Security Evaluation of Support Vector Machines in Adversarial Environments

Battista Biggio, Igino Corona, Blaine Nelson, Benjamin I. P. Rubinstein, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli

Abstract Support Vector Machines (SVMs) are among the most popular classification techniques adopted in security applications like malware detection, intrusion detection, and spam filtering. However, if SVMs are to be incorporated in real-world security systems, they must be able to cope with attack patterns that can either mislead the learning algorithm (poisoning), evade detection (evasion), or gain information about their internal parameters (privacy breaches). The main contributions of this chapter are twofold. First, we introduce a formal general framework for the empirical evaluation of the security of machine-learning systems. Second, according to our framework, we demonstrate the feasibility of evasion, poisoning and privacy attacks against SVMs in real-world security problems. For each attack technique, we evaluate its impact and discuss whether (and how) it can be countered through an adversary-aware design of SVMs. Our experiments are easily reproducible thanks to *open-source* code that we have made available, together with all the employed datasets, on a public repository.

Battista Biggio, Igino Corona, Davide Maiorca, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli
Department of Electrical and Electronic Engineering, University of Cagliari,
Piazza d'Armi 09123, Cagliari, Italy.
e-mail: {battista.biggio,igino.corona,davide.maiorca}@diee.unica.it
e-mail: {fumera,giacinto,roli}@diee.unica.it
Blaine Nelson
Institut für Informatik, Universität Potsdam, August-Bebel-Straße 89, 14482 Potsdam, Germany.
e-mail: blaine.nelson@gmail.com
Benjamin I. P. Rubinstein
IBM Research, Lvl 5 / 204 Lygon Street, Carlton, VIC 3053, Australia.
e-mail: ben@bipr.net

1 Introduction

Machine-learning and pattern-recognition techniques are increasingly being adopted in security applications like spam filtering, network intrusion detection, and malware detection due to their ability to generalize, and to potentially detect novel attacks or variants of known ones. Support Vector Machines (SVMs) are among the most successful techniques that have been applied for this purpose [28, 54].

However, learning algorithms like SVMs assume *stationarity*: that is, both the data used to train the classifier and the operational data it classifies are sampled from the same (though possibly unknown) distribution. Meanwhile, in adversarial settings such as the above mentioned ones, intelligent and adaptive adversaries may purposely manipulate data (violating *stationarity*) to exploit existing vulnerabilities of learning algorithms, and to impair the entire system. This raises several open issues, related to whether machine-learning techniques can be safely adopted in security-sensitive tasks, or if they must (and can) be re-designed for this purpose. In particular, the main open issues to be addressed include:

1. analyzing the vulnerabilities of learning algorithms;
2. evaluating their security by implementing the corresponding attacks; and
3. eventually, designing suitable countermeasures.

These issues are currently addressed in the emerging research area of *adversarial machine learning*, at the intersection between computer security and machine learning. This field is receiving growing interest from the research community, as witnessed by an increasing number of recent events: the NIPS Workshop on "Machine Learning in Adversarial Environments for Computer Security" (2007) [43]; the subsequent Special Issue of the Machine Learning journal titled "Machine Learning in Adversarial Environments" (2010) [44]; the 2010 UCLA IPAM workshop on "Statistical and Learning-Theoretic Challenges in Data Privacy"; the ECML-PKDD Workshop on "Privacy and Security issues in Data Mining and Machine Learning" (2010) [27]; five consecutive CCS Workshops on "Artificial Intelligence and Security" (2008-2012) [2, 3, 34, 22, 19], and the Dagstuhl Perspectives Workshop on "Machine Learning for Computer Security" (2012) [37].

In Section 2, we review the literature of adversarial machine learning, focusing mainly on the issue of *security evaluation*. We discuss both theoretical work and applications, including examples of how learning can be attacked in practical scenarios, either during its training phase (*i.e.*, *poisoning* attacks that contaminate the learner's training data to mislead it) or during its deployment phase (*i.e.*, *evasion* attacks that circumvent the learned classifier).

In Section 3, we summarize our recently defined framework for the empirical evaluation of classifiers' security [12]. It is based on a *general* model of an adversary that builds on previous models and guidelines proposed in the literature of adversarial machine learning. We expound on the assumptions of the adversary's goal, knowledge and capabilities that comprise this model, which also easily accommodate application-specific constraints. Having detailed the assumptions of his adversary, a security analyst can formalize the adversary's strategy as an optimization problem.

We then demonstrate our framework by applying it to assess the security of SVMs. We discuss our recently devised evasion attacks against SVMs [8] in Section 4, and review and extend our recent work [14] on poisoning attacks against SVMs in Section 5. We show that the optimization problems corresponding to the above attack strategies can be solved through simple gradient-descent algorithms. The experimental results for these evasion and poisoning attacks show that the SVM is vulnerable to these threats for both linear and non-linear kernels in several realistic application domains including handwritten digit classification and malware detection for PDF files. We further explore the threat of privacy-breaching attacks aimed at the SVM's training data in Section 6 where we apply our framework to precisely describe the setting and threat model.

Our analysis provides useful insights into the potential security threats from the usage of learning algorithms (and, particularly, of SVMs) in real-world applications, and sheds light on whether they can be safely adopted for security-sensitive tasks. The presented analysis allows a system designer to *quantify* the security risk entailed by an SVM-based detector so that he may weigh it against the benefits provided by the learning. It further suggests guidelines and countermeasures that may mitigate threats and thereby improve overall system security. These aspects are discussed for evasion and poisoning attacks in Sections 4 and 5. In Section 6 we focus on developing countermeasures for privacy attacks that are endowed with strong theoretical guarantees within the framework of *differential privacy*. We conclude with a summary and discussion in Section 7.

In order to support the reproducibility of our experiments, we published all the code and the data employed for the experimental evaluations described in this paper [24]. In particular, our code is released under open-source license, and carefully documented, with the aim of allowing other researchers to not only reproduce, but also customize, extend and improve our work.

2 Background

In this section, we review the main concepts used throughout this chapter. We first introduce our notation and summarize the SVM learning problem. We then motivate the need for the proper assessment of the security of a learning algorithm so that it can be applied to security-sensitive tasks.

Learning can be generally stated as a process by which data is used to form a hypothesis that performs better than an *a priori* hypothesis formed without the data. For our purposes, the hypotheses will be represented as functions of the form $f : \mathscr{X} \to \mathscr{Y}$, which assign an input sample point $\mathbf{x} \in \mathscr{X}$ to a class $y \in \mathscr{Y}$; that is, given an observation from the input space $\mathscr{X}$, a hypothesis f makes a prediction in the output space $\mathscr{Y}$. For *binary classification*, the output space is binary and we use $\mathscr{Y} = \{-1, +1\}$. In the classical *supervised learning* setting, we are given a paired training dataset $\{(\mathbf{x}_i, y_i) \mid \mathbf{x}_i \in \mathscr{X}, y_i \in \mathscr{Y}\}_{i=1}^{n}$, we assume each pair is drawn independently from an unknown joint distribution $P(\mathbf{X}, Y)$, and we want to infer a

classifier f able to *generalize* well on $P(\mathbf{X},Y)$; *i.e.*, to accurately predict the label y of an unseen sample $\mathbf{x}$ drawn from that distribution.

2.1 Support Vector Machines

In its simplest formulation, an SVM learns a linear classifier for a binary classification problem. Its decision function is thus $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$, where $\text{sign}(a) = +1$ (-1) if $a \geq 0$ $(a < 0)$, and $\mathbf{w}$ and b are learned parameters that specify the position of the decision hyperplane in feature space: the hyperplane's normal $\mathbf{w}$ gives its orientation and b is its displacement. The learning task is thus to find a hyperplane that well-separates the two classes. While many hyperplanes may suffice for this task, the SVM hyperplane both separates the training samples of the two classes and provides a maximum distance from itself to the nearest training point (this distance is called the classifier's *margin*), since maximum-margin learning generally reduces *generalization error* [65]. Although originally designed for linearly-separable classification tasks (*hard-margin* SVMs), SVMs were extended to non-linearly-separable classification problems by Vapnik [25] (*soft-margin* SVMs), which allow some samples to violate the margin. In particular, a soft-margin SVM is learned by solving the following convex quadratic program (QP):

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\sum_{i=1}^{n} \xi_i \\ \text{s. t.} \quad & \forall\, i = 1,\ldots,n \quad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \ , \end{aligned}$$

where the margin is maximized by minimizing $\frac{1}{2}\mathbf{w}^\top \mathbf{w}$, and the variables ξ_i (referred to as *slack variables*) represent the extent to which the samples, $\mathbf{x}_i$, violate the margin. The parameter C tunes the trade-off between minimizing the sum of the slack violation errors and maximizing the margin.

While the primal can be optimized directly, it is often solved via its (Lagrangian) dual problem written in terms of Lagrange multipliers, α_i, which are constrained so that $\sum_{i=1}^{n} \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$ for $i = 1,\ldots,n$. Solving the dual has a computational complexity that grows according to the size of the training data as opposed to the feature space's dimensionality. Further, in the dual formulation, both the data and the slack variables become implicitly represented—the data is represented by a *kernel matrix*, $\mathbf{K}$, of all inner products between pairs of data points (that is, $K_{i,j} = \mathbf{x}_i^\top \mathbf{x}_j$) and each slack variable is associated with a Lagrangian multiplier via the KKT conditions that arise from duality. Using the method of Lagrangian multipliers, the *dual problem* is derived, in matrix form, as

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^\top \mathbf{Q}\alpha - \mathbf{1}_n^\top \alpha \\ \text{s. t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \forall\, i = 1,\ldots,n \quad 0 \leq \alpha_i \leq C \ , \end{aligned}$$

where $\mathbf{Q} = \mathbf{K} \circ yy^\top$ (the Hadamard product of $\mathbf{K}$ and $yy^\top$) and $\mathbf{1}_n$ is a vector of n ones.

Through the kernel matrix, SVMs can be extended to more complex feature spaces (where a linear classifier may perform better) via a *kernel function*—an implicit inner product from the alternative feature space. That is, if some function $\phi : \mathscr{X} \to \Phi$ maps training samples into a higher-dimensional feature space, then K_{ij} is computed via the space's corresponding kernel function, $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$. Thus, one need not explicitly know ϕ, only its corresponding *kernel function*.

Further, the dual problem and its KKT conditions elicit interesting properties of the SVM. First, the optimal primal hyperplane's normal vector, $\mathbf{w}$, is a linear combination of the training samples;[1] *i.e.*, $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$. Second, the dual solution is *sparse*, and only samples that lie on or within the hyperplane's margin have a non-zero α-value. Thus, if $\alpha_i = 0$, the corresponding sample $\mathbf{x}_i$ is correctly classified, lies beyond the margin (*i.e.*, $y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 1$) and is called a *non-support vector*. If $\alpha_i = C$, the i^{th} sample violates the margin (*i.e.*, $y_i(\mathbf{w}^\top \mathbf{x}_i + b) < 1$) and is an *error vector*. Finally, if $0 < \alpha_i < C$, the i^{th} sample lies exactly on the margin (*i.e.*, $y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$) and is a *support vector*. As a consequence, the optimal displacement b can be determined by averaging $y_i - \mathbf{w}^\top \mathbf{x}_i$ over the support vectors.

2.2 Machine Learning for Computer Security: Motivation, Trends, and Arms Races

In this section, we motivate the recent adoption of machine-learning techniques in computer security and discuss the novel issues this trend raises. In the last decade, security systems increased in complexity to counter the growing sophistication and variability of attacks; a result of a long-lasting and continuing arms race in security-related applications such as malware detection, intrusion detection and spam filtering. The main characteristics of this struggle and the typical approaches pursued in security to face it are discussed in Section 2.3.1. We now discuss some examples that better explain this trend and motivate the use of modern machine-learning techniques for security applications.

In the early years, the attack surface (*i.e.*, the vulnerable points of a system) of most systems was relatively small and most attacks were simple. In this era, signature-based detection systems (*e.g.*, rule-based systems based on string-matching techniques) were considered sufficient to provide an acceptable level of security. However, as the complexity and exposure of sensitive systems increased in the Internet Age, more targets emerged and the incentive for attacking them became increasingly attractive, thus providing a means and motivation for developing sophisticated and diverse attacks. Since signature-based detection systems can only detect attacks matching an existing *signature*, attackers used minor variations of

[1] This is an instance of the Representer Theorem which states that solutions to a large class of regularized ERM problems lie in the span of the training data [60].

their attacks to evade detection (*e.g.*, string-matching techniques can be evaded by slightly changing the attack code). To cope with the increasing variability of attack samples and to detect never-before-seen attacks, machine-learning approaches have been increasingly incorporated into these detection systems to complement traditional signature-based detection. These two approaches can be combined to make accurate and agile detection: signature-based detection offers fast and lightweight filtering of most known attacks, while machine-learning approaches can process the remaining (unfiltered) samples and identify new (or less well-known) attacks.

The quest of image spam. A recent example of the above arms race is *image spam* (see, *e.g.*, [10]). In 2006, to evade the textual-based spam filters, spammers began rendering their messages into images included as attachments, thus producing "image-based spam," or *image spam* for short. Due to the massive volume of image spam sent in 2006 and 2007, researchers and spam-filter designers proposed several different countermeasures. Initially, suspect images were analyzed by OCR tools to extract text for standard spam detection, and then signatures were generated to block the (known) spam images. However, spammers immediately reacted by *randomly* obfuscating images with *adversarial* noise, both to make OCR-based detection ineffective, and to evade signature-based detection. The research community responded with (fast) approaches mainly based on machine-learning techniques using visual features extracted from images, which could accurately discriminate between spam images and legitimate ones (*e.g.*, photographs, plots, *etc.*). Although image spam volumes have since declined, the exact cause for this decrease is debatable—these countermeasures may have played a role, but the image spam were also more costly to the spammer as they required more time to generate and more bandwidth to deliver, thus limiting the spammers' ability to send a high volume of messages. Nevertheless, had this arms race continued, spammers could have attempted to evade the countermeasures by *mimicking* the feature values exhibited by legitimate images, which would have, in fact, forced spammers to increase the number of colors and elements in their spam images thus further increasing the size of such files, and the cost of sending them.

Misuse and anomaly detection in computer networks. Another example of the above arms race can be found in network intrusion detection, where *misuse detection* has been gradually augmented by *anomaly detection*. The former approach relies on detecting attacks on the basis of signatures extracted from (known) intrusive network traffic, while the latter is based upon a statistical model of the *normal profile* of the network traffic and detects *anomalous* traffic that deviates from the assumed model of normality. This model is often constructed using machine-learning techniques, such as one-class classifiers (*e.g.*, one-class SVMs), or, more generally, using density estimators. The underlying assumption of anomaly-detection-based intrusion detection, though, is that *all anomalous* network traffic is, in fact, intrusive. Although intrusive traffic often does exhibit anomalous behavior, the opposite is not necessarily true: some non-intrusive network traffic may also behave anomalously. Thus, accurate anomaly detectors often suffer from high false-alarm rates.

2.3 Adversarial Machine Learning

As witnessed by the above examples, the introduction of machine-learning techniques in security-sensitive tasks has many beneficial aspects, and it has been somewhat necessitated by the increased sophistication and variability of recent attacks and zero-day exploits. However, there is good reason to believe that machine-learning techniques themselves will be subject to carefully designed attacks in the near future, as a logical next step in the above-sketched arms race. Since machine-learning techniques were not originally designed to withstand manipulations made by intelligent and adaptive adversaries, it would be reckless to naively trust these learners in a secure system. Instead, one needs to carefully consider whether these techniques can introduce novel vulnerabilities that may degrade the overall system's security, or whether they can be safely adopted. In other words, we need to address the question raised by Barreno *et al.* [5]: *can machine learning be secure?*

At the center of this question is the effect an adversary can have on a learner by violating the *stationarity assumption* that the *training data* used to train the classifier comes from the same distribution as the *test data* that will be classified by the learned classifier. This is a conventional and natural assumption underlying much of machine learning and is the basis for performance-evaluation-based techniques like cross-validation and bootstrapping as well as for principles like empirical risk minimization (ERM). However, in security-sensitive settings, the adversary may purposely manipulate data to mislead learning. Accordingly, the data distribution is subject to *change*, thereby potentially violating non-stationarity, albeit, in a limited way subject to the adversary's assumed capabilities (as we discuss in Section 3.1.3). Further, as in most security tasks, predicting how the data distribution will change is difficult, if not impossible [12, 36]. Hence, adversarial learning problems are often addressed as a *proactive* arms race [12], in which the classifier designer tries to anticipate the next adversary's move, by simulating and hypothesizing proper attack scenarios, as discussed in the next section.

2.3.1 Reactive and Proactive Arms Races

As mentioned in the previous sections, and highlighted by the examples in Section 2.2, security problems are often cast as a long-lasting *reactive* arms race between the classifier designer and the adversary, in which each player attempts to achieve his/her goal by reacting to the changing behavior of his/her opponent. For instance, the adversary typically crafts samples to evade detection (*e.g.*, a spammer's goal is often to create spam emails that will not be detected), while the classifier designer seeks to develop a system that accurately detects most malicious samples while maintaining a very low false-alarm rate; *i.e.*, by not falsely identifying legitimate examples. Under this setting, the arms race can be modeled as the following cycle [12]. First, the adversary analyzes the existing learning algorithm and manipulates her data to evade detection (or more generally, to make the learning algorithm ineffective). For instance, a spammer may gather some knowledge of the words used

by the targeted spam filter to block spam and then manipulate the textual content of her spam emails accordingly; *e.g.*, words like "cheap" that are indicative of spam can be misspelled as "che4p". Second, the classifier designer reacts by analyzing the novel attack samples and updating his classifier. This is typically done by retraining the classifier on the newly collected samples, and/or by adding features that can better detect the novel attacks. In the previous spam example, this amounts to retraining the filter on the newly collected spam and, thus, to adding novel words into the filter's dictionary (*e.g.*, "che4p" may be now learned as a spammy word). This *reactive* arms race continues in perpetuity as illustrated in Figure 1.

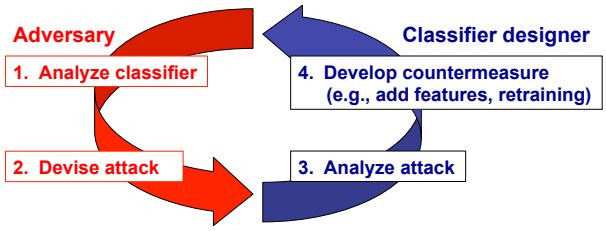

Fig. 1 A conceptual representation of the *reactive* arms race [12].

However, *reactive* approaches to this arms race do not anticipate the next generation of security vulnerabilities and thus, the system potentially remains vulnerable to new attacks. Instead, computer security guidelines traditionally advocate a *proactive* approach[2]—the classifier designer should *proactively anticipate* the adversary's strategy by (i) identifying the most relevant threats, (ii) designing proper countermeasures into his classifier, and (iii) repeating this process for his new design *before* deploying the classifier. This can be accomplished by modeling the adversary (based on knowledge of the adversary's goals and capabilities) and using this model to simulate attacks, as is depicted in Figure 2 to contrast the reactive arms race. While such an approach does not account for unknown or changing aspects of the adversary, it can indeed lead to an improved level of security by delaying each step of the *reactive* arms race because it should reasonably force the adversary to exert greater effort (in terms of time, skills, and resources) to find new vulnerabilities. Accordingly, proactively designed classifiers should remain useful for a longer time, with less frequent supervision or human intervention and with less severe vulnerabilities.

Although this approach has been implicitly followed in most of the previous work (see Section 2.3.2), it has only recently been formalized within a more general framework for the empirical evaluation of a classifier's security [12], which we summarize in Section 3. Finally, although security evaluation may suggest specific countermeasures, designing general-purpose *secure* classifiers remains an open problem.

[2] Although in certain abstract models we have shown how regret-minimizing online learning can be used to define reactive approaches that are competitive with proactive security [6].

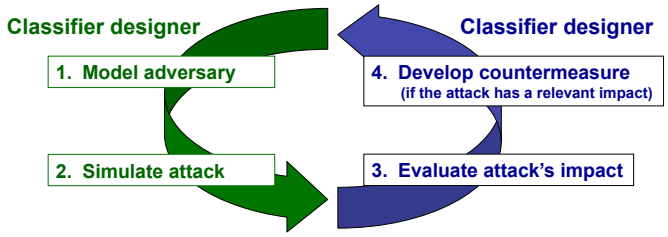

Fig. 2 A conceptual representation of the *proactive* arms race [12].

2.3.2 Previous Work on Security Evaluation

Previous work in adversarial learning can be categorized according to the two main steps of the proactive arms race described in the previous section. The first research direction focuses on identifying potential vulnerabilities of learning algorithms and assessing the impact of the corresponding attacks on the targeted classifier; *e.g.*, [4, 5, 18, 36, 40, 41, 42, 46]. The second explores the development of proper countermeasures and learning algorithms robust to known attacks; *e.g.*, [26, 41, 57].

Although some prior work does address aspects of the *empirical* evaluation of classifier security, which is often implicitly defined as the performance degradation incurred under a (simulated) attack, to our knowledge a systematic treatment of this process under a unifying perspective was only first described in our recent work [12]. Previously, security evaluation is generally conducted within a specific application domain such as spam filtering and network intrusion detection (*e.g.*, in [26, 31, 41, 47, 66]), in which a different application-dependent criteria is separately defined for each endeavor. Security evaluation is then implicitly undertaken by defining an attack and assessing its impact on the given classifier. For instance, in [31], the authors showed how *camouflage* network packets can mimic legitimate traffic to evade detection; and, similarly, in [26, 41, 47, 66], the content of spam emails was manipulated for evasion. Although such analyses provide indispensable insights into specific problems, their results are difficult to generalize to other domains and provide little guidance for evaluating classifier security in a different application. Thus, in a new application domain, security evaluation often must begin anew and it is difficult to directly compare with prior studies. This shortcoming highlights the need for a more general set of security guidelines and a more systematic definition of classifier security evaluation, that we began to address in [12].

Apart from application-specific work, several theoretical models of adversarial learning have been proposed [4, 17, 26, 36, 40, 42, 46, 53]. These models frame the secure learning problem and provide a foundation for a proper security evaluation scheme. In particular, we build upon elements of the models of [4, 5, 36, 38, 40, 42], which were used in defining our framework for security evaluation [12]. Below we summarize these foundations.

2.3.3 A Taxonomy of Potential Attacks against Machine Learning Algorithms

A taxonomy of potential attacks against pattern classifiers was proposed in [4, 5, 36] as a baseline to characterize attacks on learners. The taxonomy is based on three main features: the kind of *influence* of attacks on the classifier, the kind of *security violation* they cause, and the *specificity* of an attack. The attack's influence can be either **causative**, if it aims to undermine learning, or **exploratory**, if it targets the classification phase. Accordingly, a causative attack may manipulate both training and testing data, whereas an exploratory attack only affects testing data. Examples of causative attacks include work in [14, 38, 40, 52, 58], while exploratory attacks can be found in [26, 31, 41, 47, 66]. The security violation can be either an **integrity** violation, if it aims to gain unauthorized access to the system (*i.e.*, to have malicious samples be misclassified as legitimate); an **availability** violation, if the goal is to generate a high number of errors (both false-negatives and false-positives) such that normal system operation is compromised (*e.g.*, legitimate users are denied access to their resources); or a **privacy** violation, if it allows the adversary to obtain confidential information from the classifier (*e.g.*, in biometric recognition, this may amount to recovering a protected biometric template of a system's client). Finally, the attack specificity refers to the samples that are affected by the attack. It ranges continuously from **targeted** attacks (*e.g.*, if the goal of the attack is to have a specific spam email misclassified as legitimate) to **indiscriminate** attacks (*e.g.*, if the goal is to have *any* spam email misclassified as legitimate).

Each portion of the taxonomy specifies a different type of attack as laid out in Barreno *et al.* [4] and here we outline these with respect to a PDF malware detector. An example of a **causative integrity** attack is an attacker who wants to mislead the malware detector to falsely classify malicious PDFs as benign. The attacker could accomplish this goal by introducing benign PDFs with malicious features into the training set and the attack would be **targeted** if the features corresponded to a particular malware or otherwise an **indiscriminate** attack. Similarly, the attacker could cause a **causative availability** attack by injecting malware training examples that exhibited features common to benign messages; again, these would be **targeted** if the attacker wanted a particular set of benign PDFs to be misclassified. A **causative privacy** attack, however, would require both manipulation of the training and information obtained from the learned classifier. The attacker could inject malicious PDFs with features identifying a particular author and then subsequently test if other PDFs with those features were labeled as malicious; this observed behavior may leak private information about the authors of other PDFs in the training set.

In contrast to the causative attacks, **exploratory** attacks cannot manipulate the learner, but can still exploit the learning mechanism. An example of an **exploratory integrity** attack involves an attacker who crafts a malicious PDF for an existing malware detector. This attacker queries the detector with candidate PDFs to discover which attributes the detector uses to identify malware, thus, allowing her to re-design her PDF to avoid the detector. This example could be **targeted** to a single PDF exploit or **indiscriminate** if a set of possible exploits are considered. An **exploratory privacy** attack against the malware detector can be conducted in the

same way as the **causative privacy** attack described above, but without first injecting PDFs into the training data. Simply by probing the malware detector with crafted PDFs, the attacker may divulge secrets from the detector. Finally, **exploratory availability** attacks are possible in some applications but are not currently considered to be of interest.

3 A Framework for Security Evaluation

In Sections 2.3 and 2.3.1, we motivated the need for simulating a proactive arms race as a means for improving system security. We further argued that evaluating a classifier's security properties through simulations of different, potential attack scenarios is a crucial step in this arms race for identifying the most relevant vulnerabilities and for suggesting how to potentially counter them. Here, we summarize our recent work [12] that proposes a new framework for designing proactive secure classifiers by addressing the shortcomings of the reactive security cycle raised above. Namely, our approach allows one to empirically evaluate a classifier's security during its design phase by addressing the first three steps of the proactive arms race depicted in Figure 2: (i) identifying potential attack scenarios, (ii) devising the corresponding attacks, and (iii) systematically evaluating their impact. Although it may also suggest countermeasures to the hypothesized attacks, the final step of the proactive arms race remains unspecified as a unique design step that has to be addressed separately in an application-specific manner.

Under our proposed security evaluation process, the analyst must clearly scrutinize the classifier by considering different attack scenarios to investigate a set of distinct potential vulnerabilities. This amounts to performing a more systematic *what-if analysis* of classifier security [56]. This is an essential step in the design of security systems, as it not only allows the designer to identify the most important and relevant threats, but also it forces him/her to consciously decide whether the classifier can be reasonably deployed, after being made aware of the corresponding risks, or whether it is instead better to adopt additional countermeasure to mitigate the attack's impact *before* deploying the classifier.

Our proposed framework builds on previous work and attempts to systematize and unify their views under a more coherent perspective. The framework defines how an analyst can conduct a security audit of a classifier, which we detail in the remainder of this section. First, in Section 3.1, we explain how an adversary model is constructed according to the adversary's anticipated goals, knowledge and capabilities. Based on this model, a simulation of the adversary can be conducted to find the corresponding *optimal* attack strategies and produce simulated attacks, as described in Section 3.1.4. These simulated attack samples are then used to evaluate the classifier by either adding them to the training or test data, in accordance with the adversary's capabilities from Section 3.1.3. We conclude this section by discussing how to exploit our framework in specific application domains in Section 3.2.

3.1 Modeling the Adversary

The proposed model of the adversary is based on specific assumptions about her goal, knowledge of the system, and capability to modify the underlying data distribution by manipulating individual samples. It allows the classifier designer to model the attacks identified in the attack taxonomy described as in Section 2.3.3 [4, 5, 36]. However, in our framework, one can also incorporate application-specific constraints into the definition of the adversary's capability. Therefore, it can be exploited to derive practical guidelines for developing optimal attack strategies and to guide the design of adversarially resilient classifiers.

3.1.1 Adversary's Goal

According to the taxonomy presented first by Barreno *et al.* [5] and extended by Huang *et al.* [36], the adversary's goal should be defined based on the anticipated security violation, which might be an integrity, availability, or privacy violation (see Section 2.3.3), and also depending on the attack's specificity, which ranges from targeted to indiscriminate. Further, as suggested by Laskov and Kloft [42] and Kloft and Laskov [40], the adversary's goal should be defined in terms of an objective function that the adversary is willing to maximize. This allows for a formal characterization of the *optimal* attack strategy.

For instance, in an indiscriminate integrity attack, the adversary may aim to maximize the number of spam emails that evade detection, while minimally manipulating their content [26, 46, 53], whereas in an indiscriminate availability attack, the adversary may aim to maximize the number of classification errors, thereby causing a general denial-of-service due to an excess of false alarms [52, 14].

3.1.2 Adversary's Knowledge

The adversary's knowledge of the attacked system can be defined based on the main components involved in the design of a machine learning system, as described in [29] and depicted in Figure 3.

According to the five design steps depicted in Figure 3, the adversary may have various degrees of knowledge (ranging from no information to complete information) pertaining to the following five components:

- (*k.i*) the training set (or part of it);
- (*k.ii*) the feature representation of each sample; *i.e.*, how *real* objects (emails, network packets, *etc.*) are mapped into the feature space;
- (*k.iii*) the learning algorithm and its decision function; *e.g.*, that logistic regression is used to learn a linear classifier;
- (*k.iv*) the learned classifier's parameters; *e.g.*, the actual learned weights of a linear classifier;

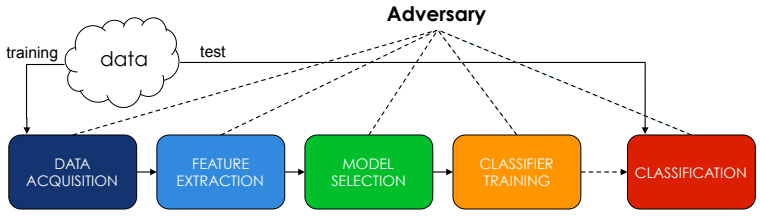

Fig. 3 A representation of the design steps of a machine learning system [29] that may provide sources of knowledge for the adversary.

(*k.v*) feedback from the deployed classifier; *e.g.*, the classification labels assigned to some of the samples by the targeted classifier.

These five elements represent different levels of knowledge about the system being attacked. A typical hypothesized scenario assumes that the adversary has *perfect knowledge* of the targeted classifier (*k.iv*). Although potentially too pessimistic, this worst-case setting allows one to compute a lower bound on the classifier performance when it is under attack [26, 41]. A more realistic setting is that the adversary knows the (untrained) learning algorithm (*k.iii*), and she may exploit feedback from the classifier on the labels assigned to some *query* samples (*k.v*), either to directly find optimal or nearly-optimal attack instances [46, 53], or to learn a surrogate classifier, which can then serve as a template to guide the attack against the actual classifier. We refer to this scenario as a *limited knowledge* setting in Section 4.

Note that one may also make more restrictive assumptions on the adversary's knowledge, such as considering partial knowledge of the feature representation (*k.ii*), or a complete lack of knowledge of the learning algorithm (*k.iii*). Investigating classifier security against these uninformed adversaries may yield a higher level of security. However, such assumptions would be contingent on *security through obscurity*; that is, the provided security would rely upon *secrets* that must be kept unknown to the adversary even though such a high level of secrecy may not be practical. Reliance on unjustified secrets can potentially lead to catastrophic unforeseen vulnerabilities. Thus, this paradigm should be regarded as being complementary to *security by design*, which instead advocates that systems should be designed from the ground-up to be secure and, if secrets are assumed, they must be well-justified. Accordingly, security is often investigated by assuming that the adversary knows at least the learning algorithm and the underlying feature representation.

3.1.3 Adversary's Capability

We now give some guidelines on how the attacker may be able to manipulate samples and the corresponding data distribution. As discussed in Section 2.3.3 [4, 5, 36], the adversary may control both training and test data (causative attacks), or only on test data (exploratory attacks). Further, training and test data may follow different

distributions, since they can be manipulated according to different attack strategies by the adversary. Therefore, we should specify:

- (*c.i*) whether the adversary can manipulate training (TR) and/or testing (TS) data; *i.e.*, the attack influence from the taxonomy in [4, 5, 36]);
- (*c.ii*) whether and to what extent the attack affects the class priors, for TR and TS;
- (*c.iii*) which and how many samples can be modified in each class, for TR and TS;
- (*c.iv*) which features of each attack sample can be modified and how can these features' values be altered; *e.g.*, correlated feature values can not be modified independently.

Assuming a generative model $p(\mathbf{X},Y) = p(Y)p(\mathbf{X}|Y)$ (where we use p_{tr} and p_{ts} for training and test distributions, respectively), assumption (*c.ii*) specifies how an attack can modify the priors $p_{\text{tr}}(Y)$ and $p_{\text{ts}}(Y)$ while assumptions (*c.iii*) and (*c.iv*) specifies how it can alter the class-conditional distributions $p_{\text{tr}}(\mathbf{X}|Y)$ and $p_{\text{ts}}(\mathbf{X}|Y)$.

To perform security evaluation according to the hypothesized attack scenario, it is thus clear that the collected data and generated attack samples should be resampled according to the above distributions to produce suitable training and test set pairs. This can be accomplished through existing resampling algorithms like cross-validation or bootstrapping, when the attack samples are independently sampled from an identical distribution (i.i.d.). Otherwise, one may consider different sampling schemes. For instance, in Biggio *et al.* [14] the attack samples had to be injected into the training data, and each attack sample depended on the current training data, which also included past attack samples. In this case, it was sufficient to add one attack sample at a time, until the desired number of samples was reached.[3]

3.1.4 Attack Strategy

Once specific assumptions on the adversary's goal, knowledge, and capability are made, one can compute the optimal attack strategy corresponding to the hypothesized attack scenario; *i.e.*, the adversary model. This amounts to solving the optimization problem defined according to the adversary's goal, under proper constraints defined in accordance with the adversary's assumed knowledge and capabilities. The attack strategy can then be used to produce the desired attack samples, which then have to be merged consistently to the rest of the data to produce suitable training and test sets for the desired security evaluation, as explained in the previous section. Specific examples of how to derive optimal attacks against SVMs, and how to resample training and test data to properly include them are discussed in Sections 4 and 5.

[3] See [12] for more details on the definition of the data distribution and the resampling algorithm.

3.2 How to use our Framework

We summarize here the steps that can be followed to correctly use our framework in specific application scenarios:

1. hypothesize an attack scenario by identifying a proper adversary's goal, and according to the taxonomy in [4, 5, 36];
2. define the adversary's knowledge according to (*k.i-v*), and capabilities according to (*c.i-iv*);
3. formulate the corresponding optimization problem and devise the corresponding attack strategy;
4. resample the collected (training and test) data accordingly;
5. evaluate classifier's security on the resampled data (including attack samples);
6. repeat the evaluation for different levels of adversary's knowledge and/or capabilities, if necessary; or hypothesize a different attack scenario.

In the next sections we show how our framework can be applied to investigate three security threats to SVMs: evasion, poisoning, and privacy violations. We then discuss how our findings may be used to improve the security of such classifiers to the considered attacks. For instance, we show how careful kernel parameter selection, which trades off between security to attacks and classification accuracy, may complicate the adversary's task of subverting the learning process.

4 Evasion Attacks against SVMs

In this section, we consider the problem of SVM evasion at *test time*; *i.e.*, how to optimally manipulate samples at test time to avoid detection. The problem of evasion at test time has been considered in previous work, albeit either limited to simple decision functions such as linear classifiers [26, 46], or to cover any convex-inducing classifiers [53] that partition the feature space into two sets, one of which is convex, but do not include most interesting families of non-linear classifiers such as neural nets or SVMs. In contrast to this prior work, the methods presented in our recent work [8] and in this section demonstrate that evasion of kernel-based classifiers at test time can be realized with a straightforward gradient-descent-based approach derived from Golland's technique of discriminative directions [33]. As a further simplification of the attacker's effort, we empirically show that, even if the adversary does not precisely know the classifier's decision function, she can learn a *surrogate* classifier on a surrogate dataset and reliably evade the targeted classifier.

This section is structured as follows. In Section 4.1, we define the model of the adversary, including her attack strategy, according to our evaluation framework described in Section 3.1. Then, in Section 4.2 we derive the attack strategy that will be employed to experimentally evaluate evasion attacks against SVMs. We report our experimental results in Section 4.3. Finally, we critically discuss and interpret our research findings in Section 4.4.

4.1 Modeling the Adversary

We show here how our framework can be applied to evaluate the security of SVMs against evasion attacks. We first introduce our notation, state our assumptions about attack scenario, and then derive the corresponding optimal attack strategy.

Notation. We consider a classification algorithm $f : \mathscr{X} \mapsto \mathscr{Y}$ that assigns samples represented in some feature space $\mathbf{x} \in \mathscr{X}$ to a label in the set of predefined classes $y \in \mathscr{Y} = \{-1, +1\}$, where -1 $(+1)$ represents the legitimate (malicious) class. The label $f_\mathbf{x} = f(\mathbf{x})$ given by a classifier is typically obtained by thresholding a continuous discriminant function $g : \mathscr{X} \mapsto \mathbb{R}$. Without loss of generality, we assume that $f(\mathbf{x}) = -1$ if $g(\mathbf{x}) < 0$, and $+1$ otherwise. Further, note that we use $f_\mathbf{x}$ to refer to a label assigned by the classifier for the point $\mathbf{x}$ (rather than the true label y of that point) and the shorthand f_i for the label assigned to the i^{th} training point, $\mathbf{x}_i$.

4.1.1 Adversary's Goal

Malicious (positive) samples are manipulated to evade the classifier. The adversary may be satisfied when a sample $\mathbf{x}$ is found such that $g(\mathbf{x}) < -\varepsilon$ where $\varepsilon > 0$ is a small constant. However, as mentioned in Section 3.1.1, these attacks may be easily defeated by simply adjusting the decision threshold to a slightly more conservative value (*e.g.*, to attain a lower false negative rate at the expense of a higher false positive rate). For this reason, we assume a *smarter* adversary, whose goal is to have her attack sample misclassified as legitimate with the largest confidence. Analytically, this statement can be expressed as follows: find an attack sample $\mathbf{x}$ that minimizes the value of the classifier's discriminant function $g(\mathbf{x})$. Indeed, this adversarial setting provides a worst-case bound for the targeted classifier.

4.1.2 Adversary's Knowledge

We investigate two adversarial settings. In the first, the adversary has *perfect* knowledge (PK) of the targeted classifier; *i.e.*, she knows the feature space (*k.ii*) and function $g(\mathbf{x})$ (*k.iii-iv*). Thus, the labels from the targeted classifier (*k.v*) are not needed. In the second, the adversary is assumed to have *limited* knowledge (LK) of the classifier. We assume she knows the feature representation (*k.ii*) and the learning algorithm (*k.iii*), but that she does not know the learned classifier $g(\mathbf{x})$ (*k.iv*). In both cases, we assume the attacker does not have knowledge of the training set (*k.i*).

Within the LK scenario, the adversary does not know the true discriminant function $g(\mathbf{x})$ but may approximate it as $\hat{g}(\mathbf{x})$ by learning a *surrogate* classifier on a surrogate training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n_q}$ of n_q samples. This data may be collected by the adversary in several ways; *e.g.*, she may sniff network traffic or collect legitimate and spam emails from an alternate source. Thus, for LK, there are two sub-cases related to assumption (*k.v*), which depend on whether the adversary can query the classifier. If so, the adversary can build the training set by submitting a set of n_q

queries $\mathbf{x}_i$ to the targeted classifier to obtain their classification labels, $y_i = f(\mathbf{x}_i)$. This is indeed the adversary's true learning task, but it requires her to have access to classifier feedback; *e.g.*, by having an email account protected by the targeted filter (for public email providers, the adversary can reasonably obtain such accounts). If not, the adversary may use the true class labels for the surrogate data, although this may not correctly approximate the targeted classifier (unless it is very accurate).

4.1.3 Adversary's Capability

In the evasion setting, the adversary can only manipulate testing data (*c.i*); *i.e.*, she has no way to influence training data. We further assume here that the class priors can not be modified (*c.ii*), and that all the malicious testing samples are affected by the attack (*c.iii*). In other words, we are interested in simulating an *exploratory*, *indiscriminate* attack. The adversary's capability of manipulating the features of each sample (*c.iv*) should be defined based on application-specific constraints. However, at a more general level we can bound the attack point to lie within some maximum distance from the original attack sample, $d_{\max}$, which then is a parameter of our evaluation. Similarly to previous work, the definition of a suitable distance measure $d : \mathscr{X} \times \mathscr{X} \mapsto \mathbb{R}$ is left to the specific application domain [26, 46, 53]. Note indeed that this distance should reflect the adversary's effort or cost in manipulating samples, by considering factors that can limit the overall attack impact; *e.g.*, the increase in the file size of a malicious PDF, since larger files will lower the infection rate due to increased transmission times. For spam filtering, distance is often given as the number of modified words in each spam [26, 46, 52, 53], since it is assumed that highly modified spam messages are less effectively able to convey the spammer's message.

4.1.4 Attack Strategy

Under the attacker's model described in Sections 4.1.1, 4.1.2 and 4.1.3, for any target malicious sample $\mathbf{x}^0$ (the adversary's true objective), an optimal attack strategy finds a sample $\mathbf{x}^*$ to minimize g or its estimate $\hat{g}$, subject to a bound on its modification distance from $\mathbf{x}^0$:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \hat{g}(\mathbf{x}) \quad \text{s.t.} \quad d(\mathbf{x}, \mathbf{x}^0) \leq d_{\max} \ .$$

For several classifiers, minimizing $g(\mathbf{x})$ is equivalent to maximizing the estimated posterior $p(f_{\mathbf{x}} = -1|\mathbf{x})$; *e.g.*, for neural networks, since they directly output a posterior estimate, and for SVMs, since their posterior can be estimated as a sigmoidal function of the distance of $\mathbf{x}$ to the SVM hyperplane [55].

Generally, this is a non-linear optimization, which one may optimize with many well-known techniques (*e.g.*, gradient descent, Newton's method, or BFGS) and below we use a gradient descent procedure. However, if $\hat{g}(\mathbf{x})$ is not convex, descent

approaches may not find a global optima. Instead, the descent path may lead to a flat region (local minimum) outside of the samples' support where $p(\mathbf{x}) \approx 0$ and the classification behavior of g is unspecified and may stymie evasion attempts (see the *upper left* plot in Figure 4).

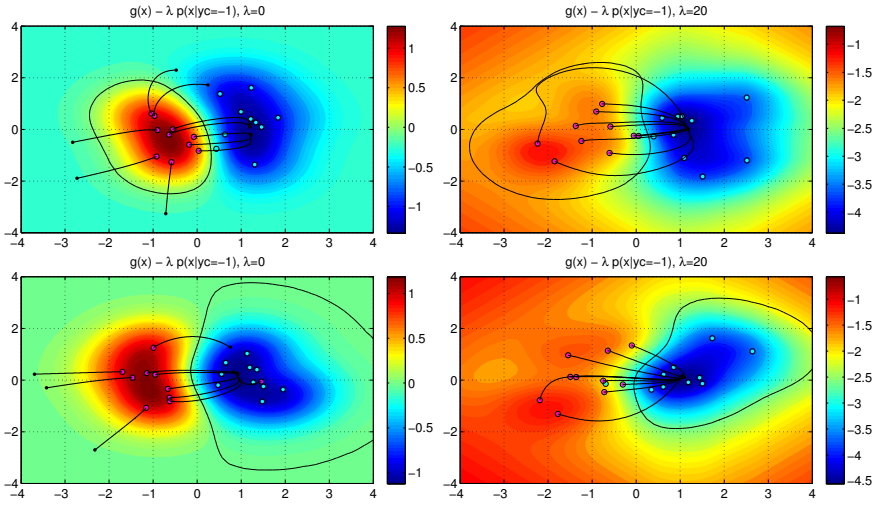

Fig. 4 Different scenarios for gradient-descent-based evasion procedures. In each, the function $g(\mathbf{x})$ of the learned classifier is plotted with a color map with high values (red-orange-yellow) corresponding to the malicious class, low values (green-cyan-blue) corresponding to the benign class, and a black decision boundary for the classifier. For every malicious sample, we plot the path of a simple gradient descent evasion for a classifier with a closed boundary around the malicious class (**upper left**) or benign class (**bottom left**). Then, we plot the modified objective function of Equation (1) and the paths of the resulting density-augmented gradient descent for a classifier with a closed boundary around the malicious (**upper right**) or benign class (**bottom right**).

Unfortunately, our objective does not utilize the evidence we have about the distribution of data $p(\mathbf{x})$, and thus gradient descent may meander into unsupported regions ($p(\mathbf{x}) \approx 0$) where g is relatively unspecified. This problem is further compounded since our estimate $\hat{g}$ is based on a finite (and possibly small) training set making it a poor estimate of g in unsupported regions, which may lead to false evasion points in these regions. To overcome these limitations, we introduce an additional component into the formulation of our attack objective, which estimates $p(\mathbf{x}|f_{\mathbf{x}} = -1)$ using density-estimation techniques. This second component acts as a penalizer for $\mathbf{x}$ in low density regions and is weighted by a parameter $\lambda \geq 0$ yielding the following modified optimization problem:

$$\begin{aligned} \arg\min_{\mathbf{x}} \;\; & E(\mathbf{x}) = \hat{g}(\mathbf{x}) - \frac{\lambda}{n} \sum_{i|f_i=-1} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) \\ \text{s.t.} \;\; & d(\mathbf{x}, \mathbf{x}^0) \leq d_{\max} \;\; , \end{aligned} \tag{1}$$

where h is a bandwidth parameter for a kernel density estimator (KDE), and n is the number of benign samples ($f_{\mathbf{x}} = -1$) available to the adversary. This alternate objective trades off between minimizing $\hat{g}(\mathbf{x})$ (or $p(f_{\mathbf{x}} = -1|\mathbf{x})$) and maximizing the estimated density $p(\mathbf{x}|f_{\mathbf{x}} = -1)$. The extra component favors attack points to imitate features of known samples classified as legitimate, as in mimicry attacks [31]. In doing so, it reshapes the objective function and thereby biases the resulting *density-augmented gradient descent* towards regions where the negative class is concentrated (see the *bottom right* plot in Figure 4).

Finally, note that this behavior may lead our technique to disregard attack patterns within unsupported regions ($p(\mathbf{x}) \approx 0$) for which $g(\mathbf{x}) < 0$, when they do exist (see, *e.g.*, the *upper right* plot in Figure 4). This may limit classifier evasion especially when the constraint $d(\mathbf{x}, \mathbf{x}^0) \leq d_{\max}$ is particularly strict. Therefore, the trade-off between the two components of the objective function should be carefully considered.

4.2 Evasion Attack Algorithm

Algorithm 1 details a gradient-descent method for optimizing problem of Equation (1). It iteratively modifies the attack point $\mathbf{x}$ in the feature space as $\mathbf{x}' \leftarrow \mathbf{x} - t\nabla E$, where ∇E is a unit vector aligned with the gradient of our objective function, and t is the step size. We assume g to be differentiable almost everywhere (subgradients may be used at discontinuities). When g is non-differentiable or is not smooth enough for a gradient descent to work well, it is also possible to rely upon the mimicry / KDE term in the optimization of Equation (1).

Algorithm 1 Gradient-descent attack procedure

Input: the initial attack point, $\mathbf{x}^0$; the step size, t; the trade-off parameter, λ; and $\varepsilon > 0$.
Output: $\mathbf{x}^*$, the final attack point.

1: $k \leftarrow 0$.
2: **repeat**
3: $k \leftarrow k + 1$
4: Set $\nabla E(\mathbf{x}^{k-1})$ to a unit vector aligned with $\nabla g(\mathbf{x}^{k-1}) - \lambda \nabla p(\mathbf{x}^{k-1}|f_{\mathbf{x}} = -1)$.
5: $\mathbf{x}^k \leftarrow \mathbf{x}^{k-1} - t\nabla E(\mathbf{x}^{k-1})$
6: **if** $d(\mathbf{x}^k, \mathbf{x}^0) > d_{\max}$ **then**
7: Project $\mathbf{x}^k$ onto the boundary of the feasible region (enforcing application-specific constraints, if any).
8: **end if**
9: **until** $E\left(\mathbf{x}^k\right) - E\left(\mathbf{x}^{k-1}\right) < \varepsilon$
10: **return:** $\mathbf{x}^* = \mathbf{x}^k$

In the next sections, we show how to compute the components of ∇E; namely, the gradient of the discriminant function $g(\mathbf{x})$ of SVMs for different kernels, and the

gradient of the mimicking component (density estimation). We finally discuss how to project the gradient ∇E onto the feasible region in *discrete* feature spaces.

4.2.1 Gradient of Support Vector Machines

For SVMs, $g(\mathbf{x}) = \sum_i \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b$. The gradient is thus given by $\nabla g(\mathbf{x}) = \sum_i \alpha_i y_i \nabla k(\mathbf{x}, \mathbf{x}_i)$. Accordingly, the feasibility of the approach depends on the computability of this kernel gradient $\nabla k(\mathbf{x}, \mathbf{x}_i)$, which is computable for many numeric kernels. In the following, we report the kernel gradients for three main cases: (a) the linear kernel, (b) the RBF kernel, and (c) the polynomial kernel.

(a) Linear kernel. In this case, the kernel is simply given by $k(\mathbf{x}, \mathbf{x}_i) = \langle \mathbf{x}, \mathbf{x}_i \rangle$. Accordingly, $\nabla k(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}_i$ (we remind the reader that the gradient has to be computed with respect to the current attack sample $\mathbf{x}$), and $\nabla g(\mathbf{x}) = \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$.

(b) RBF kernel. For this kernel, $k(\mathbf{x}, \mathbf{x}_i) = \exp\{-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2\}$. The gradient is thus given by $\nabla k(\mathbf{x}, \mathbf{x}_i) = -2\gamma \exp\{-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2\}(\mathbf{x} - \mathbf{x}_i)$.

(c) Polynomial kernel. In this final case, $k(\mathbf{x}, \mathbf{x}_i) = (\langle \mathbf{x}, \mathbf{x}_i \rangle + c)^p$. The gradient is thus given by $\nabla k(\mathbf{x}, \mathbf{x}_i) = p(\langle \mathbf{x}, \mathbf{x}_i \rangle + c)^{p-1} \mathbf{x}_i$.

4.2.2 Gradients of Kernel Density Estimators

As with SVMs, the gradient of kernel density estimators depends on the gradient of its kernel. We considered generalized RBF kernels of the form

$$k\left(\tfrac{\mathbf{x}-\mathbf{x}_i}{h}\right) = \exp\left(-\tfrac{d(\mathbf{x},\mathbf{x}_i)}{h}\right),$$

where $d(\cdot,\cdot)$ is any suitable distance function. We used here the same distance $d(\cdot,\cdot)$ used in Equation (1), but they can be different, in general. For ℓ_2- and ℓ_1-norms (*i.e.*, RBF and Laplacian kernels), the KDE (sub)gradients are respectively given by:

$$-\frac{2}{nh} \sum_{i|f_i=-1} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|_2^2}{h}\right)(\mathbf{x} - \mathbf{x}_i) \ ,$$

$$-\frac{1}{nh} \sum_{i|f_i=-1} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|_1}{h}\right)(\mathbf{x} - \mathbf{x}_i) \ .$$

Note that the scaling factor here is proportional to $O(\frac{1}{nh})$. Therefore, to influence gradient descent with a significant mimicking effect, the value of λ in the objective function should be chosen such that the value of $\frac{\lambda}{nh}$ is comparable to (or higher than) the range of the discriminant function $\hat{g}(\mathbf{x})$.

4.2.3 Gradient Descent Attack in Discrete Spaces

In discrete spaces, gradient approaches may lead to a path through infeasible portions of the feature space. In such cases, we need to find feasible neighbors $\mathbf{x}$ that yield a steepest descent; *i.e.*, maximally decreasing $E(\mathbf{x})$. A simple approach to this problem is to probe E at every point in a small neighborhood of $\mathbf{x}$: $\mathbf{x}' \leftarrow \arg\min_{\mathbf{z}\in\mathscr{N}(\mathbf{x})} E(\mathbf{z})$. However, this approach requires a large number of queries. For classifiers with a differentiable decision function, we can instead use the neighbor whose difference from $\mathbf{x}$ best aligns with $\nabla E(\mathbf{x})$; *i.e.*, the update becomes

$$\mathbf{x}' \leftarrow \arg\max_{\mathbf{z}\in\mathscr{N}(\mathbf{x})} \frac{(\mathbf{z}-\mathbf{x})}{\|\mathbf{z}-\mathbf{x}\|}^{\top} \nabla E(\mathbf{x}) \ .$$

Thus, the solution to the above alignment is simply to modify a feature that satisfies $\arg\max_i |\nabla E(\mathbf{x})_i|$ for which the corresponding change leads to a feasible state. Note however that, sometimes, such a step may be relatively quite large, and may lead the attack out of a local minimum potentially increasing the objective function. Therefore, one should consider the best alignment that effectively reduces the objective function by disregarding features that lead to states where the objective function is higher.

4.3 Experiments

In this section, we first report some experimental results on the MNIST handwritten digit classification task [32, 45], that visually demonstrate how the proposed algorithm modifies digits to mislead classification. This dataset is particularly useful because the visual nature of the handwritten digit data provides a *semantic meaning* for attacks. We then show the effectiveness of the proposed attack on a more realistic and practical scenario: the detection of malware in PDF files.

4.3.1 Handwritten Digits

We first focus on a two-class sub-problem of discriminating between two distinct digits from the MNIST dataset [45]. Each digit example is represented as a gray-scale image of 28×28 pixels arranged in raster-scan-order to give feature vectors of $d = 28 \times 28 = 784$ values. We normalized each feature (pixel) $x_f \in [0,1]^d$ by dividing its value by 255, and we constrained the attack samples to this range. Accordingly, we optimized Equation (1) subject to $0 \leq x_f \leq 1$ for all f.

For our attacker, we assume the perfect knowledge (PK) attack scenario. We used the *Manhattan* distance (ℓ_1-norm) as the distance function, d, both for the kernel density estimator (*i.e.*, a Laplacian kernel) and for the constraint $d(\mathbf{x},\mathbf{x}^0) \leq d_{\max}$ of Equation (1), which bounds the total difference between the gray level values of

the original image $\mathbf{x}^0$ and the attack image $\mathbf{x}$. We used an upper bound of $d_{\max} = \frac{5000}{255}$ to limit the total change in the gray-level values to 5000. At each iteration, we increased the ℓ_1-norm value of $\mathbf{x} - \mathbf{x}^0$ by $\frac{10}{255}$, which is equivalent to increasing the difference in the gray level values by 10. This is effectively the gradient step size.

For the digit discrimination task, we applied an SVM with the linear kernel and $C = 1$. We randomly chose 100 training samples and applied the attacks to a correctly-classified positive sample.

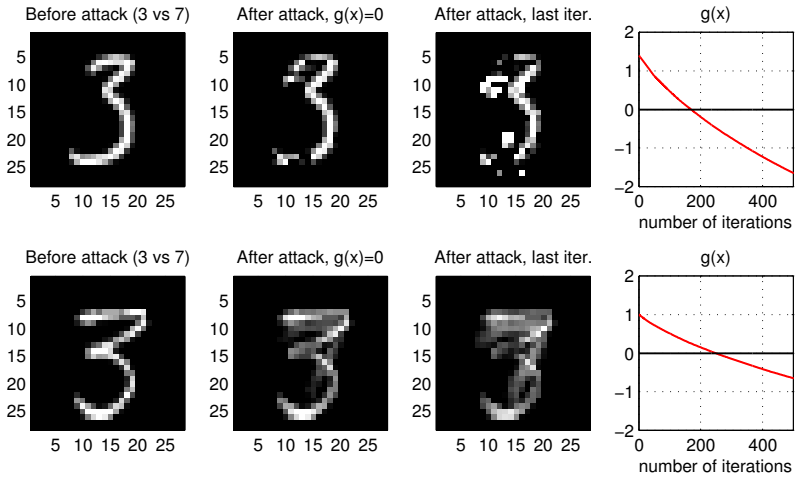

Fig. 5 Illustration of the gradient attack on the MNIST digit data, for $\lambda = 0$ (**top row**) and $\lambda = 10$ (**bottom row**). Without a mimicry component ($\lambda = 0$) gradient descent quickly decreases g but the resulting attack image does not resemble a "7". In contrast, the attack minimizes g slower when mimicry is applied ($\lambda = 10$) but the final attack image closely resembles a mixture between "3" and "7", as the term "mimicry" suggests.

In Figure 5 we illustrate gradient attacks in which a "3" is to be misclassified as a "7". The left image shows the initial attack point, the middle image shows the first attack image misclassified as legitimate, and the right image shows the attack point after 500 iterations. When $\lambda = 0$, the attack images exhibit only a weak resemblance to the target class "7" but are, nevertheless, reliably misclassified. This is the same effect we observed in the left plot of Figure 4: the classifier is evaded by making the attack sample dissimilar to the malicious class. Conversely, when $\lambda = 10$ the attack images strongly resemble the target class because the mimicry term favors samples that are more similar to the target examples. This is the same effect illustrated in the rightmost plot of Figure 4.

4.3.2 Malware Detection in PDF Files

We focus now on the problem of discriminating between legitimate and malicious PDF files, a popular medium for disseminating malware [67]. PDF files are excellent vectors for malicious-code, due to their flexible *logical structure*, which can described by a hierarchy of interconnected objects. As a result, an attack can be easily hidden in a PDF to circumvent file-type filtering. The PDF format further allows a wide variety of resources to be embedded in the document including `JavaScript`, `Flash`, and even binary programs. The type of the embedded object is specified by *keywords*, and its content is in a *data stream*. Several recent works proposed machine-learning techniques for detecting malicious PDFs use the file's logical structure to accurately identify the malware [49, 62, 63]. In this case study, we use the feature representation of Maiorca *et al.* [49] in which each feature corresponds to the tally of occurrences of a given keyword in the PDF file. Similar feature representations were also exploited in [62, 63].

The PDF application imposes natural constraints on attacks. Although it is difficult to *remove* an embedded object (and its corresponding keywords) without corrupting the PDF's file structure, it is rather easy to *insert* new objects (and, thus, keywords) through the addition of a new *version* to the PDF file [1]. In our feature representation, this is equivalent to allowing only feature increments,;*i.e.*, requiring $\mathbf{x}^0 \leq \mathbf{x}$ as an additional constraint in the optimization problem given by Equation (1). Further, the total difference in keyword counts between two samples is their *Manhattan* distance, which we again use for the kernel density estimator and the constraint in Equation (1). Accordingly, $d_{\max}$ is the maximum number of additional keywords that an attacker can add to the original $\mathbf{x}^0$.

Experimental setup. For experiments, we used a PDF corpus with 500 malicious samples from the *Contagio* dataset[4] and 500 benign samples collected from the web. We randomly split the data into five pairs of training and testing sets with 500 samples each to average the final results. The features (keywords) were extracted from each training set as described in [49]; on average, 100 keywords were found in each run. Further, we also bounded the maximum value of each feature (keyword count) to 100, as this value was found to be close to the 95$^{\text{th}}$ percentile for each feature. This limited the influence of outlying samples having very high feature values.

We simulated the *perfect* knowledge (PK) and the *limited* knowledge (LK) scenarios described in Section 4.1.2. In the LK case, we set the number of samples used to learn the surrogate classifier to $n_q = 100$. The reason is to demonstrate that even with a dataset as small as the 20% of the original training set size, the adversary may be able to evade the targeted classifier with high reliability. Further, we assumed that the adversary uses feedback from the *targeted* classifier f; *i.e.*, the labels $\hat{y}_i = f_i = f(\mathbf{x}_i)$ for each surrogate sample $\mathbf{x}_i$. Similar results were also obtained using the true labels (without relabeling), since the targeted classifiers correctly classified almost all samples in the test set.

[4] `http://contagiodump.blogspot.it`

As discussed in Section 4.2.2, the value of λ is chosen according to the scale of the discriminant function $g(\mathbf{x})$, the bandwidth parameter h of the kernel density estimator, and the number of samples n labeled as legitimate in the surrogate training set. For computational reasons, to estimate the value of the KDE at a given point $\mathbf{x}$ in the feature space, we only consider the 50 nearest (legitimate) training samples to $\mathbf{x}$; therefore $n \leq 50$ in our case. The bandwidth parameter was set to $h = 10$, as this value provided a proper rescaling of the Manhattan distances observed in our dataset for the KDE. We thus set $\lambda = 500$ to be comparable with $O(nh)$.

For each targeted classifier and training/testing pair, we learned five different surrogate classifiers by randomly selecting n_q samples from the test set, and averaged their results. For SVMs, we sought a surrogate classifier that would correctly match the labels from the targeted classifier; thus, we used parameters $C = 100$, and $\gamma = 0.1$ (for the RBF kernel) to heavily penalize training errors.

Experimental results. We report our results in Figure 6, in terms of the false negative (FN) rate attained by the targeted classifiers as a function of the maximum allowable number of modifications, $d_{\max} \in [0, 50]$. We compute the FN rate corresponding to a fixed false positive (FP) rate of FP$= 0.5\%$. For $d_{\max} = 0$, the FN rate corresponds to a standard performance evaluation using unmodified PDFs. As expected, the FN rate increases with $d_{\max}$ as the PDF is increasingly modified, since the adversary has more flexibility in his attack. Accordingly, a more secure classifier will exhibit a more graceful increase of the FN rate.

Results for $\lambda = 0$. We first investigate the effect of the proposed attack in the PK case, without considering the mimicry component (Figure 6, top row), for varying parameters of the considered classifiers. The linear SVM (Figure 6, top-left plot) is almost always evaded with as few as 5 to 10 modifications, independent of the regularization parameter C. It is worth noting that attacking a linear classifier amounts to always incrementing the value of the same highest-weighted feature (corresponding to the `/Linearized` keyword in the majority of the cases) until it is bounded. This continues with the next highest-weighted non-bounded feature until termination. This occurs simply because the gradient of $g(\mathbf{x})$ does not depend on $\mathbf{x}$ for a linear classifier (see Section 4.2.1). With the RBF kernel (Figure 6, top-right plot), SVMs exhibit a similar behavior with $C = 1$ and various values of its γ parameter,[5] and the RBF SVM provides a higher degree of security compared to linear SVMs (*cf.* top-left plot and middle-left plot in Figure 6).

In the LK case, without mimicry (Figure 6, middle row), classifiers are evaded with a probability only *slightly* lower than that found in the PK case, even when only $n_q = 100$ surrogate samples are used to learn the surrogate classifier. This aspect highlights the threat posed by a skilled adversary with incomplete knowledge: only a small set of samples may be required to successfully attack the target classifier using the proposed algorithm.

Results for $\lambda = 500$. When mimicry is used (Figure 6, bottom row), the success of the evasion of linear SVMs (with $C = 1$) decreases both in the PK (*e.g.*, compare the

[5] We also conducted experiments using $C = 0.1$ and $C = 100$, but did not find significant differences compared to the presented results using $C = 1$.

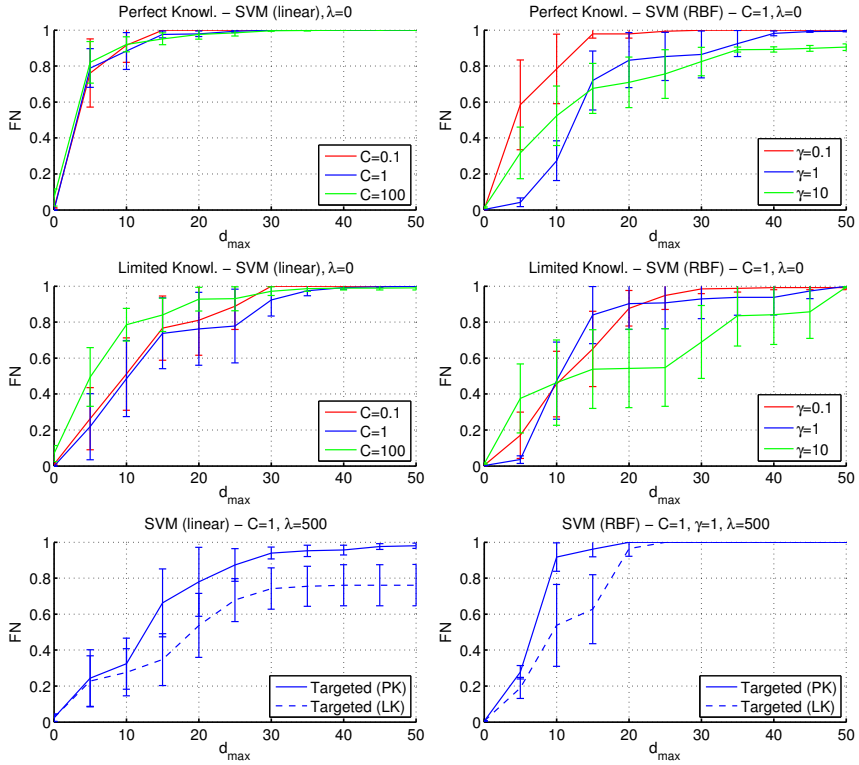

Fig. 6 Experimental results for SVMs with the linear and the RBF kernel (first and second column). We report the FN values (attained at FP=0.5%) for increasing $d_{\max}$. For the sake of readability, we report the average FN value $\pm$ half standard deviation (shown with error bars). Results for perfect knowledge (PK) attacks with $\lambda = 0$ (without mimicry) are shown in the first row, for different values of the considered classifier parameters, *i.e.*, the regularization parameter C for the linear SVM, and the kernel parameter γ for the SVM with RBF kernel (with $C = 1$). Results for limited knowledge (LK) attacks with $\lambda = 0$ (without mimicry) are shown in the second row for the linear SVM (for varying C), and the SVM with RBF kernel (for varying γ, with $C = 1$). Results for perfect (PK) and limited knowledge (LK) with $\lambda = 500$ (with mimicry) are shown in the third row for the linear SVM (with $C = 1$), and the SVM with RBF kernel (with $\gamma = 1$ and $C = 1$).

blue curve in the top-left plot with the solid blue curve in the bottom-left plot) and LK case (*e.g.*, compare the blue curve in the middle-left plot with the dashed blue curve in the bottom-left plot). The reason is that the computed direction tends to lead to a slower descent; *i.e.*, a less direct path that often requires more modifications to evade the classifier. In the non-linear case (Figure 6, bottom-right plot), instead, mimicking exhibits some beneficial aspects for the attacker, although the constraint on feature addition may make it difficult to properly mimic legitimate samples. In particular, note how the targeted SVMs with RBF kernel (with $C = 1$ and $\gamma = 1$) in the PK case (*e.g.*, compare the blue curve in the top-right plot with the solid blue

curve in the bottom-right plot) is evaded with a significantly higher probability than in the case when $\lambda = 0$. The reason is that a pure descent strategy on $g(\mathbf{x})$ may find local minima (*i.e.*, attack samples) that do not evade detection, while the mimicry component biases the descent towards regions of the feature space more densely populated by legitimate samples, where $g(\mathbf{x})$ eventually attains lower values. In the LK case (*e.g.*, compare the blue curve in the middle-right plot with the dashed blue curve in the bottom-right plot), however, mimicking does not exhibit significant improvements.

Analysis. Our attacks raise questions about the feasibility of detecting malicious PDFs solely based on logical structure. We found that `/Linearized`, `/OpenAction`, `/Comment`, `/Root` and `/PageLayout` were among the most commonly manipulated keywords. They indeed are found mainly in legitimate PDFs, but can be easily added to malicious PDFs by the versioning mechanism. The attacker can simply insert comments inside the malicious PDF file to augment its `/Comment` count. Similarly, she can embed *legitimate* OpenAction code to add `/OpenAction` keywords or she can add new pages to insert `/PageLayout` keywords.

In summary, our analysis shows that even detection systems that accurately classify non-malicious data can be significantly degraded with only a few malicious modifications. This aspect highlights the importance of developing detection systems that are accurate, but also *designed to be robust* against adversarial manipulation of attack instances.

4.4 Discussion

In this section, we proposed a simple algorithm that allows for evasion of SVMs with differentiable kernels, and, more generally, of any classifier with a differentiable discriminant function. We investigated the attack effectiveness in the case of perfect knowledge of the attacked system. Further, we empirically showed that SVMs can still be evaded with high probability even if the adversary can only learn a classifier's copy on surrogate data (limited knowledge). We believe that the proposed attack formulation can easily be extended to classifiers with non-differentiable discriminant functions as well, such as decision trees and k-nearest neighbors.

Our analysis also suggests some ideas for improving classifier security. In particular, when the classifier tightly *encloses* the legitimate samples, the adversary must increasingly mimic the legitimate class to evade (see Figure 4), and this may not always be possible; *e.g.*, malicious network packets or PDF files still need to embed a valid exploit, and some features may be immutable. Accordingly, a guideline for designing secure classifiers is that learning should encourage a tight enclosure of the legitimate class; *e.g.*, by using a regularizer that penalizes classifying "blind spots"—regions with low $p(\mathbf{x})$—as legitimate. Generative classifiers can be modified, by explicitly modeling the attack distribution, as in [11], and discriminative classifiers can be modified similarly by adding generated attack samples to the training set. However, these security improvements may incur higher FP rates.

In the above applications, the feature representations were *invertible*; *i.e.*, there is a direct mapping from the feature vectors $\mathbf{x}$ to a corresponding real-world sample (*e.g.*, a spam email, or PDF file). However, some feature mappings can not be trivially inverted; *e.g.*, n-gram analysis [31]. In these cases, one may modify the real-world object corresponding to the initial attack point at each step of the gradient descent to obtain a sample in the feature space that as close as possible to the sample that would be obtained at the next attack iteration. A similar technique has been already exploited in to address the pre-image problem of kernel methods [14].

Other interesting extensions include (i) considering more effective strategies such as those proposed by [46, 53] to build a small but representative set of surrogate data to learn the surrogate classifier and (ii) improving the classifier estimate $\hat{g}(\mathbf{x})$; *e.g.*using an ensemble technique like bagging to average several classifiers [16].

5 Poisoning Attacks against SVMs

In the previous section, we devised a simple algorithm that allows for evasion of classifiers at test time and showed experimentally how it can be exploited to evade detection by SVMs and kernel-based classification techniques. Here we present another kind of attack, based on our work in [14]. Its goal is to force the attacked SVM to misclassify as many samples as possible at test time through *poisoning* of the training data, that is, by injecting well-crafted attack samples into the training set. Note that, in this case, the test data is assumed not to be manipulated by the attacker.

Poisoning attacks are staged during classifier training, and they can thus target *adaptive* or *online* classifiers, as well as classifiers that are being re-trained on data collected during test time, especially if in an unsupervised or semi-supervised manner. Examples of these attacks, besides our work [14], can be found in [13, 7, 9, 39, 40, 52, 58]. They include specific application examples in different areas, such as intrusion detection in computer networks [7, 39, 40, 58], spam filtering [7, 52], and, most recently, even biometric authentication [9, 13].

In this section, we follow the same structure of Section 4. In Section 5.1, we define the adversary model according to our framework; then, in Sections 5.1.4 and 5.2 we respectively derive the optimal poisoning attack and the corresponding algorithm; and, finally, in Sections 5.3 and 5.4 we report our experimental findings and discuss the results.

5.1 Modeling the Adversary

Here, we apply our framework to evaluate security against poisoning attacks. As with the evasion attacks in Section 4.1, we model the attack scenario and derive the corresponding optimal attack strategy for poisoning.

Notation. In the following, we assume that an SVM has been trained on a dataset $\mathscr{D}_{\text{tr}} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$. The matrix of kernel values between two sets of points is denoted with $\mathbf{K}$, while $\mathbf{Q} = \mathbf{K} \circ yy^\top$ denotes its label-annotated version, and α denotes the SVM's dual variables corresponding to each training point. Depending on the value of α_i, the training points are referred to as margin support vectors ($0 < \alpha_i < C$, set $\mathscr{S}$), error support vectors ($\alpha_i = C$, set $\mathscr{E}$) or reserve vectors ($\alpha_i = 0$, set $\mathscr{R}$). In the sequel, the lower-case letters s, e, r are used to index the corresponding parts of vectors or matrices; *e.g.*, $\mathbf{Q}_{ss}$ denotes the sub-matrix of $\mathbf{Q}$ corresponding to the margin support vectors.

5.1.1 Adversary's Goal

For a poisoning attack, the attacker's goal is to find a set of points whose addition to $\mathscr{D}_{\text{tr}}$ maximally decreases the SVM's classification accuracy. For simplicity, we start considering the addition of a single attack point $(\mathbf{x}^*, y^*)$. The choice of its label y^* is arbitrary but fixed. We refer to the class of this chosen label as *attacking* class and the other as the *attacked* class.

5.1.2 Adversary's Knowledge

According to Section 3.1.2, we assume that the adversary knows the training samples (*k.i*), the feature representation (*k.ii*), that an SVM learning algorithm is used (*k.iii*) and the learned SVM's parameters (*k.iv*), since they can be inferred by the adversary by solving the SVM learning problem on the *known* training set. Finally, we assume that no feedback is exploited by the adversary (*k.v*).

These assumptions amount to considering a worst-case analysis that allows us to compute the maximum error rate that the adversary can inflict through poisoning. This is indeed useful to check whether and under what circumstances poisoning may be a relevant threat for SVMs.

Although having perfect knowledge of the training data is very difficult in practice for an adversary, collecting a surrogate dataset sampled from the same distribution may not be that complicated; for instance, in network intrusion detection an attacker may easily sniff network packets to build a surrogate learning model, which can then be poisoned under the perfect knowledge setting. The analysis of this limited knowledge poisoning scenario is however left to future work.

5.1.3 Adversary's Capability

According to Section 3.1.3, we assume that the attacker can manipulate only training data (*c.i*), can manipulate the class prior and the class-conditional distribution of the attack point's class y^* by essentially adding a number of attack points of that class into the training data, one at a time (*c.ii-iii*), and can alter the feature values of the

attack sample within some lower and upper bounds (*c.iv*). In particular, we will constrain the attack point to lie within a box, that is $\mathbf{x}_{\text{lb}} \leq \mathbf{x} \leq \mathbf{x}_{\text{ub}}$.

5.1.4 Attack Strategy

Under the above assumptions, the optimal attack strategy amounts to solving the following optimization problem:

$$\mathbf{x}^* = \text{argmax}_{\mathbf{x}} \;\; P(\mathbf{x}) = \sum_{k=1}^{m} (1 - y_k f_{\mathbf{x}}(\mathbf{x}_k))_+ = \sum_{k=1}^{m} (-g_k)_+ \tag{2}$$

$$\text{s.t.} \;\; \mathbf{x}_{\text{lb}} \leq \mathbf{x} \leq \mathbf{x}_{\text{ub}} \;\; , \tag{3}$$

where the hinge loss has to be maximized on a separate validation set $\mathscr{D}_{\text{val}} = \{\mathbf{x}_k, y_k\}_{k=1}^{m}$ to avoid considering a further regularization term in the objective function. The reason is that the attacker aims to maximize the SVM *generalization error* and not only its empirical estimate on the training data.

5.2 Poisoning Attack Algorithm

In this section, we assume the role of the attacker and develop a method for optimizing $\mathbf{x}^*$ according to Equation (2). Since the objective function is non-linear, we use a gradient-ascent algorithm, where the attack vector is initialized by cloning an arbitrary point from the attacked class and flipping its label. This initialized attack point (at iteration 0) is denoted by $\mathbf{x}^0$. In principle, $\mathbf{x}^0$ can be any point *sufficiently deep* within the attacking class's margin. However, if this point is too close to the boundary of the attacking class, the iteratively adjusted attack point may become a reserve point, which halts further progress.

The computation of the gradient of the validation error crucially depends on the assumption that the structure of the sets $\mathscr{S}$, $\mathscr{E}$ and $\mathscr{R}$ does not change during the update. In general, it is difficult to determine the largest step t along the gradient direction ∇P, which preserves this structure. Hence, the step t is fixed to a small constant value in our algorithm. After each update of the attack point $\mathbf{x}^p$, the optimal solution can be efficiently recomputed from the solution on $\mathscr{D}_{\text{tr}}$, using the incremental SVM machinery [20]. The algorithm terminates when the change in the validation error is smaller than a predefined threshold.

5.2.1 Gradient Computation

We now discuss how to compute the gradient ∇P of our objective function. For notational convenience, we now refer to the attack point as $\mathbf{x}_c$ instead of $\mathbf{x}$.

Algorithm 2 Poisoning attack against an SVM

Input: $\mathscr{D}_{\text{tr}}$, the training data; $\mathscr{D}_{\text{val}}$, the validation data; y^*, the class label of the attack point; $\mathbf{x}^0$, the initial attack point; t, the step size.
Output: $\mathbf{x}^*$, the final attack point.

1: $\{\alpha_i, b\} \leftarrow$ learn an SVM on $\mathscr{D}_{\text{tr}}$.
2: $p \leftarrow 0$.
3: **repeat**
4: Re-compute the SVM solution on $\mathscr{D}_{\text{tr}} \cup \{\mathbf{x}^p, y^*\}$ using the incremental SVM [20]. This step requires $\{\alpha_i, b\}$. If computational complexity is manageable, a full SVM can be learned at each step, instead.
5: Compute ∇P on $\mathscr{D}_{\text{val}}$ according to Equation (8).
6: Normalize ∇P to have unit norm.
7: $p \leftarrow p+1$ and $\mathbf{x}^p \leftarrow \mathbf{x}^{p-1} + t\nabla P$
8: **if** $\mathbf{x}_{\text{lb}} > \mathbf{x}^p$ or $\mathbf{x}^p > \mathbf{x}_{\text{ub}}$ **then**
9: Project $\mathbf{x}^p$ onto the boundary of the feasible region (enforce application-specific constraints, if any).
10: **end if**
11: **until** $P(\mathbf{x}^p) - P\left(\mathbf{x}^{p-1}\right) < \varepsilon$
12: **return:** $\mathbf{x}^* = \mathbf{x}^p$

First, we explicitly account for all terms in the margin conditions g_k that are affected by the attack point $\mathbf{x}_c$:

$$\begin{aligned} g_k &= \sum_j \mathbf{Q}_{kj}\alpha_j + y_k b - 1 \\ &= \sum_{j \neq c} \mathbf{Q}_{kj}\alpha_j(\mathbf{x}_c) + \mathbf{Q}_{kc}(\mathbf{x}_c)\alpha_c(\mathbf{x}_c) + y_k b(\mathbf{x}_c) - 1 \ . \end{aligned} \tag{4}$$

As already mentioned, $P(\mathbf{x}_c)$ is a non-convex objective function, and we thus exploit a gradient ascent technique to iteratively optimize it. We denote the initial location of the attack point as $\mathbf{x}_c^0$. Our goal is to update the attack point as $\mathbf{x}_c^p = \mathbf{x}_c^{(p-1)} + t\nabla P$ where p is the current iteration, ∇P is a unit vector representing the attack direction (*i.e.*, the normalized objective gradient), and t is the step size. To maximize our objective, the attack direction ∇P is computed at each iteration.

Although the hinge loss is not everywhere differentiable, this can be overcome by only considering point indices k with non-zero contributions to P; *i.e.*, those for which $-g_k > 0$. Contributions of such points to ∇P can be computed by differentiating Equation (4) with respect to $\mathbf{x}_c$ using the product rule:

$$\frac{\partial g_k}{\partial \mathbf{x}_c} = \mathbf{Q}_{ks}\frac{\partial \alpha}{\partial \mathbf{x}_c} + \frac{\partial \mathbf{Q}_{kc}}{\partial \mathbf{x}_c}\alpha_c + y_k \frac{\partial b}{\partial \mathbf{x}_c}, \tag{5}$$

where, by denoting the l^{th} feature of $\mathbf{x}_c$ as x_{cl}, we use the notation

$$\frac{\partial \alpha}{\partial \mathbf{x}_c} = \begin{bmatrix} \frac{\partial \alpha_1}{\partial x_{c1}} & \cdots & \frac{\partial \alpha_1}{\partial x_{cd}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \alpha_s}{\partial x_{c1}} & \cdots & \frac{\partial \alpha_s}{\partial x_{cd}} \end{bmatrix}, \text{ simil. } \frac{\partial \mathbf{Q}_{kc}}{\partial \mathbf{x}_c}, \frac{\partial b}{\partial \mathbf{x}_c} \ .$$

The expressions for the gradient can be further refined using the fact that the gradient step must preserve the optimal SVM solution. This can expressed as an adiabatic update condition using the technique introduced in [20]. In particular, for the i^{th} *training* point, the KKT conditions of the optimal SVM solution are:

$$g_i = \sum_{j \in \mathscr{D}_{\text{tr}}} \mathbf{Q}_{ij} \alpha_j + y_i b - 1 \begin{cases} > 0; \; i \in \mathscr{R} \\ = 0; \; i \in \mathscr{S} \\ < 0; \; i \in \mathscr{E} \end{cases} \tag{6}$$

$$h = \textstyle\sum_{j \in \mathscr{D}_{\text{tr}}} (y_j \alpha_j) = 0 \ . \tag{7}$$

The form of these conditions implies that an infinitesimal change in the attack point x_c causes a smooth change in the optimal solution of the SVM, under the restriction that the composition of the sets $\mathscr{S}$, $\mathscr{E}$ and $\mathscr{R}$ remains intact. This equilibrium allows us to predict the *response* of the SVM solution to the variation of x_c, as shown below.

By differentiation of the x_c-dependent terms in Equations (6)–(7) with respect to each feature x_{cl} ($1 \leq l \leq d$), we obtain, for any $i \in \mathscr{S}$,

$$\frac{\partial g}{\partial x_{cl}} = \mathbf{Q}_{ss} \frac{\partial \alpha}{\partial x_{cl}} + \frac{\partial \mathbf{Q}_{sc}}{\partial x_{cl}} \alpha_c + y_s \frac{\partial b}{\partial x_{cl}} = 0$$
$$\frac{\partial h}{\partial x_{cl}} = y_s^\top \frac{\partial \alpha}{\partial x_{cl}} = 0 \ .$$

Solving these equations and computing an inverse matrix via the Sherman-Morrison-Woodbury formula [48] yields the following gradients:

$$\frac{\partial \alpha}{\partial \mathbf{x}_c} = -\frac{1}{\zeta} \alpha_c (\zeta \mathbf{Q}_{ss}^{-1} - \upsilon \upsilon^\top) \cdot \frac{\partial \mathbf{Q}_{sc}}{\partial \mathbf{x}_c}$$
$$\frac{\partial b}{\partial \mathbf{x}_c} = -\frac{1}{\zeta} \alpha_c \upsilon^\top \cdot \frac{\partial \mathbf{Q}_{sc}}{\partial \mathbf{x}_c} \ ,$$

where $\upsilon = \mathbf{Q}_{ss}^{-1} y_s$ and $\zeta = y_s^\top \mathbf{Q}_{ss}^{-1} y_s$. We thus obtain the following gradient of the objective used for optimizing our attack, which only depends on $\mathbf{x}_c$ through gradients of the kernel matrix, $\frac{\partial \mathbf{Q}_{kc}}{\partial \mathbf{x}_c}$:

$$\nabla P = \sum_{k=1}^{m} \left\{ M_k \frac{\partial \mathbf{Q}_{sc}}{\partial \mathbf{x}_c} + \frac{\partial \mathbf{Q}_{kc}}{\partial \mathbf{x}_c} \right\} \alpha_c \ , \tag{8}$$

where $M_k = -\frac{1}{\zeta} (\mathbf{Q}_{ks} (\zeta \mathbf{Q}_{ss}^{-1} - \upsilon \upsilon^T) + y_k \upsilon^T)$.

5.2.2 Kernelization

From Equation (8), we see that the gradient of the objective function at iteration p may depend on the attack point $\mathbf{x}_c^p = \mathbf{x}_c^{p-1} + t\nabla P$ only through the gradients of the matrix $\mathbf{Q}$. In particular, this depends on the chosen kernel. We report below the expressions of these gradients for three common kernels (see Section 4.2.1):

- Linear kernel: $\frac{\partial K_{ic}}{\partial \mathbf{x}_c} = \frac{\partial(\mathbf{x}_i \cdot \mathbf{x}_c)}{\partial \mathbf{x}_c} = \mathbf{x}_i$
- Polynomial kernel: $\frac{\partial K_{ic}}{\partial \mathbf{x}_c} = \frac{\partial(\mathbf{x}_i \cdot \mathbf{x}_c + R)^d}{\partial \mathbf{x}_c} = d(\mathbf{x}_i \cdot \mathbf{x}_c + R)^{d-1}\mathbf{x}_i$
- RBF kernel: $\frac{\partial K_{ic}}{\partial \mathbf{x}_c} = \frac{\partial e^{-\gamma||\mathbf{x}_i - \mathbf{x}_c||^2}}{\partial \mathbf{x}_c} = 2\gamma K(\mathbf{x}_i, \mathbf{x}_c)(\mathbf{x}_i - \mathbf{x}_c)$

The dependence of these gradients on the current attack point, $\mathbf{x}_c$, can be avoided by using the previous attack point, provided that t is sufficiently small. This approximation enables a straightforward extension of our method to arbitrary differentiable kernels.

5.3 Experiments

The experimental evaluation presented in the following sections demonstrates the behavior of our proposed method on an artificial two-dimensional dataset and evaluates its effectiveness on the classical MNIST handwritten digit recognition dataset [32, 45].

5.3.1 Two-dimensional Toy Example

Here we consider a two-dimensional example in which each class follows a Gaussian with mean and covariance matrices given by $\mu_- = [-1.5, 0]$, $\mu_+ = [1.5, 0]$, $\Sigma_- = \Sigma_+ = 0.6I$. The points from the *negative* distribution have label -1 (shown as red in subsequent figures) and otherwise $+1$ (shown as blue). The training and the validation sets, $\mathscr{D}_{\text{tr}}$ and $\mathscr{D}_{\text{val}}$, consist of 25 and 500 points per class, respectively.

In the experiment presented below, the red class is the *attacking* class. That is, a random point of the blue class is selected and its label is flipped to serve as the starting point for our method. Our gradient ascent method is then used to refine this attack until its termination condition is satisfied. The attack's trajectory is traced as the black line in Figure 7 for both the linear kernel (upper two plots) and the RBF kernel (lower two plots). The background of each plot depicts an error surface: hinge loss computed on a validation set (leftmost plots) and the classification error (rightmost plots). For the linear kernel, the range of attack points is limited to the box $\mathbf{x} \in [-4, 4]^2$ shown as a dashed line. This implements the constraint of Equation (3).

For both kernels, these plots show that our gradient ascent algorithm finds a reasonably good local maximum of the non-convex error surface. For the linear kernel, it terminates at the corner of the bounded region, since the error surface is un-

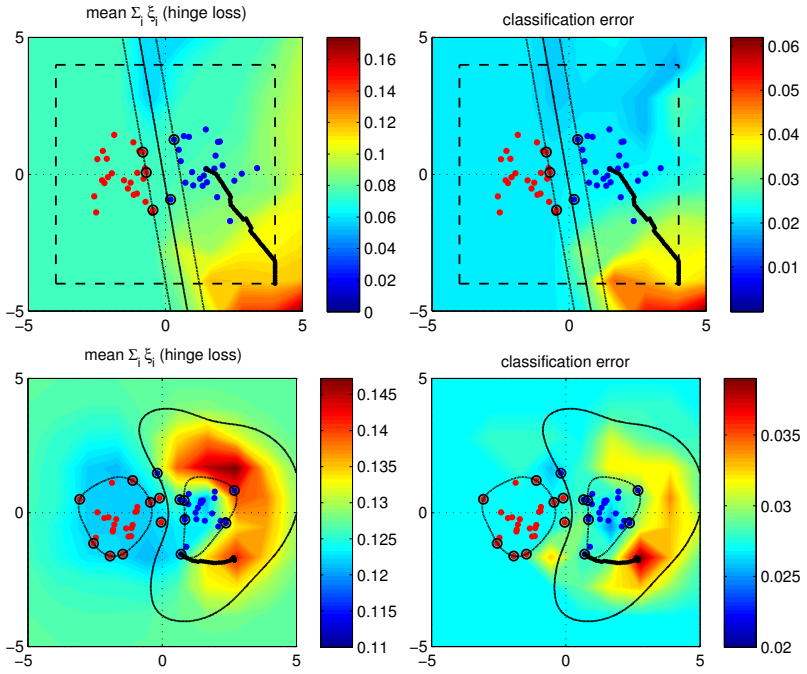

Fig. 7 Behavior of the gradient-based attack strategy on the Gaussian datasets, for the linear (top row) and the RBF kernel (bottom row) with $\gamma = 0.5$. The regularization parameter C was set to 1 in both cases. The solid black line represents the gradual shift of the attack point $\mathbf{x}_c^p$ toward a local maximum. The hinge loss and the classification error are shown in colors, to appreciate that the hinge loss provides a good approximation of the classification error. The value of such functions for each point $\mathbf{x} \in [-5,5]^2$ is computed by learning an SVM on $\mathscr{D}_{\text{tr}} \cup \{\mathbf{x}, y = -1\}$ and evaluating its performance on $\mathscr{D}_{\text{val}}$. The SVM solution on the clean data $\mathscr{D}_{\text{tr}}$, and the training data itself, are reported for completeness, highlighting the support vectors (with black circles), the decision hyperplane and the margin bounds (with black lines).

bounded. For the RBF kernel, it also finds a good local maximum of the hinge loss which, incidentally, is the maximum classification error within this area of interest.

5.3.2 Handwritten Digits

We now quantitatively validate the effectiveness of the proposed attack strategy on the MNIST handwritten digit classification task [32, 45], as with the evasion attacks in Section 4.3. In particular, we focus here on the following two-class sub-problems: 7 vs. 1; 9 vs. 8; 4 vs. 0. Each digit is normalized as described in Section 4.3.1. We consider again a linear SVM with $C = 1$. We randomly sample a training and a validation data of 100 and 500 samples, respectively, and retain the complete testing

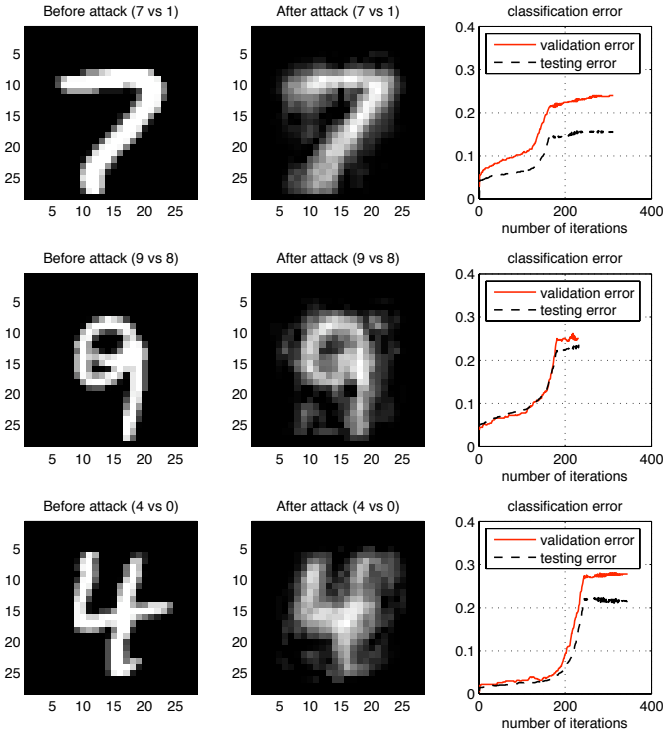

Fig. 8 Modifications to the initial (mislabeled) attack point performed by the proposed attack strategy, for the three considered two-class problems from the MNIST dataset. The increase in validation and testing errors across different iterations is also reported.

data given by MNIST for $\mathscr{D}_{\text{ts}}$. Although it varies for each digit, the size of the testing data is about 2000 samples per class (digit).

Figure 8 shows the effect of *single* attack points being optimized by our descent method. The leftmost plots of each row show the example of the attacked class used as starting points in our algorithm. The middle plots show the final attack point. The rightmost plots depict the increase in the validation and testing errors as the attack progresses. For this experiment we run the attack algorithm 5 times by re-initializing the gradient ascent procedure, and we retain the best result.

Visualizing the attack points reveals that these attacks succeed by blurring the initial prototype to appear more like examples of the attacking class. In comparing the initial and final attack points, we see that the bottom segment of the 7 straightens to resemble a 1, the lower segment of the 9 is rounded to mimicking an 8, and *ovular* noise is added to the outer boundary of the 4 to make it similar to a 0. These blurred images are thus consistent with one's natural notion of visually confusing digits.

The rightmost plots further demonstrate a striking increase in error over the course of the attack. In general, the validation error overestimates the classification

error due to a smaller sample size. Nonetheless, in the exemplary runs reported in this experiment, a *single* attack data point caused the classification error to rise from initial error rates of 2–5% to 15–20%. Since our initial attack points are obtained by flipping the label of a point in the attacked class, the errors in the first iteration of the rightmost plots of Figure 8 are caused by single random label flips. This confirms that our attack can achieve significantly higher error rates than random label flips, and underscores the vulnerability of the SVM to poisoning attacks.

The latter point is further illustrated in a multiple point, multiple run experiment presented in Figure 9. For this experiment, the attack was extended by repeatedly injecting attack points into the same class and averaging results over multiple runs on randomly chosen training and validation sets of the same size (100 and 500 samples, respectively). These results exhibit a steady rise in classification error as the percentage of attack points in the training set increases. The variance of the error is quite high, which can be attributed to the relatively small sizes of the training and validation sets. Also note that, in this experiment, to reach an error rate of 15–20%, the adversary needs to control at least 4–6% of the training data, unlike in the single point attacks of Figure 8. This is because Figure 8 displays the best single point attack from five restarts whereas here initial points are selected without restarts.

5.4 Discussion

The poisoning attack presented in this section, summarized from our previous work in [14], is a first step toward the security analysis of SVM against training data attacks. Although our gradient ascent method is not optimal, it attains a surprisingly large impact on the SVM's classification accuracy.

Several potential improvements to the presented method remain to be explored in future work. For instance, one may investigate the effectiveness of such an attack with surrogate data, that is, when the training data is not known to the adversary, who may however collect samples drawn from the same distribution to learn a classifier's copy (similarly to the limited knowledge case considered in the evasion attacks of Section 4). Another improvement may be to consider the simultaneous optimization of multi-point attacks, although we have already demonstrated that greedy, sequential single-point attacks may be rather successful.

An interesting analysis of the SVM's vulnerability to poisoning suggested from this work is to consider the attack's impact under loss functions other than the hinge loss. It would be especially interesting to analyze *bounded* loss functions, like the ramp loss, since such losses are designed to limit the impact of any single (attack) point on the outcome. On the other hand, while these losses may lead to improved security to poisoning, they also make the SVM's optimization problem non-convex, and, thus, more computationally demanding. This may be viewed as another trade-off between computational complexity of the learning algorithm and security.

An important practical limitation of the proposed method is the assumption that the attacker controls the labels of injected points. Such assumptions may not hold

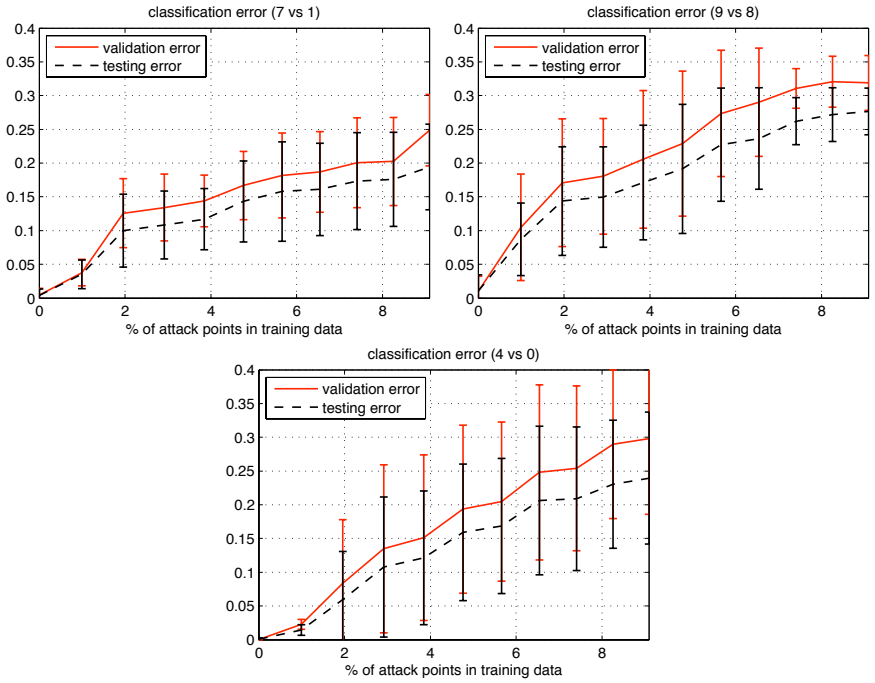

Fig. 9 Results of the multi-point, multi-run experiments on the MNIST dataset. In each plot, we show the classification errors due to poisoning as a function of the percentage of training contamination for both the validation (red solid line) and testing sets (black dashed line). The top-left plot is for the 7 vs.1 task, the top-right plot is for the 9 vs. 8 task, and the bottom-middle plot is for the 4 vs. 0 task.

if the labels are assigned by trusted sources such as humans, *e.g.*, anti-spam filters use their users' labeling of messages as ground truth. Thus, although an attacker can send arbitrary messages, he cannot guarantee that they will have the labels necessary for his attack. This imposes an additional requirement that the attack data must satisfy certain side constraints to fool the labeling oracle. Further work is needed to understand and incorporate these potential side constraints into attacks.

6 Privacy Attacks against SVMs

We now consider a third scenario in which the attacker's goal is to affect a breach of the training data's confidentiality. We review our recent work [59] deriving mechanisms for releasing SVM classifiers trained on privacy-sensitive data while maintaining the data's privacy. Unlike previous sections, our focus here lies primary in the study of countermeasures, while we only briefly consider attacks in the context

of lower bounds. We adopt the formal framework of Dwork *et al.* [30], in which a randomized mechanism is said to preserve β*-differential privacy*, if the likelihood of the mechanism's output changes by at most β when a training datum is changed arbitrarily (or even removed). The power of this framework, which gained near-universal favor after its introduction, is that it quantifies privacy in a rigorous way and provides strong guarantees even against powerful adversaries with knowledge of almost all of the training data, knowledge of the mechanism (barring its source of randomness), arbitrary access to the classifier output by the mechanism, and the ability to manipulate almost all training data prior to learning.

This section is organized as follows. In Section 6.1 we outline our model of the adversary, which makes only weak assumptions. Section 6.2 provides background on differential privacy, presents a mechanism for training and releasing privacy-preserving SVMs—essentially a countermeasure to many privacy attacks—and provides guarantees on differential privacy and also utility (*e.g.*, controlling the classifier's accuracy). We then briefly touch on existing approaches for evaluation via lower bounds and discuss other work and open problems in Section 6.3.

6.1 Modeling the Adversary

We first apply our framework to define the threat model for defending against privacy attacks within the broader context of differential privacy. We then focus on specific countermeasures in the form of modifications to SVM learning that provide differential privacy.

6.1.1 Adversary's Goal

The ultimate goal of the attacker in this section is to determine features and/or the label of an individual training datum. The overall approach of the adversary towards this goal, is to inspect (arbitrary numbers of) test-time classifications made by a released classifier trained on the data, or by inspecting the classifier directly. The definition of differential privacy, and the particular mechanisms derived here, can be modified for related goals of determining properties of several training data; we focus on the above conventional case without loss of generality.

6.1.2 Adversary's Knowledge

As alluded to above, we endow our adversary with significant knowledge of the learning system, so as to derive countermeasures that can withstand very strong attacks. Indeed the notion of differential privacy, as opposed to more syntactic notions of privacy such as *k-anonymity* [64], was inspired by decades-old work in cryptography that introduced mathematical formalism to an age-old problem, yielding

significant practical success. Specifically, we consider a scenario in which the adversary has complete knowledge of the raw input feature representation, the learning algorithm (the entire mechanism including the form of randomization it introduces, although not the source of randomness) and the form of its decision function (in this case, a thresholded SVM), the learned classifier's parameters (the kernel/feature mapping, primal weight vector, and bias term), and arbitrary/unlimited feedback from the deployed classifier (*k.ii-v*). We grant the attacker near complete knowledge of the training set (*k.i*): the attacker may have complete knowledge of all but one training datum, for which she has no knowledge of input feature values or its training label, and it is these attributes she wishes to reveal. For simplicity of exposition, but without loss of generality, we assume this to be the last datum in the training sample.

6.1.3 Adversary's Capability

Like our assumptions on the attacker's knowledge, we impose weak limitations on the adversary's capability. We assume an adversary that can manipulate both training and test data (*c.i*), although the latter is subsumed by the attacker's complete knowledge of the decision function and learned parameters—*e.g.*, she may implement her own classifier and execute it arbitrarily, or she may submit or manipulate test points presented to a deployed classifier.

Our attack model makes no assumptions about the origins of the training or test data. The data need not be sampled independently or even according to a distribution—the definition of differential privacy provided below makes worst-case assumptions about the training data, and again the test data could be arbitrary. Thus the adversary may have arbitrary capability to modify class priors, training data features and labels (*c.ii-iv*) except that the adversary attacking the system may not directly modify the targeted training datum because she does not have knowledge of it. That said, however, differential privacy makes worst-case (no distributional) assumptions about the datum and thus one could consider even this data point as being adversarially manipulated by nature (*i.e.*, nature does not collude with the attacker to share information about the target training datum, but that may collude to facilitate a privacy breach by selecting a "convenient" target datum).

6.1.4 Attack Strategy

While no practical privacy attacks on SVMs have been explored in the past—an open problem discussed in Section 6.3—a general approach would be to approximate the inversion of the learning map on the released SVM parametrization (either primal weight vector, or dual variables) around the known portion of the training data. In practice this could be achieved by taking a similar approach as done in Section 5 whereby an initial guess of a missing training point is iterated on by taking gradient steps of the differential in the SVM parameter vector with respect to the

missing training datum. An interpretation of this approach is one of simulation: to guess a missing training datum, given access to the remainder of the training set and the SVM solution on all the data, simulate the SVM on guesses for the missing datum, updating the guesses in directions that appropriately shift the intermediate solutions. As we discuss briefly in the sequel, theoretical lower bounds on achievable privacy relate to attacks in pathological cases.

6.2 Countermeasures with Provable Guarantees

Given an adversary with such strong knowledge and capabilities as described above, it may seem difficult to provide effective countermeasures particularly considering the complication of abundant access to side information that is often used in publicized privacy attacks [51, 64]. However, the crux that makes privacy-preservation under these conditions possible lies in the fact that the learned quantity being released is an aggregate statistic of the sensitive data; intuitively the more data being aggregated, the less sensitive a statistic should be to changes or removal of any single datum. We now present results from our recent work that quantifies this effect [59], within the framework of *differential privacy*.

6.2.1 Background on Differential Privacy

We begin by recalling the key definition due to Dwork *et al.* [30]. First, for any training set $\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ denote set $\mathscr{D}'$ to be a *neighbor* of $\mathscr{D}$ (or $\mathscr{D}' \sim \mathscr{D}$) if $\mathscr{D}' = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n-1} \cup \{(\mathbf{x}'_n, y'_n)\}$ where $(\mathbf{x}_n, y_n) \neq (\mathbf{x}'_n, y'_n)$. In the present context, differential privacy is a desirable property of *learning maps*, which maps a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ to a continuous discriminant function of the form $g : \mathscr{X} \to \mathbb{R}$—here a learned SVM—in some space of functions, $\mathscr{H}$. We say that a *randomized*[6] learning map $\mathscr{L}$ preserves β-differential privacy if for all datasets $\mathscr{D}$, all neighboring sets $\mathscr{D}'$ of $\mathscr{D}$, and all possible functions $g \in \mathscr{H}$, the following relation holds

$$\Pr(\mathscr{L}(\mathscr{D}) = g) \leq \exp(\beta) \Pr(\mathscr{L}(\mathscr{D}') = g) \ .$$

Intuitively, if we initially fix a training set and neighboring training set, differential privacy simply says that the two resulting distributions induced on the learned functions are point-wise close—and closer for smaller β. For a patient deciding whether to submit her datum to a training set for a cancer detector, differential privacy means that the learned classifier will reveal little information about that datum. Even an adversary with access to the inner-workings of the learner, with access to

[6] That is, the learning map's output is not a deterministic function of the training data. The probability in the definition of differential privacy is due to this randomness. Our treatment here is only as complex as necessary, but to be completely general, the events in the definition should be on measurable sets $G \subset \mathscr{H}$ rather than individual $g \in \mathscr{H}$.

all other patients' data, and with the ability to guess-and-simulate the learning process repeatedly with various possible values of her datum, cannot reverse engineer her datum from the classifier released by the hospital because the adversary cannot distinguish the classifier distribution on one training set, from that on neighboring sets. Moreover, variations of this definition (which do not significantly affect the presented results) allow for neighboring databases to be defined as those missing a datum; or having several varying data, not just a single one.

For simplicity of exposition, we drop the explicit bias term b from our SVM learning process and instead assume that the data feature vectors are augmented with a unit constant, and that the resulting additional normal weight component corresponds to the bias. This is an equivalent SVM formulation that allows us to focus only on the normal's weight vector.

A classic route to establish differential privacy is to define a randomized map $\mathscr{L}$ that returns the value of a deterministic, non-random $\hat{\mathscr{L}}$ plus a noise term. Typically, we use an exponential family in a term that matches an available Lipschitz condition satisfied by $\hat{\mathscr{L}}$: in our case, for learning maps that return weight vectors in $\mathbb{R}^d$, we aim to measure *global sensitivity* of $\hat{\mathscr{L}}$ via the L_1 norm as

$$\Delta(\hat{\mathscr{L}}) = \max_{\mathscr{D},\mathscr{D}' \sim \mathscr{D}} \left\| \hat{\mathscr{L}}(\mathscr{D}) - \hat{\mathscr{L}}(\mathscr{D}') \right\|_1 \ .$$

With respect to this sensitivity, we can easily prove that the randomized mechanism

$$\mathscr{L}(\mathscr{D}) = \hat{\mathscr{L}}(\mathscr{D}) + Laplace(0, \Delta(\hat{\mathscr{L}})/\beta) \ ,$$

is β-differential private.[7] The well-established proof technique [30] follows from the definition of the Laplace distribution involving the same norm as used in our measure of global sensitivity, and the triangle inequality: for any training set $\mathscr{D}$, $\mathscr{D}' \sim \mathscr{D}$, response $g \in \mathscr{H}$, and privacy parameter β

$$\begin{aligned} \frac{\Pr(\mathscr{L}(\mathscr{D}) = g)}{\Pr(\mathscr{L}(\mathscr{D}') = g)} &= \frac{\exp\left(\left\| \hat{\mathscr{L}}(\mathscr{D}') - g \right\|_1 \beta / \Delta(\hat{\mathscr{L}})\right)}{\exp\left(\left\| \hat{\mathscr{L}}(\mathscr{D}) - g \right\|_1 \beta / \Delta(\hat{\mathscr{L}})\right)} \\ &\leq \exp\left(\left\| \hat{\mathscr{L}}(\mathscr{D}') - \hat{\mathscr{L}}(\mathscr{D}) \right\|_1 \beta / \Delta(\hat{\mathscr{L}})\right) \\ &\leq \exp(\beta) \ . \end{aligned}$$

We take the above route to develop a differentially-private SVM. As such, the onus is on calculating the SVM's global sensitivity, $\Delta(\hat{\mathscr{L}})$.

6.2.2 Global Sensitivity of Linear SVM

Unlike much prior work applying the "Laplace mechanism" to achieving differential privacy, in which studied estimators are often decomposed as linear functions of

[7] Recall that the zero-mean multi-variate Laplace distribution with scale parameter s has density proportional to $\exp(-\|\mathbf{x}\|_1/s)$.

Algorithm 3 Privacy-preserving SVM

Input: $\mathscr{D}$ the training data; $C > 0$ soft-margin parameter; kernel k inducing a feature space with finite dimension F and κ-bounded L_2-norm; privacy parameter $\beta > 0$.
Output: learned weight vector $\mathbf{w}$.

1: $\hat{\mathbf{w}} \leftarrow$ learn an SVM with parameter C and kernel k on data $\mathscr{D}$.
2: $\mu \leftarrow$ draw i.i.d. sample of F scalars from $Laplace\left(0, \frac{4C\kappa\sqrt{F}}{\beta}\right)$.
3: **return:** $\mathbf{w} = \hat{\mathbf{w}} + \mu$

data [15], measuring the sensitivity of the SVM appears to be non-trivial owing to the non-linear influence an individual training datum may have on the learned SVM. However, perturbations of the training data were studied by the learning-theory community in the context of *algorithmic stability*: there the goal is to establish bounds on classifier risk, from stability of the learning map, as opposed to leveraging combinatorial properties of the hypothesis class (*e.g.*, the VC dimension, which is not always possible to control, and for the RBF kernel SVM is infinite) [61]. In recent work [59], we showed how these existing stability measurements for the SVM can be adapted to provide the following L_1-global sensitivity bound.

Lemma 1. *Consider SVM learning with a kernel corresponding to linear SVM in a feature space with finite-dimension F and L_2-norm bounded*[8] *by κ, with hinge loss (as used throughout this chapter), and chosen parameter $C > 0$. Then the L_1 global sensitivity of the resulting normal weight vector is upper-bounded by $4C\kappa\sqrt{F}$.*

We omit the proof, which is available in the original paper [59] and which follows closely the previous measurements for algorithmic stability. We note that the result extends to any convex Lipschitz loss.

6.2.3 Differentially-Private SVMs

So far we have established that Algorithm 3, which learns an SVM and returns the resulting weight vector with added Laplace noise, preserves β-differential privacy. More noise is added to the weight vector when either (i) a higher degree of privacy is desired (smaller β), (ii) the SVM fits closer to the data (higher C) or (iii) the data is more distinguishable (higher κ or F—the curse of dimensionality). Hidden in the above is the dependence on n: typically we take C to scale like $1/n$ to achieve consistency in which case we see that noise decreases with larger training data—akin to less individual influence—as expected [59].

Problematic in the above approach, is the destruction to utility due to preserving differential privacy. One approach to quantifying this effect, involves bounding the following notion of utility [59]. We say a privacy-preserving learning map $\mathscr{L}$ has (ε, δ)-utility with respect to non-private map $\hat{\mathscr{L}}$ if for all training sets $\mathscr{D}$,

[8] That is $\forall \mathbf{x}$, $k(\mathbf{x}, \mathbf{x}) \leq \kappa^2$; *e.g.* for the RBF the norm is uniformly unity $\kappa = 1$; more generally, we can make the standard assumption that the data lies within some κ L_2-ball.

$$\Pr\left(\left\|\mathscr{L}(\mathscr{D}) - \hat{\mathscr{L}}(\mathscr{D})\right\|_\infty \leq \varepsilon\right) \geq 1-\delta \ .$$

The norm here is in the function space of continuous discriminators, learned by the learning maps, and is the point-wise L_∞ norm which corresponds to $\|g\|_\infty = \sup_{\mathbf{x}} |g(\mathbf{x})|$—although for technical reasons we will restrict the supremum to be over a set $\mathscr{M}$ to be specified later. Intuitively, this indicates that the continuous predictions of the learned private classifier are close to those predictions of the learned non-private classifier, for all test points in $\mathscr{M}$, with high probability (again, in the randomness due to the private mechanism). This definition draws parallels with PAC learnability, and in certain scenarios is strictly stronger than requiring that the private learner achieves good risk (*i.e.*, PAC learns) [59]. Using the Chernoff tail inequality and known moment-generating functions, we establish the following bound on the utility of this private SVM [59].

Theorem 1. *The β-differentially-private SVM of Algorithm 3 achieves (ε,δ)-utility with respect to the non-private SVM run with the same C parameter and kernel, for* $0 < \delta < 1$ *and*

$$\varepsilon \geq 8C\kappa\Phi\sqrt{F}\left(F + \log\frac{1}{\delta}\right)/\beta \ ,$$

where the set $\mathscr{M}$ supporting the supremum in the definition of utility is taken to be the pre-image of the feature mapping on the L_∞ ball of radius $\Phi > 0$.[9]

As expected, the more confidence δ or privacy β required, the less accuracy is attainable. Similarly, when the training data is fitted more tightly via higher C, or when the data is less tightly packed for higher κ, Φ, F, less accuracy is possible. Note that like the privacy result, this result can hold for quite general loss functions.

6.3 Discussion

In this section, we have provided a summary of our recent results on strong countermeasures to privacy attacks on the SVM. We have shown how, through controlled addition of noise, SVM learning in finite-dimensional feature spaces can provide both privacy and utility guarantees. These results can be extended to certain translation-invariant kernels including the infinite-dimensional RBF [59]. This extension borrows a technique from large-scale learning where finding a dual solution of the SVM for large training data size n is infeasible. Instead, a primal SVM problem is solved using a random kernel that uniformly approximates the desired kernel. Since the approximating kernel induces a feature mapping of relatively small, finite dimensions, the primal solution becomes feasible. For privacy preservation, we use the same primal approach but with this new kernel. Fortunately, the distribution of

[9] Above we previously bounded the L_2 norms of points in features space by κ, the additional bound on the L_∞ norm here is for convenience and is standard practice in learning-theoretic results.

the approximating kernel is independent of the training data, and thus we can reveal the approximating kernel without sacrificing privacy. Likewise the uniform approximation of the kernel composes with the utility result here to yield an analogous utility guarantee for translation-invariant kernels.

While we demonstrated here a mechanism for private SVM learning with upper bounds on privacy and utility, we have previously also studied lower bounds that expose limits on the achievable utility of *any* learner that provides a given level of differential privacy. Further work is needed to sharpen these results. In a sense, these lower bounds are witnessed by pathological training sets and perturbation points and, as such, serve as attacks in pathological (unrealistic) cases. Developing practical attacks on the privacy of an SVM's training data remains unexplored.

Finally, it is important to note that alternate approaches to differentially-private SVMs have been explored by others. Most notable is the work (parallel to our own) of Chaudhuri *et al.* [21]. Their approach to finite-dimensional feature mappings is, instead of adding noise to the primal solution, to add noise to the primal objective in the form of a dot product of the weight with a random vector. Initial experiments show their approach to be very promising empirically, although it does not allow for non-differentiable losses like the hinge loss.

7 Concluding Remarks

In security applications like malware detection, intrusion detection, and spam filtering, SVMs may be attacked through patterns that can either evade detection (evasion), mislead the learning algorithm (poisoning), or gain information about their internal parameters or training data (privacy violation). In this chapter, we demonstrated that these attacks are feasible and constitute a relevant threat to the security of SVMs, and to machine learning systems, in general.

Evasion. We proposed an *evasion* algorithm against SVMs with differentiable kernels, and, more generally, against classifiers with differentiable discriminant functions. We investigated the attack's effectiveness in perfect and limited knowledge settings. In both cases, our attack simulation showed that SVMs (both linear and RBF) can be evaded with high probability after a few modifications to the attack patterns. Our analysis also provides some general hints for tuning the classifier's parameters (*e.g.*, the value of γ in SVMs with the RBF kernel) and for improving classifier security. For instance, if a classifier tightly *encloses* the legitimate samples, the adversary's samples must closely mimic legitimate samples to evade it, in which case, if such exact mimicry is still possible, it suggests an inherent flaw in the feature representation.

Poisoning. We presented an algorithm that allows the adversary to find an attack pattern whose addition to the training set maximally decreases the SVM's classification accuracy. We found that the increase in error over the course of attack is especially striking. A single attack data point may cause the classification error to rise from the initial error rates of 2–5% to 15–20%. This confirms that our attack

can achieve significantly higher error rates than random label flips, and underscores the vulnerability of the SVM to poisoning attacks. As a future investigation, it may be of interest to analyze the effectiveness of poisoning attacks against non-convex SVMs with bounded loss functions, both empirically and theoretically, since such losses are designed to limit the impact of any single (attack) point on the resulting learned function. This has been also studied from a more theoretical perspective in [23], exploiting the framework of Robust Statistics [35, 50]. A similar effect is obtained by using bounded kernels (*e.g.*, the RBF kernel) or bounded feature values.

Privacy. We developed an SVM learning algorithm that preserves *differential privacy*, a formal framework for quantifying the threat of a potential training set privacy violation incurred by releasing learned classifiers. Our mechanism involves adding Laplace-distributed noise to the SVM weight vector with a scale that depends on the algorithmic stability of the SVM and the desired level of privacy. In addition to presenting a formal guarantee that our mechanism preserves privacy, we also provided bounds on the utility of the new mechanism, which state that the privacy-preserving classifier makes predictions that are point-wise close to those of the non-private SVM, with high probability. Finally we discussed potential approaches for attacking SVMs' training data privacy, and known approaches to differentially-private SVMs with (possibly infinite-dimensional feature space) translation-invariant kernels, and lower bounds on the fundamental limits on utility for private approximations of the SVM.

Acknowledgements This work has been partly supported by the project CRP-18293 funded by Regione Autonoma della Sardegna, L.R. 7/2007, Bando 2009, and by the project "Advanced and secure sharing of multimedia data over social networks in the future Internet" (CUP F71J1100069 0002) funded by the same institution. Davide Maiorca gratefully acknowledges Regione Autonoma della Sardegna for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.). Blaine Nelson thanks the Alexander von Humboldt Foundation for providing additional financial support. The opinions expressed in this chapter are solely those of the authors and do not necessarily reflect the opinions of any sponsor.

References

1. Adobe: PDF Reference, sixth edition, version 1.7
2. Balfanz, D., Staddon, J. (eds.): Proceedings of the 1st ACM Workshop on AISec (AISec). ACM (2008)
3. Balfanz, D., Staddon, J. (eds.): Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence (AISec). ACM (2009)
4. Barreno, M., Nelson, B., Joseph, A., Tygar, J.: The security of machine learning. Machine Learning **81**, 121–148 (2010). 10.1007/s10994-010-5188-5
5. Barreno, M., Nelson, B., Sears, R., Joseph, A.D., Tygar, J.D.: Can machine learning be secure? In: ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, pp. 16–25. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1128817.1128824

6. Barth, A., Rubinstein, B.I.P., Sundararajan, M., Mitchell, J.C., Song, D., , Bartlett, P.L.: A learning-based approach to reactive security. IEEE Transactions on Dependable and Secure Computing **9**(4), 482–493 (2012)
7. Biggio, B., Corona, I., Fumera, G., Giacinto, G., Roli, F.: Bagging classifiers for fighting poisoning attacks in adversarial environments. In: the 10th International Workshop on Multiple Classifier Systems (MCS), *Lecture Notes in Computer Science*, vol. 6713, pp. 350–359. Springer-Verlag (2011)
8. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: H. Blockeel et al. (ed.) European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Part III, *Lecture Notes in Artificial Intelligence*, vol. 8190, pp. 387–402. Springer-Verlag Berlin Heidelberg (2013)
9. Biggio, B., Didaci, L., Fumera, G., Roli, F.: Poisoning attacks to compromise face templates. In: the 6th IAPR International Conference on Biometrics (ICB) (2013)
10. Biggio, B., Fumera, G., Pillai, I., Roli, F.: A survey and experimental evaluation of image spam filtering techniques. Pattern Recognition Letters **32**(10), 1436 – 1446 (2011)
11. Biggio, B., Fumera, G., Roli, F.: Design of robust classifiers for adversarial environments. In: IEEE Int'l Conf. on Systems, Man, and Cybernetics (SMC), pp. 977–982 (2011)
12. Biggio, B., Fumera, G., Roli, F.: Security evaluation of pattern classifiers under attack. IEEE Transactions on Knowledge and Data Engineering **99**(PrePrints), 1 (2013)
13. Biggio, B., Fumera, G., Roli, F., Didaci, L.: Poisoning adaptive biometric systems. In: Structural, Syntactic, and Statistical Pattern Recognition, *Lecture Notes in Computer Science*, vol. 7626, pp. 417–425 (2012)
14. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. In: Proceedings of the 29th International Conference on Machine Learning (2012)
15. Blum, A., Dwork, C., McSherry, F., Nissim, K.: Practical privacy: the SuLQ framework. In: Proceedings of the 24th Symposium on Principles of Database Systems, pp. 128–138 (2005)
16. Breiman, L.: Bagging predictors. Machine Learning **24**(2), 123–140 (1996)
17. Brückner, M., Kanzow, C., Scheffer, T.: Static prediction games for adversarial learning problems. Journal of Machine Learning Research **13**, 2617–2654 (2012)
18. Cárdenas, A.A., Baras, J.S.: Evaluation of classifiers: Practical considerations for security applications. In: AAAI Workshop on Evaluation Methods for Machine Learning (2006)
19. Cárdenas, A.A., Nelson, B., Rubinstein, B.I. (eds.): The 5th ACM Workshop on Artificial Intelligence and Security (AISec). ACM (2012)
20. Cauwenberghs, G., Poggio, T.: Incremental and decremental support vector machine learning. In: T.K. Leen, T.G. Dietterich, V. Tresp (eds.) NIPS, pp. 409–415. MIT Press (2000)
21. Chaudhuri, K., Monteleoni, C., Sarwate, A.D.: Differentially private empirical risk minimization. Journal of Machine Learning Research **12**(Mar), 1069–1109 (2011)
22. Chen, Y., Cárdenas, A.A., Greenstadt, R., Rubinstein, B. (eds.): Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence (AISec). ACM (2011)
23. Christmann, A., Steinwart, I.: On robust properties of convex risk minimization methods for pattern recognition. Journal of Machine Learning Research **5**, 1007–1034 (2004)
24. Corona, I., Biggio, B., Maiorca, D.: Adversarialib: a general-purpose library for the automatic evaluation of machine learning-based classifiers under adversarial attacks. URL `http://sourceforge.net/projects/adversarialib/` (2013)
25. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**, 273–297 (1995)
26. Dalvi, N., Domingos, P., Mausam, Sanghai, S., Verma, D.: Adversarial classification. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 99–108 (2004)
27. Dimitrakakis, C., Gkoulalas-Divanis, A., Mitrokotsa, A., Verykios, V.S., Saygin, Y. (eds.): International ECML/PKDD Workshop on Privacy and Security Issues in Data Mining and Machine Learning (2010)
28. Drucker, H., Wu, D., Vapnik, V.N.: Support vector machines for spam categorization. IEEE Transactions on Neural Networks **10**(5), 1048–1054 (1999)

29. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley-Interscience (2000)
30. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Proceedings of the 3rd Theory of Cryptography Conference (TCC 2006), pp. 265–284 (2006)
31. Fogla, P., Sharif, M., Perdisci, R., Kolesnikov, O., Lee, W.: Polymorphic blending attacks. In: Proceedings of the 15th Conference on USENIX Security Symposium (2006)
32. Globerson, A., Roweis, S.T.: Nightmare at test time: robust learning by feature deletion. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 353–360 (2006)
33. Golland, P.: Discriminative direction for kernel classifiers. In: Neural Information Processing Systems (NIPS), pp. 745–752 (2002)
34. Greenstadt, R. (ed.): Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security (AISec). ACM (2010)
35. Hampel, F.R., Ronchetti, E.M., Rousseeuw, P.J., Stahel, W.A.: Robust Statistics: The Approach Based on Influence Functions. Probability and Mathematical Statistics. John Wiley and Sons, New York, NY, USA (1986). URL http://www.worldcat.org/isbn/0471735779
36. Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B., Tygar, J.D.: Adversarial machine learning. In: Proceedings of the 4th ACM Workshop on Artificial Intelligence and Security (AISec), pp. 43–57 (2011)
37. Joseph, A.D., Laskov, P., Roli, F., Tygar, D. (eds.): Dagstuhl Perspectives Workshop on Machine Learning Methods for Computer Security (2012). Workshop 12371
38. Kloft, M., Laskov, P.: Online anomaly detection under adversarial impact. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), pp. 405–412 (2010)
39. Kloft, M., Laskov, P.: A 'poisoning' attack against online anomaly detection. In: Joseph et al. [37]. Workshop 12371
40. Kloft, M., Laskov, P.: Security analysis of online centroid anomaly detection. Journal of Machine Learning Research **13**, 3647–3690 (2012)
41. Kolcz, A., Teo, C.H.: Feature weighting for improved classifier robustness. In: Proceedings of the 6th Conference on Email and Anti-Spam (CEAS) (2009)
42. Laskov, P., Kloft, M.: A framework for quantitative security analysis of machine learning. In: Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, pp. 1–4 (2009)
43. Laskov, P., Lippmann, R. (eds.): NIPS Workshop on Machine Learning in Adversarial Environments for Computer Security (2007)
44. Laskov, P., Lippmann, R.: Machine learning in adversarial environments. Machine Learning **81**, 115–119 (2010)
45. LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Müller, U., Säckinger, E., Simard, P., Vapnik, V.: Comparison of learning algorithms for handwritten digit recognition. In: International Conference on Artificial Neural Networks, pp. 53–60 (1995)
46. Lowd, D., Meek, C.: Adversarial learning. In: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 641–647 (2005)
47. Lowd, D., Meek, C.: Good word attacks on statistical spam filters. In: Proceedings of the 2nd Conference on Email and Anti-Spam (CEAS) (2005)
48. Lütkepohl, H.: Handbook of matrices. John Wiley & Sons (1996)
49. Maiorca, D., Giacinto, G., Corona, I.: A pattern recognition system for malicious PDF files detection. In: MLDM, pp. 510–524 (2012)
50. Maronna, R.A., Martin, R.D., Yohai, V.J.: Robust Statistics: Theory and Methods. Probability and Mathematical Statistics. John Wiley and Sons, New York, NY, USA (2006). URL http://www.worldcat.org/isbn/0471735779
51. Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: Proceedings of the 29th IEEE Symposium on Security and Privacy, pp. 111–125 (2008)
52. Nelson, B., Barreno, M., Chi, F.J., Joseph, A.D., Rubinstein, B.I.P., Saini, U., Sutton, C., Tygar, J.D., Xia, K.: Exploiting machine learning to subvert your spam filter. In: Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats, pp. 1–9 (2008)

53. Nelson, B., Rubinstein, B.I., Huang, L., Joseph, A.D., Lee, S.J., Rao, S., Tygar, J.D.: Query strategies for evading convex-inducing classifiers. Journal of Machine Learning Research **13**, 1293–1332 (2012)
54. Perdisci, R., Gu, G., Lee, W.: Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems. In: Proceedings of the International Conference on Data Mining (ICDM), pp. 488–498 (2006)
55. Platt, J.: Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In: Advances in Large Margin Classifiers, pp. 61–74 (2000)
56. Rizzi, S.: What-if analysis. Encyclopedia of Database Systems pp. 3525–3529 (2009)
57. Rodrigues, R.N., Ling, L.L., Govindaraju, V.: Robustness of multimodal biometric fusion methods against spoof attacks. Journal of Visual Languages and Computing **20**(3), 169–179 (2009)
58. Rubinstein, B.I., Nelson, B., Huang, L., Joseph, A.D., Lau, S.h., Rao, S., Taft, N., Tygar, J.D.: Antidote: understanding and defending against poisoning of anomaly detectors. In: Proceedings of the 9th Conference on Internet Measurement Conference (IMC), pp. 1–14 (2009)
59. Rubinstein, B.I.P., Bartlett, P.L., Huang, L., Taft, N.: Learning in a large function space: Privacy-preserving mechanisms for SVM learning. Journal of Privacy and Confidentiality **4**(1), 65–100 (2012)
60. Schölkopf, B., Herbrich, R., Smola, A.J.: A generalized representer theorem. In: Computational Learning Theory, *Lecture Notes in Computer Science*, vol. 2111, pp. 416–426 (2001)
61. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press (2001)
62. Smutz, C., Stavrou, A.: Malicious PDF detection using metadata and structural features. In: Proceedings of the 28th Annual Computer Security Applications Conference, pp. 239–248 (2012)
63. Šrndić, N., Laskov, P.: Detection of malicious PDF files based on hierarchical document structure. In: Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS) (2013)
64. Sweeney, L.: k-anonymity: a model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems **10**(5), 557–570 (2002)
65. Vapnik, V.N.: The nature of statistical learning theory. Springer-Verlag, Inc. (1995)
66. Wittel, G.L., Wu, S.F.: On attacking statistical spam filters. In: Proceedings of the 1st Conference on Email and Anti-Spam (CEAS) (2004)
67. Young, R.: 2010 IBM x-force mid-year trend & risk report. Tech. rep., IBM (2010)