Detecting unknown computer worm activity via support vector machines and active learning

Nir Nissim, Robert Moskovitch, Lior Rokach, Yuval Elovici

Department of Information Systems Engineering,
Ben Gurion University of the Negev,
P.O.B. 653, Beer-Sheva, Israel 84105
Deutsche Telekom Laboratories at Ben Gurion University,
nirni, robertmo, liorrk, elovici@bgu.ac.il

Abstract. To detect the presence of unknown worms, we propose a technique based on computer measurements extracted from the operating system. We designed a series of experiments to test the new technique by employing several computer configurations and background application activities. In the course of the experiments, 323 computer features were monitored. Four feature-ranking measures were used to reduce the number of features required for classification. We applied support vector machines to the resulting feature subsets. In addition, we used active learning as a selective sampling method to increase the performance of the classifier and improve its robustness in the presence of misleading instances in the data. Our results indicate a mean detection accuracy in excess of 90%, and an accuracy above 94% for specific unknown worms using just 20 features, while maintaining a low false-positive rate when the active learning approach is applied.

1 Introduction

The detection of malicious code (malcode) transmitted over computer networks has been researched intensively in recent years. Worms, a particularly widespread malcode, proactively propagate across networks while exploiting vulnerabilities in operating systems or in installed programs. Other types of malcode include computer viruses, Trojan horses, spyware, and adware.

Nowadays, excellent technology (i.e., antivirus software packages) exists for detecting *known* malicious code. Typically, antivirus software packages inspect each file that enters the system, looking for known signs (signatures) that uniquely identify a malcode. *Antivirus* technology cannot, however, be used for detecting an unknown malcode, since it is based on prior explicit knowledge of malcode signatures. Following the appearance of a new worm instance, operating system providers provide a patch to deal with the problem, while antivirus vendors update their signatures-base accordingly. This solution has obvious demerits, however, since worms propagate very rapidly. By the time the antivirus software has been updated with the new worm, very expensive damage has already been inflicted [1].

Intrusion detection, termed a network-based intrusion detection system (NIDS), is commonly implemented at the network level. NIDS has been substantially researched but remains limited in its detection capabilities (like any detection system). In order to detect malcodes that have slipped through the NIDS at the network level, detection operations are performed locally by implementing a host-based intrusion detection system (HIDS). To monitor activities at the host level, HIDS usually compare various states of the computer, such as the changes in the file system, using checksum comparisons. The main drawback of this approach is the ability of malcodes to disable antiviruses. The main problem in using HIDS, however, is detection knowledge maintenance, which is usually performed manually by the domain expert. This is apt to be time-consuming and expensive.

Recent studies have proposed methods for detecting unknown malcode using machine-learning techniques. Given a training set of malicious and benign binary executables, a classifier is trained to identify and classify unknown malicious executables as malicious [2–4]. While this approach is potentially a good solution, it is not complete. It can detect only executable files, and not malcodes located entirely in the memory, such as the Slammer worm [5]. In a previous research report, we presented a new method for detecting unknown computer worms [6, 7]. The underlying assumption was that malcode within the same category (e.g., worms, Trojans, spyware, adware) share similar characteristics and behavior patterns and that these patterns can be induced using machine-learning techniques. By continuously monitoring and matching the computer's vital signs (such as CPU and hard disk usage) against the previously induced malcode patterns, we can gain an indication as to whether the computer is infected. While this approach does not prevent infection, it enables its fast detection. Relevant decisions and policies, such as disconnecting a single computer or a cluster, can then be implemented.

The goal of this study is to assess the viability of employing support vector machines (SVM) in an individual computer host to detect *unknown* worms based on their behavior (measurements), and examine whether selective sampling can improve the detection performance. The behavior of some of the worms is unstable, so that some of the time they tend to behave as a legitimate application does. Thus by monitoring their behavior we would derive instances that would negatively affect the model (hereafter we will call these instances it misleading instances). The selection of the right instances to be included in the training-set is therefore also very challenging.

This paper makes four contributions to our armory for combating malware:

1. Development of a selective sampling procedure using active learning: Active learning is commonly used to reduce the amount of labeling required from an expert (a time-consuming task). The Oracle is actively asked to label specific examples in the dataset that the learner considers the most informative, based on its current knowledge, which eventually reduces the acquisition cost. In this study, all the training examples are labeled in advance. However, the

goal is to select intelligently the best examples that will increase the accuracy by filtering misleading or non-informative instances.
2. Adaptation of support vector machines for malware detection: In our previous study [6] we used the algorithms decision trees, naïve Bayes, Bayesian networks, and neural networks. In this paper, we study the performance of support vector machines (SVM). Specifically, we examine which of the three kernel functions (linear, polynomial, RBF) is most suitable for detecting unknown worms. We argue that SVM will achieve better results when using active learning as a selective sampling method.
3. Comparison of feature selection methods for improving malware detection: We examine experimentally which feature selection method (if any) best fits the worm detection task using SVM.
4. We investigate the contribution of specific worms to the detection performance and examine if all worms are equally informative.

The rest of the article is structured as follows. Section 2 surveys the relevant background for this work, while Section 3 describes the support vector machines, relevant kernel functions and active learning methods used in this study. Section 4 discusses the research question, corresponding experimental plan, and evaluation results. Finally, in Section 5 we conclude the paper with a discussion of the evaluation results, conclusions, and future work.

2 Background and Related Work

2.1 Malicious Code and Worms

The term 'malicious code' (malcode) refers to a piece of code, not necessarily an executable file, the intention of which is to harm. In [8], the authors define a worm according to how it can be distinguished from other types of malcode: 1) network propagation or human intervention - worms propagate actively over a network, while other types of malicious codes, such as viruses, commonly require human activity to propagate; 2) standalone or file infecting - while viruses infect a host, a worm does not require a host file and sometimes does not even require an executable file since it may reside entirely in the memory. This was the case with the Code Red worm [9].

Worm developers have different purposes and motivations [10]. Some are motivated by experimental curiosity (ILoveYou worm [11]), while pride and the desire for power lead others to flaunt their knowledge and skill through the harm caused by the worm. Still other motivations are commercial advantage, extortion and criminal gain, random and political protest, and terrorism and cyber warfare.

The wide variety of motivations that we find among worm developers indicates that computer worms will be a long-lasting phenomenon. To address the challenge posed by worms effectively, as much meaningful experience and knowledge as possible should be extracted from known worms by analyzing them. Today, given the number of known worms, we have an excellent opportunity to

learn from these examples. We argue that active learning methods can be very useful for learning and generalizing from previously encountered worms in order to detect previously unseen worms effectively.

2.2 Detecting Malicious Code Using Supervised Learning Techniques

Supervised learning techniques have already been used for detecting malicious codes and creating protection against them. For example, in [12], the authors proposed a framework consisting of a set of algorithms for extracting anomalies from a user's normal behavior patterns. A normal behavior is learned and any abnormal activity is considered intrusive. In order to determine what constitutes normal, the authors suggest several techniques, such as classification, meta-learning, association rules, and frequent episodes. The extracted knowledge forms the basis of an anomaly-based intrusion detection system.

A naïve Bayesian classifier was suggested in [13], referring to its implementation within the ADAM system, developed in 2001 [14]. The classifier consists of three main parts: (a) a network data monitor listening to TCP/IP protocol; (b) a learning engine for acquiring association rules from the network data; and (c) a classification module that classifies the nature of the traffic into two possible classes, normal and abnormal, that can later be linked to specific attacks. Other soft computing algorithms proposed for detecting malicious code include: artificial neural networks (ANN) [15–18]; self-organizing maps (SOM) [19], and fuzzy logic [20–22].

2.3 Active Learning

Labeled examples are crucial when training classifiers. However, in certain domains the labeling operation is costly and time-consuming. Active learning (AL) [23] refers to learning policies, in which a learner actively selects unlabeled instances for labeling, based on some criterion. The objective of most AL methods is to minimize the cost of acquiring the labeled instances needed for inducing an accurate model. In this paper, we scrutinize another aspect of AL. Instead of minimizing the acquisition costs, our objective is to increase the generalization accuracy by using an approach that disregards misleading instances. Several AL frameworks are presented in the literature. In pool-based active learning [24], the learner has access to a pool of unlabeled data and can request the true class label for a certain number of instances in the pool. Other approaches focus on the expected improvement of class entropy [25], or minimizing both labelling and misclassification costs [26]. Although in our problem all the examples are actually labeled, we decided to apply AL as the selective sampling approach for choosing the most informative examples to reduce the number of the misleading instances in the training data. In section 3.4, we explain how AL can be used to achieve this goal.

3 Methods

3.1 Dataset Creation

Since no benchmark dataset exists that we could use for this study, we created our own. A controlled network with various computers (configurations) was deployed into which we could inject worms, and monitor and log the computer operating system features using a dedicated agent. In order to create the datasets, we isolated the local network of computers, simulating a real Internet network that allowed worms to propagate.

We designed several experiments centered around eight datasets, which we created based on three aspects: hardware configuration, background applications, and user activities. Using this model, we designed several experiments to achieve our research goals:

a. To find out whether a classifier, trained on data collected from a computer with a certain hardware configuration and specific background activity, is capable of classifying correctly the behavior of a computer that has other configurations.
b. To select the minimal subset of features required to classify new cases correctly. Reducing the number of features used in the model implies that less monitoring effort would be needed in an operational system.

In the course of experimentation, we applied four classification algorithms on the given datasets in a varied series of experiments in order to detect, first, known worms in different environments and, later, completely new, previously unseen worms.

Figure 1 depicts the process that was used in this study. The upper part refers to the training phase. We collected a set of worms and used them to infect the hosts in the controlled environment. An agent, which was installed on each host, then recorded its behavior. Based on the collected dataset, we trained the classifiers. The bottom part of Figure 1 refers to the test phase. In this phase, we examined whether the induced classifier can be used to identify the existence of an unknown worm.

Environment Description The laboratory network consisted of seven computers, which contained heterogenic hardware, and a server computer simulating the Internet. We used the Windows performance counters [1] which enabled us to monitor system features that appeared in the following categories (the number of features in each category appears in parentheses): Internet Control Message Protocol (27), Internet Protocol (17), Memory (29), Network Interface (17), Physical Disk (21), Process (27), Processor (15), System (17), Transport Control Protocol (9), Thread (12), User Datagram Protocol (5). In addition, we used VTrace [27], a software tool that can be installed for monitoring purposes on a PC running

[1] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/counter/counters2_lbfc.asp

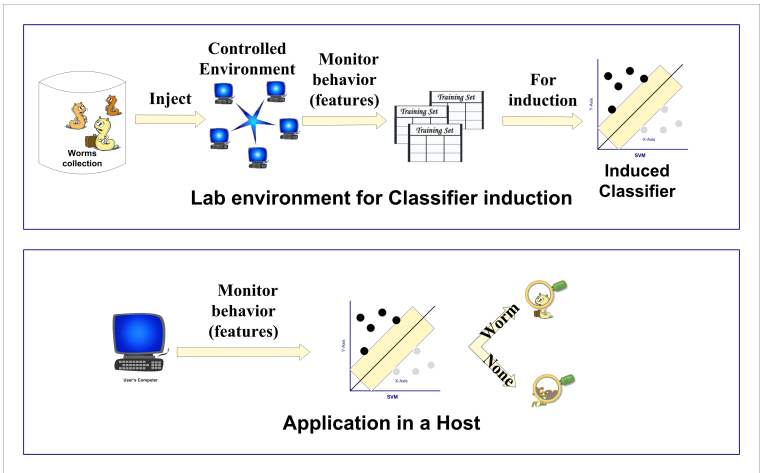

Fig. 1. Outline of the Train phase and the Test Phase. The worms are injected into the computers, which are monitored. Features are extracted and a SVM classifier is induced. In the test step the monitored features are provided to the classifier, which classifies whether there is worm activity or not.

Windows. VTrace collects traces of the file system, the network, the disk drive, processes, threads, inter-process communication, cursor changes, etc. The Windows performance counters were configured to measure the features every second and to store them in a log file as a vector. VTrace stored, time-stamped events were aggregated into the same fixed intervals, and merged with the Windows performance log files. This body of data eventually consisted of a vector of 323 features collected every second. We worked with this granularity because these loggers' most granular level was one second. Larger time windows, in which we could aggregate the measurements over longer time periods, might have been too slow for worm activity detection.

Injected Worms When selecting worms for injection, we tried to include every variety. Some of the worms had a heavy payload of Trojans for installation, in parallel, on the distribution process of the network; others focused entirely on distribution. Another feature that we desired to obtain was that the worms would have different strategies for IP scanning that would result in varying communication behaviors, CPU consumption, and network usage. While all the worms were different, we wanted to find common characteristics, which could be used to detect an unknown worm. We briefly describe here the main characteristics of the five worms included in this study. The information is based on the virus libraries on the Web[2] [3] [4]

[2] Symantec - www.symantec.com
[3] Kasparsky - www.viruslist.com
[4] Macfee - http://vil.nai.com

W32.Dabber.A
This worm randomly scans IP addresses and uses the W32.Sasser.D worm to propagate and open the FTP server in order to upload itself to the vicitom's computer. The worm registers itself for implementation at the next user login (human-based activation). It drops a backdoor, which listens in on a predefined port. This worm is distinguished by its use of an external worm in order to propagate.
W32.Deborm.Y
W32.Deborm.Y is a self-carried worm that prefers local IP addresses. It registers itself as an MS Windows service and is implemented upon user login (human-based activation). This worm contains and implements three Trojans as a payload: Backdoor.Sdbot, Backdoor.Litmus, and Trojan.KillAV. We chose this worm because of its heavy payload.
W32.Korgo.X This is a self-carrying worm that uses a completely random method for scanning IP addresses. It is self-activated and tries to inject itself via a new thread of MS Internet Explorer. It contains a payload code that enables it to connect to predefined websites in order to receive orders or download newer worm versions.
W32.Sasser.D
W32.Sasser.D has a preference for local address optimization while scanning the network. It divides its time, approximately half and half, between scanning local addresses and random addresses. In particular, it opens 128 threads for scanning the network. This requires heavy CPU consumption, as well as significant network traffic. It is a self-carried worm and uses a shell to connect to the infected computer's FTP server and to upload itself.
W32.Slackor.A This is a self-carried worm that propagates by exploiting MS Windows' sharing vulnerability to propagate. The worm registers itself for execution upon user login. It contains a Trojan payload and opens an IRC server on the infected computer in order to receive orders.

Computers Measurements We examined the influence of computer hardware configuration, applications running in the background, and user activity.

1. Computer hardware configuration: We used two different configurations. Both ran on Windows XP, which is considered the most widely used operating system, having two configuration types: the "old" configuration has a Pentium 3 800Mhz CPU, a bus speed of 133Mhz, and 512 Mb memory; the "new" configuration has a Pentium 4 3Ghz CPU, a bus speed of 800Mhz, and 1 Gb memory.
2. Background application: We ran an application that affects mainly the following features: processor object, processor time (usage of 100%); page faults/sec; physical disk object; average disk bytes/transfer avg disk bytes/write, and disk writes/sec.
3. User activity included several applications, among them: browsing, downloading, and streaming operations through Internet Explorer, Word, Excel, chat through MSN messenger, and Windows Media Player. These activities

were implemented in such a way as to imitate user activity in a scheduled way.

The data were collected in the presence or absence of Background Application and User Activity in each of the hardware configurations. We therefore had three binary aspects, which resulted in eight possible feature-collecting conditions, shown in Table 1, representing a variety of dynamic computer configurations and usage patterns. Each dataset contained monitored instances of all the five injected worms separately, and instances of *normal* computer behavior without any injected worm. Each instance was labeled with the relevant worm (class), or 'none' for "clean" instances; Each worm was monitored for a period of 20 minutes with a resolution of 1 second. Thus, each instance contained a vector of measurements that represents a 1 second snapshot. Accordingly, each dataset contained a few thousand such labeled instances. Worms and legitimate applications were monitored in different configurations (computer hardware configuration, existence of background application and also existence user-activity). The outcome of this monitoring process was features that represent the application's (worm/non worm) behavior. Some of the worms tend to behave in one environment similarly to a legitimate application in another environment; similarly, a legitimate application might be perceived as non legitimate when its behavior is monitored in different environments. Thus, these cases are also a source of misleading instances in the data. In order to derive a training set that included applications with distinctive behavior in any environment, we chose to disregard applications whose behavior is not stable in all the environments.

Computer	Background Application	User Activity	Dataset Name
Old	No	No	o
Old	No	Yes	ou
Old	Yes	No	oa
Old	Yes	Yes	oau
Old	Yes	Yes	oau
New	No	Yes	nu
New	Yes	No	na
New	Yes	Yes	nau

Table 1. The three aspects resulting in eight datasets, representing a variety of feature collecting conditions of a monitored computer

3.2 Feature Selection

In machine-learning applications, the large number of features in many domains presents a huge challenge. Typically, some of the features do not contribute to the accuracy of the classification task and may even hamper it. Moreover, in our approach, reducing the amount of features while maintaining a high level

of detection accuracy is crucial for meeting computer and resource consumption requirements for the monitoring operations (measurements) and the classifier computations. This state can be achieved using the feature selection technique. Since this is not the focus of this paper, we will describe the feature selection preprocessing only very briefly. In order to compare the performance of the different kernels in the SVM, we used the filter approach, which is applied on the dataset and is independent of any classification algorithm (unlike wrappers, in which the best subset is chosen using an iterative evaluation experiment). Using filters, a measure was calculated to quantify the correlation of each feature with the class, in our case, the presence or absence of worm activity. Each feature received a rank representing its expected contribution in the classification task. Finally, the top ranked features were selected.

Feature Ranking Metrics We used three feature metrics, which resulted in a list of ranked features for each metric and an ensemble incorporating all three of them. We used chi-square (CS), gain ratio (GR) and RELIEF implemented in the WEKA environment [28] and their ensemble.

Chi-Square

Chi-Square measures the lack of independence between a feature f and a class c_i (such as W32.Dabber.A) and can be compared to the chi-square distribution with one degree of freedom to judge extremeness. Equation 1 shows how the chi-square measure is defined and computed, where N is the total number of documents, f refers to the presence of the feature (and $\bar{f}$ its absence), and c_i refers to its membership in c_i. $P(f, c_i)$ is the probability that the feature f occurs in c_i and the probability $P(\bar{f}, c_i)$ is the probability that the feature f does not occur in c_i. Similarly, $P(f, \bar{c}_i)$ and $P(\bar{f}, \bar{c}_i)$ are the probabilities that the feature does or does not occur in a file that is not labeled to c_i, respectively. $P(f)$ is the probability that the feature appears in a file, and $P(\bar{f})$ is the probability that the feature does not appear in the file. $P(c_i)$ is the probability that a file is labeled to c_i, and $P(\bar{c}_i)$ is the probability that a file is not to be labeled to class c_i.

$$\chi^2(f, c_i) = \frac{N[P(f, c_i)P(\bar{f}, \bar{c}_i) - P(f, \bar{c}_i)P(\bar{f}, c_i)]^2}{P(f)P(\bar{f})P(c_i)P(\bar{c}_i)}. \quad (1)$$

Gain Ratio

Gain Ratio (GR) was originally presented in 1993 [29] in the context of decision trees [30]. It was designed to overcome a bias in the information gain (IG) measure, and measures the expected reduction of entropy caused by partitioning the examples according to a chosen feature. Given entropy $E(S)$ as a measure of the impurity in a collection of items, it is possible to quantify the effectiveness of a feature in classifying the training data. Equation 3 presents the entropy of a set of items S, based on C subsets of S (for example, classes of the items), presented by S_c. IG measures the expected reduction of entropy caused by partitioning the examples according to attribute A, in which V is the set of possible values of A,

as shown in Equation 2. These equations refer to discrete values; however, it is possible to extend them to continuous valued attribute.

$$IG(S,A) = E(S) - \sum_{v \in V(A)} \frac{\mid S_v \mid}{\mid S \mid} \times E(S_v) \tag{2}$$

$$E(S) = \sum_{c \in C} -\frac{\mid S_c \mid}{\mid S \mid} \cdot \log_2 \frac{\mid S_c \mid}{\mid S \mid}. \tag{3}$$

The IG measure favors features having a high variety of values over those with only a few. GR overcomes this problem by considering how the feature splits the data (Equations 4 and 5). S_i are d subsets of examples resulting from partitioning S by the *d-valued* feature A.

$$GR(S,A) = \frac{IG(S,A)}{SI(S,A)} \tag{4}$$

$$SI(S,A) = -\sum_{i=1}^{d} \frac{\mid S_i \mid}{\mid S \mid} \cdot \log_2 \frac{\mid S_i \mid}{\mid S \mid}. \tag{5}$$

Relief

Relief [31] estimates the quality of the features according to how well their values distinguish between instances that are near each other. Given a randomly selected instance x, from a dataset s with k features, Relief searches the dataset for its two nearest neighbors from the same class, an action termed "nearest hit H", and from a different class, referred to as "nearest miss M". The quality estimation $W[A_i]$ is stored in a vector of the features A_i, based on the values of a difference function $diff()$ given x, H and M as shown in Equation 6.

$$diff(A_i, x_{1i}, x_{2i}) = \begin{cases} \mid x_{1i} - x_{2i} \mid & \text{if } A_i \text{ is numeric,} \\ 0 & \text{if } A_i \text{ is nominal and } x_{1i} = x_{2i}, \\ 1 & \text{if } A_i \text{ is nominal and } x_{1i} \neq x_{2i}. \end{cases} \tag{6}$$

Features Ensembles

Instead of selecting features based on feature selection methods, one can use the ensemble strategy (see for instance [32–34]), which combines the feature subsets that are obtained from several feature selection methods. Specifically, we combine several methods by averaging the feature ranks as shown in Equation 7:

$$rank(f_i) = \frac{\sum_{j=1}^{k} rank^j(f_i)}{k} \tag{7}$$

where f_i is a feature and *filter* is one of the k filtering (feature selection) methods. Specifically in our case $k = 3$.

3.3 Support Vector Machines

We employed the SVM classification algorithm [35] using three different kernel functions in a supervised learning approach. We now briefly introduce the SVM classification algorithm and the principles and implementation of the active learning method we used in this study. SVM is a binary classifier that finds a linear hyperplane that separates the given examples into the two given classes. SVM is known for its capability to handle a large amount of features, such as text. We used the SVM-light implementation [36], given a training set in which an example is a vector $x_i = \langle f_1, f_2 \ldots n \rangle$ labeled by $y_i = \{-1, +1\}$ where f_i is a feature.

The SVM attempts to specify a linear hyperplane that has a maximal margin, defined by the maximal (perpendicular) distance between the examples of the two classes. Figure 2 illustrates a two-dimensional space, in which the examples are located according to their features; the hyperplane splits them according to their label. The examples lying closest to the hyperplane are the "supporting vectors". W, the normal of the hyperplane, is a linear combination of the most important examples (supporting vectors), multiplied by LaGrange multipliers (alphas).

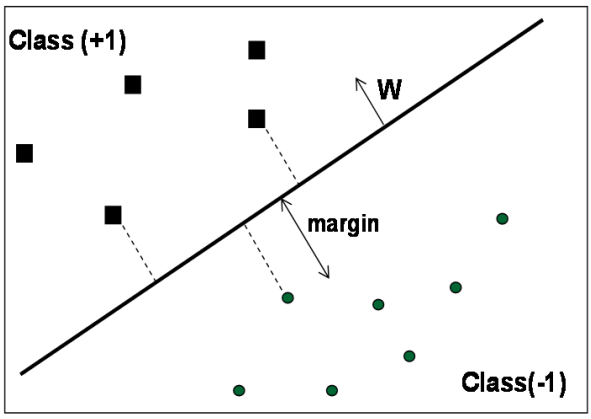

Fig. 2. SVM that separates the training set into two classes with maximal margin

Since the dataset in the original space often cannot be separated linearly, a kernel function K is used. By using a kernel function, the SVM actually projects the examples into a higher dimensional space in order to create linear separation of the examples. Note that when the kernel function satisfies Mercer's condition [37], then K can be written as shown in Equation (8), where Φ is a function that maps the example from the original feature space into higher dimensional space, while K captures the "inner product" between the mappings of examples x_1, x_2.

For the general case, the SVM classifier will be in the form shown in Equation (9). Note that n is the number of examples in the training set. Equation 10 defines W.

$$K(x_1, x_2) = \Phi(x_1) \cdot \Phi(x_2) \tag{8}$$

$$f(x) = w \cdot \Phi(x) = \sum_{1}^{n} \alpha_i y_i K(x_i x) \tag{9}$$

$$w = \sum_{1}^{n} \alpha_i y_i \Phi(x_i) \tag{10}$$

The use of kernel functions, often referred to as the *kernel trick* [38], is of great importance. Equation (8) means that inner products in the higher dimensional space can be evaluated simply using the kernel function; it is therefore not necessary to work explicitly in the higher dimensional whenever only inner products are required. Therefore, the problem that arises from the high dimensional feature space is alleviated, because it allows the computations to take place in the original feature space of the problem, which involves the computation of inner products in Equation (8). After projecting the examples into the higher dimension space, the SVM tries to identify the optimal hyperplane that separates the two classes. Logically there can be more than one separating hyperplane for a specific projection of a dataset; therefore, as a criterion of selection, the one maximizing the margin is selected in an attempt to achieve a better generalization capability in order to increase the expected accuracy.

Since the kernel function is derived from the theoretic basis of SVM, one should select a kernel function that has the appropriate parameter configurations, as was empirically demonstrated in [36].

Each kernel function creates a different separating plane in the original space as demonstrated in Figure 3 and 4. Commonly, the *kernel* functions used are the *Polynomial* and *RBF kernel*. One should note that the performance of the kernel also depends on the true data distribution, which is usually unknown, and thus one should scrutinize different kernels in order to determine the best kernel for the specific problem and dataset.

Polynomial kernel

The polynomial kernel creates values of degree p, where the output depends on the direction of the two vectors (examples x_1, x_2), as shown in Equation 11, in the original problem space. A private case of the polynomial kernel, having $P = 1$, is actually the *linear kernel*.

$$K(x_1, x_2) = (x_1 \cdot x_2 + 1)^P \tag{11}$$

In order to convey the significance of the kernels, we provided an explanation combined with visualizations. As can be seen in Figure 3, the same training set is given to SVM with *linear* and *polynomial* kernels. While the SVM with a linear

kernel (right side) has not determined a hyperplane that separates the training set, the SVM with the *polynomial* kernel (left side) has successfully determined such a one:

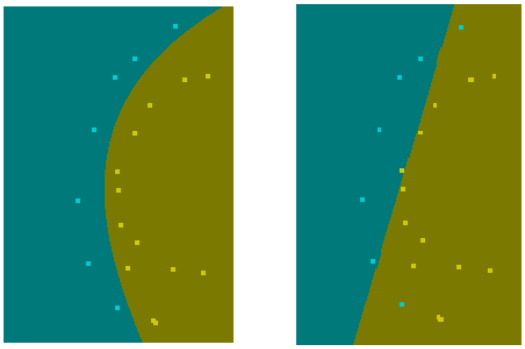

Fig. 3. The same training set was given to *polynomial* (left) and *linear* kernels (right); the *polynomial* kernel achieved better separation with its induced model. The figure was produced by the applet provided in the LIBSVM software [39].

Radial Basis Function (RBF) kernel

The second most used kernel is a *radial basis function* (RBF), as shown in Equation 12, in which a Gaussian is used as the RBF and the output of the kernel depends on the Euclidean distance of examples x_1, x_2.

$$K(x_1, x_2) = e\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right). \tag{12}$$

As Figure 4 shows, the same training set is given to the SVM with the *RBF* and the *polynomial* kernels. While the SVM with the polynomial kernel (left side) has not determined the hyperplane that separates the training set, the SVM with the *RBF* kernel (right side) has successfully determined such a one:

The *RBF* has successfully separated the dataset, whereas the *polynomial* has not.

There are several reasons for using the SVM as the classification algorithm. Primarily, SVM was successfully used for the detection of worms as indicated in four previous works [40–43]. Moreover, in the first work [40], it was indicated that "SVM learns a black-box classifier that is hard for worm writers to interpret". In addition, SVM was very efficient when combined with AL methods in closely related domains, as was presented in [44]. An additional reason is related to the fact that the data contain misleading data (the misleading issue will be explained in more detail in Section 3.4). In a nutshell, misleading data means that it is hard to find a clear separation in the dataset between the worm and non-worm instances due to the similarity between the behaviors of these two classes. Thus,

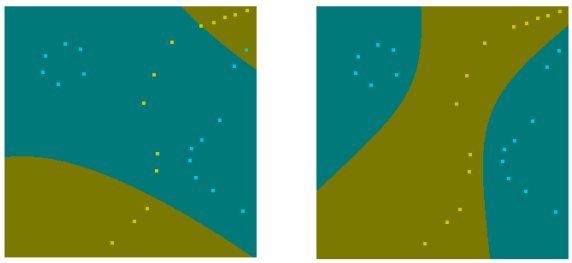

Fig. 4. The same training set was given to *polynomial* (left) and *RBF* kernels (right), the *RBF* kernel achieved better separation with its induced model. The figure was produced by the applet provided in the LIBSVM software [39].

our goal was to detect the worm activity through system behavior, since, on the one hand, the RBF kernel might help due to the fact that it is sophisticated and very sensitive to misleading data, and, on the other hand, the linear kernel, which is the simplest one, might find a simple separation between the classes.

3.4 Active Learning

Active learning (AL) is usually used to reduce the effort expended on labeling examples, generally a time-consuming and costly task, while obtaining a high accuracy rate. In AL, the learner actively acquires the labels of the most informative instances. In our study, all the examples were labeled, since we knew which worm was active during the measurements. However, since the data were misleading, we used AL as a selective sampling technique that increases accuracy by selecting only those examples that lead to a better classifier. Some of the worms behave as a legitimate application part of the time, and as a consequence, they generate misleading instances. In order to prevent these instances from effecting the detection accuracy of our model negatively, we did not select them for inclusion in the training set. The worms are not always active and even when active they do not always behave in an illegitimate way. Thus, according to their monitored behavior in this period, they seem to act like a non-worm instances. This creates misleading instances in the dataset and makes their detection much more difficult. The misleading data here were a greater problem; in the security domain, worms are created in such a sophisticated way that they behave similarly to a legitimate application in order to make their detection harder. Thus, monitoring worm behavior creates many instances (snapshots) that are very similar to non-worm instances and are therefore considered as misleading instances that confuse the SVM. This phenomenon is called "malicious noise", as presented in [45]. In most other domains, the misleading instances are not purposefully created, but exist naturally. Here in worm detection, the task is more complicated since the misleading data are inherent in the class, and even present in a larger amount in the class we want to detect. Here we used the AL

idea to select the most informative instances among the existing ones so that the misleading ones would not be selected, as was done in previous work [46] and discussed in other work [47].

Misleading instances usually create confusion in the classification processes and cause degradation in the classifier's performance. Thus, these misleading instances, generally, will not meet the AL selection criterion, that is, for the Error reduction method, the instances whose addition to the training set will create a classifier that is more confident in its capability to classify unknown instances correctly. Selecting the misleading instances has the opposite effect: it changes the decision function of the classifier so that the classifier is less accurate and thus less confident in its capability to classify the unknown instances correctly.

In this study, we implemented an AL, termed "error-reduction" [25] the objective of which is to reduce the expected *generalization error*. The task is to select the most contributive examples for the classifier from a given pool of unlabeled examples denoted by P. By estimating the expected error reduction achieved through labeling an example and adding it to the training set, the example that has the maximal error reduction will be selected for true labeling and will also be added to the training set of the current actual classifier. Since the real future error rates are unknown, the learner utilizes its current classifier in order to estimate these errors, as will now be elaborated. At the beginning of an AL trial, an initial classifier $\hat{P_D}(y \mid x)$ is trained over a randomly selected initial set D, and for every candidate example (x_i, y_1) where $x_i \in P$ and its possible labels $y_1 \in Y$, the algorithm induces a new classifier $\hat{P_D}(y_i \mid x_i)$ trained on the extended training set $D' = D + (x_i, y_i)$. One should note that this new classifier actually represents the addition of the new example with a specific label into the training set - and these are the classifiers by which the selection criterion is being calculated in order to select the most suitable examples.

$$E_{\hat{P_{D'(y_i|x_i)}}} = \frac{1}{P} \sum_{x \in P} \sum_{y \in Y} \hat{P_{D'(y_i|x_i)}}(y \mid x) \cdot \mid \log \left(\hat{P_{D'(y_i|x_i)}}(y \mid x) \right) \mid \tag{13}$$

$$SE_{x_i} = \sum_{y \in Y} \hat{P_D}(y_i \mid x_i) \cdot E_{\hat{P_{D'(y_i|x_i)}}} \tag{14}$$

The future expected generalization error of the new classifier is then estimated using the entropy of the new induced classifier itself, averaged over $\mid P \mid$, as given in Equation 13. From Equation 13, it can be understood that the error calculation is being done over all the examples in the pool, and the more confident the new classifier $\hat{P_{D'(y_i|x_i)}}$ is in knowing x's true label, the lower is the expected error. For example, $\hat{P_{D'(y_i|x_i)}}(y \mid x) = 1$ means that the classifier is confident that y is the true label of x; thus, error $= 0$, and $\hat{P_{D'(y_i|x_i)}}(y \mid x) = 0$ means it is confident that y is not the true label of x and error=0, while $\hat{P_{D'(y_i|x_i)}}(y \mid x) = 0.5$ means that the classifier is most uncertain that y is the true label of x, and thus the error is maximal. Note that Equation 13 actually calculates the expected error of the new classifier over all the examples in the pool. Yet it is not enough,

since it was calculated only for one label of x_i; thus, for each candidate example, Equation 13 is calculated one time for each possible label and it is averaged using Equation 14 (in our context, there are only two labels: worm and non worm). Equation 14 is the self-estimated average error of candidate example x_i denoted by SE_{x_i}, which is actually a weighted average of the error for all the examples and its possible labels, as shown in Equation 13. The example x_i with the lowest expected self-estimated error (SE_{x_i}) is chosen and added to the training set. In brief, an example is chosen from the pool only if it dramatically improves the confidence of the current classifier for all the examples in the pool.

3.5 Evaluation Measures

For evaluation purposes, we measured: the *true positive rate (TPR)* measure, which is the number of *positive instances* correctly classified, as shown in Equation 15; the *false positive rate (FPR)*, which is the number of misclassified *negative* instances (Equation 15); and the *Total Accuracy*, which measures the number of absolutely correct classified instances, either positive or negative, divided by the entire number of instances shown in Equation 16.

$$TPR = \frac{\mid TP \mid}{\mid TP + FN \mid}; \quad FPR = \frac{\mid FP \mid}{\mid FP + TN \mid} \tag{15}$$

$$Total\ Accuracy = \frac{\mid TP + TN \mid}{\mid TP + FP + TN + FN \mid} \tag{16}$$

We also measured a *confusion matrix*, which depicts the number of instances from each class that were classified in each one of the classes (ideally all the instances would be in their actual class).

4 Experiments and Results

In the first part of the study, our objective was to identify the best feature selection measure, the best kernel function, and the minimal features required to maintain a high level of accuracy. In the second part, we wanted to measure the possibility of classifying unknown worms using a training set of known worms, and to examine the possibility of increasing the detection performance using selective sampling. In order to elucidate these issues, we designed three experimental plans. We applied four different feature selection measures to generate seventeen training sets such that each measure was used to extract the *Top 5, 10, 20 and 30*. In addition, the *full* feature set was also used.

4.1 Experiment I - Analysis the Effect of Feature Selection

We performed a wide set of experiments in order to evaluate how feature selection affects the detection of unknown worms. Specifically, we compared the accuracy

performance obtained after selecting features by using each one of the above-mentioned feature-ranking metrics (chi-square, gain ratio, relief, and ensemble). Note that in this experimental study we examined the task of identifying worm or no-worm activity. In certain scenarios, this binary classification is sufficient. While identifying the exact worm is considered to be more challenging, we decided to explore this direction because of the possibility of obtaining additional insights. The evaluations were performed on different conditions, based on the following factors:

1. Top n - Select top (best) 5, 10, 20, 30 or all features according to the features ranking;
2. Feature consolidation (unified, averaged). In the first option, features were ranked on a *unified* dataset, which contains all the eight datasets presented in Table 1. In the second option, features were ranked separately on each dataset. We then computed the average rank for each feature;
3. SVM Kernel Functions: Linear, Polynomial and RBF kernel;
4. Training set (selected from the eight datasets in Table 1) for inducing the SVM classifier;
5. Test set (selected from Table 1) for evaluating the SVM classifier.

When the training and test sets were collected under the same conditions (i.e., the same computer configuration, background application, and user activity), we employed a *ten-fold cross-validation procedure* for evaluating the accuracy. In all other cases, we simply used the entire training/test set for the corresponding training/testing. To analyze the results, we performed a factorial ANOVA. Section 4.1.1 presents the results obtained when the training and test set were collected in the same condition. Section 4.1.2 presents the results for all other cases.

Training and Test on the same feature collecting condition Figure 5 presents the accuracy obtained by different feature ranking measures on different features subset sizes. The results indicate that for this scenario feature selection reduces the accuracy. The GainRatio measure was particularly less effective in selecting the most relevant features, especially in small subsets (top 5 and top 10). The null-hypothesis, that all feature-ranking measures perform equally and the observed differences are merely random, was rejected. Similarly, the null-hypothesis that all feature subset sizes perform equally was also rejected. Moreover, the interaction between the feature-ranking measure and the top select features was found to be statistically significant with $F(12, 752) = 7.9533$ and $p < 1\%$.

The trend in the results is that the more features added to the training set, the higher the accuracy of worm detection is, where here it is strongly related to the fact that the same feature collecting conditions were used for the training and test sets. The more features given to the classifier, the more information the classifier receives. Consequently, due to the fact that over a small set of features some instances might appear similar, while expanding this set of features with

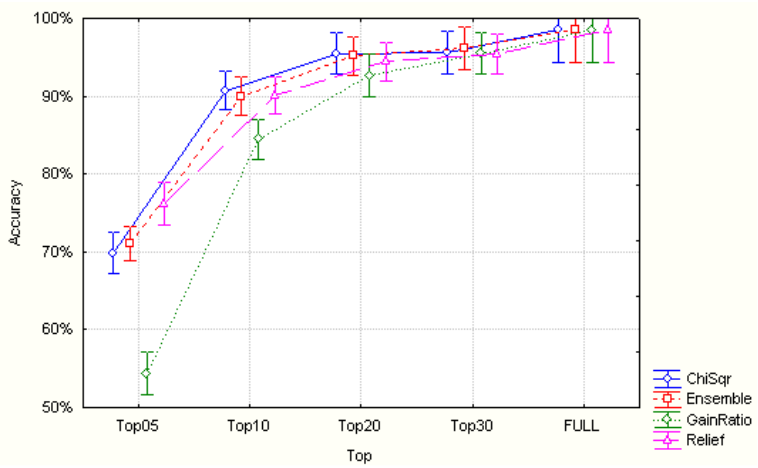

Fig. 5. The interaction between feature-ranking measures and the top selected features. In general the FS reduces the accuracy, but the chi-square was found most effective. The vertical bars denote 0.95 confidence intervals.

additional features will probably reveal a difference between them (if indeed it exists), these additional features help the classifier to cope with the Misleading instances. In addition, it implies that the features we have monitored are relevant and actually help the classifier find patterns in the dataset that are necessary for distinguishing between a worm and non-worm behavior.

Figure 6 presents the mean accuracy obtained by the different kernel functions using different feature subset sizes. The results indicate that the polynomial function performs best in terms of accuracy. The null-hypothesis, that all kernel functions perform equally and the observed differences are merely random, was rejected. Moreover, the interaction between the kernel function and the top select features was found to be statistically significant with $F(7, 716) = 7.483$ and $p < 1\%$.

Again, it can be seen that the amount of features that are taken into consideration has a positive influence on the detection rate, and, although there are differences in the accuracy rates among the various kernels, the trend is quite similar: a steep incline in the accuracy when moving from top 5 to top 10, while from top 10 to Full there is a moderate increase. That the linear kernel achieved significantly lower results (mostly under top 20) implies a complicated dataset that cannot be easily separated linearly: the instances of the worm and non-worm tend to be similar and thus more features are needed to distinguish between them.

Figure 7(a) shows a comparison of the accuracy obtained by the unified and the averaged consolidation approaches. The one-way ANOVA indicates that the averaged approach significantly outperforms the unified approach with $F(1, 770) = 85.1$, $p < 1\%$. Nevertheless, a further investigation, shown in Figure 7(b), indi-

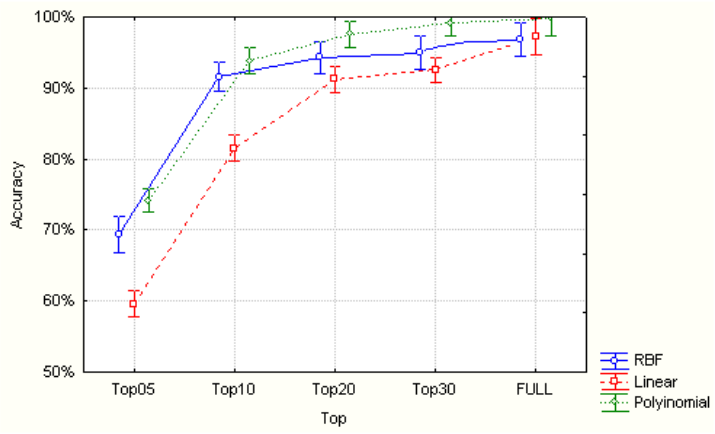

Fig. 6. The interaction between the kernel function and the top selected features. The vertical bars denote 0.95 confidence intervals.

cates that the averaged approach outperforms the unified approach only when small feature sets are used. For the Top 30 and FULL feature sets the unified approach was found to be better. Moreover, the interaction effect of the consolidation factor and feature subset size factor was found to be statistically significant with $F(3, 700) = 10.474$ and $p < 1\%$.

Training and Test on different feature collecting conditions We trained each classifier on a single dataset and tested on it each one of the remaining *seven* datasets. Thus, we had a set of eight iterations in which a dataset was used for training, and seven corresponding evaluations of each one of the datasets. In short, there were 56 evaluation runs for each combination.

Figure 8 presents the accuracy obtained by various feature-ranking measures on different features subset sizes. As expected, it can be seen that the accuracy level in this case is significantly lower than the accuracy obtained in section 4.1.1. Contrary to the results appearing in Figure 5, the GainRatio measure outperforms other measures. Generally, it can be seen that the above 20 features do not improve performance, possibly because the additional features correlate less with the classes. The *Top5* significant features when using *GainRatio* included: A_1ICMP: *Sent_Echo_sec, Messages_Sent_sec, Messages_sec*, and A_1TCP: *Connections_Passive and Connection_Failures*, which are Windows' performance counters, related to ICMP and TCP, describing general communication properties.

The null-hypothesis, that all feature-ranking measures perform equally and that the observed differences are merely random, was rejected. Similarly, the null-hypothesis that all feature subset sizes perform equally was also rejected.

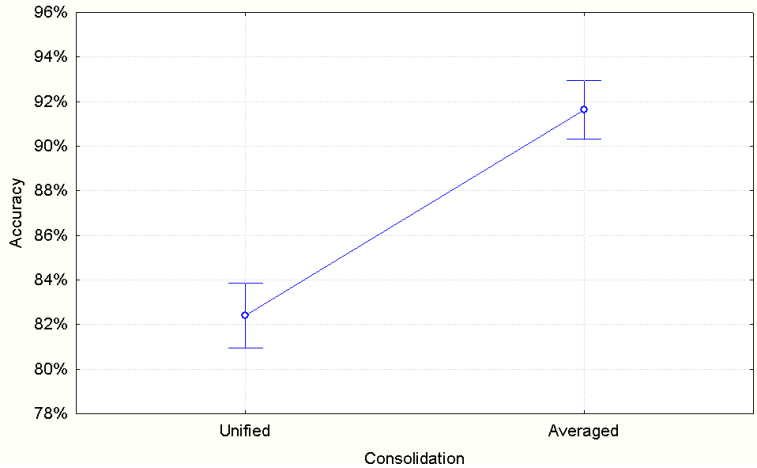

Fig. 7. (a). One-way ANOVA of the consolidation method. The averaged approach outperformed the unified approach. The vertical bars denote 0.95 confidence intervals.

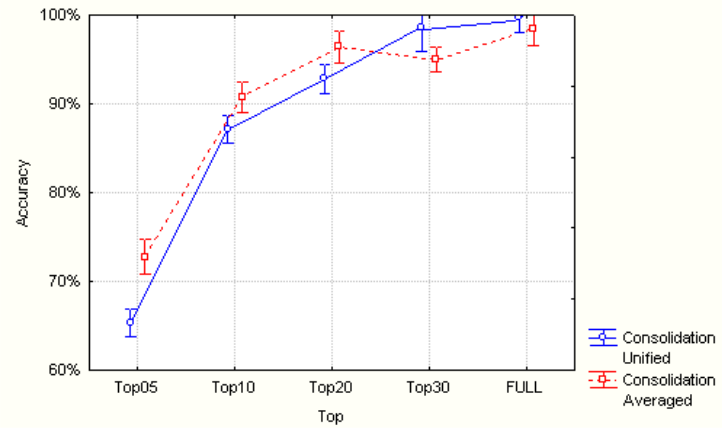

Fig. 7. (b). The interaction between the consolidation method and the top selected features. The averaged consolidation method was better for the small number of features selected (5-20), while the unified was better for the Top30 and FULL. The vertical bars denote 0.95 confidence intervals.

Moreover, the interaction between the feature-ranking measure and the top select features was found to be statistically significant with $F(12, 5350) = 7.479$ and $p < 1\%$.

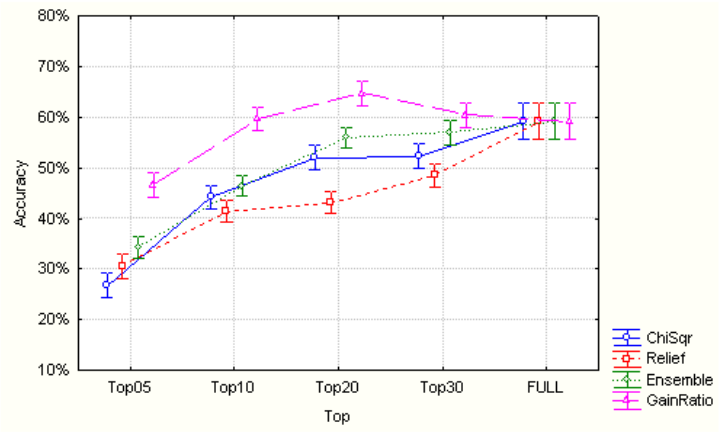

Fig. 8. The interaction between feature ranking measures and the top selected features. The GainRatio outperforms all the methods and selecting more than 20 features reduces the accuracy. The vertical bars denote 0.95 confidence intervals.

Figure 9 presents the accuracy obtained by the different kernel functions using different feature-ranking methods. The results indicate that the best combination is obtained by first using GainRatio for selecting the features and then building the SVM using the RBF function. The null-hypothesis, that all kernel functions perform equally and that observed differences are merely random, was rejected. Moreover, the interaction between the kernel function and the feature-ranking measures was found to be statistically significant with $F(7, 5392) = 7.2035$ and $p < 1\%$.

According to the experimental results given in 4.1.1 and 4.1.2, the different feature selection methods performed significantly differently in our context of worm detection. We understand that this is a result of the critical influence of each relevant feature that was selected. Different methods select different features, and it seems that the features we monitored are very diverse, which means that we have a complementary set of features that are independent of each other. Thus, the selection of different features had a significant impact on the results. In addition, the results reveal an interesting phenomenon. Previous research [48] showed a correlation between chi-square and information-gain. However, in our experimental study, on the one hand, in the same feature-collecting conditions (Figure 5), chi-square provided the best results, while information gain yielded

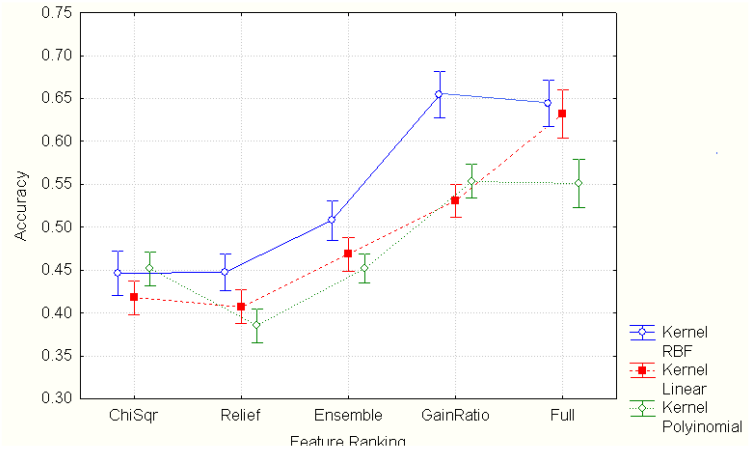

Fig. 9. The interaction between the kernel function and the top selected features. The RBF kernel outperforms the other kernels. The vertical bars denote 0.95 confidence intervals.

the worst results; on the other hand, in the different feature collecting conditions (Figure 8), the dominance relation was reversed.

4.2 Experiment II - Unknown Worms Detection

To evaluate the capability of the suggested approach to classify *unknown worm* activity, which was our main objective, an additional experiment was performed. In this experiment the training set consisted of $(5-k)$ worms and the testing set contained the k excluded worms; the *none* activity appeared in both datasets. This process was repeated for all the possible combinations of the k worms, for $k = 1$ to 4. Note that in these experiments, unlike in the previous section, there were two classes: (generally) *worm*, for any type of worm, and *none* for any other cases. For selecting the features, we used the Top20 features of the GainRatio with unified consolidation. The full training set, when no worm was excluded, contained 7126 instances of worm and non-worm, while the full training set, when 1,2,3,4 worms were excluded, contained 5881,4689,3497,2305 instances, respectively.

Figure 10 presents the results when all training data were used. As more worms were included in the training set, a monotonic improvement was observed. However, RBF was less affected by the number of excluded worms. Consequently, we prefer the RBF kernel when there are fewer worms in the training set. Additionally the Linear kernel consistently outperformed the polynomial kernel. Note that the number of worms in the x axis refers to the *number of excluded worms*. The RBF outperformed all the other kernels, while the *polynomial* kernel performed worse than the *linear* kernel.

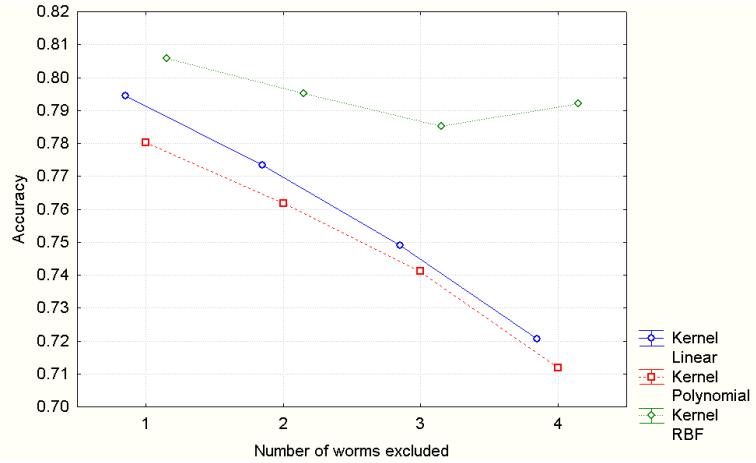

Fig. 10. The performance monotonically increases as fewer worms are excluded (and more worms appear in the training set). The RBF kernel presents a different behavior, in which a high level of accuracy is achieved even when learning from a single worm.

4.3 Experiment 3 - Using Selective Sampling

In this set of experiments, we wanted to maximize the performance achieved by the RBF kernel function. Specifically, we examined whether improved results can be achieved by using a selective sampling approach to reduce the number of misleading instances in the training set, which poses a challenge for the RBF. Thus, we employed a selective sampling approach based on AL.

The evaluation was made using the same setup as in the previous section, in which worms excluded from the training set appeared in the test set. Specifically, we repeated the same experiment with the entire set of examples as a baseline using the selective sampling method. For the selective sampling process, first, we randomly selected six examples from each type of class (worms/none); subsequently, in each AL iteration, an additional example was selected. Performances were noted after selecting 50, 100 and 150 additional instances.

Figure 11 presents the obtained results. Two main outcomes can be observed. First, the selective sampling significantly improved the baseline accuracy by more than 10%. Second, actively selecting only 50 instances can be sufficient for obtaining high accuracy. When we used the entire dataset, the accuracy increased as more worms were removed from the training set. This can be explained by the fact that some worms behave most of the time as legitimate applications. Thus, adding all their instances to the training set, without filtering out the confusing instances, might affect the training of the SVM negatively. Another observation that supports these insights is related to the fact that the fewer worms excluded, the larger is the gap between the results of selective sampling and the learning from the full training set, with the largest gap being at 1 excluded worm. This means that as more misleading instances exist in the training set, thus the selec-

tive sampling is more contributive in filtering the worm instances that are very similar to the non-worm behavior.

One should note that most of the instances in the test set that are presented to the SVM seem to be legitimate, yet the detection of the worm is done according to its own process in which there are also abnormal instances by which the SVM successfully determines that it is a indeed a worm; all the instances that belong to the same process are classified as worm, although they seem to be legitimate.

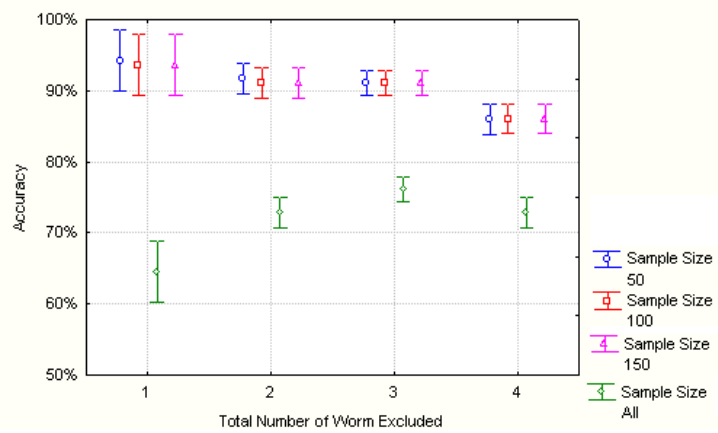

Fig. 11. The selective sampling approach significantly improves accuracy. Generally, an improvement of above 10% in accuracy was achieved.

Figure 12 presents the accuracy obtained when different worms were excluded from the training set. It can be seen that not all worms have the same detection accuracy. The differences were found to be statistically significant with $F(4, 155) = 7.84$ and $p < 1\%$. Further investigation, as shown in Figure 13, indicated that the exclusion of the W32.Deborm.Y decreased the performance of both W32.Deborm.Y and W32.Sasser.D worms. This implies that some worms can be detected by learning patterns sampled from other worms.

5 Conclusion and Future Work

We have presented a concept for detecting *unknown* computer worms based on host behavior, using the SVM classification algorithm with different kernels. The results show that the use of support vector machines in the task of detecting unknown computer worms is possible. We used a feature-selection method that enabled us to identify the most important computer features in order to detect unknown worm activity. This sort of work is currently performed by human experts. We specifically focused on the use of active learning as a selective

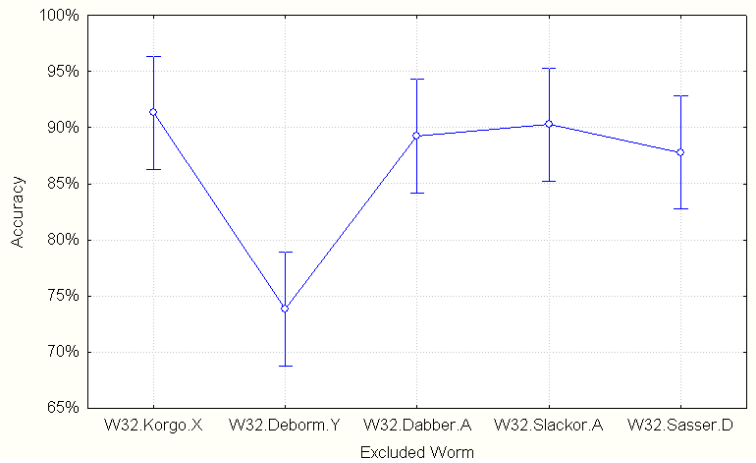

Fig. 12. The accuracy obtained when different worms are excluded. The W32.Deborm.Y seems to be most informative and crucial for use in the training set. The vertical bars denote 0.95 confidence intervals.

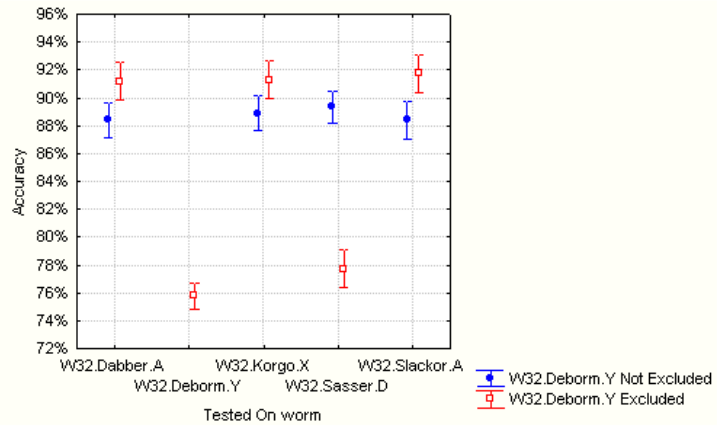

Fig. 13. The interaction effect of excluding W32.Deborm.Y worm and the tested worm. The vertical bars denote 0.95 confidence intervals. The interaction is statistically significant with $F(3, 2100) = 44.496$ and $p < 1\%$.

sampling method to increase the performance of the unknown computer worm detection with minimal human efforts. We rigorously analyzed the data from the large set of experiments that we performed. In the case of different conditions (in the training set and test set), the *GainRatio* measure for feature selection was most effective. On average, the *Top20* features produced the highest results and the *RBF* kernel commonly outperformed other kernels. For detecting unknown worms, the results show that it was possible to achieve a high level of accuracy; accuracy improved as more worms were included in the training set. To reduce the number of misleading instances in the training set and improve the learning, we show that the AL approach, as a selective method, can improve the performance. Selecting only 50 examples increased the accuracy to about 90%, and 94% when the training set contained four worms, in comparison to about 65% and 75%, respectively. When we selected 100 and 150 examples, no improvement was observed over the performance achieved with 50 examples. Furthermore, we analyzed the importance of using each worm in the training set. We found that a significant decrease in the performance occurred only when the W32.Deborm.Y was excluded from the training set. This can be explained by its nature, which is probably more general in its activity than are the other worms.

We conclude that selective sampling can be used to select the most informative examples from data that include misleading instances. These results are highly encouraging and show that the propagation of unknown worms, which commonly spread intensively, can be stopped in real time. The advantage of the suggested approach is the automatic acquisition and maintenance of knowledge based on inductive learning. This avoids the need for a human expert who may not always be available or familiar with generating rules or signatures.

References

1. C. Fosnock. Computer worms: Past, present and future. Technical report, East Carolina University, 2008.
2. Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, SP '01, pages 38–, Washington, DC, USA, 2001. IEEE Computer Society.
3. Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. N-gram-based detection of new malicious code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts - Volume 02*, COMPSAC '04, pages 41–42, Washington, DC, USA, 2004. IEEE Computer Society.
4. J. Zico Kolter and Marcus A. Maloof. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.*
5. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *Security Privacy, IEEE*, 1(4):33 – 39, july-aug. 2003.
6. Robert Moskovitch, Yuval Elovici, and Lior Rokach. Detection of unknown computer worms based on behavioral classification of the host. *Comput. Stat. Data Anal.*, 52(9):4544–4566, May 2008.

7. Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Improving malware detection by applying multi-inducer ensemble. *Comput. Stat. Data Anal.*, 53(4):1483–1494, February 2009.
8. Darrell M. Kienzle and Matthew C. Elder. Recent worms: a survey and trends. In *Proceedings of the 2003 ACM workshop on Rapid malcode*, WORM '03, pages 1–10, New York, NY, USA, 2003. ACM.
9. David Moore, Colleen Shannon, and k claffy. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 273–284, New York, NY, USA, 2002. ACM.
10. Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode*, WORM '03, pages 11–18, New York, NY, USA, 2003. ACM.
11. Cert. Multiple Denial-of-Service Problems in ISC BIND. `http://www.cert.org/advisories/CA-2000-20.html`, 2000. [Online; accessed July 23, 2012)].
12. Wenke Lee, S.J. Stolfo, and K.W. Mok. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 120 –132, 1999.
13. Peyman Kabiri and Ali A. Ghorbani. Research on intrusion detection and response: A survey. *International Journal of Network Security*, 1:84–102, 2005.
14. Daniel Barbará, Ningning Wu, and Sushil Jajodia. Detecting Novel Network Intrusions using Bayes Estimators. In *Proceedings of the First SIAM Conference on Data Mining*, April 2001.
15. Stefano Zanero and Sergio M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 412–419, New York, NY, USA, 2004. ACM.
16. H.G. Kayacik, A.N. Zincir-Heywood, and M.I. Heywood. On the capability of an som based intrusion detection system. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1808 – 1813 vol.3, july 2003.
17. J.Z. Lei and A. Ghorbani. Network intrusion detection using an improved competitive learning neural network. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 190 – 197, may 2004.
18. Dima Stopel, Robert Moskovitch, Zvi Boger, Yuval Shahar, and Yuval Elovici. Using artificial neural networks to detect unknown computer worms. *Neural Computing and Applications*, 18:663–674, 2009.
19. PingZhao Hu and M.I. Heywood. Predicting intrusions with local linear models. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1780 – 1785 vol.3, july 2003.
20. J.E. Dickerson and J.A. Dickerson. Fuzzy network profiling for intrusion detection. In *Fuzzy Information Processing Society, 2000. NAFIPS. 19th International Conference of the North American*, pages 301 –306, 2000.
21. Susan M. Bridges, Rayford B. Vaughn, Associate Professor, and Associate Professor. Fuzzy data mining and genetic algorithms applied to intrusion detection. In *In Proceedings of the National Information Systems Security Conference (NISSC*, pages 16–19, 2000.
22. Martin Botha and Rossouw von Solms. Utilising fuzzy logic and trend analysis for effective intrusion detection. *Computers and amp; Security*, 22(5):423 – 434, 2003.
23. David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. Technical report, Cambridge, MA, USA, 1995.

24. David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 3–12, New York, NY, USA, 1994. Springer-Verlag New York, Inc.
25. Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 441–448, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
26. Dragos D. Margineantu. Active cost-sensitive learning. In *IJCAI*, pages 1622–1613, 2005.
27. Jacob R. Lorch and Alan Jay Smith. Building vtrace, a tracer for windows nt and windows 2000. Technical Report UCB/CSD-00-1093, EECS Department, University of California, Berkeley, Feb 2000.
28. Francisco Azuaje. Witten ih, frank e: Data mining: Practical machine learning tools and techniques. *BioMedical Engineering OnLine*, 5:1–2, 2006.
29. J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
30. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
31. J Pearl. Fusion, propagation, and structuring in belief networks. *Artif. Intell.*, 29(3):241–288, September 1986.
32. Lior Rokach, Oded Maimon, and Reuven Arbel. Selective voting - getting more for less in sensor fusion. *IJPRAI*, 20(3):329–350, 2006.
33. Lior Rokach, Barak Chizi, and Oded Maimon. A methodology for improving the performance of non-ranker feature selection filters. *IJPRAI*, 21(5):809–830, 2007.
34. L. Rokach, R. Romano, and O. Maimon. Negation recognition in medical narrative reports. *Information Retrieval*, 11(6):499–538, 2008.
35. Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.
36. Thorsten Joachims. Advances in kernel methods. chapter Making large-scale support vector machine learning practical, pages 169–184. MIT Press, Cambridge, MA, USA, 1999.
37. Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.
38. A. Aizerman, E.M. Braverman, and LI Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.
39. Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*
40. Xun Wang, Wei Yu, Adam Champion, Xinwen Fu, and Dong Xuan. Detecting worms via mining dynamic program execution. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 412 –421, sept. 2007.
41. Mohammad M. Masud, Latifur Khan, and Bhavani Thuraisingham. Feature based techniques for auto-detection of novel email worms. In *Proceedings of the 11th Pacific-Asia conference on Advances in knowledge discovery and data mining*, PAKDD'07, pages 205–216, Berlin, Heidelberg, 2007. Springer-Verlag.
42. Robert Moskovitch, Nir Nissim, Dima Stopel, Clint Feher, Roman Englert, and Yuval Elovici. Improving the detection of unknown computer worms activity using

active learning. In *Proceedings of the 30th annual German conference on Advances in Artificial Intelligence*, KI '07, pages 489–493, Berlin, Heidelberg, 2007. Springer-Verlag.

43. Y. ZHU, X. WANG, and H. SHEN. Detection method of computer worms based on svm. *Mechanical & Electrical Engineering Magazine*, 8, 2008.
44. Robert Moskovitch, Nir Nissim, and Yuval Elovici. Malicious code detection using active learning. In Francesco Bonchi, Elena Ferrari, Wei Jiang, and Bradley Malin, editors, *Privacy, Security, and Trust in KDD*, volume 5456 of *Lecture Notes in Computer Science*, pages 74–91. Springer Berlin Heidelberg, 2009.
45. Rocco A. Servedio. Smooth boosting and learning with malicious noise. *J. Mach. Learn. Res.*, 4:633–648, December 2003.
46. Chen Yabi and Zhan Yongzhao. Co-training semi-supervised active learning algorithm based on noise filter. In *Proceedings of the 2009 WRI Global Congress on Intelligent Systems - Volume 03*, GCIS '09, pages 524–528, Washington, DC, USA, 2009. IEEE Computer Society.
47. Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 839–846, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
48. George Forman. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*