Detecting Bots via Incremental LS-SVM Learning with Dynamic Feature Adaptation

Feilong Chen
Dept. of Computer Science
Michigan State University
East Lansing, MI 48824
chenfeil@msu.edu

Supranamaya Ranjan
Narus Corporation
570 Maude Court
Sunnyvale, CA 94085
soups@narus.com

Pang-Ning Tan
Dept. of Computer Science
Michigan State University
East Lansing, MI 48824
ptan@cse.msu.edu

ABSTRACT

As botnets continue to proliferate and grow in sophistication, so does the need for more advanced security solutions to effectively detect and defend against such attacks. In particular, botnets such as Conficker have been known to encrypt the communication packets exchanged between bots and their command-and-control server, making it costly for existing botnet detection systems that rely on deep packet inspection (DPI) methods to identify compromised machines. In this paper, we argue that, even in the face of encrypted traffic flows, botnets can still be detected by examining the set of server IP-addresses visited by a client machine in the past. However there are several challenges that must be addressed. First, the set of server IP-addresses visited by client machines may evolve dynamically. Second, the set of client machines used for training and their class labels may also change over time. To overcome these challenges, this paper presents a novel incremental LS-SVM algorithm that is adaptive to both changes in the feature set and class labels of training instances. To evaluate the performance of our algorithm, we have performed experiments on two large-scale datasets, including real-time data collected from peering routers at a large Tier-1 ISP. Experimental results showed that the proposed algorithm produces classification accuracy comparable to its batch counterpart, while consuming significantly less computational resources.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

General Terms

Algorithms

1. INTRODUCTION

Recently, many security solutions have begun using online supervised learning algorithms to address a variety of malware and spam detection problems. For instance, Wang et al. [17] presented an online support vector machine (SVM) classifier that uses "bag of keyword" features from e-mail content to label future e-mails as ham or spam. More recently, Ma et al. [8, 9] applied the online SVM and confidence-weighted (CW) algorithms to detect URLs suspected of hosting malware on the basis of the lexical and host-based features extracted from each URL.

A third application that we consider in this paper centers around the problem of detecting *botnets*, defined as a collection of machines that have been infected by malware. Some known botnets consist of tens of thousands of machines and botnets in general are responsible for launching most of the malicious attacks on the Internet including sending spam e-mails, hosting phishing sites, stealing credit card information via keyloggers, *etc.* The bot owner co-ordinates all the bot machines by sending them common commands about the next attack to launch, or sending them new spam templates. In this regards, one way of detecting botnets is by examining the command-and-control traffic exchanged between bots and the group of command-and-control servers that the bots have to obtain their instructions from.

However, some sophisticated botnets such as Conficker [12] have been known to encrypt the communication packets exchanged between the bots and the command-and-control server. This makes the detection process extremely costly for many existing botnet detection systems [6, 7] that require inspecting the encrypted packet payloads to identify compromised machines. In this paper, we argue that even in the face of encrypted traffic flows, botnets can still be detected by virtue of the fact that bots *must* communicate with command-and-control servers to obtain instructions whereas legitimate clients are unlikely to communicate with the same servers. In this regards, we position the problem of botnet detection as a binary classification problem where we label each client IP-address seen in the network as good or bad depending on a classifier built for the client machines, with features being the set of server IP-addresses that each client talks to within a certain time period. We obtain ground truth labels for a set of client IP-addresses by (i) consulting with IP blacklists maintained by several reputation systems such as Spamhaus, SORBS, *etc.* as well as (ii) consulting with an Intrusion Detection System [1] that labels a client IP-address as suspicious if it is found to exchange a traffic flow that matches a set of regular expression signatures.

A key challenge in developing an effective classifier for botnet detection is that the set of server IP-addresses used to represent the features often evolves dynamically. For instance, after a new machine gets infected, it may begin com-

KDD'11, August 21–24, 2011, San Diego, California, USA.

municating with command-and-control servers, for which we may not have seen any traffic originating to or from in the past. Even if the feature set remains the same, effects due to churn in the hosting infrastructure with new machines being infected and infected machines being cleaned up leads to changes in the importance of each feature. For instance, a newly infected machine may be elevated to the position of a command-and-control server and hence the feature weight learnt for this IP-address should be modified accordingly. Similarly, an existing command-and-control server may be taken down either by law-enforcement or due to cleaning-up of that machine by anti-virus solutions, and hence the feature weight corresponding to that IP-address should be adjusted accordingly. As a result, a supervised learning model trained on an initial feature set may not be as effective when applied to the next day's data set because it would not be able to utilize the new features present in the next day's data set. To improve its detection rate without incurring significant computational overhead, the model must be updated incrementally to take these effects into account.

Current approaches deal with this problem either via batch learning, or via online learning. Batch learning based solutions build a new model from scratch by utilizing the new data points as well as new features seen. We argue that in these class of applications, the data set size can be prohibitively large such that throwing away past computations and rebuilding from scratch may be highly compute-intensive as well as suffer from detection lags, *i.e.* attackers can get around the detector by introducing new features at a rate faster than the retraining rate. Moreover, during the classification phase, the batch learning based approachs simply apply the old model to the new data set while ignoring the new features that were encountered. Clearly, due to infrequent retraining periods and being unable to make use of new features as soon as they appear, the batch learning based approaches may not achieve a high enough detection accuracy as we will show later during our experimental evaluation. In contrast, in online learning, the model is updated with every incoming data point.

In this regards, one of the primary contributions in this paper is in the development of a botnet detection approach based on a novel online least-square support vector machine (LS-SVM) algorithm, which is incremental in terms of both evolving feature-set as well as data-points. While the problem of how to update an existing SVM model in presence of incremental data-points has been dealt with before [17], we consider our work to be the first (to the best of our knowledge) to consider the problem of updating an existing SVM model when the feature set also evolves. Towards this end, we propose a novel algorithm that achieves a time-accuracy trade-off and reduces the model update time by collapsing the old features into one feature during the retraining phase. Moreover, we constrain the (re)training time of our algorithm by constraining the number of training instances that are used, and in this regards, we introduce an approach to choose which training instances be re-used during the retraining phase.

Finally, we evaluate the performance of our proposed algorithm on two real-world datasets. First, we use the dataset evaluated in [8, 9], which considers the problem of labeling URLs as malicious. The second dataset is obtained from peering routers at a large Tier-1 ISP and provides a who-talks-to-whom graph for a one day time period. We evaluate our incremental LS-SVM approach against both the datasets and show the following. First, our dynamic feature adaptation method achieves a bounded retraining time compared to retraining from scratch and counter-intuitively, in some cases, even achieves higher detection accuracy since we bound the number of features and thereby avoid overfitting. Second, our incremental LS-SVM with dynamic feature and data adaptation method significantly outperforms the CW algorithm [9], a state-of-the-art online classification algorithm, when applied to the large-scale botnet data from an ISP.

The rest of this paper is organized as follows. First, we provide an overview of the proposed system in Section 2. We then present the related work in Section 3. Section 4 discusses background on SVM and the various formulations of online learning. Next, Section 5 provides details about the dynamic feature adaptation part of our incremental LS-SVM approach. Section 6 discusses the approach to deal with evolving data and how we combine both the incremental feature- and incremental-data approaches. Next, we perform a comprehensive evaluation in Section 7. Finally, we conclude in Section 8.

2. SYSTEM ARCHITECTURE

As proof of concept, we develop a prototype system called *BotWatch* that passively monitors and continuously analyzes the Internet traffic in real time to detect bots. A key component of our system is an incremental LS-SVM classifier to identify emerging bots and consequently new botnets.

BotWatch relies on the following hypothesis to detect bots: the set of servers that bots communicate with are different from the set of servers that legitimate clients communicate with. This can be explained by the fact that botnet operators need to maintain communication with their bots in order to issue commands to them for the next attack to launch or to ensure that they have proper control over their bots by having the bots contact them frequently via "heart-beat" requests. However, this hypothesis can be complicated by the fact that the bot machines also communicate with legitimate servers either because the humans behind the machine are also using it or for subversion purposes. For instance, *e.g.* Conficker [12] bots first open an HTTP GET request to one from a set of over hundred legitimate servers including Google, Amazon, *etc.* to first obtain the current time of day. Regardless, as we will show later, bots can still be distinguished from legitimate clients on the basis of which unique set of machines they communicate with.

The system architecture is composed of the following components: *(i)* flow parser, *(ii)* graph feature extractor, *(iii)* external IP blacklists, and an *(iv)* an online classifier. The input to the system is traffic as obtained via the routers at an Internet Service Provider.

The flow parser reconstructs all the packets that correspond to the same traffic flow, where a flow is defined as the unique 6-tuple of client and server IP-addresses, client- and server-ports, timestamp and layer-3 protocol (TCP or UDP). On seeing each new flow, the BotWatch system updates the aggregated statistics for the pair of IP-addresses which are communicating with each other. In our implementation, we currently compute simple aggregated counts for how many flows were exchanged between the IP-pair within a pre-configured time interval. At the end of this time-interval, the following information is passed on to the

online learning module: timestamp for the end of the time window, client IP-address, server IP-address and number of flows exchanged between the IP-pair.

Note that this window based updation of statistics does not have any relationship with our online algorithm which we explain in Section 5, and is only reflective of the application area of botnet detection. Essentially, in the botnet detection application, we found that gathering statistics about each IP-pair over a time window provides better detection rates than labeling each IP-pair as it is seen. Finally, the online learning module trains a classifier that is updated on each incoming IP-pair. We use an external blacklist to train the classifier, where the blacklist contains all the known bot IP addresses, and is either updated whenever the external source releases a new list, or is queried individually for each new IP address detected by the system. Due to the temporary nature of bots, as ideally users will clean their devices and remove bots, a timestamp is assigned to each entry in the list and entries are removed after a pre-configured time interval.

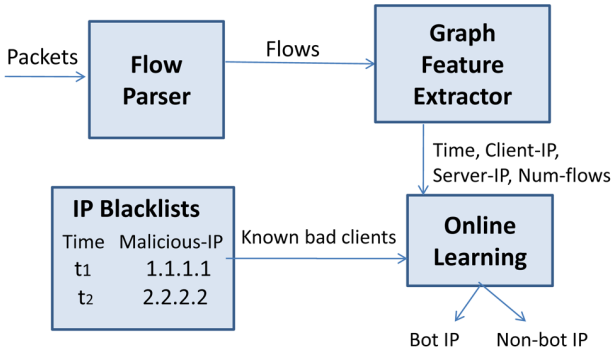

Figure 1: System Architecture

3. RELATED WORK

3.1 Botnet Detection

Botnets have emerged as one of the biggest security threats facing the Internet today. They can be broadly categorized into two types—centralized botnets and Peer-to-Peer (P2P) botnets. Centralized botnets employ one or more Command-and-Control (C&C) servers to control all infected bots and instructs them when, whom and how to attack. Most centralized botnets are based on the IRC protocol. There have been several works focusing on monitoring IRC channel traffic to detect such botnets [2, 5, 3]. Once they have been identified, the centralized botnets can be easily destroyed by blacklisting them.

Newer botnets have evolved from a centralized structure to a more distributed, P2P configuration, to make it harder for them to be detected and shut down. As P2P botnets became prevalent, more sophisticated methods are needed. Some of the recent works have focused on detecting similar communication patterns exhibited by members of the botnet ([6, 7, 10]). For instance, [11] clusters HTTP traces based on their structural similarity to detect bots. BotGrep [10] analyzes the communication graph between hosts and partitions the graph into several components based on some metrics to separate bots from non-bots. None of these approaches however employ a sophisticated online classifier such as SVM to improve their detection rate.

3.2 Online Learning Algorithms

Online learning algorithms have been developed to handle situations where the data set is extremely large and it is infeasible to load the entire data set to the main memory. A classic online learning algorithm is perceptron [13], a linear classifier that takes one data example at a time and updates the weight vector when the current classifier misclassifies the example. The perceptron is simple to implement and fast to learn but not effective when applied to nonlinearly separable problems. More recently, Ma et al. [9] applied a variety of online learning algorithms, including Passive-Aggressive (PA) and Confidence Weighted (CW), to the problem of classifying URLs as suspicious or legitimate. These algorithms were designed to minimally update the target function as new data arrives by employing different objective functions (PA minimizes a least-square loss function whereas CW uses Kullback-Leibler divergence). CW also maintains a confidence measure for each feature and updates more aggressively the weights for the features with lower confidence. However, these heuristics do not guarantee that the updated model has better generalization error. For example, if the initial model was poor, then adjusting the model minimally may not be an effective strategy. In contrast, the online LS-SVM algorithm investigated in this paper is designed to maintain a large margin to ensure good generalization performance. Though there are several online SVM classifiers proposed in the literature [16, 14], none of them were designed to adapt to new features in the data.

4. PRELIMINARIES

Let $\mathcal{D} = \mathcal{D}^{(1)}\mathcal{D}^{(2)}\cdots\mathcal{D}^{(T)}$ denote an ordered collection of labeled data sets, where each $D^{(i)} = \{(\mathbf{x}_j^{(i)}, y_j^{(i)})\}_{j=1}^{N_i}$ is a set of training instances collected during the time interval t_{i-1} and t_i. Each training instance is associated with a feature set $\mathbf{x}_j^{(i)} \in \Re^{d_i}$ and a binary class label $y_j^{(i)} \in \{-1, +1\}$. Many online learning algorithms are designed to update the classification model instantaneously, as each data example arrives (i.e., $N_i = |D^{(i)}| = 1$ for all i). However, for efficiency reasons, it may be more practical to consider the situation where the time interval is sufficiently large such that $N_i \gg 1$. Furthermore, we consider several formulations of the online classification problem:

Dynamic feature adaptation (DynF) This formulation assumes that the training instances are the same throughout the entire time period, but the feature set may grow with the addition of new features that have never been seen in the past. Furthermore, the class label of a training instance may also change over time.

Dynamic feature and data adaptation (DynFD) This formulation assumes new training instances may be collected at each time period while older instances continue to be monitored. The feature set and class labels for the training instances may evolve over time.

The classification algorithm considered in this study is the well-known support vector machine (SVM) classifier. SVM is designed to learn a linear decision surface, either in the original feature space or in a projected high-dimensional feature space, that maximizes the geometric margin of the

separating hyperplanes between different classes. In this study, we consider a variation of the classifier known as least-square support vector machine (LS-SVM) [15]. Unlike regular SVM, which minimizes a hinge loss function with linear inequality constraints, LS-SVM optimizes the following least-square loss function with equality constraint:

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2 \tag{1}$$
$$\text{s.t.} \qquad y_i\left[\mathbf{w}^T\mathbf{x}_i + b\right] = 1 - e_i, \quad \forall i \in \{1,\ldots,N\}$$

where γ is a user-specified parameter and N is the number of training instances. An appealing feature of the optimization problem for LS-SVM is that it has a closed form solution, compared to the optimization problem for regular SVM, which requires numerical methods to solve a quadratic programming problem. Previous studies have found that the generalization performance of LS-SVM is comparable to that of regular SVM [4, 19]. Furthermore, it has been theoretically proven the equivalence between linear LS-SVM and hard margin SVM with Mahalanobis distance measure [18].

The constraint optimization problem given in (1) can be cast into the following Lagrangian formulation:

$$\begin{aligned}\mathcal{L}(\mathbf{w}, b, \mathbf{e}, \alpha) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2 \\ &- \sum_{i=1}^{N}\alpha_i\left\{y_i\left[\mathbf{w}^T\mathbf{x}_i + b\right] - 1 + e_i\right\}, \end{aligned} \tag{2}$$

where $\{\alpha_i\}$ is the set of Lagrange multipliers. The optimization problem can be solved by taking its partial derivative with respect to the parameters $\mathbf{w}$, b, $\mathbf{e}$, and α and setting them to zero:

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \mathbf{Z}^T\alpha = \mathbf{0}_d \\ \frac{\partial L}{\partial b} &= \alpha^T\mathbf{y} = 0 \\ \frac{\partial L}{\partial \mathbf{e}} &= \gamma\mathbf{e} - \alpha = \mathbf{0}_N \\ \frac{\partial L}{\partial \alpha} &= \mathbf{Z}\mathbf{w} + b\mathbf{y} + \mathbf{e} - \mathbf{1}_N = \mathbf{0}_N,\end{aligned}$$

where $\mathbf{Z}$ is a rectangular matrix whose $(i,j)^{\text{th}}$ element is equal to $y_i x_{ij}$ (i.e., the product of the class label for the i-th instance and its j-th attribute value), $\mathbf{0}_p$ is a p-dimensional column vector of all zeros, and $\mathbf{1}_p$ is a p-dimensional column vector of all ones. The preceding set of linear equations can be further reduced to the following form:

$$\begin{bmatrix}\mathbf{I}_d & 0 & 0 & -\mathbf{Z}^T \\ 0 & 0 & 0 & -\mathbf{y}^T \\ 0 & 0 & \gamma\mathbf{I}_N & -\mathbf{I}_N \\ \mathbf{Z} & \mathbf{y} & \mathbf{I}_N & 0\end{bmatrix}\begin{bmatrix}\mathbf{w} \\ b \\ \mathbf{e} \\ \alpha\end{bmatrix} = \begin{bmatrix}\mathbf{0}_d \\ 0 \\ \mathbf{0}_N \\ \mathbf{1}_N\end{bmatrix}, \tag{3}$$

where $\mathbf{I}_p$ is a $p \times p$ identity matrix. Since

$$\mathbf{I}_d\mathbf{w} - \mathbf{Z}^T\alpha = 0 \implies \mathbf{w} = \mathbf{Z}^T\alpha \tag{4}$$
$$\gamma\mathbf{I}_N\mathbf{e} - \mathbf{I}_N\alpha = 0 \implies \mathbf{e} = \gamma^{-1}\alpha, \tag{5}$$

the system of linear equations can be further simplified as follows:

$$\begin{bmatrix}0 & \mathbf{y}^T \\ \mathbf{y} & \mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N\end{bmatrix}\begin{bmatrix}b \\ \alpha\end{bmatrix} = \begin{bmatrix}0 \\ \mathbf{1}_N\end{bmatrix} \tag{6}$$

which leads to the following closed form solutions:

$$b = \frac{y^T(\mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}\mathbf{1}_N}{y^T(\mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}\mathbf{y}} \tag{7}$$
$$\alpha = (\mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}[\mathbf{1}_N - \mathbf{y}b] \tag{8}$$

Once the values for the Lagrange multipliers α are found, $\mathbf{w}$ is computed using Equation (4). Note that the solutions for α and b involve the matrix product $(\mathbf{Z}\mathbf{Z}^T)_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$. This suggests that we can easily turn LS-SVM into a nonlinear classifier by replacing the dot-product $\mathbf{x}_i \cdot \mathbf{x}_j$ with its corresponding kernel function, i.e., $\mathbf{A}_{ij} = y_i y_j \Phi(\mathbf{x}_i, \mathbf{x}_j)$.

Finally, having solved the equations above and obtain the model parameters, we can predict the class label for a new data instance, say $\mathbf{x}_{test}$, as $f(\mathbf{x}_{test}) = \text{sign}[\mathbf{w}^T\mathbf{x}_{test} + b]$. For nonlinear LS-SVM, this is equivalent to

$$f(\mathbf{x}_{test}) = \text{sign}\left[\sum_i y_i\alpha_i\phi(\mathbf{x}_i, \mathbf{x}_{test}) + b\right].$$

The runtime complexity for LS-SVM is $O(N^3)$, with most of the computational overhead spent on computing the matrix inverse $(\mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}$ in Equations (7) and (8).

5. ONLINE LS-SVM WITH DYNAMIC FEATURE ADAPTATION

Let $\mathbf{x}^t$ be the feature set at time t and $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} \cup \hat{\mathbf{x}}^{(t+1)}$ be the expanded feature set at $t+1$, where $\hat{\mathbf{x}}^{(t+1)}$ denote the newly added features. For notational convenience, we will drop the superscript in the remainder of the paper. We begin our initial discussion with an exact formulation of online LS-SVM in the presence of dynamic features. However, the formulation makes an unrealistic assumption that the class labels are unchanged. We then discuss an efficient method for approximating the solution that allows the class labels to evolve over time.

5.1 Exact Method

Let $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{v}^T\hat{\mathbf{x}} + b$ be the decision function of the LS-SVM classifier, where $\mathbf{w}$ and $\mathbf{v}$ are the weight vectors associated with the old features ($\mathbf{x}$) and new features ($\hat{\mathbf{x}}$). The weights are computed by optimizing the following objective function:

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}[\mathbf{w}^T\mathbf{w} + \mathbf{v}^T\mathbf{v}] + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2 \\ &- \sum_{i=1}^{N}\alpha_i\left\{y_i\left[\mathbf{w}^T\mathbf{x}_i + \mathbf{v}^T\hat{\mathbf{x}}_i + b\right] - 1 + e_i\right\}, \end{aligned} \tag{9}$$

which is equivalent to the objective function given in (2) when the feature set consists of $\mathbf{x} \cup \hat{\mathbf{x}}$. The closed form solutions to the objective function are as follows:

$$\begin{aligned} b &= \frac{y^T(\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T + \mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}\mathbf{1}_N}{y^T(\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T + \mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}\mathbf{y}} \\ \alpha &= (\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T + \mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N)^{-1}[\mathbf{1}_N - \mathbf{y}b] \\ \mathbf{w} &= \mathbf{Z}^T\alpha, \quad \mathbf{v} = \tilde{\mathbf{Z}}^T\alpha, \end{aligned} \tag{10}$$

where $\mathbf{Z}_{ij} = y_i x_{ij}$ and $\tilde{\mathbf{Z}}_{ij} = y_i \hat{x}_{ij}$. As can be seen from Equation (10), the exact solutions require computing the inverse of the matrix $\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T + \mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N$, which is an $O(N^3 + N^2 d + N^2 \hat{d})$ operations (where N is the number

of training instances, d is the cardinality of $\mathbf{x}$ and $\hat{d}$ is the cardinality of $\hat{\mathbf{x}}$). Thus, re-computing the matrix inverse at each time period can be very expensive especially when both number of training examples and features are large.

If we assume that the class labels of the training instances do not change, i.e., $y_i^{(t+1)} = y_i^{(t)}$ $(\forall i = \{1, 2, \cdots, N\})$, or equivalently, $\mathbf{Z}^{(t+1)} = \mathbf{Z}^{(t)}$, then the matrix inverse can be computed more efficiently as follows:

LEMMA 1. *Let $\mathbf{A}$ be an invertible $N \times N$ matrix and $\hat{\mathbf{Z}}$ be an $N \times \hat{d}$ matrix (where $\hat{d} \ll N$). The inverse of the matrix $(A + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T)^{-1}$ can be computed efficiently as follows:*

$$(\mathbf{A} + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\hat{\mathbf{Z}}\left(\mathbf{I}_{\hat{d}} + \hat{\mathbf{Z}}^T\mathbf{A}^{-1}\hat{\mathbf{Z}}\right)^{-1}\hat{\mathbf{Z}}^T\mathbf{A}^{-1}$$

The proof of this lemma follows from a special case of the well-known Sherman-Morrison-Woodbury formula. Although the formula requires computing the inverse of another matrix $(\mathbf{I}_{\hat{d}} + \hat{\mathbf{Z}}^T\mathbf{A}^{-1}\hat{\mathbf{Z}})$, the size of this matrix is only $\hat{d} \times \hat{d}$, which is less costly than computing the inverse of its original matrix. If $\mathbf{A}^{-1}$ is already known, then the overall cost for computing the matrix inverse using Lemma 1 reduces to $O(N^2\hat{d})$ instead of $O(N^3)$.

We can apply Lemma 1 to solve Equation (10) by setting $\mathbf{A} = \mathbf{Z}\mathbf{Z}^T + \gamma^{-1}\mathbf{I}_N$. However, the limitation of this approach is that it assumes the class labels of the training instances do not evolve over time. The next subsection presents an approximate algorithm that can overcome this limitation.

5.2 Approximate Method

This section approximates the LS-SVM objective function with the following assumption, that the weights associated with the original feature vector is modified uniformly as follows: $\mathbf{w}_{new} = \lambda\mathbf{w}_{old}$. This assumption leads to the following modified objective function:

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}[\lambda^2\mathbf{w}^T\mathbf{w} + \mathbf{v}^T\mathbf{v}] + \frac{\gamma}{2}\sum_{i=1}^{l} e_i^2 \\ &- \sum_{i=1}^{l}\alpha_i\left\{y_i\left[\lambda(\mathbf{w}^T\mathbf{x}_i + b) + \mathbf{v}^T\hat{\mathbf{x}}_i + \hat{b}\right] - 1 + e_i\right\} + c\lambda^2\end{aligned}$$

where we have included a regularization term $c\lambda^2$ to constrain the magnitude of λ. For brevity, the subscript for $\mathbf{w}_{old}$ has also been omitted in the remainder of this section. As will be shown below, the parameter λ is determined automatically, based on the degree of agreement between the previous classification model and the current labels of the training instances. The more agreement there is, the larger λ will be. Furthermore, even if the uniformity assumption about the modification to $\mathbf{w}$ does not hold, any inaccuracies in the new model can be compensated by the weights associated with the new features $\hat{\mathbf{x}}$. Experimental results have suggested that the approximate model performs quite as well as its batch counterpart.

Taking the partial derivative over the model parameters and setting them to zeros yield the following:

$$\begin{bmatrix}\mathbf{I}_d & 0 & 0 & 0 & -\hat{\mathbf{Z}}^T \\ 0 & 0 & 0 & 0 & -\mathbf{y}^T \\ 0 & 0 & 0 & \gamma\mathbf{I}_N & -\mathbf{I}_N \\ 0 & 0 & c + \mathbf{w}^T\mathbf{w} & 0 & -\mathbf{g}^T \\ \hat{\mathbf{Z}} & \mathbf{y} & 0 & \mathbf{I}_N & 0\end{bmatrix}\begin{bmatrix}\mathbf{v} \\ \hat{b} \\ \lambda \\ \mathbf{e} \\ \alpha\end{bmatrix} = \begin{bmatrix}\mathbf{0}_d \\ 0 \\ \mathbf{0}_N \\ 0 \\ \mathbf{1}_N\end{bmatrix}, \quad (11)$$

where $\mathbf{g} = \mathbf{Z}\mathbf{w} + b\mathbf{y} = (\mathbf{X}\mathbf{w} + b\mathbf{1}_N) \odot \mathbf{y}$ is an N-dimensional column vector that measures the agreement between the predicted values of the previous model and the current labels of the training instances. The symbol $\odot$ denotes an element-wise vector product. After some simplification, the optimization problem reduces to solving the following system of linear equations:

$$\begin{bmatrix}0 & \mathbf{y}^T \\ \mathbf{y} & \frac{\mathbf{g}\mathbf{g}^T}{c + \mathbf{w}^T\mathbf{w}} + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T + \gamma^{-1}\mathbf{I}_N\end{bmatrix}\begin{bmatrix}\hat{b} \\ \alpha\end{bmatrix} = \begin{bmatrix}0 \\ \mathbf{1}\end{bmatrix} \quad (12)$$

There are several advantages of using this formulation. First, instead of working with the $N \times d$ matrix $\mathbf{X}$, it is sufficient to collapse the old features into a column vector $\mathbf{g}$, thereby reducing the storage requirements. This is important because the number of original features d keeps growing over time in dynamic feature adaptation. Second, the formulation may accommodate changes in the class labels of training instances in subsequent time periods. This can be accomplished by storing the vector $\tilde{\mathbf{g}} = \mathbf{X}\mathbf{w} + b\mathbf{1}_N$ instead of $\mathbf{g}$. If the class labels for some training instances have changed, we only need to update the corresponding elements in $\mathbf{y}$ to reflect these changes. The updated vector $\mathbf{g}$ is obtained by computing $\tilde{\mathbf{g}} \odot \mathbf{y}$.

Note that λ is determined automatically based on the consistency between the predictions made by the previous model and the current labels of the training instances, i.e.:

$$\lambda = \frac{\alpha^T\mathbf{g}}{\mathbf{w}^T\mathbf{w} + c} = \frac{\alpha^T[(\mathbf{X}\mathbf{w} + b\mathbf{1}) \odot \mathbf{y}]}{\mathbf{w}^T\mathbf{w} + c}.$$

This formula suggests that the value for λ is large when the predictions made by the previous model are consistent with the current labels of the training instances (especially for those instances associated with large values of α). On the other hand, if the previous model becomes outdated, then the value for λ will automatically be reduced.

5.3 Efficient Computation

The preceding formulation has to solve the system of linear equations given in (12) and performs matrix inversion on $\frac{\mathbf{g}\mathbf{g}^T}{c + \mathbf{w}^T\mathbf{w}} + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T + \gamma^{-1}\mathbf{I}_N$. The problem can be further simplified by choosing $c = 1 - \mathbf{w}^T\mathbf{w}$ (though the methodology explained below is applicable to other values of c).

LEMMA 2. *Let $S = \mathbf{g}\mathbf{g}^T + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T + \gamma^{-1}\mathbf{I}_N$, N be the number of training instances, and $\hat{d}$ be the number of new features in $\hat{\mathbf{x}}$ (where $\hat{d} \ll d$). The overall cost for solving the approximate LS-SVM formulation with dynamic feature adaptation is $O(N\hat{d}^2 + \hat{d}^3 + N^2\hat{d})$, which is independent of the number of features d in the previous time period.*

PROOF. The inverse for $\mathbf{S}$ can be computed efficiently as follows. First, we use the Sherman-Morrison-Woodbury formula (see Lemma 1) to compute the inverse of $\gamma^{-1}\mathbf{I}_N + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T$:

$$(\gamma\mathbf{I}_N + \hat{\mathbf{Z}}\hat{\mathbf{Z}}^T)^{-1} = \gamma\mathbf{I}_N - \gamma^2\hat{\mathbf{Z}}(\mathbf{I}_d + \gamma\hat{\mathbf{Z}}^T\hat{\mathbf{Z}})^{-1}\hat{\mathbf{Z}}^T \quad (13)$$

Since $(\mathbf{I}_d + \gamma\hat{\mathbf{Z}}^T\hat{\mathbf{Z}})$ is a $\hat{d} \times \hat{d}$ matrix, its inverse can be computed efficiently in $O(\hat{d}^3)$. Next, we can apply the Sherman-Morrison formula to compute the inverse of $\mathbf{S}$ as follows:

$$\mathbf{S}^{-1} = (\mathbf{J} + \mathbf{g}\mathbf{g}^T)^{-1} = \mathbf{J}^{-1} - \frac{\mathbf{J}^{-1}\mathbf{g}\mathbf{g}^T\mathbf{J}^{-1}}{1 + \mathbf{g}^T\mathbf{J}^{-1}\mathbf{g}}$$

where $\mathbf{J}$ is the inverse given in Equation (13). Since $(1 + \mathbf{g}^T\mathbf{J}^{-1}\mathbf{g})$ is a scalar, we do not need to perform any matrix inversion here. In conclusion, the overall time complexity to compute the inverse for $\mathbf{S}$ is $O(N\hat{d}^2 + \hat{d}^3 + \hat{N}^2\hat{d})$. □

6. ONLINE LS-SVM WITH DYNAMIC FEATURE AND DATA ADAPTATION

This section presents an extension to the online LS-SVM method described in the previous section to incorporate new training instances. We assume the deployed system has bounded storage capacity to store at most N training instances in memory at a time. Furthermore, since the complexity of the algorithm is proportional to N^2, maintaining the growing number of training instances would make the computation more expensive. To circumvent this problem, we present an approach that maintains the size of the training set to be N at all times. In particular, we focus on training instances that can either maintain or help improve the performance of the classifier.

First, we consider only the new labeled instances that have been misclassified by the existing model to be included in the training set. Let $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ be the current training set and $\mathbf{D}_2 = \{\mathbf{x}_{N+1}, \mathbf{x}_{N+2}, \cdots, \mathbf{x}_{N+m}\}$ be the misclassified new instances. In order to include the m new instances, we need to remove the same number of instances from the training set $\mathbf{D}$. In this paper, we investigate three instance removal strategies. The first strategy (**OLD**) simply removes the m oldest training examples and replaces them with the misclassified new instances. The second strategy (**MIN**) discards the m training examples with smallest values for α. The rationale behind this approach is that such examples have the least influence on the decision surface of the classifier.

Our third strategy is designed to identify training examples whose removal have minimal effect on the current model. Let $f_{\mathbf{Q}}(\mathbf{x}) = \sum_{\mathbf{x}_i \in \mathbf{Q}} y_i\alpha_i\mathbf{x}_i \cdot \mathbf{x} + b$ be the model constructed from a training set $\mathbf{Q}$ and $\mathbf{S}^* \subset \mathbf{D}$ be the set of training instances selected for removal. We formalize the problem of selecting training instances to remove as follows:

$$\begin{aligned}
\mathbf{S}^* &= \arg\min_{\mathbf{S}} E\Big[|f_{\mathbf{D}}(\mathbf{x}) - f_{\mathbf{D}\setminus\mathbf{S}}(\mathbf{x})|\Big] \\
&= \arg\min_{\mathbf{S}} E\Big[\Big|\sum_{\mathbf{x}_i\in\mathbf{D}} y_i\alpha_i\mathbf{x}_i\cdot\mathbf{x} - \sum_{\mathbf{x}_i\in\mathbf{D}\setminus\mathbf{S}} y_i\alpha_i\mathbf{x}_i\cdot\mathbf{x}\Big|\Big] \\
&= \arg\min_{\mathbf{S}} E\Big[\sum_{\mathbf{x}_i\in\mathbf{S}} y_i\alpha_i\mathbf{x}_i\cdot\mathbf{x}\Big] \\
&= \arg\min_{\mathbf{S}} E\Big[\Big(\sum_{\mathbf{x}_i\in\mathbf{S}} y_i\alpha_i\mathbf{x}_i\Big)\cdot\mathbf{x}\Big]. \qquad (14)
\end{aligned}$$

where $E[\cdot]$ denote expected value. Solving the objective function shown above can be expensive especially when the training set size is large. The **MIN** approach (strategy 2) tries to find an approximate solution by choosing

$$\mathbf{S}^* = \arg\min_{\mathbf{S}} |\sum_{\mathbf{x}_i\in\mathbf{S}} \alpha_i|$$

The limitation of this approach is that it ignores the class label of the selected training instances unlike the objective function given in Equation (14). Instead, our third strategy (**GREEDY**) is designed to select training instances that minimize the following objective function:

$$\mathbf{S}^* = \arg\min_{\mathbf{S}} |\sum_{\mathbf{x}_i\in\mathbf{S}} y_i\alpha_i|$$

After we find $\mathbf{S}^*$, we can construct a new training set $\mathbf{D}^* = (\mathbf{D} \setminus \mathbf{S}^*) \cup \mathbf{D}_2$. Finally, we apply the online LS-SVM with dynamic feature adaptation approach described in the previous section to $\mathbf{D}^*$.

Table 1: Statistics of identified bots

Botnet	Regular Expression	#C&C servers	#bots
Conficker	URL ∼ `http://[0-9]+.[0-9]+.[0-9]+[0-9]+./search?q=[0-9]+`	28	1,835
Grum	URL ∼ `[a-zA-Z0-9\.]+/spm/s\_[a-zA-Z]+.php`	24	161
Pushdo	URL ∼ `[a-zA-Z0-9\.]+/40E800` || `[a-zA-Z0-9\.]+/C00000`	19	166
Sality	URL ∼ `musikrajt.sk` || `musikrajt.wz.cz` || `edmatrix.us`	10	232
Total		81	2,394

7. EXPERIMENTAL EVALUATION

We evaluated our algorithm on two real-world data sets. The first is a subset of the benchmark dataset used for detecting malicious URLs [9]. The dataset contains URLs collected over a 20-day period, each of which has 20,000 unique URLs. The features include 1,791,261 lexical features and 1,117,901 host-based features. Nearly one third of the data belongs to the positive class (i.e., malicious URL). The second is a botnet detection dataset we have collected by monitoring the HTTP network traffic at a large ISP in 24 hours. The data contains features derived from Layer-4 and Layer-7 load information, including source and destination IP-addresses, number of bytes transmitted, number of packets, *etc.* The labels are generated based on the approach described in Section 2. The botnets found in the network trace include Conficker, Grum, Pong, Pushdo, and Sality. Table 1 shows examples of the identified botnets, their statistics, and signatures used for detecting them in the data. We partition the data into four subsets, each containing all the client IP-addresses observed in a six-hourly time window. We then train the LS-SVM classifier on the first subset and incrementally update the model with new data from subsequent subsets.

7.1 Dynamic Feature Adaptation

First we evaluated the performance of our proposed dynamic feature adaptation method using the malicious URL dataset. For this experiment, we combine the data for all the days and partition them into five folds. For each fold, we designate half of the data as labeled (to be used for training and incremental model updating) and the remainder as unlabeled (to be used for testing purposes). In addition, we randomly split the feature set into ten equal-sized subsets

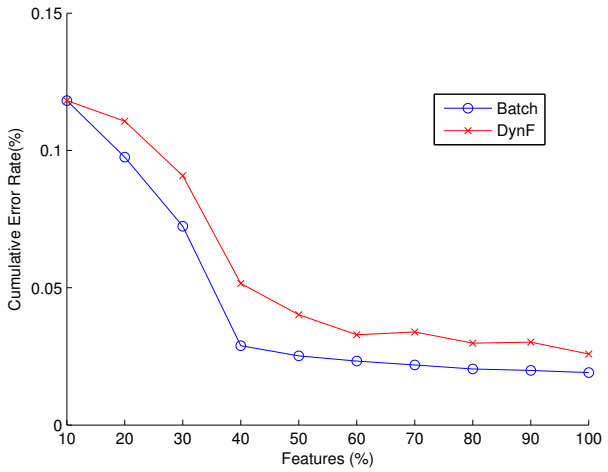

Figure 2: Comparison of classification error between batch and DynF algorithms on fold 1 data as new features are incrementally added to the classifier.

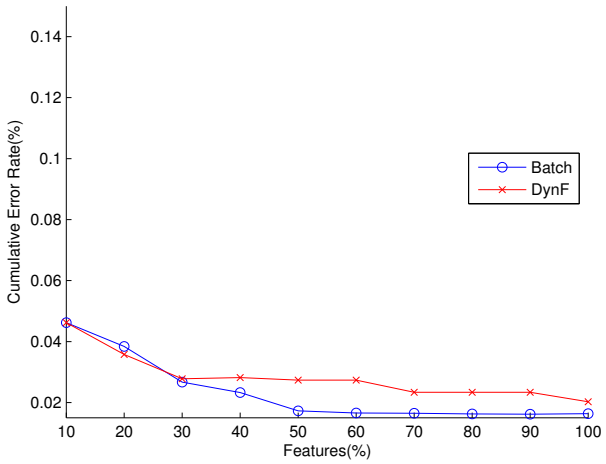

Figure 3: Comparison of classification error between batch and DynF algorithms on fold 2 data as new features are incrementally added to the classifier.

and use one of them (10% of the features) for initial model building and testing. We then iteratively add another subset of features and update the model incrementally using the approach described in Section 5 (the training and test sets are assumed to be fixed throughout this experiment). This iterative update process continues until the entire feature set is included into the model. The experiment is then repeated for the remaining four folds.

We consider two metrics for evaluating the performance of our method—misclassification error rate and training time. We compared the performance of our method against the batch LS-SVM algorithm, which retrains a new model from scratch using both the old and newly augmented features. Figures 2, 3, and 4 show the classification error rates for folds 1 to 3, respectively (we omit the results for folds 4 and 5 due to lack of space, but the results are very similar). Observe that the misclassification error of our dynamic feature adaptation method (DynF) is comparable to that of the batch LS-SVM algorithm.

Figure 5 shows the training time for both batch and DynF methods. Clearly, as more features are added, the training time for the batch algorithm grows more rapidly compared to our proposed approach. In fact, the training time for DynF is somewhat stable due to the fact that the features from previous models are shrunk into a column vector $\mathbf{g}$ while the number of new features used to update the model is the same (i.e., 10% of the entire feature space). The decrease in training time for the DynF algorithm after using the first 10% features can be explained by the fact that the computational complexity for building the initial LS-SVM model is $O(N^3)$, while subsequent updates require only $O(N^2\hat{d})$.

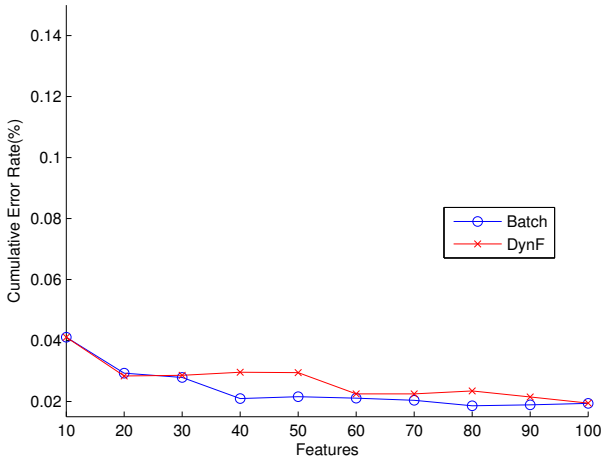

Figure 4: Comparison of classification error between batch and DynF algorithms on fold 3 data as new features are incrementally added to the classifier.

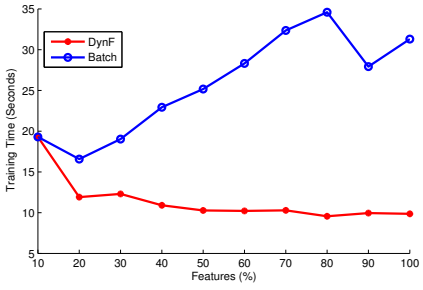

Figure 5: Comparison of training time between batch and DynF algorithms on fold 1 data as new features are incrementally added to the classifier.

7.2 Dynamic Feature and Data Adaptation

Next we evaluate the performance of our proposed Dynamic Feature with Data Adaptation (DynFD) algorithm. We compared the performance of our algorithm against the following baseline methods: (1) **Static**. In this approach, we train an initial model from the first time period and applies it to all subsequent time periods (without updat-

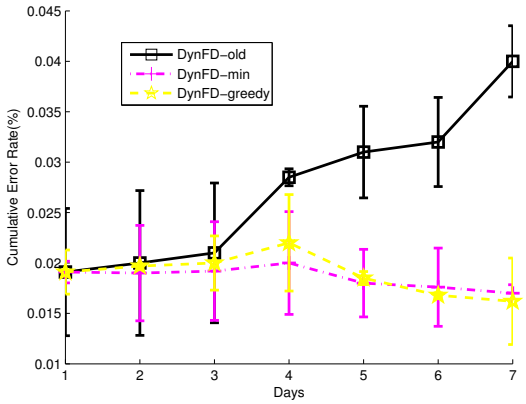

Figure 6: Cumulative error rate. The results were obtained using a benchmark suspicious URL detection dataset.

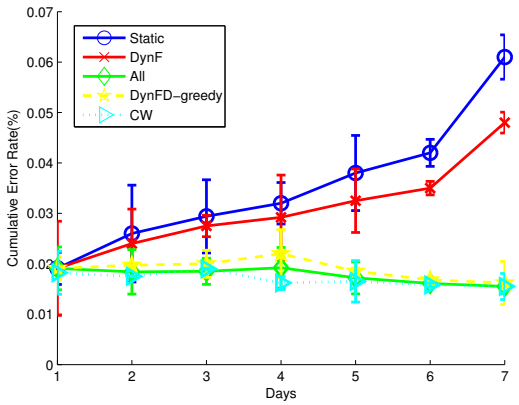

Figure 7: Cumulative error rate. The results were obtained using a benchmark suspicious URL detection dataset.

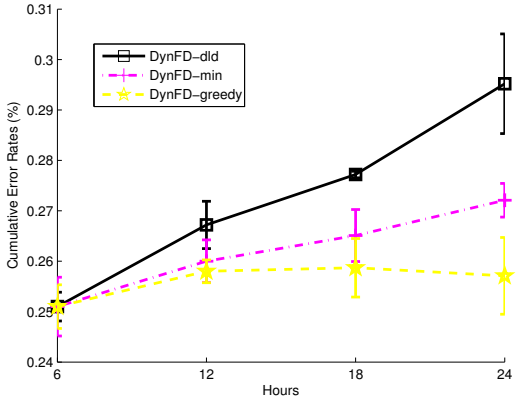

Figure 8: Cumulative error rates. The results were obtained using the botnet dataset.

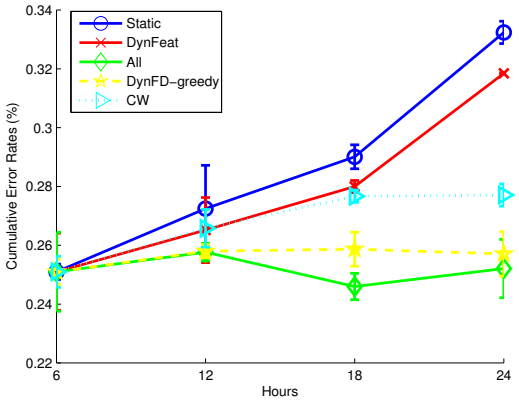

Figure 9: Cumulative error rates. The results were obtained using the botnet dataset.

ing the model). (2) **DynF**. In this approach, we train an initial model during the first time period and refines it at each subsequent time period using the *dynamic feature adaptation* approach (the training examples are assumed to be fixed). (3) **All**. In this approach, we add new features and new training examples at each iteration without removing older data examples (this approach assumes that the memory space is unbounded). (4) **CW**. This is the confidence-weighted online algorithm used in [9]. In addition to the baseline methods, we consider three variations of the proposed DynFD method: (1) **DynFD-old**. In this approach, we make room for the new training examples at each time period by removing the oldest training examples. (2) **DynFD-min**. In this approach, we accommodate the new training examples by removing training examples that have the lowest values of α. (3) **DynFD-greedy**. In this approach, we choose the older training examples to be removed according to Equation 14.

Figure 6 compares the performance of the three DynFD proposed methods on the benchmark malicious URL dataset [9]. The results suggest that both **DynFD-min** and **DynFD-greedy** achieve consistently low error rates compared to **DynFD-old**, which performed worse on datasets collected after Day 3. This is not surprising as **DynFD-old** may accidentally discard old but still useful training examples. Figure 7 compares the performance of **DynFD-greedy** against other the baseline algorithms. As shown in the figure, the method **All** which allows new examples to be added to the training set without removing other existing examples, appears to produce the best result along with the **CW** algorithm. The performance of **DynFD-greedy** is slightly worse, though the difference is not statistically significance (the error rates for all three methods are less than 2%). The other two methods, **Static** and **DynF**, appear to misclassify more test examples especially after Day 4. This is because these methods do not include newer training examples for model re-building.

Figures 8 and 9 show the performance of the seven methods on the botnet dataset. In Figure 8, we compare the three variants of the DynFD method. The results suggest **DynFD-greedy** clearly outperformed the other two methods. Furthermore, in Figure 9, we compare it against other baseline algorithms. While the **All** method once again yields

the best result, our proposed **DynFD-greedy** came in close second. On this dataset, the **CW** algorithm was significantly worse than both **All** and **DynFD-greedy**. One possible explanation is that for the botnet dataset, which is noisy and harder to classify, the SVM-based methods employ the maximum-margin principle to achieve better generalization error. **Static** and **DynF** again yield the worst performance due to the fact that they do not accommodate additional training examples to rebuild the model.

8. CONCLUSIONS

This paper focuses on the problem of detecting bots in a large network using an online learning algorithm that is adaptive to both evolving features and training sets. Our assumption is that we can identify whether a client machine is a "bot" by examining the server IP-addresses they had visited in the past. Using two real-world data sets, we show that the proposed algorithm produces an accuracy comparable to its batch counterpart, while consuming significantly less computational resources. In addition, our proposed algorithm also outperformed the state-of-the-art CW algorithm when applied to large-scale botnet data obtained from an ISP.

9. ACKNOWLEDGMENTS

Feilong Chen and Pang-Ning Tan's research are supported in part by ONR grant number N00014-09-1-0663.

10. REFERENCES

[1] Snort network intrusion prevention and detection system, `http://www.snort.org`.

[2] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *SRUTI'06: Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, pp. 43-48, San Jose, CA, USA, 2006.

[3] E. Cooke, F. Jahanian, and D. Mcpherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *SRUTI'05: Proceedings of the 1st Workshop on Steps to Reducing Unwanted Traffic on the Internet*, pp. 39-44, Cambridge, MA, USA, 2005.

[4] T. Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G.Dedene, B. D. Moor, and J. Vandewalle. Benchmarking Least Square Support Vector Machine Classifiers. *Machine Learning*, 54(1):5–32, 2004.

[5] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *HotBots'07: Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, Cambridge, MA, USA, 2007.

[6] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, pp. 167-182, Boston, MA, USA, 2007.

[7] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *NDSS'08: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, San Diego, CA, USA, 2008.

[8] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious URLs. In *KDD'09: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1245-1254, Paris, France, 2009.

[9] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious URLs: An application of large-scale online learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, Montreal, Canada, 2009.

[10] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: finding P2P bots with structured graph analysis. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, Washington, DC, USA, 2010.

[11] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *NSDI'10: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation*, San Jose, CA, USA, 2010.

[12] P. Porras, H. Saidi, and V. Yegneswaran. An analysis of conficker's logic and rendezvous points. Technical report, SRI International, Menlo Park, CA, USA, 2009.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel distributed processing: Explorations in the microstructure of cognition*, Vol 1, MIT Press, pp. 318–362, 1986.

[14] D. Sculley and G. M. Wachman. Relaxed online SVMs for spam filtering. In *SIGIR '07: Proceedings of the 30th International ACM SIGIR conference on Research and development in Information Retrieval*, pp. 415–422, Amsterdam, Netherlands, 2007.

[15] J. Suykens, T. Gestel, J. Brabanter, B. Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific Pub, Singapore, 2002.

[16] D. Tax and P. Laskov. Online SVM learning: from classification to data description and back. In *NNSP'03: Proceedings of the IEEE 13th Workshop on Neural Networks for Signal Processing*, pp. 499–508, Toulouse, France, 2003.

[17] Q. Wang, Y. Guan, and X. Wang. SVM-based spam filter with active and online learning. In *TREC '06: Proceedings of the 15th Text Retrieval Conference*, Gaithersburg, Maryland, USA, 2006.

[18] J. Ye and T. Xiong. SVM versus least square SVM. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, pp. 644–651, San Juan, Puerto Rico, 2007.

[19] L. Zhang, J. Zhu, and T. Yao. An evaluation of statistical spam filtering techniques. In *ACM Transactions on Asian Language Information Processing*, 3(4), pp. 243-269, 2004.