Approaches to Adversarial Drift

Alex Kantchelian
UC Berkeley

Sadia Afroz
Drexel University

Ling Huang
Intel Labs

Aylin Caliskan Islam
Drexel University

Brad Miller
UC Berkeley

Michael Carl Tschantz
UC Berkeley

Rachel Greenstadt
Drexel University

Anthony D. Joseph
UC Berkeley

J. D. Tygar
UC Berkeley

ABSTRACT

In this position paper, we argue that to be of practical interest, a machine-learning based security system must engage with the human operators beyond feature engineering and instance labeling to address the challenge of drift in adversarial environments. We propose that designers of such systems broaden the classification goal into an *explanatory* goal, which would deepen the interaction with system's operators.

To provide guidance, we advocate for an approach based on maintaining one classifier for each class of unwanted activity to be filtered. We also emphasize the necessity for the system to be *responsive* to the operators constant curation of the training set. We show how this paradigm provides a property we call *isolation* and how it relates to classical causative attacks.

In order to demonstrate the effects of drift on a binary classification task, we also report on two experiments using a previously unpublished malware data set where each instance is timestamped according to when it was seen.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; D.4.6 [**Security and Protection**]: Invasive software; H.1.2 [**User/Machine Systems**]: Human information processing

Keywords

Adversarial machine learning; concept drift; malware classification

1. INTRODUCTION

In the setting of security, classifiers employing machine learning must respond to instances crafted by an adversary. Such adversaries can and do introduce changes, or *adversarial drift*, to the instances they craft. The adversary may either design changes to evade the classifier immediately or to make future evasion easier. To handle such adversarial drift, classifiers must be retrained over time to learn each new attack. They must be eager to learn to react quickly to new attacks designed for immediate evasion. However, they must not naively overreact and fall for attacks designed to degrade long-term performance. Thus, a tension exists between the system being responsive and being skeptical. We argue that managing this trade off in real-world settings requires machine learning algorithms that cooperate in an understandable fashion with humans.

As decision-making tools using machine learning gain popularity in the security settings, managing such trade offs will become increasingly important. Such tools already exist for email spam filtering [14, 28, 34, 43], virus and malware detection [50, 36], and detection of malicious advertisements [41], Javascript [9], PDFs [46] and URLs [48]. It is no surprise automation in the form of machine learning has been extensively invoked for these high-volume and time-sensitive applications where no raw human resource can economically operate [49].

Most of the existing research in the development of machine learning algorithms and systems is focused on one-shot solutions and non-adversarial parties, conditions that fail to hold in security applications. Existing research often proceeded to collect a data set, learn a model on a portion of the data, evaluate the model on the other portion, and finally claim success. However, in practice, deploying and maintaining such systems for security applications involves big, non-stationary data streams that *drift* over time. One-shot approaches fail to account for changing trends in the data and are incapable of detecting novel events emerging in the data, thus degrading performance of machine learning-based security systems.

In addition, security applications often face adversaries who may game the machine learning system to evade its detection, or devise data samples to misguide the training process of the machine learning system for future evasion. For example, an adversary may issue a set of carefully crafted queries to evaluate the impact of each feature value on classification results. Using this information, the adversary can manipulate the feature values of her sample to achieve her desired classification result (i.e., craft the sample to appear benign to the present classifier).

In this paper, we examine in detail the challenges of applying machine learning systems for security applications, and discuss new requirements for designing adaptive, adversarial-resistant machine learning systems that succeed in realistic security application environments.

We argue that to be of practical interest, a machine-learning based security system must evolve over time and engage with human ex-

AISec'13, November 4, 2013, Berlin, Germany.

ACM 978-1-4503-2488-5/13/11 ...$15.00.
http://dx.doi.org/10.1145/2517312.2517320

perts beyond feature engineering and instance labeling. In particular, in this position paper, we argue that *machine learning algorithms and their results must be understandable to their human operators for each to aid the other in overcoming adversarial drift*.

This position is not vacuously true: machine learning is often used as a black box and theoretical results show that discriminative classifiers (those that do not produce probabilistic models) typically perform better than generative classifiers (those that do produce them) [33]. We argue that the practical considerations of and structure of machine learning in an adversarial setting are more important. In particular we argue that we should:

- Use an ensemble of classifiers, one for each *family* of unwanted behavior.
- Ensure that the classifier is responsive to new training data.

The main technical construction of this argument is using multiple classifiers. Along the way we make the following novel points:

- Using an ensemble of classifiers, one for each family, can *isolate* malicious campaigns from one another limiting the effectiveness of causative attacks. Whereas many alternative constructions of classifier ensembles have been previously proposed, only a few constructions display this isolation property.
- We can and should expect zero-training error from the machine learning based security system we created.

Additionally, we show experimental evidence of the presence of temporal drift using a previously unpublished dataset from the malware domain.

Prior Work.

We summarize the most closely related works here to emphasize the context of our paper. However, most related works are discussed in later sections.

Our work attempts to explain a difference between academic research and production systems and to guide research in a more applicable direction. In particular, our promotion of classifiers using family-based ensembles generalizes deployed systems such as Google's systems for detecting malicious advertisements [41]. Cretu et al. proposed a similar system of ensembles of time-sliced models to sanitize training data for anomaly sensors [8]. The goal of their work was to clean the training data whereas we are focused on detecting specific attacks. Moreover, models trained on time sliced data would fail to capture the concept of different malware campaigns as the duration of different malware campaigns varies widely, as we noticed in our practical data set (Table 2).

Our work builds on that of Sommer and Paxson [45], who studied why academic approaches to using automated anomaly detection gain little traction in real deployed intrusion detection systems (IDS). They note that classification systems where decisions are opaque to operators are unlikely to be useful in practice. This gap between the filtering policies of the human operators and the actual classification algorithm has been called the *semantic gap*. Reducing the semantic gap means that the system provides the operators with a clear explanation of why it labeled a sample as it did. We argue that using family-based ensembles of classifiers can mitigate the authors' criticism by closing the semantic gap between the automated system and its operator's understanding of both the data and the filtering policies.

Other authors have looked at families within adversarial drift. For example, Singh et al. study population drift for three families of malware [44]. They order samples from each family by the compilation timestamp found in the executable to create a sequence of instances simulating order of apparition. They measure change across time of these sequences and conclude that families remain fairly stable. They also show that classifiers for each family trained on old data can successfully classify new instances. Srndic et al. show that their classifier for malicious PDFs continues to operate with acceptable but significantly degraded performance in this setting as compared to standard cross-validation [46]. Our experiments highlight the importance of evaluating classifiers by testing them over samples drawn chronologically. Although they do not directly contradict previous claims, as such were made using different features when not on different applications domains, our results warrant some dose of caution when it comes to performance claims on future adversarial instances.

Contents.

In Section 2, we cover general background including systems that use machine learning for security, and adversarial drift. Section 3 discusses previously published attacks and defenses proposed on such systems. This section has an intensive survey flavor as it provides perspectives on the nature of adversarial drift.

In Section 4, we discuss how the identification of families of malicious behavior can improve classification in the face of adversaries. In particular, we motivate such families as a useful and intuitive abstraction already used by humans. We discuss family-based ensembles of classifiers as proposed by Sculley et al. [41]. Finally, we contribute a novel observation about such ensembles: they help *isolate* effects of types of maliciousness from one another.

Section 5 looks more closely at the interactions between the human operators and the system, and recognize the operator's expectation of a *responsive* system. Responsiveness implies zero training errors and we will argue for both attainability and desirability of this goal. In particular, it reduces misclassification of the most important instances and makes the system more intuitive to human operators.

Section 6 presents our preliminary experiments on a previously unpublished executable malware dataset containing both malicious and benign instances with chronological appearance information for each instance, ranging from 2007 to 2013. Using this dataset, we conduct two experiments demonstrating the importance of temporal drift in a very adversarial environment.

2. MACHINE LEARNING FOR SECURITY

Many different systems for providing security use machine learning. For example, Google uses machine learning to identify websites engaged in Phishing [49]. Zozzle uses machine learning to identify malicious JavaScript programs [9]. Spam is typically identified using machine learning [14].

In this section, we give an overview of how these systems work. We start by providing a concrete example before presenting a general formalism. We end with a discussion of population drift.

2.1 Example

To be concrete and to provide background for the analysis we discuss in Section 6, we here focus on large-scale systems for learning and classifying malware binaries *automatically* based on their behavioral patterns [1, 35, 36]. Such behavior-based classification systems are often employed by anti-virus companies to generalize and augment signature-based anti-virus products to classify increasing amounts of malware from diverse families. The classification task usually consists of telling whether an unknown binary is malware, and if it is malware, which family it belongs to. However,

most of our comments are applicable with slight modifications to other security problems tackled with machine learning.

In general, the classification system learns the behavior patterns of malware from labeled samples and constructs models that can classify unknown binaries. The system proceeds first by analyzing binaries (both malicious and benign) to get the behavioral features of each binary, by either doing static binary analysis to obtain call graphs, or executing the binaries in a sandboxed environment to collect system call execution traces. Then the system converts and encodes the behavioral features into high-dimensional numerical vectors, where each dimension of a vector is associated with a behavioral pattern (e.g., a sequence of function calls). Then the system applies machine learning methods to the data consisting of both malware and benign vectors to learn classification models that can discriminate between malware binaries and benign binaries, and further classify malware binaries into corresponding families.

2.2 Formalism

More formally, we can abstract away from the specifics of malware detection to discuss classification tasks in general. The system must classify instances of some set $\mathcal{X}$ (e.g., executable files) that reach a client. Each instance x in $\mathcal{X}$ is either a positive or negative instance for some concept c that the system is attempting to determine (e.g., whether the file is malware). We treat the concept c as a function such that $c(x) = +$ if x is positive instance (e.g., x is malware) and $c(x) = -$ otherwise. We call an ordered pair $\langle x, y \rangle$ of an instance x and a label y in $\{+, -\}$ a *labeled instance*. If $y = c(x)$, the instance is correctly labeled; otherwise, it is *mislabeled*.

To approximate c, the system uses a machine learning algorithm that takes a set of the available information including labeled instances and produces a model about how to classify unlabeled instances. The system has the option of retraining to learn from new instances added to the set of available information. Thus, we can treat the system as producing a series of models h_t where t ranges over time. For simplicity, we will typically treat h_t as a function from instances in $\mathcal{X}$ to $\{+, -\}$, but in some cases a model provides additional information such as its confidence. If $h_t(x) = c(x)$, then the classifier correctly classifies the instance. Otherwise, h_t misclassifies the instance.

The instances that reach a client and consequently the system for classification are influenced by adversaries. In particular, the adversary may encourage the client to encounter instances that it believes that the current hypothesis h_t will misclassify. In this paper, we consider the ramifications of the adversary having partial control over the instances that the system encounters.

2.3 Population Drift

Over time the probability that a particular instance will be encountered by a client and thus the system can change as the adversary and other factors change. Thus, we model the probability of a client encountering an instance with a family of time indexed distributions X_t such that $X_t(x)$ is the probability of encountering instance x at time t. For example, X_t might not be equal to X_{t+1} since adversaries post new malware and make old malware more attractive for downloading while others post more benign software and make malware less attractive for downloading. The condition of X_t not equaling $X_{t'}$ is *population drift*.

Since the changes to the distributions X_t are party under the control of an adversary, we cannot expect them to obey a straightforward pattern. However, in many settings, each adversary has limited control over the distributions and cannot introduce arbitrary changes affecting large numbers of instances. Nevertheless, their introduction of previously unseen instances can lead to new false negatives. A goal of the system is to retrain quickly and accurately enough to identify new positive instances while their prevalence is still low enough that its introduction has not impacted clients too greatly. This goal requires the system to rapidly respond to novelty.

3. ATTACKS AND DEFENSES

In this section we present several types of attacks against classifiers which occur in adversarial settings and result in adversarial drift. We also present defenses which may be deployed in response to attacks. The first broad class of attacks occurs when an adversary reacts to a deployed classifier and attempts to evade detection. Following Huang et al. [17], we refer to these attacks as *exploratory* since the adversary is exploring the state of the classifier in an attempt to strategically craft malicious samples.

Machine learning also introduces another class of attacks: those that involve actions taken by an adversary to adversely influence training data and reduce the quality of a future classifier. In further keeping with Huang et al. we refer to this class of attacks as *causative* since the adversary is attempting to cause a problem in the state of the classifier [17]. Due to the iterative nature of the system, an exploratory attack may ultimately have causative effects. This occurs when an attack is mislabeled by the system, and subsequently used as training data. Thus, these two classes of attacks are not entirely separate in frequently retrained systems. Nevertheless, we discuss each class in turn since the second class highlights attacks targeting machine learning in particular.

3.1 Exploratory Attacks and Defenses

While we focus on machine learning algorithms, any detection mechanism is subject to exploratory attacks. They start by the adversary performing some amount of reverse engineering (exploration) to determine how the mechanism operates. From this understanding, the adversary conjectures that a mechanism suffers from some vulnerability that will allow it to pass a malicious instance off as a benign one.

For machine learning algorithms, the reverse engineering can start with the attacker developing a notion of which features are used by the system and the influence of each of those features. Although an in-depth discussion of efficient mechanisms for identifying features is beyond the scope of this work, it is important to realize that this process will require a measure of trial-and-error by the attacker and may produce samples which suggest adversarial activity. For example, an attacker may begin with a hypothesis composed of a list of features drawn from his intuition and prior literature, and issue a set of carefully crafted queries to evaluate the impact of each feature value on classification results. This process may result in submission of samples that contain an unlikely mix of feature values indicative of different classifications. These samples may be used by the defender to identify, characterize and defend against emerging attacks.

Having approximated the set of features used by the learning system and the impact of each feature on classification, the adversary must now manipulate the feature values of his sample in order to achieve his desired classification result. We separate exploratory attacks into three categories. Categories are distinguished by the attack mechanism and consequently cause drift in different fashions.

First, the attacker may manipulate samples to mutate their feature values in such a way that they are misclassified by the present classifier. Lowd et al. develop this type of attack against spam filters which use the individual words occurring in a message for classification [26]. By adding "good" words to a message to balance the presence of "bad" words, an attacker is able to decrease the

amount of blocked spam by 87.2% and 99.9% against naive Bayes and maximum entropy classifiers, respectively. However, the spam messages can still be detected using the same feature extraction methodology and learning algorithm: once the filter is retrained on the modified input, the amount of blocked spam returns to within 2% of original levels. We refer to this as a *cat-and-mouse* attack as the classifier and evader chase each other through the feature space. Cat-and-mouse attacks fundamentally work by exploiting the *inductive bias* that the algorithm uses to generalize from training instances to unseen instances [29].

Prior work contains several approaches to defend against feature manipulation in cat-and-mouse attacks. We present two examples of fully automated systems without human effort. Lee et al. design a HMM based method for email spam de-obfuscation that operates like an automated spell-checker on manipulated email content to recover the intended words [23]. Sculley et al. use kernel SVM-inspired techniques from genomics for fast approximate string matching to deal with word manipulation [42].

Second, moving to a more complex attack, we consider the case in which the attacker is able to manipulate feature values in such a way that the original classifier and algorithm are no longer able to produce accurate results through retraining on manipulated data, but it is still possible to produce accurate classification using either different features or a different algorithm applied to the original features. We say that such attacks exploit a *blind spot* in the feature space or model class since content can still be classified using the data underlying the original features, but the necessary distinctions are not captured by either the current features or the current model class.

Privacy technologies that attempt to prevent a third-party from recovering sensitive data often employ blind spot attacks. Notice that in this case the traditional roles have been reversed, and the "white hat" is now the party attacking the classification system. Wright et al. develop a technique called *traffic morphing* which attempts to alter the features of encrypted traffic to prevent traffic analysis attacks [52]. Evaluated against the original classification algorithms, traffic morphing is able to decrease web traffic classification accuracy from 98% to 63.4%, and VoIP traffic classification accuracy from 76% to 50%. However, by applying different features and different classification algorithms to data protected using traffic morphing, Dyer et al. are able to achieve 89% accuracy [11].

Third, the last class of attacks is distinguished by the attacker attempting to entirely circumvent the type of data which is supporting a set of features. We refer to this as a *data nullification* attack since any classifier derived from the existing data, no matter how good the features and the learning algorithm are, can no longer achieve accurate classification. Data nullification attacks occur when cloaking is used to prevent the classification of malvertising and other malicious web content using features extracted from the content itself. In response to cloaking, techniques have been developed which inspect redirection chains followed by the browser in the process of resolving a URL to do classification [24].

In the data nullification attack the malicious content is not captured by the original data, so the features derived from the data can no longer be used to detect the malicious content. The model trained using the same data and features will not solve the problem no matter how good the learning algorithm is. A human expert must carefully investigate the malicious activity conducted by the adversaries in the new setting, develop a new feature space that can distinguish bad from good, and design new measurement infrastructure to collect data and extract features to feed into the ML system to model the malicious activity. Examples include classifying malicious URLs and advertisements using the structure of their redirection chains [27, 24].

Finally, we mention the game theoretic line of work [5, 6, 10, 25]. There, under simplifying assumptions, equilibrium conditions between an attacker and a defender can be translated into optimal learners. A merit of this elegant approach is the principled framework for finely studying and improving the robustness of machine learning algorithms. We note however that the resulting models are currently computationally impractical for large-scale decision problems.

3.2 Causative Attacks and Defenses

In this section we examine attempts to poison training data, referred to as causative attacks. As discussed in the previous section, exploratory attacks alter attack instances in response to the current classifier state. Consequently, drift occurring as a result of exploratory attacks is analogous to a moving target which the classifier must follow. In contrast, drift resulting from causative attacks is best understood as malicious shifts in the distribution of training data, causing the model to change and misclassify instances. Rather than attack instances drifting, the distribution of training data itself drifts and causes classification errors.

Systems which require regular iterative retraining to maintain up-to-date classifications are particularly susceptible to manipulation since some training data labels may come directly from classification results and not have an opportunity for human review. Thus, such systems expose an additional vector of attack to the adversary. Previous work has shown that small amounts of mislabeled or strategically constructed adversarial training data can have outsized bad effects on the accuracy of the ML system [7, 40]. The attacker may maliciously produce an instance which is designed to receive a particular target label from the classifier, get it into the training data set on which the classifier is retrained, and subsequently bias similar samples submitted to the classifier towards receiving the same target label.

Exploratory attacks are a common source of adverse causative influence in deployed systems [46, 49]. We refer to these as *pollution* attacks, since misclassifications have corrupted the quality of the training data and opened the door to further misclassifications. An example of this attack is seen in a recent PDF classification system [46]. Although the system initial achieved accuracy in excess of 99%, an evaluation involving iterative retraining over a period of weeks revealed a weakness in which a single mislabeled training sample caused the false negative rate to increase to 37%. The single mislabeled training point had large impact because the test data set for that week contained a large number of samples which were identical or highly similar (with respect to the feature space) to the mislabeled sample.

Amplification attacks can occur in which an attacker leverages the users of a system to act as patsies who increase the impact of the attack as the system iterates. An attacker may initially compromise a single email account to seed an attack and send an initial set of spam emails, and then induce users to forward and increasingly legitimize the spam email from the classifier's perspective. The attacker may induce users by either manipulating the user's emotions (e.g., forward this for good luck...) or through social engineering tactics (e.g., chance to win a free iPad...). As the system iterates, user behaviors which would normally identify good content will have the opposite effect and hinder the system from recognizing malicious content.

A more complex attack is the *red herring attack*, in which the adversary provides the system with numerous instances of an attack each containing the same unnecessary feature that the system

comes to associate with the attack. The adversary then uses instances of the attack without that feature in hopes that system will fail to recognize them without the unnecessary feature. Such attacks have been conducted on Polygraph, an automatic signature generator for polymorphic worms [31, 32].

Thus, we see that the need to retrain on new data in a timely fashion to react to population drift results in a new vector of attack for the adversary. Machine learning algorithms for the adversarial setting must trade off the need to be responsive with the need to not over-react to misleading, adversarially crafted training data.

One broad class of defenses is based on constraining model change over time, which can be achieved in several ways [22].

Reject on Negative Impact (RoNI)[30] is one of the defense strategies based on constraining model change. Points that disproportionately move the classifier in the direction of more examples being identified as "malicious" pose a challenge to a learning system. On the one hand, they may represent non-stationarity in the data and be the first sign of changes in the data. However, on the other hand, this point might represent a statistical fluke, or worse, a concerted attack on the learning system of the type described above. The RoNI approach to email spam [30] throws out these points, deeming the risk too high. However, we argue that they are an ideal case for human intervention to distinguish data drift from malicious action.

Another approach is to sanitize the training data before training a classifier. Cretu et al. proposed a data sanitization approach for anomaly sensors that divides the training dataset into multiple small disjoint time consecutive sets and trains multiple models ("micro models") on each set [8]. Then an ensemble of the micro-models is used to reduce the effect of poisoning in the dataset. Alternatively, Rubenstein et al. [39] proposed using robust statistical models that are not affected by poisoning in the training data.

4. FAMILIES AND ISOLATION

While in theory, an adversarial drift could be arbitrarily radical, in practice the ability of an adversary to effect drift is limited by its resources. During a campaign, an adversary typically recycles techniques from previous ones and evolves its campaign slowly over time. For example, a new malware campaign may use a vulnerability known from a previous attack but use a delivery mechanism that changes to avoid detection [15]. Spam emails also tend to evolve from previous ones by, for example, selling the same product but under a different (misspelled) name [23]. Due to the evolutionary nature of campaigns, they can be grouped into families, which organize adversarial drift into distinct trends. In addition to making understanding drift easier, often, the detection strategy of attacks grouped together in the same family are similar.

Furthermore, these families of malicious instances can both make understanding the behavior of classifiers easier and allow humans to guide classifiers. In particular, using multiple classifiers, one for each family, provides benefits to both understanding and accuracy. Such *ensemble classifiers* have proved useful in a diverse set of applications [38], but are particularly well suited to our adversarial setting with families of malicious campaigns.

For furthering the goal of understanding the malicious instances, the operator can examine the outputs of individual classifiers to understand in which families a malicious instance belongs. Examining the classifiers for each family can yield a better understanding of why a given family is or is not being caught. It can also help the human understand the most identifying characteristics of a family.

For furthering our goal of an accurate classification as either malicious instances or not, the operator can combine the decisions of each of the family's classifiers into one overall decision, which has the potential to be more accurate since each family's classifier can be more specialized for finding a type of malicious instances.

Furthermore, by selecting how to group training instances into families, the human can aid the machine learning algorithms by imparting domain knowledge into this separation. The use of families provides an intuitive way of integrating humans into the machine learning process beyond simply selecting features and labeling training data as malicious or not.

Ensemble classification techniques for robust machine learning have been used in the past. Most related is Google's system for detecting malicious advertisements [41], which also uses different classifiers for different families of attacks. Less related is the work of Biggio et al. [3, 2], who use separate classifiers for different features. The works of Kuncheva and Rodriguez are also less related by using two classifiers for different purposes: one for classification and one for detecting drift [37, 21].

Using a separate classifier for each family introduces two choices: how to introduce, train and retire each classifier and how to combine their classifications into an overall classification as to whether the instance is malicious or benign. We consider each in turn before considering an advantage of using families: isolating campaigns from one another.

Combining Classifications.

Looking first at how to combine classifications, consider the simple case of just having a single binary classifier for each family. In this case, the overall classification should be malicious if any of the classifiers is positive for its malicious family, and should be benign otherwise, since belonging to any of the malicious families implies being malicious.

In more complex cases, there could exist a classifier for benign instances, which requires a decision about which classifier takes precedence if an instance is labeled as benign by it and as malicious by some malicious family's classifier. For example, Google uses a classifier for the benign class that trumps the classifiers for malicious cases but it only reports that an instance is benign if it has high confidence, leaving non-obvious cases to the classifiers for the families [41].

In Section 5, we discuss also using a blacklist and a whitelist as classifiers that tramp all others.

Above, we have only considered classifiers that produce a simple positive or negative result. More generally, the theory of ensemble classifiers provides numerous other options for combining classifiers that produce more complex classifications, such as a probability of being in a class [38].

Introduction, Training and Retirement of Classifiers.

The second choice, of how to maintain a given classifier, can determine the influence the operator and the adversary have over the whole ensemble classifier. Leaving the questions of classifier introduction or retirement aside for a moment we would intuitively like to train each classifier in the ensemble with instances of its family. However, identifying these instances can be tricky. Despite research on the automated classification of instances into families (e.g., [13]), deployed systems often do so by hand. For example, Google uses domain experts to separate different kind of adversarial advertisements into families for training specific classifiers [41]. While separating instances by hand limits the number of training examples available, it allows for humans to make subjective decisions, such as where one family ends and another begins, in a manner consistent their understanding and to impart their domain knowledge on the system. Nevertheless, we consider algorithms

for managing the automatic introduction and retirement of classifiers to be an interesting direction for future work.

Once the training data is broken into families, by hand or otherwise, the next decision is which instances to provide to which classifiers. The typical method of training an ensemble of multiple one-class classifiers for classifying multiple classes is the *one-vs-all* method [16]. Under this method, each classifier is trained using all the labeled instances. The instances for the family that the classifier is to recognize are treated as positive and the all other instances are treated as negative.

However, Sculley et al. present a more appropriate method for our setting [41]. Their system, operating for Google, uses a method they call *one-vs-good*. Under this method, the training of the classifier for a family uses only the instances labeled as belonging to that family (treated as positive) and those labeled as good (treated as negative). Instances labeled with some other family are completely held out from the training dataset.

The authors highlight two advantages of one-vs-good over one-vs-all:

1. the adversarial classes overlap making one-vs-all an ill fit, and

2. the non-adversarial class is large.

We wish to highlight a third advantage: this method helps to *isolate* attack families from one another.

Isolation.

Under the one-vs-good method of training, instances belonging to a family will not be used in the training sets of classifiers of any other family. Thus, the instances of each family are isolated from the others with respect to their effect on the individual classifiers. Under the assumption that malware campaigns are most cost-effective using a single family, this suggests that an attacker will only affect a single family's classifier in the ensemble. This suggestion has exceptions: other classifiers can be affected if part of a campaign is labeled as being in a different family by mistake or if the campaign also employs benign but misleading instances as part of a causative attack. Nevertheless, facing a defender motivated to accurately label instances, a campaign will have to either spend more resources to use a multi-family attack or accept a limited impact on the system.

We propose that isolation provides a sweet spot in the space of trade offs between being responsive to drift and limiting causative attacks. In particular, isolation focuses the effects of a maliciously crafted instance to the single family to which it is assigned. Furthermore, isolation eases understanding the drift since changes become local to only the families to which an instance belongs.

We give a practical illustration of how isolation can be beneficial in the canonical example of the dictionary causative attack against the SpamBayes system [30, 51]. The dictionary attack takes place in a content-based only spam detection system and goes as follow:

1. The attacker sends nonsensical messages containing a high proportion of benign, frequently used words (*dictionary* words),

2. The victim correctly classifies these messages as spam and retrains her classifier,

3. The classifier assigns higher spam scores to common words, eventually overwhelming the end user with too many false positives.

Using isolation, the same attack could have a very different story:

1. The initial attacker and victim moves are similar, importantly, the victim classifies the attacker's messages as spam,

2. The system introduces a new classifier for the attacker's messages and trains it without affecting the pre-existing ones.

3. If using cross-validation for example, the performance of the newly introduced classifier turns out to be very low, the user is either prompted to disregard it, or introduce better distinguishing features.

Even though the isolation property does not solve the attack by itself, it provides a beneficial framework for designing counter measures. Clearly, the delicate part is the introduction of the new classifier. Unsupervised or semi-supervised clustering techniques could provide a fertile starting ground here.

5. RESPONSIVENESS

Using families can aid the operator in understanding adversarial drift, which could help detect and avoid causative attacks. However, the system must also respond to changes. In particular, the system must quickly react to newly found attacks and to operator feedback in terms of newly labeled instances that it has previously misclassified.

Retraining the classifiers for each family may or may not actually cause a simple ensemble using one-vs-good to respond to an instance newly labeled as malicious or benign. Typically, the classifiers used for each family would be from a standard machine learning algorithm. For example, Google used linear SVMs among others to classify malicious advertisements [41]. These algorithms are designed to perform well under aggregated measures such as accuracy rates or the area under the receiver operating characteristic curve. In particular, the ability to generalize and do well on the testing data is seen as more important than misclassifying a small number of training samples.

However, problems vary in the acceptability of even low numbers of misclassifications. As previously mentioned by Sommer et al., the targeting of online advertisements is much more accepting of errors than intrusion detection [45]. More recently, work on adversarial advertisement detection presents another case of a high cost for both false positives and false negatives [41].

In this section, we argue in favor of highly responsive systems that closely fit models to training data. Such responsiveness is almost always overlooked by machine learning experts for three reasons. First, the cost structure of many problems is such that either false positives rates or false negatives rates or even both are not particularly critical. Second, in the extreme case, responsiveness requires zero training error, which is commonly equated to overfitting and poor generalization power. Third, training data is often unreliable with mislabeled instances to which the system would be better off not responding. We have already presented the case for high-cost errors in some adversarial settings. We address the remaining two concerns by first arguing that, in some cases, zero training error really is desirable, but that blacklists and whitelists can maintain generalizability. We then consider noisy data and other practical concerns.

Required: Zero High-Impact Errors.

In some adversarial settings, misclassification of certain training instances may be unacceptable if those instances are high impact and likely to reappear in the testing data. For example, the training set for malware detection systems typically include system files necessary for the operation of a computer. A non-zero error rate on

such files would imply disabling users' computers, a completely unacceptable outcome.

A zero training error rate on high-impact instances is so important that the operator expects the system to respond to training on such instances by correctly classifying them with zero error regardless of other training instances. They would gladly trade some generalizability for zero error on such instances. That is, they expect the system to respond to changes in the training data even at expense of generalizability, a traditional metric of machine learning quality.

For example, linear Support Vector Machines are well-regraded for their ability to generalize to unseen testing instances. However, for non-linearly separable instances they will produce a non-zero training error. As a more extreme example, stochastic gradient decent does not even examine training instances in excess of those needed to reach convergence at a provided precision [4]. Thus, the algorithm will not respond at all to any high-impact instances after a certain point. Such classification algorithms could lead to unacceptable high-impact errors.

Furthermore, the human operators should be able to tell how the system is going to react to simple interactions such as adding more high-impact labeled instances or relabeling some instances in the training set. It is not sufficient to think that such instances will be correctly labeled; the operator must understand the system's reaction well enough to *know* that they will be correctly labeled. However, even in such simple cases, the behavior of most standard machine learning algorithms is highly dependent on complex interactions between various instances.

Simple Solution: Blacklists and Whitelists.

Operators are willing to trade off the ability to generalize for the ability to perfectly classify high-impact training instances, which in the extreme case where all training instances are high impact, implies zero training error. However, such a trade off is unnecessary. Equating zero-training error to poor generalization performance is a widespread but false belief. Indeed, it is possible to wrap any machine learning algorithm to create a decision function that is equally general but makes zero errors on the training data set. Here, we measure generality in terms of how the wrapped and original versions perform on instances not in the training set: they will classify such unseen instances identically.

The wrapped version simply uses a blacklist and a whitelist for recording training instances. In the training phase, it trains the original machine learning algorithm as usual but also records each positive training instance in the blacklist and each negative training instance in the whitelist. In the testing phase, it checks whether the submitted instance appears in one of the lists of training instances. If so, then it returns the corresponding label. Otherwise, it returns the prediction provided by the trained machine learning algorithm.

Despite the above construction's simplicity, it illustrates that zero-training error does not imply a lack of generality and over fitting of the data. Indeed, it shows that zero-training error is achievable without any changes to the classifications of unseen test instances.

Noisy Data and Other Practical Concerns.

The above construction must be generalized to become useful in practice. In particular, the operator might not want or be able to provide the system with every high-impact instance. Thus, the blacklists and whitelists must be replaced by more general constructs that can represent many high-impact instances with ease.

It also is crucial to recognize that labeled data, even high-impact data, comes in various qualities, ranging from machine generated to expert human labeling. In published systems, noisy data sets mostly occur when the training labels are machine generated by some other source with unknown quality [19, 48]. Even in systems where the data is carefully curated, labeling errors can still be present regardless of how careful the human experts are [18]. For example, a high-impact, popular website that is labeled as benign and whitelisted could become compromised. Thus, operators should be notified when a classifier would have labeled an instance differently than it did had it not been blacklisted or whitelisted. For example, Whittaker et al. discuss Google's system for classifying phishing pages having the ability to classify websites with a high page rank as malicious but only after a human review [49].

The system must respect the various qualities of data while learning models as well. We might expect more low-quality instances than high-quality ones, simply because expert human labeling is a scarce and hence expensive resource. Indeed, such is the case for Google's system for classifying phishing pages [49]. For learning algorithms, they should weight high-quality ones more importantly than the rest, possibly disregarding inconsistencies in the low-quality data. It is important to ensure that high-quality data provided by operators to introduce new examples of either positive or negative classes does not get washed out with low-quality data. In the unavoidable case of labeling inconsistency in the high-quality data, such as two identical instances in terms of features being assigned two distinct labels, the system should signal the discrepancy to the operators instead of silently processing the paradoxical data.

In some sense, the system must explain the training data set and report for inconsistencies. This is philosophically different from fitting a classification model to the data and adjusting the regularization such that no training error occurs. For example, Google effectively clusters the instances into similar groups (a manual effort backed by some search functionality for retrieving similar samples) [41]. Our view is that a system which acts responsively should be able to help the operators in their data understanding process, by providing the right tools for data exploration, organization and most importantly explanation.

Lastly, the time required for training a system becomes a secondary motivation for using generalized blacklists and whitelists. In practice, responsiveness is not simply measured in the effect a training instance has on the classifier but also in the time it takes for that effect to be pushed out to production servers. A timely response is particularly important in real time monitoring systems, such as Facebook's immune system [47] where spam can rapidly disseminate in the social network, and Google's adversarial advertisement detector [41], which must operate under stringent timing constraints. In both cases, the system design enables operators to write ad hoc decision rules to operate while the classifiers are retrained and tested. The existence of such methods to circumvent machine learning, at least temporarily, is a strong indicator that responsiveness is an important requirement.

6. DATA EXPLORATIONS

6.1 The Data Set

In this section, we describe a malware classification task to demonstrate the adversarial drift in a real world data set. We received a data set of static analysis-based reports of malware and benign x86 executables from an anonymous provider. The data set was sampled from two strata: the *old* stratum and the *new* stratum. The old stratum consists of malware and benign samples submitted to the provider from April 2007 to March 2013. The new stratum consists of samples submitted from April 2013 to July 2013. Table 1 shows the size of the samples from each strata, which are not proportion-

ate to the sizes of the underlying strata. The numbers of malware and benign instances of the new and old samples are not balanced: the sample from the old stratum has more benign instances and the sample from the new stratum has more malware instances. Also we noticed 63.75% of the malwares are from April 2013.

	Old: Apr '07-Mar '13	New: Apr '13- Jul '13
Benign	85549	8803
Malware	40861	82984
Total	**126410**	**91787**

Table 1: Size of the data

For each piece of malware represented in a sample, we received from the provider:

- a timestamp, which is the time that this instance was submitted to the provider;
- a label, which could be benign or a specific family it belongs to if it is malware; and
- a feature vector, which is a sparse 120K dimensional binary vector derived from the control flow graph of the instance using static binary analysis by the provider.

Table 2 shows the presence of various families in the data set. The families were assigned by the provider using a combination of factors including the exploit used, the distribution method, and the command servers to which it connected.

Family	# of instances	Duration
worm:win32/vobfus	14203	10/2008 - 06/2013
trojandownloader:win32/beebone	11125	03/2012 - 06/2013
pws:win32/zbot	5691	01/2008 - 06/2013
adware:win32/hotbar	3913	09/2010 - 07/2013
virus:win32/ramnit	2387	11/2010 - 06/2013
trojan:win32/ramnit	2078	12/2010 - 06/2013
rogue:win32/winwebsec	2022	05/2009 - 06/2013
trojan:win32/killav	1917	11/2007 - 06/2013
trojan:win32/vundo	1601	11/2007 - 06/2013
worm:win32/allaple	1567	05/2007 - 06/2013

Table 2: Top 10 families

The details of how the provider produced our samples has not been shared with us, but it was designed to ensure that machine learning results on our samples carry over to their entire data set and this has been empirically verified. Thus, we believe our samples to be at least coarsely representative of the entire data set. However, since the provider's data set is not a random sample of all malware, we cannot assume that our data set is representative of malware in general; only that it is coarsely representative of the malware reported to the provider. This limitation does not pose a problem for our purposes since we are interested in whether machine learning can aid malware detectors in classifying the malware they encounter. However, our results are limited to detectors similar to the one that provided us with our data set. Another limitation is that since the allocation of samples between strata is not proportionate to the underlying sizes of the strata, statistics computed by mixing observations across strata (e.g., a mean value computed over the whole of both samples) might not be representative of the provider's data set.

6.2 Experiments

To ascertain the presence of drift in this real life data set, we performed two classification tasks. In the first one, we split the data set into two epochs, and evaluated the performance of different prediction problems, taking time ordering into account (predicting each epoch using the other) or not (random cross-validation). In the second one, we fixed a testing data set constituted of the most recent instances, and kept evaluating the performance of models trained using instances of various freshness.

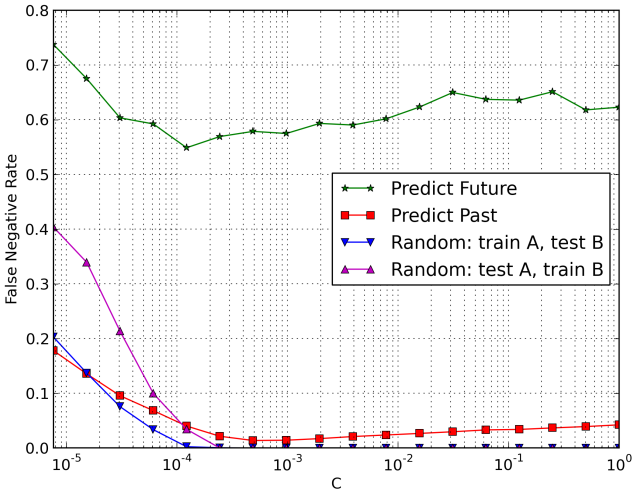

Figure 1: False negative rate at false positive rate of 1% for different regularization factors with different training-testing partition.

For all of the presented results, the learning algorithm we used is an empirical loss minimization approach closely related to Support Vector Machines with a squared hinge loss and an L2 penalty term. Namely, the risk function we are minimizing is:

$$\mathbf{w} \mapsto \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \sum_{(\mathbf{x},y)\in\mathcal{D}} \max(0, 1 - y\mathbf{x}^T\mathbf{w})^2$$

where $\mathbf{w}$ is the weight vector, $\mathcal{D}$ is the data set of labeled instances $(\mathbf{x}, y)$, and C is a regularization term. Conveniently, we used the `liblinear` [12] implementation, which uses a fast conjugate gradient descent scheme on the dual of this convex optimization problem.

On cross-validation.

For the first experiment we split the data set on mid-April, resulting in a equitable re-partition of malware before and after, with about 60 thousand malicious instances in each period. The actual proportions of malware in each epoch, 42% and 90% respectively, were however dissimilar due to the low number of benign instances in the last epoch.

We trained two-class linear SVM models on each epoch separately, using different regularization factors C, exponentially ranging from 2^{-17} to 1 (lower C values meaning more regularization).

Also, to compare with a time-agnostic cross-validation process, we randomly reassigned the instances to the two periods, making sure the original class ratios were exactly preserved within each period.

Figure 1 shows the false negative rate (rate of misidentifying malware instances) at detection thresholds adjusted such that the false positive rate is constant below 1% for all different regularization factors C. The green star curve shows the performance of the system trained on the old data period and tested on the new one, the

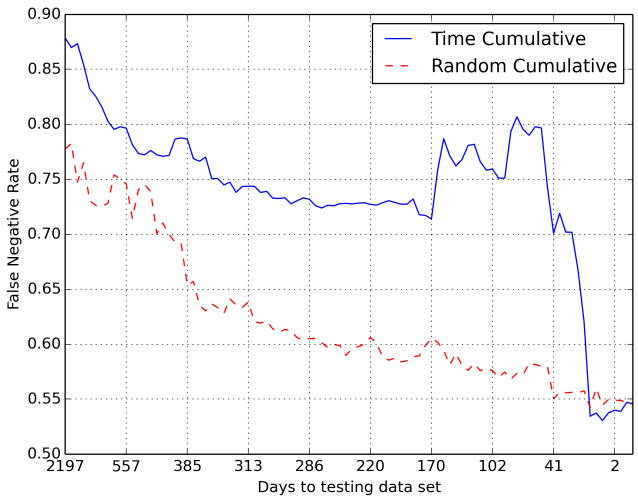

Figure 2: False negative rate at false positive rate of 1% on a fixed future testing set for increasingly large temporal prefixes of historical data.

red square curve the reverse, and the remaining two show the same experiments carried out after a random shuffling of the samples as described above, in order to simulate cross-validation.

We can make several important observations. First, no matter how much we regularize the historical model, the false negative rate stays above 50%, far above the very optimistic 0 false negative rate obtained by random cross validation. Second, time is not reversible. It is easier to predict the labels of the historical data using a model trained on current data than the reverse. Third, the optimal regularization factor is different for each experiment: any high enough C (less regularization) works in the random cross-validation case, but only low Cs (more regularization) are optimal when taking into account the temporal ordering of the instances.

Aside from limiting over fitting on such high dimensional data, another reason why lower values of C appear to work better could be a more evenly distribution of the weights across the dimensions when using the L2 regularization, therefore improving the robustness of the detector, as mentioned in [20, 3].

We conclude that to be of practical interest, the evaluation of a machine learning based security system should take the temporal nature of the instances into account and avoid relying solely on random cross-validation.

On data freshness.

The presence of data set drift is already established by the previous experiment for a particular partitioning of the data. We now turn to another experiment which partially removes the fixed time splitting and shows the importance of freshness of the data for the real prediction task.

In this experiment, we fixed the testing set once and for all to be the same as the future epoch of the previous experiment. We then trained SVM models on increasingly larger portions of the historical data, starting from the oldest samples and eventually including all the previous historical evidence in the training set. For this experiment, we fixed $C = 10^{-4}$ as hinted by Figure 1 with otherwise identical parameters.

For comparison purposes, we also performed the same experiment but disregarded the temporal ordering of the training instances by means of a random permutation (red dashed curve). Similarly to the previous experiment, the detection threshold was adjusted so that the false positive rate on the testing set always remained below 1%.

Figure 2 shows the resulting false negative rate curve as a function of time. We can see that the system's error rate is on average very high and somewhat erratic on old data. We however observe a very sudden significant improvement as instances get closer in time to the testing set instances, pointing to the existence of an important drift. On the contrary, the randomly distributed data set experiment is very similar to actual textbooks error curves as a function of data set size. The dashed red curve shows fast improvement initially and then more steady progress.

We conclude this experimental section by mentioning a previous publication by Singh et al. which studies the importance of data set drift in the context of malware detection [44]. Although, not contradictory with the authors findings of low drift *within* families, our results call for, at least, very cautious extrapolations of such results to detection of malware across families in general.

7. CONCLUSIONS AND OPEN QUESTIONS

Our position is that machine learning algorithms and their results must be understandable to their human operators for each to the aid the other in overcoming adversarial drift. First, we have argued that drift should be organized by viewing malicious instances as belonging to evolving families and classification algorithms should respect and leverage this organization. In particular, we have argued that such algorithms can isolate campaigns from one another to limit the impact of each and lead to more intuitive results.

Second, we have also argued that algorithms must respect expert domain knowledge in an understandable fashion by being able to guarantee zero training error for high-impact instances. Otherwise, the algorithm would be either unresponsive to expert input or responsive in a counterintuitive manner. Despite the common belief that zero training error implies a lack of generalizability, we show black- and whitelists can provide zero training error and generalizability.

Lastly, we have explored a data set from a malware detection provider. Our analyses are consistent with our concerns about adversarial drift. In particular, we find that a learning algorithm's accuracy is highly dependent upon whether it is trained and tested on random samples of all data (as in cross validation) or trained on data with older timestamps than it is tested upon (a more realistic manner).

Our results suggest that the standard measures of classifier performance are insufficient. Drift and temporal order must be respected while testing a classifier's accuracy. More fundamentally, to be useful in practice, human operators must be able understand how the classifier handles drift and how to alter its behavior in light of new data.

Finally, we summarize here the important open questions that this position paper raises:

Classifier lifecycle management.

In an ensemble classification system where each classifier is responsible for detecting one particular type of behavior (e.g., one type of malware family) how should classifier/family introduction, merging, splitting and retirement be implemented? Aside from manual identification of new families, are there reliable semi-automated or even automated techniques for doing so?

Responsiveness beyond black- and whitelists.

How do we design machine learning based systems that consistently and safely react to label changes concerning one or two instances? In particular, how do we guarantee zero training error efficiently, without the use of potentially large black and white lists?

Consistency.

Related to the previous point, how do we build learning systems that spot labeling inconsistencies in the training data and help the operators drive the labeling error rate towards zero?

Defenses on causative attacks with isolation.

How do we defend against causative attacks after containing their effects to a new classifier? Can we automatically prompt the operators for more distinguishing features?

Acknowledgements

We thank our anonymous data provider for the data set of malware and benign instances. We also thank our anonymous reviewers for their helpful comments on the paper. We are grateful to the Intel Science and Technology Center for Secure Computing, DARPA (grant N10AP20014), the National Science Foundation (through the TRUST Science and Technology Center), and the US Department of State (DRL) for supporting this work in part. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of any of the funding sponsors.

8. REFERENCES

[1] U. Bayer, P. M. Comparetti, C. H. C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *NDSS*, 2009.

[2] B. Biggio, I. Corona, and G. Fumera. Bagging classifiers for fighting poisoning attacks in adversarial classification tasks. In *Multiple Classifier Systems*, pages 350–359. Springer Berlin Heidelberg, 2011.

[3] B. Biggio, G. Fumera, and F. Roli. Evade hard multiple classifier systems. In *Applications of Supervised and Unsupervised Ensemble Methods*, pages 15–38. Springer Berlin Heidelberg, 2009.

[4] L. Bottou and O. Bousquet. The Tradeoffs of Large-Scale Learning. *Advances in Neural Information Processing Systems*, 20:161–168, 2008.

[5] M. Brückner, C. Kanzow, and T. Scheffer. Static prediction games for adversarial learning problems. *Journal of Machine Learning Research*, 13:2617–2654, 2012.

[6] M. Brückner and T. Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 547–555, 2011.

[7] V. Castelli and T. M. Cover. On the exponential value of labeled samples. *Pattern Recognition Letters*, 16, 1995.

[8] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 81–95. IEEE, 2008.

[9] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Fast and precise in-browser JavaScript malware detection. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 3–3, Berkeley, CA, USA, 2011. USENIX Association.

[10] N. Dalvi, P. Domingos, S. Sanghai, and D. Verma. Adversarial classification. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining KDD 04 (2004)*, page 99, New York, New York, USA, 2004. ACM Press.

[11] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 332–346, Washington, DC, USA, 2012. IEEE Computer Society.

[12] R. Fan, K. Chang, C. Hsieh, X. Wang, and Lin. LIBLINEAR : A Library for Large Linear Classification. *The Journal of Machine Learning Research*, 9(2008):1871–1874, 2008.

[13] J. Gennari and D. French. Defining malware families based on analyst insights. In *Technologies for Homeland Security (HST), 2011 IEEE International Conference on*, pages 396–401, 2011.

[14] P. Graham. A plan for spam. `http://www.paulgraham.com/spam.html`, Aug. 2002.

[15] A. Gupta, P. Kuppili, A. Akella, and P. Barford. An empirical study of malware evolution. In *First International Communication Systems and Networks and Workshops (COMSNETS 2009)*, pages 1–10, 2009.

[16] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415–425, 2002.

[17] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 43–58, New York, NY, USA, 2011. ACM.

[18] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on Amazon Mechanical Turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[19] A. Kantchelian, J. Ma, L. Huang, S. Afroz, A. D. Joseph, and J. D. Tygar. Robust detection of comment spam using entropy rate. In *Proceedings of the 5th ACM Workshop on Artificial Intelligence and Security*, AISEC 2012. ACM, 2012.

[20] A. Kołcz and C. H. Teo. Feature weighting for improved classifier robustness. In *CEAS'09: Sixth conference on email and Anti-Spam*, number 1, 2009.

[21] L. I. Kuncheva. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In O. Okun and G. Valentini, editors, *Workshop on Supervised and Unsupervised Ensemble Methods and their Applications (SUEMA)*, 2008.

[22] A. Lavoie, M. Otey, N. Ratliff, and D. Sculley. History Dependent Domain Adaptation. In *Domain Adaptation Workshop at NIPS '11*, 2011.

[23] H. Lee and A. Ng. Spam deobfuscation using a hidden markov model. In *Proceedings of the Second Conference on Email and Anti-Spam*, 2005.

[24] Z. Li, K. Zhang, Y. Xie, F. Yu, and X. Wang. Knowing your enemy: Understanding and detecting malicious web advertising. In *CCS*, 2012.

[25] W. Liu and S. Chawla. Mining adversarial patterns via regularized loss minimization. *Machine Learning*, 81(1):69–83, July 2010.

[26] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Second Conference on Email and Anti-Spam (CEAS)*, Palo Alto, CA, 2005.

[27] L. Lu, R. Perdisci, and W. Lee. Surf: Detecting and measuring search poisoning. In *CCS*, 2011.

[28] T. A. Meyer and B. Whateley. SpamBayes: Effective open-source, Bayesian based, email classification system. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, July 2004.

[29] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[30] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. In *Proceedings of the* 1^{st} *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

[31] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Security and Privacy, 2005 IEEE Symposium on*, pages 226–241. IEEE, 2005.

[32] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In *Recent Advances in Intrusion Detection*, pages 81–105. Springer, 2006.

[33] A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2001.

[34] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of the* 14^{th} *ACM conference on Computer and communications security (CCS)*, pages 342–351, New York, NY, USA, 2007. ACM.

[35] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov. Learning and classification of malware behavior. In *DIMVA*, 2008.

[36] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4), 2011.

[37] J. J. Rodríguez and L. I. Kuncheva. Combining online classification approaches for changing environments. In *Proc. of the Joint IAPR International Workshops on Structural and Syntactic Pattern Recognition and Statistical Techniques in Pattern Recognition*, pages 520–529, 2008.

[38] L. Rokach. Ensemble-based classifiers. *Artif. Intell. Rev.*, 33(1-2):1–39, Feb. 2010.

[39] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 1–14. ACM, 2009.

[40] G. Schwenk, A. Bikadorov, T. Krueger, and K. Rieck. Autonomous learning for detection of javascript attacks: Vision or reality? In *AISEC*, 2012.

[41] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou. Detecting adversarial advertisements in the wild. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 274–282. ACM, 2011.

[42] D. Sculley, G. M. Wachman, and C. E. Brodley. Spam Filtering using Inexact String Matching in Explicit Feature Space with On-Line Linear Classifiers. In *The Fifteenth Text REtrieval Conference (TREC 2006) Proceedings*, 2006.

[43] R. Segal, J. Crawford, J. Kephart, and B. Leiba. SpamGuru: An enterprise anti-spam filtering system. In *Conference on Email and Anti-Spam (CEAS)*, 2004.

[44] A. Singh, A. Walenstein, and A. Lakhotia. Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 81–92. ACM, 2012.

[45] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE, 2010.

[46] N. Srndic and P. Laskov. Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA*. The Internet Society, 2013.

[47] T. Stein, E. Chen, and K. Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11, pages 8:1—-8:8, New York, NY, USA, 2011. ACM.

[48] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time URL spam filtering service. In *2011 IEEE Symposium on Security and Privacy (SP)*, pages 447–462. IEEE, 2011.

[49] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *Proc. of 17th NDSS*, 2010.

[50] M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the* 18^{th} *Annual Computer Security Applications Conference (ACSAC)*, pages 61–68, Washington DC, USA, 2002. IEEE Computer Society.

[51] G. Wittel and S. Wu. On attacking statistical spam filters. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004.

[52] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *NDSS*. The Internet Society, 2009.