added.

Backward selection employs the same principle as forward selection, but does so (as the name suggests) in the opposite direction. It starts with a full set of features; at each stage, it reduces the number of features by one, choosing the feature that can be disposed of with the least cost in terms of performance. Even more so than forward selection, the iterative selection process makes the backward selection a time consuming process. Note also that forward selection and backward selection are not commutative. Two feature sets, with the same number of features, but generated by the two different selection methods, will generally consist of different feature variables and will perform differently, as we will see in Fig. 6.

2.3. Classifiers

2.3.1. Naïve Bayes Classifier

As noted above, classifying an executable by examining consecutive bytes is somewhat like categorizing text documents in terms of words or phrases. The Naïve Bayes classifier calculates the posterior probability that a program is either a malicious or benign file given the features that it contains. This approach has been effectively used in text document classification (Lewis, 1992; McCallum and Nigam, 1998; Yang and Ghani, 2002) and directly applied to detect malicious executables (Schultz et al., 2001a,b).

Given class labels C corresponding to malicious (M) or benign (B), and a set $\mathscr{F}$ of feature variables, the Naïve Bayes classifier calculates $P(C|\mathscr{F})$, the probability that program is malicious or benign, given $\mathscr{F}$. From Bayes rule

$$P(C|\mathscr{F}) = \frac{P(\mathscr{F}|C)P(C)}{P(\mathscr{F})}. \tag{10}$$

A key assumption of the Naïve Bayes classifier is that the features are independent. So if an executable file contains the features $\mathscr{F} = (F_1, F_2, \ldots, F_n)$, where F_i corresponds to the ith byte (or byte pattern) in the file, then Eq. (10) becomes

$$P(C|F_1, F_2, \ldots, F_n) = \frac{\prod_{i=1}^{n} P(F_i|C)P(C)}{\prod_{i=1}^{n} P(F_i)}. \tag{11}$$

Here, $P(F_i|C)$ is the average frequency of the byte sequence pattern F_i in executables of class C. $P(C)$ is the proportion of executables in class C. Since the denominator has the same value for all classes, we can say that the most likely class is given by

$$\arg\max_{C} P(C) \prod_{i=1}^{n} P(F_i|C). \tag{12}$$

At training time, we calculate prior probabilities for $P(C)$ and $P(F_i|C)$ for all training programs. Then, at classification time, for every testing program, we compute the relative likelihood $L(C)$ for each class C:

$$L(C) = P(C) \prod_{i=1}^{n} P(F_i|C) = P(C) \prod_{F} P(F|C)^{n_F}, \tag{13}$$

where n_F is the number of occurrences of pattern F in that testing program. Taking the logarithm of Eq. (13) yields

$$\log P(C) + \sum_F n_F \log P(F|C). \tag{14}$$

Since it is a binary classification problem, the equation can be further simplified to

$$K + \sum_F n_F \log\left(\frac{P(F|B)}{P(F|M)}\right). \tag{15}$$

Here, K, depends on the prior probabilities for malicious and benign files; it can be varied to give different threshold values to be malicious or benign. If the calculated value of Eq. (15) is greater than 0, the testing file is predicted as benign, otherwise, it is malicious.

As an aside, we remark that Eq. (15) in the $K = 0$ case is equivalent to a minimum distance classifier, using the Kullback–Leibler distance. To see this, note that for a given testing file, its byte pattern occurring frequency distribution is n_F/n. The Kullback–Leibler distance for the testing file to the probability distribution of benign programs $P(F|B)$ or malicious programs $P(F|M)$ will be

$$\frac{1}{n}\sum_F n_F \log\left(\frac{n_F/n}{P(F|B)}\right) \tag{16}$$

or

$$\frac{1}{n}\sum_F n_F \log\left(\frac{n_F/n}{P(F|M)}\right). \tag{17}$$

The minimal distance is given by the smaller of Eqs. (16) and (17); subtracting Eq. (16) from Eq. (17) gives exactly the expression we derived on Eq. (15). This demonstrates the equivalence of Naïve Bayes and minimum distance, where the distance is Kullback–Leibler.

2.3.2. Product classifier

Using different feature selection methods described before, we know the benign and malicious files in general have markedly different probability distributions of their byte sequence patterns. For example, the byte `00` has much larger occurrence frequency (13.3% vs. 7%) in benign programs than in malicious ones. The product of a set of such variables should amplify these differences and give an indication whether a testing file should be labeled as benign or malicious; in particular the ratio of those products can be used to compare benign against malicious

$$\prod_{F\in B} n_F \Big/ \prod_{F\in M} n_F. \tag{18}$$

When the ratio is large, then there are many more occurrences of the benign bytes than the malicious bytes; taking the logarithm of Eq. (18) gives

$$K + \sum_{F\in B} \log(n_F) - \sum_{F\in M} \log(n_F). \tag{19}$$

The variable K has been introduced as an arbitrary threshold; when this expression is positive, the file is classified as benign; when it is negative, the file is taken to be malicious.

2.3.3. Entropy classifier

The third method we used is based on a similar rationale as the product classifier. The difference is to use entropy instead of product

$$K + \sum_{F\in B} n_F \log(n_F) - \sum_{F\in M} n_F \log(n_F). \tag{20}$$

2.3.4. Support vector machine classifiers

Support vector machines (SVMs) were introduced by Vapnik and his colleagues for solving two-class pattern recognition problem (Cortes and Vapnik, 1995; Vapnik, 1998). For an excellent tutorial on SVMs, please refer to (Burges, 1998). The training set consists of instance-label pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, N$ where $\boldsymbol{x}_i \in \Re^n$ corresponds to the feature vector for the ith file and $\boldsymbol{y}_i \in \{-1, 1\}$ corresponds to the label on the ith file; -1 corresponds to benign, and 1 corresponds to malicious. The linear SVM computes a function $f(\boldsymbol{x}) = \omega^{\mathrm{T}}\boldsymbol{x} + b$ from which the label is predicted by the sign of $f(\boldsymbol{x})$. The expression $yf(\boldsymbol{x})$ is referred to as the "margin"—when it is positive, $f(\boldsymbol{x})$ has the same sign as y and the label will be correctly predicted. When the margin is large, then the prediction has more "confidence" and in fact the SVM tries to maximize margin by optimizing the loss function

$$\frac{1}{2}\omega^{\mathrm{T}}\omega + C\sum_i \xi_i, \tag{21}$$

where $\xi_i = \max(0, 1 - y_i f(\boldsymbol{x}_i))$ are the so-called "slack" variables; ξ_i is zero whenever the margin is one or larger, but if the margin is small or even negative (signifying a misclassified data point), then ξ_i is positive and indicates how "far" the data point is from being classified with a margin of one. The constant $C > 0$ specifies how much weight is assigned to classifying points correctly compared to the penalty for large coefficients ω.

The SVM can be extended to nonlinear discriminating functions $f(\boldsymbol{x})$ by mapping data vectors $\boldsymbol{x}_i$ into a higher-dimensional space by the function ϕ. Then the SVM is a function that is linear in the higher-dimensional space: $f(\boldsymbol{x}) = \omega^{\mathrm{T}}\phi(\boldsymbol{x}) + b$. The solution can be expressed in terms of a dual problem

$$\begin{aligned} \min_{\alpha} \quad & \tfrac{1}{2}\alpha^{\mathrm{T}} Q\alpha - e^{\mathrm{T}}\alpha \\ \text{subject to} \quad & 0 \leqslant \alpha_i \leqslant C \text{ and } y^{\mathrm{T}}\alpha = 0. \end{aligned} \tag{22}$$

Here, e is the vector of all ones, Q is a $N \times N$ positive semidefinite matrix with elements $Q_{ij} \equiv y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)$ is the kernel. Then $\omega = \sum_i \alpha_i y_i \phi(x_i)$, and

$$f(\boldsymbol{x}) = \omega^{\mathrm{T}}\phi(\boldsymbol{x}) + b = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{23}$$

The software package we used to generate SVM classifiers is called *LIBSVM* (Chang and Lin, 2001). *LIBSVM* is an integrated software package for support vector classification, regression, and distribution estimation. It supports multi-class classification. The basic algorithm is a simplification of the sequential minimal optimization (SMO) scheme (Platt, 1998) and *SVMLight* (Joachimes, 1998a,b). It is also a simplification of the revised version of SMO described in (Keerthi et al., 2001). The software is available for downloading at http://www.csie.ntu.edu.tw/∼cjlin/libsvm/.

2.4. Performance measures

The individual feature selection methods are used to select features that distinguish malicious executables from benign programs. Their effectiveness is evaluated using the performance of different classifiers defined in the last section. To enable direct comparison, the model performance uses a similar measure to that used by (Schultz et al., 2001a,b).

1. True positives (TP), the number of malicious executables correctly classified as malicious;
2. True negatives (TN), the number of benign programs correctly classified as benign;
3. False positives (FP), the number of benign programs falsely classified as malicious,
4. False negatives (FN), the number of malicious executables falsely classified as benign.

$$\text{Detection rate (DTR)} = \frac{TP}{TP + FN}, \tag{24}$$

$$\text{False positive rate (FPR)} = \frac{FP}{TN + FP}, \tag{25}$$

$$\text{Overall accuracy (OA)} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{26}$$

2.5. Experimental setting

The malicious and benign files were randomly partitioned into five sets. One of the five (comprising 20% of the data samples) was held out as a test set and the remaining four (80% of the data) were concatenated into a training set. The results will be reported based on the performance on the hold-out set unless they are specified differently.

The experiment was first tested on the seven proposed feature selection methods, using three different classifiers (Naïve Bayes, product, and entropy). This was done initially using single byte patterns, but we also examined whether using multiple-byte sequence patterns would improve the classification performance. Finally, SVMs with a selection of kernel functions were trained and tested.

3. Results

3.1. Graphical analysis of feature selection methods

In Fig. 3, the plot shows how the average frequency of a byte pattern F is associated with either benign or malicious files. The y-axis is the average frequency of each variable in the malicious files, i.e. $P(F|M)$. The average occurrence of each variable in the benign files, i.e. $P(F|B)$, is plotted in the x-axis. The dashed line with a slope of 45° divides the quadrant into two sections. When a variable occurs at the upper left part of the quadrant, e.g. the byte pattern "cd", the variable more likely occurs in a malicious file than in a benign file. Conversely, if a byte pattern occurs on the lower right quadrant, we expect it would be present more in a benign file than in a malicious one. Those byte patterns,

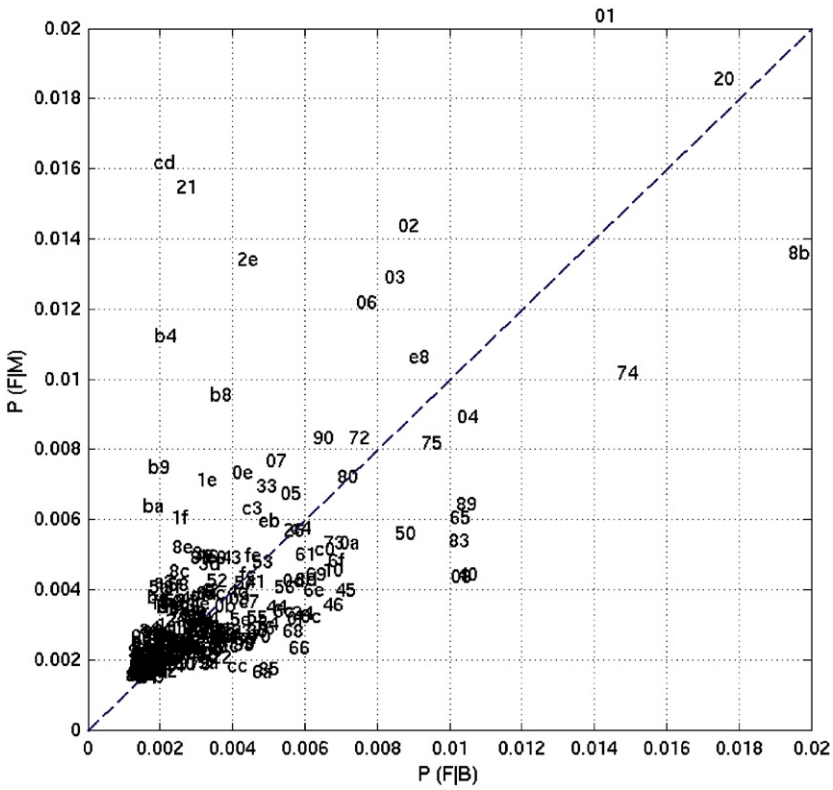

Fig. 3. This plot displays all 256 byte patterns, from "00" to "ff", in terms of the average frequency of occurrence in both benign (x-axis) and malicious (y-axis) files. Each byte pattern label is marked at its corresponding coordinate ($P(F|B)$, $P(F|M)$). The diagonal line from the bottom left corner to the upper right corner divides the quadrant to two equal triangular regions. If a byte pattern falls into the upper half of the quadrant, it occurs more frequently in malicious files than in benign ones. On the flip side, a byte pattern would have more occurrences in benign files than in malicious ones if it scatters on the lower half of the quadrant.

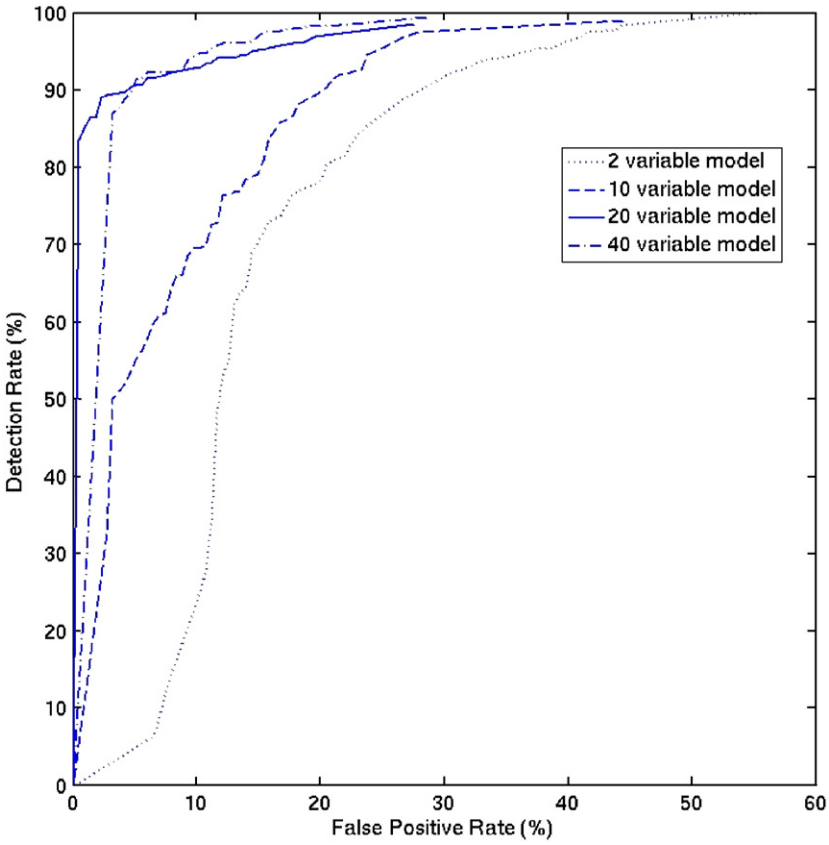

Fig. 4. Compares the performance of the product classifier as a function of the number of feature variables. Here, the maximal difference method was used to select four sets of feature variables with 2, 10, 20, and 40 variables respectively.

which are clustered near the diagonal line, are "common" to both benign and malicious executables. These "common" byte patterns are not expected to be good discriminants for classification. In order to illustrate the distribution of most byte patterns, both x and y values are capped at 0.02, which makes the pattern "00" not be displayed in the plot with its ordinate as 0.07 and the abscissa as 0.13. The distance from a pattern's location in Fig. 3 to the diagonal line is $|P(F|M) - P(F|B)|/\sqrt{2}$. A larger distance suggests that the variable is more likely to occur in one class than the other, and therefore is likely to be a more informative variable.

3.2. Choices of feature selection methods and classifiers

In Fig. 4, we illustrate classification performance with a plot of the ROC (receiver operating characteristic) curve that shows the trade-off between detection rate (y-axis) and false positive rate (x-axis).

Fig. 4 compares the performance of the product classifier as a function of number of feature variables. Here, using the maximal difference feature selection method, we computed the ROC curves using 2, 10, 20, and 40 feature variables.

At the 90% detection rate, the classifier with only 2 feature variables (dotted line) gives a 28% false positive rate. That rate drops to 21% using the model with 10 variables (dashed line). When the number of feature variables increases to 20 (solid line), the false positive rate decreases dramatically to about 3%, but then the model performance deteriorates with 40 feature variables, achieving a 5% false positive rate.

The experiment suggests the number of feature variables chosen clearly impact the model performance. Using the same data set with the same type of classifier, selecting a different number of feature variables yields very different performance. Our next experiment systematically compares how performance varies as different combinations of feature selection methods and classifier types are combined.

Table 1
Compares the performance among seven feature selection methods with three types of classifiers

	Feature selection method							
	Maximal difference	Maximal normalized difference	Maximal ratio	Maximal weighted ratio	Maximal K–L distance	Bi-normal separation	Forward selection	Average FPR
Classifier type	False positive rate (at detection rate 90%)							
Naïve Bayes	11.0	11.0	7.0	10.0	10.0	5.0	7.0	8.7
Entropy	6.0	9.0	8.0	7.0	8.0	5.0	4.0	6.7
Product	3.0	3.0	6.0	2.0	5.0	8.0	2.0	4.1
Average FPR	6.7	7.7	7.0	6.3	7.7	6.0	4.3	

Of the first six feature selection methods, the number of variables ranges from 2 to 256, and the set with the lowest false positive rate (or best performance) at about 90% detection rate using the corresponding classifier is reported in the table. In the forward selection method, all 256 variables were screened. At each run, only one variable that gave the highest overall accuracy was retained as a new feature variable. The set that gave the lowest false positive rate at the 90% detection rate is documented in Table 1.

In Table 1, we compare the performance among seven feature selection methods with three types of classifiers. For each combination of the seven feature selection methods and the three classifiers, we chose the number of features so as to give the best performance. The false alarm rate was computed at the 90% detection rate for each of the feature selection methods and classifiers.

Of all three types of classifiers, the product classifier gave the most accurate performance with a lowest false positive rate of about 2%, and an average false positive rate of about 4.1% (averaged over all seven feature selection methods). The entropy classifier gave a 6.7% false positive rate on average. The overall false positive rate for the Naïve Bayes classifier is about 8.7%. Comparing all seven feature selection methods, the iterative forward selection method takes the longest time to do variable selection, but gives the most accurate classification. The average false positive rate is about 4.3% averaged over all three classifiers.

The two selection methods, the maximal difference (MD), and the maximal weighted difference (MWD), give the largest accumulated distance, and also produce the best performance using the product classifier (Table 1). Interestingly, the BNS method shows the least accumulated distance when the first 100 features are selected and yields the highest false positive rate at a detection rate 90% paired with the product classifier. On the other hand, of all seven feature selection methods, the BNS produces the best performance using both the entropy classifier and Naïve Bayes classifier.

3.3. Impact of feature patterns size

Having compared the performance using different feature selection methods and classifiers on single byte patterns, we also checked whether multiple byte sequences would give better performance than single bytes. In Table 2, the feature variables contain 1-, 2-, 5- or 10-bytes in each pattern. All results were generated using the product classifier and the MD method of feature selection.

For m-byte sequences, there are 256^m total patterns; for $m = 10$, this works out to over 10^{24}. We have already seen that optimal performance can be achieved with a small set of selected feature variables and that performance deteriorates when too many variables are used. To avoid unnecessary computational overhead, for multiple byte patterns, we first generated all possible patterns in each file and calculated the occurring frequency of each pattern. Then the patterns were sorted in a descending order based on their frequencies. The top 1000 patterns with the largest occurring frequencies were selected, renormalized, and kept as the candidates of feature variables. We validated the performances of two byte sequence patterns, one with a full set of variables, and the other using only the top 1000 variables as proposed, and their results did not show a significant difference.

In Table 2, the number of feature variables was chosen as 10, 20, and 40. The detection rates were jointly compared for different pattern size when their false positive rates at 1–3%, respectively. When as few as 10 feature variables were used, the performance was generally poor no matter the size of patterns used. As the number of feature variables was increased to 20, the overall performance was improved significantly. For example, the detection rates at 3% false positive rate were increased more than two-fold for 1-, 2-, and 5-byte patterns. However, the corresponding increase

Table 2
Performance comparisons made when 1-, 2-, 5-, 10-byte sequences were used as the features

Byte(s) per observation window	10 Variable model			20 Variable model			40 Variable model		
	False positive rate			False positive rate			False positive rate		
	1 %	2 %	3 %	1 %	2 %	3 %	1 %	2 %	3%
	Detection rate (%)			Detection rate (%)			Detection rate (%)		
1	10	27	50	85	87	90	74	80	85
2	24	26	34	63	74	81	68	77	82
5	20	21	22	30	31	54	35	54	57
10	5	6	39	8	40	42	8	41	42

All results were generated using the product classifier and the maximal difference method of feature selection. The number of feature variables was chosen as 10, 20, and 40. The detection rates were jointly compared for different pattern size when their false positive rates were 1%, 2% and 3%, respectively.

Table 3
The results from experiments using SVM classifiers and single byte patterns with 10, 20, 40, 100, and all 256 feature variables

Number of feature variables used in SVM	Performance				
	False positive rate (%)	Detection rate (%)	False positive rate (%)	Detection rate (%)	Average training time (s)
10	3.0 (5.0)	90.0 (90.0)	10.0 (7.0)	95.0	8.0
20	1.0	89.0 (90.0)	5.0 (2.0)	95.0	10.0
40	1.0	92.0 (94.0)	4.0 (2.0)	95.0	16.8
100	1.0	95.0 (93.0)	1.0 (2.0)	95.0	32.8
256	1.0	93.0 (93.0)	2.0 (2.0)	95.0	75.0

Feature variables were selected using the maximal difference method and the BNS method (the results for which are in parentheses), and the radial basis functions were the kernel functions used throughout. The performance was benchmarked with respect to two different criteria: (1) the detection rate when the false positive rate was 1%; and (2) the false alarm rate when the detection rate was 95%.

for 10-byte patterns was merely 3%. When the number of feature variables was raised to 40, the performance for single byte pattern was actually decreased. Also, for the size of patterns at 2-, 5-, and 10-bytes, the performance improvement at 3% false positive rate was very marginal. Overall, when a small number of selected feature variables are used, the classifier with single byte patterns can perform better than those with multiple byte patterns.

3.4. Performances of SVM classifiers with different features and kernel functions

In the last set of experiments, we examined how performance of SVMs varied as a function of feature variable selection, and a choice of different kernel functions.

Table 3 summarizes the results from experiments using single byte patterns with 10, 20, 40, 100, and all 256 feature variables. To compare the two best feature selection methods, feature variables were selected using the MD method and the BNS methods, respectively. The radial basis functions were the kernel functions used throughout. The performance was benchmarked with respect to two different criteria: (1) detection rate when the false positive rate was 1%; and (2) false alarm rate when the detection rate was 95%.

For the first set of experiments, when only 10 variables were used, to achieve 1% false positive rate, we found it difficult to choose a proper set of SVM parameters without overfitting the training data. Therefore, the result was reported at a 3% false positive rate for the MD method, and a 5% false positive rate for the bi-normal separation, respectively. When the number of feature variables was increased from 10 to 40, the BNS method appeared to reach its peak performance with a detection rate 94% and a false positive rate of about 1%. Adding more variables by the BNS,

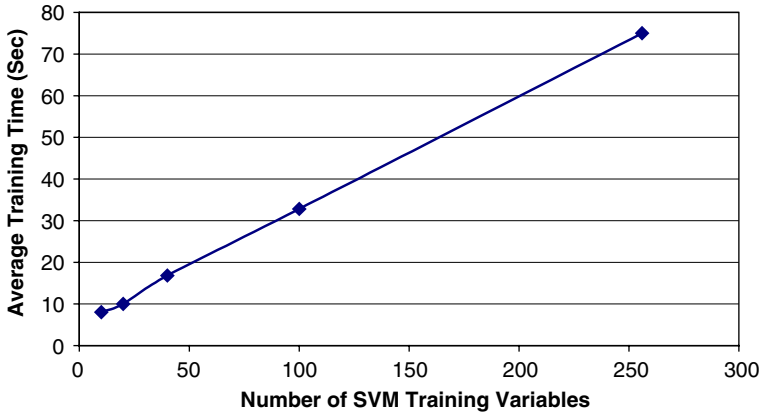

Fig. 5. The relationship between training time (excluding pre-processing time) of SVMs and the number of training variables. The features were selected using the maximal difference method. The radial basis function was used as the kernels of SVMs.

the classification accuracy did not improve. For the MD method, the performance continued to improve when more feature variables were selected from 10, 40, to 100. The MD method's detection rate improved from about 89% to 95%. When all 256 variables were provided to the classifier, the performance decreased slightly. Similar trends were observed among the false positive rates when their detection rates were fixed at 95%. With 10 variables, the false positive rate could be as high as 7–10%. The best performance occurred when 40–100 variables were selected with only 1% false positive rates for both feature selection methods. The false positive rate increased to 2% when the SVM classifier was trained using all 256 variables. Overall, the SVMs gave the best classification accuracy out of all four classifiers. Interestingly, the two feature selection methods came up with different sets of feature variables, but the performance differed only when a few variables were selected. As the number of the variables increased, the performance of SVMs seemed more comparable between the MD and BNS feature selection methods.

In the last column of Table 3, we compared the average training time when a different number of features was imported. The training set was randomly split into 80/20 for five times, and the average time for each training cycle was reported in the table. It is interesting to see the training time is linearly proportional to the number of SVM training variables, as shown in Fig. 5.

To test how robust the classifiers performed, for the SVM classifier with 100 feature variables chosen by the MD method shown in Table 3, we ran the backward and forward selection. The results are shown in Fig. 6.

The performance was measured in overall accuracy (top panel), detection rate (middle panel) and false positive rate (bottom panel). The results of backward selection (solid lines) were overlaid with those of the forward selection (dashed lines). For backward selection, as the number of variables was reduced from 100 to 58, the overall performance did not change too much. The false positive rates were below 1%, and the detection rates were above 95%. As the number of feature variables varied in the range of 63–68, these features gave the best performance with a 0.93% false positive rate at 96.3% detection rate. However, when the number of variables continued to be reduced to 31, the detection rates were maintained between 93–95%, but the false positive rates increased from 1% to 12%. The performance tended to deteriorate more when the backward selection proceeded. In contrast, forward selection seemed to select a set of models with less accurate but more stable performance. When the number of variables was increased to above 10 but below 93, the false positive rates were between 6–12% with the detection rates 94–97%. When 93–100 variables were selected, not surprisingly, the performance was about 1% false positive rate with a 95% detection rate.

Our last set of experiments was to compare the performances of SVMs with linear, polynomial, radial basis, and sigmoid kernel functions.

The same training data set with 100 variables was used for all classifiers. The linear kernel function had the least training time (22 s), but the highest false positive rate (7%) when the detection rate was 95%. The sigmoid kernel had a performance similar to the linear kernel with slightly longer training time. The best overall performance was achieved using the radial basis function. Its training time was about $\frac{1}{3}$ longer than that of the linear kernel function. Nevertheless, considering the total training time is on the order of tens of seconds, the difference is not significant.

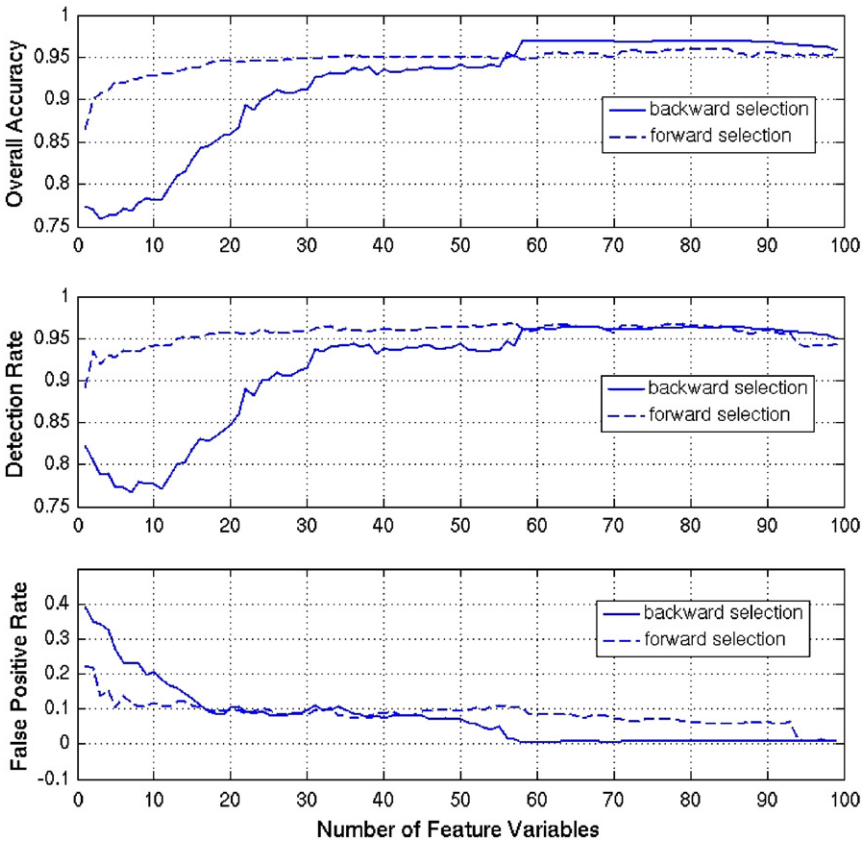

Fig. 6. Results of how robust the classifiers performed. For the SVM classifier with 100 feature variables shown in Table 3, we ran both backward and forward selection and the results are shown in this figure. The performance was measured in overall accuracy (the top panel), detection rate (the middle panel) and false positive rate (the bottom panel). The results of backward selection (the solid lines) were overlaid with those of forward selection (the dashed lines).

4. Discussion

Using frequency-derived statistical measures of byte sequences to detect malicious executables is very different from a program analysis approach. Code analysis examines meaningful information on operation codes, address ranges, function calls, and data flow, either statically or by actually running and monitoring the code to infer potential malicious behavior. "Clean room" execution may lead to the identification of malicious codes not detected by static analysis. Single bytes in binary executable do not retain any interpretable meanings of function, data flow, or even instruction. However, by examining correlations of these byte patterns between malicious and benign programs, we have found that classifiers can accurately distinguish between malicious and benign programs.

In this study, we extracted byte value frequencies from each testing file. Using pattern frequencies, instead of their binary presence or absence, follows the approach of McCallum and Nigam in text categorization (McCallum and Nigam, 1998). We note that the method of normalizing byte pattern frequency across multiple files in the testing data set affects the performance results. For a byte sequence pattern, one can calculate its frequency in each file, and then average the frequencies of the same patterns across all the files in the same class. This approach gives equal weight to all files. Alternatively, one can sum occurrences of the same patterns across all the files, and then get an average frequency of the class. The calculation skews more weight towards patterns that appear in large files. We found that the first approach, normalizing within a file and then within the class so that all files are given equal weight regardless of their size, gave better performance, and we used this method throughout this study.

In all seven feature selection methods we compared, forward selection proved most reliable for selecting the best set of features for a given classifier in most cases. Between classifiers and feature selection methods, we remark that the

overall performance of the models depended more on the choice of classifier than the method of feature selection (see Table 1).

Even though the use of SVMs as classifiers has been successfully applied to text categorization (Joachimes, 1998; Dumais, 1998; Yang, 1999) and malicious executable detection (Kolter and Maloof, 2004), our study of the interaction between the feature selection methods and the classifiers identified several notable advantages for SVM classifiers.

1. *Accuracy*: Compared with other types of classifiers, the SVM classifier gave the best overall performance in terms of detection rate and false positive rate. It was observed that the SVM classifier, using a radial basis function and the backward selection method (see Fig. 6), achieved 96.3% detection rate at only 0.93% false positive rate. This classification accuracy is well comparable to the previous results using boosted decision trees.
2. *Stability*: The performance of the SVM classifiers is less dependent on the number of features used. For instance, as shown in Table 3, when the number of features varied from 20 to 100, the false positive rates changed in a range of 1–5% at 95% detection rate. When all possible 256 features were used, the performance of the classifier deteriorated merely 1–2%, which confirms the reputation of SVMs being able to handle large feature spaces without data overfitting. The high prediction accuracy indicates the features representing malicious and benign programs might well be linearly separable when they are mapped into a high dimensional feature space using an appropriate kernel functions; e.g. a radial basis function.
3. *Training time*: With nearly 4000 training files, plus up to several hundred feature dimensions, the model training time (excluding pre-processing time) is slightly over 1 min (see Table 3).

We also investigated the use of multiple byte sequences as features but found no advantage to doing so. For one thing, multiple byte patterns require a substantial increase in system and computing overhead. Further, using multiple byte patterns makes the distribution of features highly diverse, especially in the malicious executables. For instance, when two-byte patterns were used, retaining the top 1000 most frequently occurring patterns, there were a total of 50,177 unique patterns for the benign programs, and 65,351 patterns for the malicious executables. When 10-byte patterns were used, the total number of unique patterns increased to 551,996 for the benign ones, and 1,789,295 for the malicious. The diversity of multiple byte patterns makes each pattern contribute much less weight to characterize a file. That could explain why, as shown in Table 2, when the number of variables was increased from 20 to 40, the performance of the classifiers using 2-, 5-, and 10-bytes barely changed. This is in contrast to a previous study (Kolter and Maloof, 2004) which found 4-byte patterns optimal.

Besides the high classification accuracy, the proposed feature selection methods and classifiers can be trained on a modestly equipped machine, with relatively short training time, and very short classification time. All experiments of modeling construction and validation were performed on a *Linux*-based desktop computer with dual 1 G Hz Pentium III processors and 512 Megabyte memories. It took less than 2 h to pre-process all 4754 benign and malicious hexadecimal files, extracting single byte patterns, calculating and normalizing their frequencies. The training time varied from tens of seconds to train a SVM classifier to tens of days for a forward selection to screen all 256 single byte patterns. After an optimal model is constructed, using the model to classify the executables took an average of only 2 s for each

Table 4
Performance comparison of SVMs with linear, polynomial, radial basis and sigmoid kernel functions

Kernel type	Performance				
	False positive rate (%)	Detection rate (%)	False positive rate (%)	Detection rate (%)	Average training time (s)
Linear	1.0	90.0	7.0	95.0	22.0
Polynomial	1.0	86.0	4.0	95.0	33.0
Radial basis function	1.0	95.0	1.0	95.0	32.8
Sigmoid	1.0	87.0	6.0	95.0	27.4

The same training data set with 100 variables was used for all classifiers. The performance was benchmarked with respect to two different criteria: (1) when the false positive rate was 1%, the maximal detection rate could be achieved; and (2) as the detection rate was 95%, what could be the corresponding minimal false positive rate? The last column gives the average training time (excluding pre-processing time) of SVMs when different kernel functions were applied.

file in the dataset. Of the first three classifiers compared, namely, Naïve Bayes, entropy, and product based classifier, the product classifier outperformed the other two classification methods using each given feature selection technique. Also, we observed that the product classifier required the least computation overhead of the three non-SVM classifiers. Since the product classifier only uses byte frequencies of an executable as features, and does simple multiplication and logic comparison, we expect that this classifier would be amenable to acceleration using programmable hardware.

In summary, the proposed malicious executable classifiers have the capability to automatically extract features from binary executables, accurately and expeditiously distinguish those malicious files from the benign, and potentially provide another layer of computer and network security to prevent future malicious attacks (Table 4).

Acknowledgment

We would like to thank the anonymous reviewers for their time and very helpful feedback. We are also grateful to Dr. Salvatore Stolfo and the colleagues at the Columbia University for letting us use their data. We also appreciate Drs. Chih-Jen Lin and Chih-Chung Chang for providing the *LIBSVM* package. This work was supported by the LDRD program at Los Alamos National Laboratory.

References

Arnold, W., Tesauro, G., 2000. Automatically generated Win32 heuristic virus detection. Proceedings of the 2000 International Virus Bulletin Conference.

Burges, C.J.C., 1998. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2, 121–167.

Chang, C.-C., Lin, C.-J., 2001. *LIBSVM*: a Library for Support Vector Machines. Software available at ⟨http://www.csie.ntu.edu.tw/∼ cjlin/libsvm/⟩.

Cohen, F., 1987. A cryptographic checksum for integrity protection. Computers and Security 6, 505–510.

Cortes, C., Vapnik, V., 1995. Support-vector network. Machine Learning 20, 273–297.

Crawford, R., Lo, R., Crossley, J., Fink, G., Kerchen, P., Ho, W., Levitt, K., Olsson, R., Archer, M., 1991. A testbed for malicious code detection: a synthesis of static and dynamic analysis techniques. Proceedings of the Department of Energy Computer Security Group Conference, vol. 17, pp. 1–23.

Crocker, S., Pozzo, M.M., 1989. A proposal for a verification to malicious code detection. Proceedings of IEEE Computer Society Symposium on Security and Privacy, pp. 319–324.

Dumais, S., Platt, J., Heckeman, D., Sahami, M., 1998. Inductive learning algorithm and representations for text categorization. Proceedings of the Seventh International Conference on Information Retrieval and Knowledge Management (ACM-CIKM-98), pp. 148–155.

Forman, G., 2003. An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. 3, 1289–1306.

Joachimes, T., 1998a. Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (Eds.), Advances in Kernel Methods-Support Vector Learning. MIT Press, Cambridge, MA.

Joachimes, T., 1998b. Text categorization with support vector machines: learning with many relevant features. European Conference on Machine Learning (ECML), pp. 137–142.

Keerthi, S.S., Shevade, S.K., Bhattacharyya, C., Murthy, K.R.K., 2001. Improvements to Platt's SMO algorithm for SVM classifier design. Neural Comput. 13, 637–649.

Kephart, J.O., Arnold, W.C., 1994. Automatic extraction of computer virus signatures. Proceedings of the 4th International Virus Bulletin Conference, pp. 179–194.

Kohavi, R., John, G.H., 1997. Wrappers for feature subset selection. Artificial Intelligence 97, 273–324.

Kolter, J.Z., Maloof, M.A., 2004. Learning to detect malicious executables in the Wild. KDD'04, pp. 470–478.

Lee, W., Stolfo, S., Mok, K., 1999. A data mining framework for building intrusion detection models. IEEE Symposium on Security and Privacy, pp. 120–132.

Lewis, D., 1992. An evaluation of phrasal and clustered representations on a text categorization task. 15th Annual International ACM SIGER Conference on Research and Development in Information Retrieval, Copenhagen, pp. 37–50.

Lewis, D., Ringuette, M., 1994. A comparison of two learning algorithms for text categorization. Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieval (SDAIR'94), pp. 82–93.

Lo, R., Kerchen, P., Crawford, R., Ho, W., Crossley, J., Fink, G., Levitt, K., Olsson, R., Archer, M., 1991. Towards a testbed for malicious code detection. IEEE Computer Society International Conference, pp. 160–166.

Lo, R., Levitt, K., Olsson, R.A., 1995. MCF: a malicious code filter. Comput. Security 14, 541–566.

McCallum, A., Nigam, K., 1998. A comparison of event models for Naïve Bayes text classification. AAAI-98 Workshop on Learning for Text Categorization.

MacKay, D.J.C., 2003. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, Cambridge, MA.

Platt, J.C., 1998. Fast Training of support vector machines using sequential minimal optimization. In: Schölkopf, B., Burges, C.J.C., Smola, A.J. (Eds.), Advances in Kernel Methods-Support Vector Learning. MIT Press, Cambridge, MA.

Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J., 2001a. Data mining methods for detection of new malicious executables. Proceedings of IEEE Symposium on Security and Privacy, Oakland, CA.

Schultz, M.G., Eskin, E., Stolfo, S.J., 2001b. Malicious email filter—A UNIX mail filter that detects malicious windows executables. Proceedings of USENIX Annual Technical Conference—FREENIX Track, Boston, MA.

Tesauro, G., Kephart, J.O., Sorkin, G.B., 1996. Neural networks for computer virus recognition. IEEE Expert 11, 5–6.

Vapnik, V., 1998. Statistical Learning Theory. Wiley, New York.

Wang, J.-H., Deng, P.S., Fan, Y.-S., Jaw, L.-J., Liu, Y.-C., 2003. Virus detection using data mining techniques. IEEE Annual International Carnahan Conference on Security Technology, pp. 71-77.

Yang, Y., Ghani, R., 2002. A study of approach to hypertext categorization. J. Intelligent Inform. Systems 18(2/3), 219–241.

Yang, Y.Y., Liu, X., 1999. A re-examination of text categorization methods. Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99), pp. 42–49.

Yang, Y., Pedersen, J.O., 1997. A comparative study on feature selection in text categorization. Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97), pp. 412–420.