Identifying Suspicious URLs: An Application of Large-Scale Online Learning

Justin Ma JTMA@CS.UCSD.EDU
Lawrence K. Saul SAUL@CS.UCSD.EDU
Stefan Savage SAVAGE@CS.UCSD.EDU
Geoffrey M. Voelker VOELKER@CS.UCSD.EDU
Department of Computer Science & Engineering, UC San Diego — 9500 Gilman Drive, La Jolla, CA 92093-0404

Abstract

This paper explores online learning approaches for detecting malicious Web sites (those involved in criminal scams) using lexical and host-based features of the associated URLs. We show that this application is particularly appropriate for online algorithms as the size of the training data is larger than can be efficiently processed in batch *and* because the distribution of features that typify malicious URLs is changing continuously. Using a real-time system we developed for gathering URL features, combined with a real-time source of labeled URLs from a large Web mail provider, we demonstrate that recently-developed online algorithms can be as accurate as batch techniques, achieving classification accuracies up to 99% over a balanced data set.

1. Introduction

As new communications technologies drive new opportunities for commerce, they inevitably create new opportunities for criminal actors as well. The World Wide Web is no exception to this pattern, and today millions of rogue Web sites advance a wide variety of scams including marketing counterfeit goods (e.g., pharmaceuticals or luxury watches), perpetrating financial fraud (e.g., "phishing") and propagating malware (e.g., via "drive-by" exploits or social engineering). What all of these activities have in common is the use of the Uniform Resource Locator (URL) as a vector to bring Internet users into their influence. Thus, each time a user decides whether to click on an unfamiliar URL they must implicitly evaluate the associated risk. Is that URL safe to click on, or will it expose the user to potential exploitation? Not surprisingly, this can be a difficult judgment for individual users to make.

As a result, security researchers have developed various systems to protect users from their uninformed choices. By far the most common technique, deployed in browser toolbars, Web filtering appliances and search engines, is "blacklisting." Using this approach, a third-party service compiles the names of "known bad" Web sites (labeled by combinations of user feedback, Web crawling and heuristic analysis of site content) and distributes the list to its subscribers. While such systems have minimal query overhead (just searching for a URL within the list) they can only offer modest accuracy since it is impractical for any blacklist to be comprehensive and up-to-date (Sinha et al., 2008). Thus, a user may click on a malicious URL before it appears on a blacklist (if it ever does). Alternatively, some systems also intercept and analyze full Web site content as it is downloaded. This approach can offer higher accuracy, but requires far more run-time overhead, and may inadvertently expose users to the very threats they seek to avoid.

In this paper, we focus on a complementary technique — lightweight real-time classification of the URL itself to *predict* whether the associated site is malicious. We use various lexical and host-based features of the URL for classification, but *exclude* Web page content. We are motivated by prior studies noting distinguishing commonalities in such features for malicious URLs (Chou et al., 2004; McGrath & Gupta, 2008). If properly automated, this technique can afford the low classification overhead of blacklisting while offering far greater accuracy (and may also be used as a low-cost pre-filter for more expensive techniques).

To that end, our paper's primary contribution is the successful application of online learning algorithms to the problem of predicting such malicious URLs. Our earlier work (Ma et al., 2009) is among several previous systems for URL classification that have relied on batch learning algorithms. However, we argue that online methods are far better suited to the practical nature of this problem for two reasons: (1) Online methods can process large numbers of examples far more efficiently than batch methods. (2) We need to adapt to changes in malicious URLs and their features over time.

To demonstrate this approach, we have built a URL classification system that uses a live feed of labeled URLs from a large Web mail provider, and that collects features for the

Appearing in *Proceedings of the 26^{th} International Conference on Machine Learning*, Montreal, Canada, 2009.

URLs in *real time* (see Figure 2). Using this data, we show that online algorithms can be more accurate than batch algorithms in practice because the amount of data batch algorithms can train on is resource-limited. We compare classical and modern online learning algorithms and find the Confidence-Weighted algorithm achieves accuracies up to 99% over a balanced data set. Finally, we show that continuous retraining over newly-encountered features is critical for adapting the classifier to detect new, malicious URLs.

We begin the rest of the paper by providing more background on the application of detecting malicious URLs, then describing the online algorithms we use for classification. Next, we describe our data collection methodology and evaluate the models over our data set of labeled URLs. Finally, we conclude with an overall discussion.

2. Application

Our goal is to detect malicious Web sites from the lexical and host-based features of their URLs. This section provides background for our application with respect to the features we use for URL classification, as well as placing our work in context with related work.

2.1. Features

Like our previous study (Ma et al., 2009), we analyze lexical and host-based features because they contain information about the URL and host that is straightforward to collect using automated crawling tools. Thus, the list of features is extensive, but not necessarily exhaustive.

We do not include the content of the Web page or the context of the URL (e.g., the page or email containing the URL) as features for several reasons. (1) Avoiding page content downloads is strictly safer. (2) Classifying a URL with a trained model is a lightweight operation compared to first downloading the page contents and then analyzing them. (3) We want to apply our methods on URLs regardless of the context in which they appear (pages, email, chat, calendars, games, etc.), so we are not tied to a particular application setting. And (4) reliably obtaining the content of a page can become an issue due to content "cloaking," whereby a malicious site may serve benign versions of a page to a honeypot IP address run by a security practitioner, but serve malicious versions to other users. Nevertheless, the evaluations in Section 5 show that classification with just lexical and host-based features of URLs, without any context, can still be highly accurate.

Table 1 lists the lexical and host-based feature types we consider and the number contributed by each type. Overall, lexical types account for 62% of features and host-based types account for 38%. We next describe the feature types and the motivation behind including them for classification.

Table 1. Feature breakdown on Day 100 of the experiments.

Lexical		Host-Based	
Feature type	**Count**	**Feature type**	**Count**
Hostname	835,764	WHOIS info	917,776
Primary domain	738,201	IP prefix	131,930
Path tokens	124,401	AS number	39,843
Last path token	92,367	Geographic	28,263
TLD	522	Conn. speed	52
Lexical misc.	6	Host misc.	37
Lexical	1,791,261	Host-Based	1,117,901

Lexical features: These features allow us to capture the property that malicious URLs tend to "look different" from benign URLs. For example, the appearance of the token '.com' in the URL 'www.ebay.com' is not unusual. However, the appearance of '.com' in 'www.ebay.com.phishy.biz' or 'phish.biz/www.ebay.com/index.php' could indicate an attempt by criminals to spoof the domain name of a legitimate commercial Web site. Similarly, we would like to capture the fact that there are certain red flag keywords that tend to appear in malicious URLs — e.g., 'ebayisapi' would appear frequently in the path of URLs attempting to spoof an eBay page.

To implement these features, we use a bag-of-words representation of tokens in the URL, where '/', '?', '.', '=', '-', and '_' are delimiters. We distinguish tokens that appear in the hostname, path, the top-level domain (TLD), primary domain name (the domain name given to a registrar), and last token of the path (to capture file extensions). Thus, 'com' in the TLD position of a URL would be a different token from 'com' in other parts of the URL. We also use the lengths of the hostname and the URL as features.

Host-based features: These features describe properties of the Web site host as identified by the hostname portion of the URL. They allow us to approximate "where" malicious sites are hosted, "who" own them, and "how" they are managed. We examine the following sets of properties to construct host-based features:

WHOIS information — This includes domain name registration dates, registrars, and registrants. So if a set of malicious domains are registered by the same individual, we would like to treat such ownership as a malicious feature.

Location — This refers to the host's geography, IP address prefix, and autonomous system (AS) number. So if malicious URLs tend to be hosted in a specific IP prefix of an Internet service provider (ISP), then we want to account for that disreputable ISP when classifying URLs.

Connection speed — If some malicious sites tend to reside on compromised residential machines (connected via cable or DSL), then we want to record the host connection speed.

Membership in blacklists — Over our experiments, 55% of malicious URLs were present in blacklists. Thus, although this feature is useful, it is still not comprehensive.

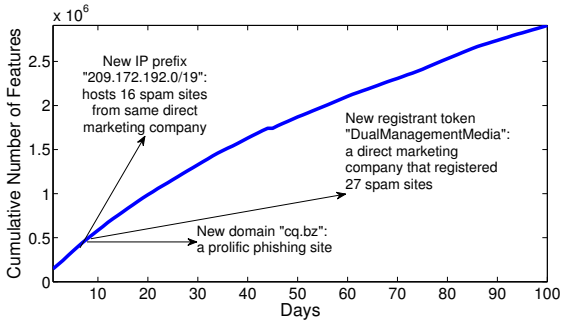


Figure 1. Cumulative number of features observed over time for our live URL feeds. We highlight a few examples of new features at the time they were introduced by new malicious URLs.

Other DNS-related properties — These include time-to-live (TTL), spam-related domain name heuristics (Rudd, 2007), and whether the DNS records share the same ISP.

Figure 1 shows the cumulative number of features for each day of the evaluations. Each day's total includes new features introduced that day and all old features from previous days (see Section 5 on our methodology for new features). The dimensionality grows quickly because we assign a binary feature for every token we encounter among the URL lexical tokens, as well as WHOIS and location properties. As we will show in Section 5.3, accounting for new features like the ones in Figure 1 is beneficial for detecting new malicious URLs.

2.2. Related Work

The most direct comparison to our work comes from Garera et al. (2007), who classify phishing URLs using logistic regression over 18 hand-selected features. The features include red-flag keywords, Google Page Rank and Web quality guidelines scores. Their classifier achieves 97.3% accuracy over a set of 2,500 URLs. Although similar in motivation and methodology, our study differs in scope (with evaluations focused on detecting spamming and phishing sites), scale (orders of magnitude more features and URLs), and algorithmic approach (online vs. batch learning).

Provos et al. (2008) study drive-by exploit URLs, and use a patented machine learning algorithm along with features derived from Web page content. Fette et al. (2007) and Bergholz et al. (2008) examine select properties of URLs contained within an email to aid the machine learning classification of phishing emails (not the URLs themselves).

Several projects have also explored operating systems-level techniques whereby the client visits the Web site using an instrumented virtual machine (VM) (Moshchuk et al., 2006; Wang et al., 2006; Provos et al., 2007). The VM can emulate any client-side exploits that occur as a result of visiting a malicious site, and the instrumentation can detect whether an infection has occurred. In this way, the VM serves as a protective buffer for the user.

Finally, many commercial efforts exist to protect users from visiting malicious URLs such as McAfee's SiteAdvisor, IronPort Web Reputation, WebSense ThreatSeeker Network, WOT Web of Trust, and Google Toolbar. These approaches are based on blacklist construction, user feedback, and proprietary feature analysis.

3. Online Algorithms

This section briefly describes the online learning algorithms we use for our evaluations. Formally, the algorithms are trying to solve an online classification problem over a sequence of pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_T, y_T)\}$, where each $\mathbf{x}_t$ is an example's feature vector and $y_t \in \{-1, +1\}$ is its label. At each time step t during training, the algorithm makes a label prediction $h_t(\mathbf{x}_t)$, which for linear classifiers is $h_t(\mathbf{x}) = \text{sign}(\mathbf{w}_t \cdot \mathbf{x})$.

After making a prediction, the algorithm receives the actual label y_t. (If $h_t(\mathbf{x}_t) \neq y_t$, we record an error for time t.) Then, the algorithm constructs the hypothesis for the next time step h_{t+1} using h_t, $\mathbf{x}_t$ and y_t.

As practitioners, we have no vested interest in any particular strategy for online learning. We simply want to determine the approach that scales well to problems of our size and yields the best performance. To that end, the online methods we evaluate are a mix of classical and recent algorithms. We present the models in order of increasing sophistication with respect to the objective functions and the treatment of classification margin (which we can also interpret as classification confidence).

Perceptron: This classical algorithm is a linear classifier that makes the following update to the weight vector whenever it makes a mistake (Rosenblatt, 1958):

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t \tag{1}$$

The advantage of the Perceptron is its simple update rule. However, because the update rate is fixed, the Perceptron cannot account for the severity of the misclassification. As a result, the algorithm can overcompensate for mistakes in some cases and undercompensate for mistakes in others.

Logistic Regression with Stochastic Gradient Descent: Many batch algorithms use gradient descent to optimize an objective function that is expressed as a sum of the examples' individual objective functions. Stochastic gradient descent (SGD) provides an *online* means for approximating the gradient of the original objective, whereby the model parameters are updated incrementally by the gradients of individual objectives. In this paper we evaluate SGD as applied to logistic regression.

Let $P(y_t = +1|\mathbf{x}_t) = \sigma(\mathbf{w} \cdot \mathbf{x}_t)$ be the likelihood that example t's label is $+1$, where the sigmoid function is $\sigma(z) = [1 + e^{-z}]^{-1}$. Moreover, let $L_t(\mathbf{w}) = \log \sigma(y_t(\mathbf{w} \cdot \mathbf{x}_t))$ be the log-likelihood for example t. Then the update for each example in logistic regression with SGD is as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \gamma \frac{\partial L_t}{\partial \mathbf{w}} = \mathbf{w}_t + \gamma \Delta_t \mathbf{x}_t \tag{2}$$

where $\Delta_t = \frac{y_t+1}{2} - \sigma(\mathbf{w}_t \cdot \mathbf{x}_t)$ and γ is a constant training rate. We do not decrease γ over time so that the parameters can continually adapt to new URLs. The update resembles a Perceptron, except with a learning rate that is proportional to Δ_t, the difference between the actual and predicted likelihood that the label is $+1$. This multiplier allows the model to be updated (perhaps by a small factor) even when there is no prediction mistake.

SGD has received renewed attention because of recent results on the convergence of SGD algorithms and the casting of classic algorithms as SGD approximations (Bottou, 1998; Bottou & LeCun, 2004). For example, the Perceptron can be viewed as an SGD minimization of the hinge-loss function $Loss(\mathbf{w}) = \sum_t \max\{0, -y_t(\mathbf{w} \cdot \mathbf{x}_t)\}$.

Passive-Aggressive (PA) Algorithm: The goal of the Passive-Aggressive algorithm is to change the model as little as possible to correct for any mistakes and low-confidence predictions it encounters (Crammer et al., 2006). Specifically, with each example PA solves the following optimization:

$$\begin{array}{rcll} \mathbf{w}_{t+1} & \leftarrow & \underset{\mathbf{w}}{\operatorname{argmin}} & \frac{1}{2}\|\mathbf{w}_t - \mathbf{w}\|^2 \\ & & \text{s.t.} & y_i(\mathbf{w} \cdot \mathbf{x}_t) \geq 1 \end{array} \tag{3}$$

Updates occur when the inner product does not exceed a fixed confidence margin — i.e., $y_t(\mathbf{w}_t \cdot \mathbf{x}_t) < 1$. The closed-form update for all examples is as follows:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t y_t \mathbf{x}_t \tag{4}$$

where $\alpha_t = \max\{\frac{1-y_t(\mathbf{w}_t \cdot \mathbf{x}_t)}{\|\mathbf{x}_t\|^2}, 0\}$. (The details of the derivation are in Crammer et al., 2006.) The PA algorithm has been successful in practice because the updates explicitly incorporate the notion of classification confidence.

Confidence-Weighted (CW) Algorithm: The idea behind Confidence-Weighted classification is to maintain a different confidence measure for each feature so that less confident weights are updated more aggressively than more confident weights. The "Stdev" update rule for CW is similar in spirit to PA. However, instead of describing each feature with a single coefficient, CW describes per-feature confidence by modeling uncertainty in weight w_i with a Gaussian distribution $\mathcal{N}(\mu_i, \Sigma_i)$ (Dredze et al., 2008; Crammer et al., 2009). Let us denote $\boldsymbol{\mu}$ as the vector of feature means, and $\boldsymbol{\Sigma}$ as the *diagonal* covariance matrix (i.e., the confidence) of the features. Then the decision rule becomes $h_t(\mathbf{x}) = \text{sign}(\boldsymbol{\mu}_t \cdot \mathbf{x})$ — which is the result of computing the average signed margin $\mathbf{w}_t \cdot \mathbf{x}$, where $\mathbf{w}_t$ is drawn from $\mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$, and then taking the sign.

The CW update rule adjusts the model as little as possible so that $\mathbf{x}_t$ can be correctly classified with probability η. Specifically, CW minimizes the KL divergence between Gaussians subject to a confidence constraint at time t:

$$\begin{array}{c} (\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) \leftarrow \underset{\boldsymbol{\mu}, \boldsymbol{\Sigma}}{\operatorname{argmin}} \, \mathrm{D}_{KL}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)) \\ \text{s.t. } y_i(\boldsymbol{\mu} \cdot \mathbf{x}_t) \geq \Phi^{-1}(\eta) \sqrt{\mathbf{x}_t^\top \boldsymbol{\Sigma} \mathbf{x}_t} \end{array} \tag{5}$$

where Φ is the cumulative distribution function of the standard normal distribution. This optimization yields the following closed-form update:

$$\begin{array}{rcl} \boldsymbol{\mu}_{t+1} & \leftarrow & \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \mathbf{x}_t \\ \boldsymbol{\Sigma}_{t+1}^{-1} & \leftarrow & \boldsymbol{\Sigma}_t^{-1} + \alpha_t \phi u_t^{-\frac{1}{2}} \, \text{diag}^2(\mathbf{x}_t) \end{array} \tag{6}$$

where α_t, u_t and ϕ are defined in Crammer et al. (2009). However, we can see that if the variance of a feature is large, the update to the feature mean will be more aggressive. As for performance, the run time of the update is linear in the number of non-zero features in $\mathbf{x}$.

Overall, because the CW algorithm makes a fine-grain distinction between each feature's weight confidence, CW can be especially well-suited to detecting malicious URLs since our data feed continually introduces a dynamic mix of new and recurring features.

Related Algorithms: We experimented with nonlinear classification using online kernel-based algorithms such as the Forgetron (Dekel et al., 2008) and the Projectron (Orabona et al., 2008). To make computation tractable, these algorithms budget (or at least try to reduce) the size of the support set used for kernel calculations. Our preliminary evaluations revealed no improvement over linear classifiers. However, we are continuing to evaluate budget algorithms for long term use.

4. Data Collection

This section describes our live sources of labeled URLs and the system we deploy to collect features in real time. Figure 2 illustrates our data collection architecture, which starts with two feeds of malicious and benign URLs.

We obtain examples of malicious URLs from a large Web mail provider, whose live, real-time feed supplies 6,000-7,500 examples of spam and phishing URLs per day. The malicious URLs are extracted from email messages that users manually label as spam, run through pre-filters to extract easily-detected false positives, and then verified manually as malicious.

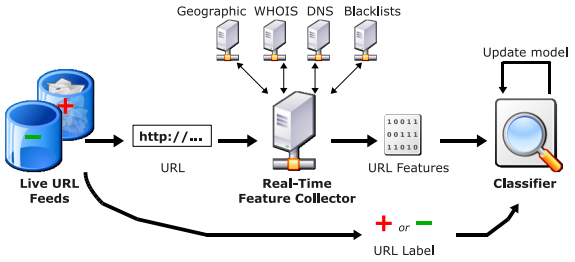


Figure 2. Overview of real-time URL feed, feature collection, and classification infrastructure.

We randomly draw our examples of benign URLs from Yahoo's directory listing. A random sample from this directory can be generated by visiting the link `http://random.yahoo.com/bin/ryl`.

Combined, we collect a total of 20,000 URLs per day from the two URL feeds, and the average ratio of benign-to-malicious URLs is 2-to-1. We ran our experiments for 100 days, collecting nearly 2 million URLs (there were feed outages during Days 35–40). However, the feeds only provide URLs, not the accompanying features.

Thus, we deploy a system to gather features in real-time. The real-time aspect is important because we want the values to reflect the features a URL had when it was first introduced to the feed (which ideally reflects values for when it was introduced to the wild). For every incoming URL, our feature collector immediately queries DNS, WHOIS, blacklist and geographic information servers, as well as processing IP address-related and lexical-related features.

Our live feed is notably different from data sets such as the `webspam` set from the PASCAL Large Scale Learning Challenge (Sonnenburg et al., 2008). Our application uses URLs as a starting point, and it is our responsibility to fetch lexical and host-based features in real-time to construct the data set on an ongoing basis. By contrast, the `webspam` set is a static representation of Web pages (using strings), not URLs, and provides no notion of the passage of time.

Finally, our live feed provides a real-time snapshot of malicious URLs that reflect the evolving strategies of Internet criminals. The freshness of this data suggests that good classification results over the set will be a strong indicator of future success in real-world deployments.

5. Evaluation

In this section, we evaluate the effectiveness of online learning over the live URL feed. To demonstrate this effectiveness, we address the following questions: Do online algorithms provide any benefit over batch algorithms? Which online algorithms are most appropriate for our application? And is there a particular training regimen that fully realizes the potential of these online classifiers?

By "training regimen", we refer to (1) when the classifier is allowed to retrain itself after attempting to predict the label of an incoming URL, and (2) how many features the classifier uses during training.

For (1), we compare "continuous" vs. "interval-based" training. Under the "continuous" training regimen, the classifier may retrain its model after each incoming example (the typical operating mode of online algorithms). In the "interval-based" training regimen, the classifier may only retrain after a specified time interval has passed. In our experiments, we set the interval to be one day. Batch algorithms are restricted to interval-based training, since continuous retraining would be computationally impractical. Unless otherwise specified, we use continuous retraining for all experiments with online algorithms (and then evaluate the benefit of doing so in Section 5.3).

For (2), we compare training using a "variable" vs. "fixed" number of features. Under the fixed-feature regimen, we train using a pre-determined set of features for all evaluation days. For example, if we fix the features to those encountered up to Day 1, then we use those 150,000 features for the whole experiment (see Figure 1). Under the variable-feature regimen, we allow the dimensionality of our models to grow with the number of new features encountered; on Day 8, for instance, we classify with up to 500,000 features. Implicitly, examples that were introduced before a feature i was first encountered will have value 0 for feature i. Unless otherwise specified, we use the variable-feature training regimen for all algorithms (and then evaluate the benefit of doing so in Section 5.3).

As for the sizes of the training sets, online algorithms implicitly train on a cumulative data set, since they can incrementally update models from the previous day. For batch algorithms, we vary the training set size to include day-long and multi-day sets (details in Section 5.1).

5.1. Advantages of Online Learning

We start by evaluating the benefit of using online over batch algorithms for our application in terms of classification accuracy — in particular, whether the benefit of efficient computation in online learning comes at the expense of accuracy. Specifically, we compare the online Confidence-Weighted (CW) algorithm against four different training set configurations of a support vector machine. We use the `LIBLINEAR` implementation of an SVM with a linear-kernel as our canonical batch algorithm (Fan et al., 2008). Evaluations with other batch algorithms such as logistic regression yielded similar results.

Figure 3 shows the classification rates for CW and for SVM using four types of training sets. We tuned all classifier parameters over one day of holdout data, setting $C = 100$

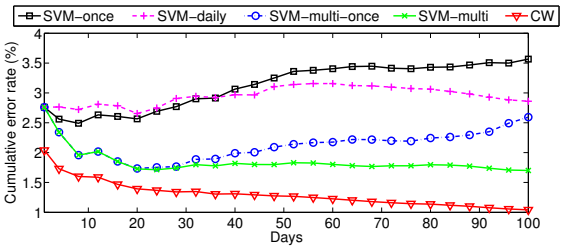


Figure 3. Cumulative error rates for CW and for batch algorithms under different training sets. Note the y-axis starts at 1%.

for SVM, and $\eta = 0.90$ for CW. The x-axis shows the number of days in the experiment, and the y-axis shows the cumulative error rate: the percentage of misclassified examples for all URLs encountered up to that date.

The SVM-once curve represents training once on Day 0's data and using that model for testing on all other days. The cumulative error steadily worsens to 3.5%, and the per-day false negative rate gets as high as 10–15%. These high error rates suggests that, to achieve better accuracy, the model must train on fresh data to account for new features of malicious *and* benign URLs encountered over time.

SVM-daily retrains only on data collected the previous day — e.g., Day 6 results reflect training on the URLs collected on Day 5, and testing on Day 6 URLs. The only exception is that we do not retrain during the feed outages on Days 35–40. As a result, the cumulative error is just under 3%, most of which is due to high per-day false negatives (5–15% on some days), whereas per-day false positives are around 1.5%. Although fresh data eventually helps SVM-daily improve over SVM-once, one day's training data is still insufficient.

We use multi-day training sets to address this issue by training on as much data as our evaluation machine with 4 GB RAM can handle (which is 14–17 days worth, or 280,000–340,000 examples). SVM-multi-once is the multi-day analogue to SVM-once. Here, SVM-multi-once trains on data from Days 0 to 16, and from Day 17 on it uses that fixed model for testing on subsequent days. The improvement over SVM-once shows the benefit of more training data, but the steadily worsening error again demonstrates the nonstationarity of the URL data set.

SVM-multi is the multi-day analogue of SVM-daily. Here, SVM-multi trains on the previous 14–17 days worth of data (depending on what can fit in memory). The resulting cumulative error reaches 1.8%. SVM-multi's improvement over SVM-multi-once suggests the URL feature distribution evolves over time, thus requiring us to use as much fresh data as possible to succeed. Overall, these results suggest that more training data yields better accuracy. However, this accuracy is fundamentally limited by the amount of computing resources available.

Fortunately, online algorithms do not have that limitation. Moreover, they have the added benefit that they can incrementally adapt to new data. As we see in Figure 3, the accuracy for CW beats SVM-multi. Since the online algorithm is making a *single pass* over a cumulative training set, it does not incur the overhead of loading the entire data set in memory. Because its training is incremental, it is capable of adapting to new examples in real time, whereas batch algorithms are restricted to retraining at the next available interval (more on interval-based training in Section 5.3). These advantages allow the online classifier to have the best accuracy in our experiments.

5.2. Comparison of Online Algorithms

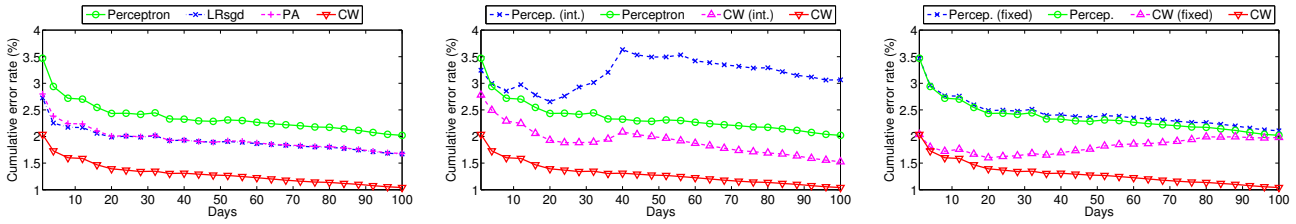
Given the demonstrated benefits of online learning over batch learning, we next evaluate which of the online algorithms from Section 3 are best suited to malicious URL detection. The main issue that these experiments address is whether recent developments in online algorithms, which include optimizing different objective functions, adjusting for classification confidence, and treating features differently, can benefit the classifiers in our application.

Figure 4(a) shows the cumulative error rates for the online algorithms. All algorithms in this experiment adopt the continuous training regimen. We also note that the error rates improve steadily over time for all classifiers, reaffirming that training on cumulative data is beneficial.

The Perceptron is the simplest of the algorithms, but it also has the highest error rates across all of the days at around 2–3%. This result suggests that because the Perceptron treats mistakes equally (and ignores all correct classifications), its updates are too coarse to accurately keep up with new examples. There needs to be a more fine-grain distinction between misclassified and correctly-classified examples with respect to their impact on model updates.

Both logistic regression with stochastic gradient descent (LRsgd) and the Passive-Aggressive (PA) algorithm achieve a cumulative error approaching 1.6%, improving over the Perceptron results. (Here we tuned the LRsgd learning rate to $\gamma = 0.01$ over one day of holdout data.) Presumably, this improvement occurs because LRsgd and PA account for classification confidence. Specifically, LRsgd updates are proportional to Δ_t, and PA updates are proportional to the normalized classification margin α_t. These results are comparable to SVM-multi.

The CW results suggest that the final leap comes from *treating features differently* — both in terms of how they affect classification confidence, and how quickly they should be updated. With an error approaching 1%, CW clearly outperforms the other algorithms. Most of the gap between CW and the other online methods comes from CW's lower



(a) Error rates for online algorithms. All use continuous/variable-feature training.

(b) Benefits of using continuous training over interval-based training.

(c) Benefits of using variable-feature sets over fixed-feature sets.

Figure 4. Comparing the effectiveness of various online algorithms, their use of continuous vs. interval training, and their use of fixed vs. variable feature sets.

false negatives — CW has 1–2% false negatives per day, whereas others have 2–4%. We hypothesize the gap occurs because CW can update select portions of its model very aggressively to account for new malicious features, all without perturbing more established features.

Overall, we find that the more recent online algorithms outperform the simpler ones. Because the live combined URL feed contains a dynamic mix of new and recurring features, CW's per-feature confidence weighting can exploit that structure to achieve the best accuracy.

5.3. Training Regimen

In this section, we show that there is a significant advantage to continuous training vs. interval-based training. We also demonstrate that there is significant benefit to adding newly-encountered features as opposed to using a fixed feature set. The aforementioned training regimens can help online algorithms stay abreast of changing trends in URL features. Thus, choosing the right training regimen can be just as important as choosing the right algorithm.

Figure 4(b) shows the value of using continuous training over interval training with the CW and Perceptron algorithms. The higher error rates for interval training show that there is enough variation between days that a model can become stale if it is not retrained soon enough. In particular, the higher number of false negatives for interval-trained CW is responsible for the persistent gap with continuously-trained CW. Notwithstanding the aforementioned feed outages on Days 35–40, the 1% error difference between continuous and interval-based Perceptron is due to spikes in the false positive/negative rates for the interval-trained Perceptron. Thus, continuous retraining yields as much improvement for the simpler Perceptron as it does for CW.

In addition to continuous retraining, accounting for new features is critical to an algorithm's success. Figure 4(c) shows the value of using variable-feature training over fixed-feature training. In this graph, "fixed features" means that we restrict the model to using the features encountered on Day 1 only (150,000 features total). We see that the performance for fixed-feature CW degrades to a point that it is no better than a Perceptron. Interestingly, variable-feature Perceptron only achieves a marginal improvement over fixed-feature Perceptron. One explanation is that, even though variable-feature Perceptron can occasionally benefit from adding new features, it does not update the new feature weights aggressively enough to correct for future errors. By contrast, the CW algorithm updates new features aggressively *by design*, and hence can reap the full benefits of variable-feature training.

Overall, continuous retraining with a variable feature set allows a model to successfully adapt to new data and new features on a sub-day granularity. And this adaptiveness is critical to the full benefits of online algorithms.

6. Conclusion

Malicious Web sites are a prominent and undesirable Internet scourge. To protect end users from visiting those sites, the identification of suspicious URLs using lexical and host-based features is an important part of a suite of defenses. However, URL classification is a challenging task because new features are introduced daily — as such, the distribution of features that characterize malicious URLs evolves continually. With an eye toward ultimately constructing a real-time malicious URL detection system, we evaluated batch and online learning algorithms for our application to study their benefits and tradeoffs.

Experiments over a live URL feed revealed the limitations of batch algorithms in this setting, where we were forced to make a tradeoff between accuracy and coping with resource limitations (e.g., running out of memory). We demonstrated that recently-developed online algorithms such as CW can be highly accurate classifiers, capable of achieving classification accuracies up to 99%. Furthermore, we showed that retraining algorithms continuously with new features is crucial for adapting successfully to the ever-evolving stream of URLs and their features.

Acknowledgments

We thank Koby Crammer, Mark Dredze, Fernando Pereira and Boris Babenko for many useful discussions. We also thank the anonymous reviewers for their valuable feedback. This work was supported by National Science Foundation grants NSF-0238323, NSF-0433668 and NSF-0829469 and by generous research, operational and in-kind support from Cisco, Google, Microsoft, Yahoo and the UCSD Center for Networked Systems.

References

Bergholz, A., Chang, J.-H., Paaß, G., Reichartz, F., & Strobel, S. (2008). Improved Phishing Detection using Model-Based Features. *Proceedings of the Conference on Email and Anti-Spam (CEAS)*. Mountain View, CA.

Bottou, L. (1998). Online Learning and Stochastic Approximations. In *Online Learning and Neural Networks*, 9–42. Cambridge, UK: Cambridge University Press.

Bottou, L., & LeCun, Y. (2004). Large Scale Online Learning. In S. Thrun, L. K. Saul and B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems 16*, 217–224. Cambridge, MA: MIT Press.

Chou, N., Ledesma, R., Teraguchi, Y., Boneh, D., & Mitchell, J. C. (2004). Client-Side Defense against Web-Based Identity Theft. *Network and Distributed System Security (NDSS)*. San Diego, CA.

Crammer, K., Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2006). Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, *7*, 551–585.

Crammer, K., Dredze, M., & Pereira, F. (2009). Exact Convex Confidence-Weighted Learning. *Advances in Neural Information Processing Systems 21* (pp. 345–352).

Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2008). The Forgetron: A Kernel-Based Perceptron on a Budget. *SIAM Journal on Computing*, *37*, 1342–1372.

Dredze, M., Crammer, K., & Pereira, F. (2008). Confidence-Weighted Linear Classification. *Proceedings of the International Conference on Marchine Learning (ICML)* (pp. 264–271). Helsinki, Finland: Omnipress.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). LIBLINEAR: A Library for Large Linear Classification. http://www.csie.ntu.edu.tw/ cjlin/liblinear/.

Fette, I., Sadeh, N., & Tomasic, A. (2007). Learning to Detect Phishing Emails. *Proceedings of the International World Wide Web Conference (WWW)* (pp. 649–656). Banff, Alberta, Canada.

Garera, S., Provos, N., Chew, M., & Rubin, A. D. (2007). A Framework for Detection and Measurement of Phishing Attacks. *Proceedings of the ACM Workshop on Rapid Malcode (WORM)* (pp. 1–8). Alexandria, VA.

Ma, J., Saul, L. K., Savage, S., & Voelker, G. M. (2009). Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs. *Proceedingsof the SIGKDD Conference*. Paris, France.

McGrath, D. K., & Gupta, M. (2008). Behind Phishing: An Examination of Phisher Modi Operandi. *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. San Francisco, CA.

Moshchuk, A., Bragin, T., Gribble, S. D., & Levy, H. M. (2006). A Crawler-Based Study of Spyware on the Web. *Network and Distributed System Security (NDSS)*. San Diego, CA.

Orabona, F., Keshet, J., & Caputo, B. (2008). The Projectron: A Bounded Kernel-Based Perceptron. *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 720–727). Helsinki, Finland: Omnipress.

Provos, N., Mavrommatis, P., Rajab, M. A., & Monrose, F. (2008). All Your iFRAMEs Point to Us. *Proceedings of the USENIX Security Symposium* (pp. 1–15). San Jose, CA.

Provos, N., McNamee, D., Mavrommatis, P., Wang, K., & Modadugu, N. (2007). The Ghost in the Browser Analysis of Web-based Malware. *Proceedings of the USENIX Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, *65*, 386–408.

Rudd, J. (2007). Botnet plugin for SpamAssassin. `http://people.ucsc.edu/~jrudd/spamassassin/`.

Sinha, S., Bailey, M., & Jahanian, F. (2008). Shades of Grey: On the Effectiveness of Reputation-Based Blacklists. *Proceedings of the International Conference on Malicious and Unwanted Software (Malware)* (pp. 57–64). Alexandria, VA.

Sonnenburg, S., Franc, V., Yom-Tov, E., & Sebag, M. (2008). PASCAL Large Scale Learning Challenge. http://largescale.first.fraunhofer.de/workshop/.

Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., & King, S. (2006). Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities. *Network and Distributed System Security (NDSS)*. San Diego, CA.