Università degli Studi di Cagliari

DOTTORATO DI RICERCA

Ingegneria Elettronica ed Informatica

XXX Ciclo

Securing Machine Learning against Adversarial Attacks

Settore scientifico disciplinare di afferenza
ING-INF/05: Sistemi di elaborazione delle informazioni

Presentata da:	**Ambra Demontis**
Coordinatore Dottorato:	**Prof. Fabio Roli**
Tutor:	**Prof. Fabio Roli**
Co-tutor:	**Dott. Ing. Battista Biggio**

Esame finale anno accademico 2016/2017
Tesi discussa nella sessione d'esame Febbraio-Marzo 2018

University of Cagliari

Department of Electrical and Electronic Engineering

Ph.D. in Electronic and Computer Engineering

XXX Cycle

Ph.D. Thesis

Securing Machine Learning against Adversarial Attacks

S.S.D ING-INF/05

Candidate
Ambra Demontis

Advisors
Prof. Fabio Roli
Dott. Ing. Battista Biggio

Ph.D. Coordinator
Prof. Fabio Roli

Final Examination-Academic year 2016/2017
March 2018

Dedicated to my family

Ringraziamenti

Se dovessi commentare con una sola frase il percorso intrapreso direi: "Le strade in salita sono le più faticose ma sono sempre le più belle". Fare ricerca credo sia un mestiere difficile quanto affascinante. Muovere i primi passi in questa strada non è per niente semplice, non solo per le numerose capacità richieste, ma anche perchè, dati i numerosi impegni è necessario saper gestire veramente bene il proprio tempo, senza pensare di poter riuscire a fare tutto subito ma facendo delle scelte e assegnando delle priorità. La difficoltà e la fatica vengono tuttavia senza dubbio ripagate da tutto ciò che si impara e si scopre di volta in volta. Durante questi tre anni credo di aver imparato tanto, non solo riguardo agli argomenti di ricerca e al fatto che tra i vari impegni bisogna inserire anche un numero di ore sufficienti di sonno, ma anche su come affrontare la vita e le difficoltà. Questa crescita sono sicura non sarebbe mai avvenuta senza il supporto e l'esempio delle persone che, giunta al termine di questo dottorato mi sembra doveroso ringraziare.

Prima di tutto vorrei ringraziare il ricercatore Battista Biggio, che mi ha seguita da vicino durante questo percorso spendendo gran parte del suo tempo per insegnarmi tutto il necessario per fare ricerca, dalle soft skills necessarie per presentare il proprio lavoro in forma orale e scritta alle skill più tecniche come scrivere del codice ben ingegnerizzato usabile e riutilizzabile. Vorrei ringraziarlo per essere stato sempre presente per consigliarmi, correggermi e discutere idee di ricerca anche durante le festività e a prescindere dalla parte del mondo nella quale si trovasse. Vorrei ringraziarlo, inoltre, per avermi spronanta con le sue parole a non arrendermi e con il suo esempio a cercare sempre di dare il massimo.

Vorrei ringraziare in particolare anche il prof. Fabio Roli, non solo per il suo prezioso supporto tecnico ma anche perchè, nonostante i suoi numerosissimi impegni, è sempre stato disponibile, ogni qual volta rilevassi una particolare difficotà in qualcosa, a parlarne. In quelle occasioni mi ha dato sempre illuminanti consigli su come affrontare e superare i problemi, e anche interessantissime spiegazioni su qualcuno dei numerossissimi argomenti di suoi interesse (economia, psicologia ecc..).

Ringrazio entrambi perchè è stato decisamente un piacere e onore poter lavorare con loro e li ringrazio inoltre per aver creduto in me.

Mi sembra doveroso ringraziare anche gli altri membri del PRALab che durante questo percorso mi hanno fornito il loro supporto. Ringrazio il Prof. Giorgio Fumera e il Prof. Giorgio Giacinto per il loro supporto tecnico e in particolare Prof. Fumera

per tutte le sue spiegazioni sulle tecniche di insegnamento. I post doc del gruppo Luca, Igino e in particolare Davide Ariu e Davide Maiorca per i numerosissimi consigli che mi hanno dato durante il corso di questo dottorato. I colleghi Marco e Paolo con i quali abbiamo lavorato a diversi articoli. Guido per alcuni utili consigli. Il Responsabile della disseminazione scientifica del gruppo Matteo tutto il suo aiuto nella preparazione del materiale pubblicitario di diverse attività e nell'organizzazione di un concorso per gli studenti delle scuole superiori. Il sistemista del gruppo Mauro per tutto il suo aiuto nella realizzazione del sito della IEEE Student Branch. Elena per il suo aiuto nell'organizzazione di attività per gli studenti. Gli altri officers della nostra IEEE Student branch Matteo, Elizavet, Arslan Michele, Graziana e Andrea Pinna che ha contribuito all'organizzazione di attività della branch. Gli altri studenti di dottorato Mohammad, Mansur, Mariam, Farideh, Bahram. La segretaria del gruppo Carla. Sono grata inoltre al Prof. Gavin Brown per avermi dato la possibilità di svolgere una intership presso la sua università e per i suoi insegnamenti. Vorrei ringraziare inoltre i ragazzi dell'MLO group, in particolare Emily, Sarah, Kostas Sechidis, Nicos, Henry, Georgiana, Tameem, Dongjiao Ainura e Joe. Oltre a professori e colleghi che mi hanno fornito il loro suppporto durante il dottorato, vorrei ringraziare anche il mio primo insegnante di informatica, Gianfranco Ciaschetti per avermi fatto appassionare all'informatica e per avermi trasmesso l'amore per la ricerca. La mia insegnante Giuliana Corrias che mi ha sempre fatto sentire la sua vicinanza anche nei periodi nei quali non sono riuscita a frequentare la sala di danza. Gli amici, in particolare le mie migliori amiche Serena e Guendalina perchè hanno sempre saputo capirmi, supportarmi, consigliarmi e aiutarmi ad analizzare le cose da angolazioni differenti. Ringrazio i miei familiari. In particolare mio didino Alessio che fin da piccolissima mi ha fatta avvicinare ai computer. Mia nonna che mi ha sempre dato preziosissimi consigli. Loredana, Mariano, Luisa, Nadia e la piccola Tamara per la loro vicinanza. Infine, un ringraziamento speciale va ai miei genitori, dai quali ho ereditato la curiosità, che fin da piccola mi hanno insegnato a seguire le mie passioni e ad impegnarmi sempre per raggiungere i miei obbiettivi.

Grazie di cuore

Ambra

Ambra Demontis gratefully acknowledges Sardinia Regional Government for the financial support of her PhD scholarship (P.O.R. Sardegna F.S.E. Operational Program of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.).

Abstract

Machine learning techniques are nowadays widely used in different application domains, ranging from computer vision to computer security, despite it has been shown that they are vulnerable to well-crafted attacks performed by skilled attackers. These include evasion attacks aimed to mislead detection at test time, and poisoning attacks in which malicious samples are injected into the training data to compromise the learning procedure. Different defenses have been proposed so far. However, the majority of them is computationally expensive, and it is not clear under which attack conditions they can be considered optimal. There is moreover a lack of a security evaluation methodology that allows comparing the security of different classifiers. This thesis aims to provide a contribution to the study of machine learning system security. Through this thesis, we firstly provide an adversarial framework that can help us to perform the security evaluation of different classifiers. We exploit this provided tool to assess the security of different machine learning systems, focusing our attention on systems with limited hardware resources. Thanks to this analysis we discover an interesting relationship between sparsity and security. Then, we propose a poisoning attack that, respect to the state-of-art ones, can be exploited against a broad subset of classifiers (neural network included). Finally, we provide theoretically well-founded and efficient countermeasures, demonstrating their effectiveness on two case studies involving Android malware detection and robot vision.

Contents

Notation

d Number of sample features

$\boldsymbol{x}$ Sample, is a vector $1 \times d$

y Sample class

n Number of samples

$\mathcal{X}$ Matrix with shape $n \times d$ where each row represent a sample

$\boldsymbol{y}$ Sample classes, is a vector with dimension $1 \times n$

$\mathcal{D}$ Dataset, $\mathcal{D} = (\mathcal{X}, \boldsymbol{y})$

n_c Number of different dataset classes

$\mathcal{D}_{tr}$ Training Dataset

$\mathcal{D}_{val}$ Validation Dataset

$\mathcal{D}_{ts}$ Test Dataset

f Classifier Discriminat Function

ε Small positive constant for ensuring algorithm convergence

b Classifier bias

$\boldsymbol{w}$ Classifier Parameters

L Classifier Loss Function

d Distance between two samples in a specified ℓ_p space

$\|\boldsymbol{x}\|_p$ Norm in ℓ_p space

$\|\boldsymbol{x}\|_p^*$ Dual Norm in ℓ_p space

$\|\boldsymbol{x}\|_{8gon}$ Octagonal norm

$\mathcal{U}$ Uncertainty set

$\boldsymbol{u}$ One of possible dataset perturbations. Is a vector $1 \times d$

$:=$ Equal for definition

K Kernel function

α Support Vector coefficient

$\boldsymbol{\alpha}$ Vector containing Support Vector coefficients

C Support Vector Machine Costs

ϕ Feature extraction/selection function

M Learning Algorithm

Θ Adversary's Knowledge Space 4.2

$\boldsymbol{\theta}$ Adversary's Knowledge $\theta = (\mathcal{D}, \phi, M, L, f, \boldsymbol{w})$ 4.2

$\boldsymbol{x}_{lb}$ Vector containing the minimum feature values that each feature can assume

$\boldsymbol{x}_{ub}$ Vector containing the maximum feature values that each feature can assume

$\preceq$ Element-wise $\leq$ operator (where $\boldsymbol{u} \preceq \boldsymbol{v}$ means that each element of $\boldsymbol{u}$ has to be not greater than the corresponding element in $\boldsymbol{v}$)

d_{max} Maximum amount of perturbation between two samples measured with a chosen $\ell_p - norm$

Π Projection operator. Function that project a given sample $\boldsymbol{x}$ inside the feasible space defined by the specified constraints

(PK) Attack Perfect Knowledge Attack 4.2.1

(LK) Attack Limited Knowledge Attack 4.2.2

(LK-SD) Attack Limited Knowledge Attack with Surrogate Data 4.2.2

(LK-SL) Attack Limited Knowledge Attack with Surrogate Learner 4.2.2

$\mathcal{D}_a$ Adversary's attack samples 4.4

Ψ Feasible transformations 4.3

A Adversary's loss function 4.4

$\mathcal{L}$ Loss function defined from the attacker in order to poison the classifier 4.4

k True class for Error-generic Evasion 5.1.1 or target class for Error-specific Evasion Attack 5.1.2

Chapter 1

Introduction

While a decade ago Internet and operative systems were used mainly from technicians, nowadays, because of smart-phones and the so-called Internet of things devices, each of us has always one or more of them in his pocket. As a consequence, a great deal of public administrations and companies have started to provide online services. The number of data produced daily has, therefore, grown exponentially. If non-sensitive data of a single person may be useless, this huge amount of data can be analyzed and transformed into valuable information. Machine Learning, an ensemble of techniques that enables machines to learn from data has gained, therefore, popularity in different application domains. One of those domains is Information Security where machine learning is used to detect attacks of different types (like malware or network intrusion). However, has been shown that Machine Learning algorithms can be easily misled by a skilled attacker. To solve this problem, the scientific community started to study the security of Machine Learning system giving rise to a field of study called Adversarial Machine Learning. As this thesis aims to give a contribution to this field, in this Chapter we provide a brief introduction to Machine Learning techniques and to two of the main problems that are being studied from the Adversarial Machine Learning Community.

1.1 Machine Learning

Machine Learning techniques are extremely useful as allows us, given a bunch of data, to find an answer to problems for which we are not able to write down an algorithm. To understand how machine learning works we can consider a simple problem solvable with those techniques. Suppose that you are asked to tell, given an object, that can be either, a screw or a hammer to which class it belongs to. This is a simple classification problem. The first step to create a machine-learning system that can distinguish between two different classes of objects is to choose some discriminant characteristics (*features*) that can be used to distinguish between them. As hammers are always considerably bigger than screw, two possible discriminant

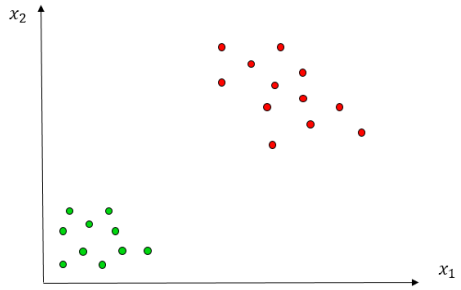


Figure 1.1: Dataset of screw and hammers in feature space.

features are length and width. You can measure them and represent every single object with a vector $\boldsymbol{x} = (x_1, ..x_d)$ where each vector element is the measurement of one of the features that you decided to take into account. Machine-learning systems learn to classify objects relying on a set of examples. You will collect therefore this measurements from many hammers and screw. With this data, you can construct a training dataset that is made up of a set of couples each one composed of a object measurements and label. The object label y is dependent on the class in which the object belongs i.e. hammer or screw. In a two-class classification problem to each class is usually assigned a number $y \in \{-1, 1\}$. We can, therefore, label the screw as -1 and the hammer as 1. Our training dataset can, therefore, be represented as $D = (X, \boldsymbol{y})$ where X is a matrix with a number of rows n equal to the number of objects considered in our dataset and a number of column d equal to the considered features. $\boldsymbol{y}$ is a vector $1 \times n$ that contains the object labels. We can visualize our dataset representing it on a plane as in Fig. 1.1 where screws are represented with green points and hammers with red points.

Your goal is then to infer a discriminant function f that you can use to predict the object class $y = +1$ if $f(\boldsymbol{x}) \geq 0$, -1 otherwise. You can make different assumptions on the shape of the function needed to separate the objects that belong to the two different classes. Let's assume that they are separable with a hyperplane. To create your classifier, you need to find the parameters of a linear function:

$$f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b\,, \tag{1.1}$$

where $\boldsymbol{w} \in \mathbb{R}^{\mathsf{d}}$ denotes the vector of *feature weights*, and $b \in \mathbb{R}$ is the so-called *bias*.

Clearly, you will search for a hyperplane that allows you to make few *errors* namely, classify few objects in a wrong way. However, you can find many different hyperplanes that are able to separate your data making almost the same number of errors. How can you choose a good hyperplane between them? Let's assume that you are asked to choose between the two hyperplanes that are shown in Fig. 1.2. With the hyperplane that is shown on the right, an object should be pretty different from the known objects to be wrongly classified. With the hyperplane that is shown on the left, instead, a hammer that is slightly little than the ones that you have

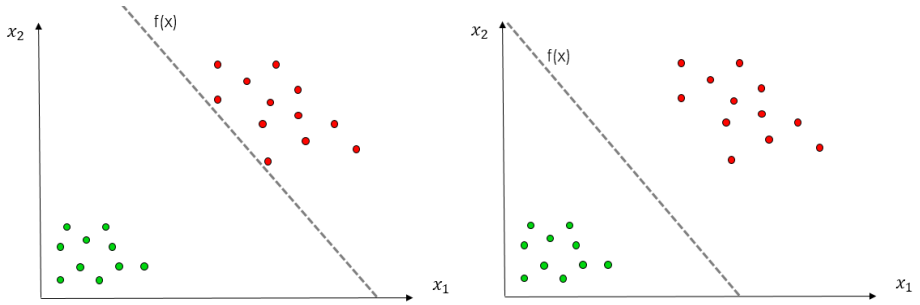


Figure 1.2: Different classification hyperplanes.

measured will be misclassified. It is said that the hyperplane on the right has a bigger *generalization capability* than the one on the left, as it is more probably able to correctly classify objects that are not between the ones that you used to learn the classifier parameters (your training dataset). To find a good hyperplane taking into account both these properties, namely the number of committed error and the generalization capability, you can define an objective function that is made up of two different terms e.g.:

$$\min_{\boldsymbol{w},b} \mathcal{L}(\mathcal{D}, f) = \underbrace{\tfrac{1}{2}\boldsymbol{w}^\top \boldsymbol{w}}_{R(f)} + C \underbrace{\sum_{i=1}^{\mathsf{n}} \max(0, 1 - y_i f(\boldsymbol{x}_i))}_{L(f,\mathcal{D})} \tag{1.2}$$

where $L(f, \mathcal{D})$ denotes a loss function that measure the number of errors committed by f on the samples in D. $R(f)$ is a regularization term to avoid overfitting (i.e., to avoid that the classifier overspecializes its decisions on the training data, losing generalization capability on unseen data), and C is a trade-off parameter. In particular, this objective function is the one used to learn a classifier called Support Vector Machine (SVM). It exploits an ℓ_2 regularizer on the feature weights and the so-called *hinge loss* as loss function. You can find the hyperplane parameters ($\boldsymbol{w}, b$ of Eq. (1.1)) optimizing the objective function in Eq. (1.2). This operation is called *training*. This learned hyperplane allows you to predict, after the training (test time) the labels of objects that you have never seen before (test samples).

1.2 Adversarial Machine Learning

Machine-learning algorithms have been increasingly applied in security-related tasks, in response to the increasing variability and sophistication of attacks [8, 88, 1, 4, 62] as their ability to *generalize* allows one to detect never-before-seen attacks or variants of known ones. However, as first pointed out by Barreno et al. [7, 6], machine-learning algorithms have been designed under the assumption that training and test data follow the same underlying probability distribution, which makes them vulnerable to well-crafted attacks violating this assumption. This means that

machine learning itself can be the weakest link in the security chain [3]. The field of study whose aim is asses machine learning system security is called *Adversarial Machine Learning*. Among the different attacks that can threaten a machine learning system the two that have been mostly considered by the adversarial machine learning community are named *evasion* and *poisoning*. We provide a brief explanation of them below.

1.2.1 Evasion

Mainly but not exclusively in security-related applications, some users can be interested to have the samples that they submit to the system misclassified. An application example is malware detection where an attacker would like to have her malware misclassified by the system as benign application.[1] To mislead the system she can exploit her knowledge or make guess about the system structure and carefully modify her malware accordingly. This attack is called *evasion* and has been, to date, the most studied one. Different evasion attacks have been devised against almost all machine learning systems, neural network [12] and Deep Neural Network [102] included.

1.2.2 Poisoning

Among the different attack scenarios envisaged against machine learning, *poisoning attacks* are considered one of the most relevant and emerging security threats for data-driven technologies, i.e., technologies relying upon the collection of large amounts of data in the wild [54]. In a poisoning attack, the attacker is assumed to control a fraction of the training data used by the learning algorithm, with the goal of subverting the entire learning process, or facilitate subsequent system evasion [81, 93, 19, 108, 73, 57]. Usually, an attacker can not have access to the training dataset, however, she can often provide new training data. Honeypots, for example, often collect malware training samples, which provides an opportunity for the adversary to poison the training data. Another example are applications that rely on user feedback for improving their performance as it can be intentionally wrongly provided to mislead the system. To date, this attack has been devised only for a little subset of machine learning systems as we will discuss more in detail later on in this thesis.

1.3 Outlook of this Thesis

In this Chapter Machine Learning and Adversarial Machine Learning fields have been introduced.

[1] We refer to the attacker here as feminine due to the common interpretation as is commonly named “Eve” or “Carol” in cryptography and security.

In Chapter 2 Machine Learning Systems that are used in this thesis are briefly summarized. The state-of-art defenses against some of the attacks that can threaten them are presented and their limitations are discussed.

The contributions that this thesis provides to the state of the art are discussed in Chapter 3.

In order to provide a methodology to evaluate and compare the security of different machine learning systems an adversarial attack framework is presented in Chapter 4.

In Chapter 5 two different defenses aimed to improve the security respectively of binary linear and multiclass classifiers are presented. Moreover, the relationship between the number of features on which a machine learning system rely on during classification and security is analyzed.

A new poisoning strategy devised for a broad subset of machine learning systems compared to the state of the art is presented in Chapter 6 together with a new countermeasure for kernel machines against this attack.

Two case studies where the proposed defenses are used to enhance respectively the security of an Android Malware detector and a Robot-vision system are finally presented respectively in Chapter 8 and in Chapter 9.

Chapter 2

Background

In this Chapter, we provide an overview of the machine learning systems that are considered through this thesis. Then we briefly review the state-of-art defenses against *evasion* and *poisoning* attacks. Finally, we highlight which are, to date, the main limitations and open issues concerning Machine Learning Security.

2.1 Machine Learning Systems

There are many problems for which machine learning techniques have been devised e.g.. Clustering, Regression, Classification. In this thesis, we focus on classification problems. In this section, we give a brief overview of the classification algorithms that we consider through this thesis.

2.1.1 Support Vector Machines

Support Vector Machine [29] is one of the most used classifiers as it is less computationally expensive than many other classifiers and allows to obtaining a high accuracy in a great deal of classification problems. As we have seen in the previous chapter, its objective function (eq. 1.2)) is made up by an ℓ_2 regularizer and the so-called *hinge loss*. It learns a hyperplane in feature space, therefore it will have d parameters (equal to the number of features). This is the so called *primal form* of the linear SVM.

Dual SVM

It has been shown that you can equivalently solve the dual problem:

$$\max_{\alpha} \quad -\frac{1}{2}\sum_{i,j=1}^{n} y_i y_j \boldsymbol{x_i}\boldsymbol{x_j} + \sum_{i=1}^{n}\alpha_i \tag{2.1}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C\,, \tag{2.2}$$

$$\sum_{i=1}^{n} y_i \alpha_i = 0\,, \tag{2.3}$$

The decision function of the dual svm is $f(\boldsymbol{x}) = \sum_i^n \alpha_i y_i (\boldsymbol{x_i}'\boldsymbol{x}) + b$. This formulation is useful when the number of features is consistent as it has n parameters, where n is equal to the number of training samples. A considerable number of α coefficients are, moreover, usually equal to zero.

Kernel trick

When the sample distributions are highly not linear a linear svm has often poor performance. A solution is therefore to map the points in a different feature space where they are easily separable. If this space is composed by many feature this operation as well as the optimization of the classifier is computationally expensive. The dual form of the SVM allows the so called kernel trick, namely to learn the classifier in a high dimensional feature spaces, without having to map those points in that space. Basically a chosen function called *Kernel* (K) which carry out the feature mapping is applied to the scalar product between samples $K(x_1, x_2)$. One of the most famous kernel is the Radial Basis Function (RBF) kernel as it allows to map the samples in an infinite dimensional features space.

$$K(x_1, x_2) = exp(\frac{-||x_1 - x_2||^2}{2\sigma^2}) \tag{2.4}$$

2.1.2 Neural Networks

Neural networks [22] are classifiers inspired to the human brain. They are composed by neurons. Each neuron receives different inputs. Those inputs are weighted and summed. A function called *activation function* is applied to the sum result. A set of neurons compose a layer of the network and a network is often composed by many layers: an input layer, one or many hidden layers and an output layer. When neural networks are used to solve binary problems the last layer is composed by a single neuron and the sample is classified as belonging to the positive class if the neuron output is bigger than 0.5, to the negative class elsewhere. When the neural network is instead used to solve multiclass problems the last layer is composed by

a number of neurons equal to the number of classes. In this case the predicted class is the one for which the corresponding neuron output the maximum scores. The weights of the network are randomly initialized and during the training they are updated using a gradient based learning algorithm to reduce the classification error committed by the network. The gradient of the loss respect to each weight is required by the algorithm. To this end the backpropagation algorithm is exploited. It basically exploits the chain rule computing the gradient of a neuron as a weighted composition made up by the derivative of the node in the previous layers.

Convolutional Neural Networks

Convolutional neural networks are deep neural networks (networks with more than one hidden layer) inspired to the animal visual cortex. The neurons of the visual cortex were shown to be activated from different stimuli (eg. Horizontal or vertical bars). This mechanism is imitated from the Convolutional neural network using two operations called convolution and pooling. Those classifiers have recently shown groundbreaking performance in computer vision applications.

2.1.3 Multiclass Classification

When the number of classes is higher than two one can use different approaches. One of them is to use a multiclass classifier as Neural Network. The other is to combine binary classifiers. One of the possible strategies is called One Versus All.

One-Versus-All (OVA). The one-versus-all scheme combines a set of c binary classifiers, being c the number of known classes. If we denote the discriminant functions of the aforementioned binary classifiers as $f_1(\boldsymbol{x}), \ldots, f_{\mathsf{c}}(\boldsymbol{x})$. The predicted class of the one-versus-all classifier $c^\star$ for a sample $\boldsymbol{x}$ is determined as the class whose discriminant function for that sample is maximum:

$$c^\star = \arg\max_{k=1,\ldots,\mathsf{c}} f_k(\boldsymbol{x})\,. \tag{2.5}$$

2.1.4 Sparse Machine Learning

Sparse Machine learning is referred to a collection of learning techniques that tries to minimize the amount of data that is used from the machine learning systems to makes decisions. As we have seen before, if trained using the Dual formulation SVM learns a sparse solution where sparsity is referred to the number of zero's α and therefore to the number of training samples considered. If SVM is learned in the primal, it can be enforced to learn a sparse solution, where sparsity is referred to the number of features used to make decisions [117], changing the ℓ_2 regularizer with one that enforces sparsity. Some sparse classifiers that can be obtained substituting the regularizer are:

1-Norm SVM (1-norm).:

$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|_1 + C\sum_{i=1}^{\mathsf{m}} (1 - y_i g(\boldsymbol{x}_i))_+ \ . \tag{2.6}$$

The ℓ_1 regularizer induces $\boldsymbol{w}$ sparsity, while retaining convexity and linearity.

Elastic-net SVM (el-net). The elastic-net regularizer [118] allows, combined with the hinge loss to obtain an SVM formulation with tunable sparsity:

$$\min_{\boldsymbol{w},b} \quad (1-\lambda)\|\boldsymbol{w}\|_1 + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{\mathsf{m}} (1 - y_i g(\boldsymbol{x}_i))_+ \ . \tag{2.7}$$

The sparsity level can be tuned through the trade-off parameter $\lambda \in (0, 1)$.

2.2 Security Measures for Machine Learning

Different defenses aimed to counter evasion and poisoning attacks have been proposed to date. In this section we briefly summarize the main idea on which they are based.

2.2.1 Defenses against Evasion Attacks

To account for this potential adversarial drift between training and testing distributions, various adversary-aware learning algorithms have been developed, based on robust optimization, probabilistic and game-theoretical models (see, e.g., [28, 46, 105]). The underlying idea of these algorithms is to incorporate knowledge of the potential adversarial data manipulations into the learning phase. Game-theoretical models simulate such manipulation at training time. Basically there are two players, an attacker and a defender (the classifier). The attacker goal is to produce samples that increase the classifier error. The classifier goal is try to decrease the error on both, the original training samples and the ones produced by the attacker. Robust optimization and probabilistic models include, instead, the distribution drift directly into the model objective function. In practice, both options reflect a similar effect, i.e., an adversarial shift of the malicious distribution, as witnessed by the aforementioned probability model. The only difference is the level at which assumptions on the attacker model are made, i.e., either at the level of each malicious sample, or at the higher level of their global probability distribution. Clearly, making assumptions at the sample level allows one to more finely define the potential adversarial data manipulations, which can be advantageous when application-specific constraints on data manipulation can be accounted for. Another line of defense [76] is supporting, instead, the main classifier with another one that is called *detector* as it is learned with the purpose to detect attacks. If it recognizes that the sample is malicious it is

classified as adversarial, otherwise, it is classified as belonging to the class predicted from the main classifier. Secure learning techniques based on the aforementioned approaches tend to exhibit a much higher training complexity compared to the corresponding non-secure version, especially in terms of computational time and space. This is one of the main factors that hinder the adoption of these algorithms in practice, along with the difficulty of meeting some theoretical requirements, and, in some cases, the complexity of their implementation.

2.2.2 Defenses against Poisoning Attacks

In order to enhance the security of machine learning systems against poisoning attacks, different defenses have been proposed so far. The main idea on which the proposed defences are based is that poisoning points are outlier. The main motivation behind this idea is that adversary aims to "deviate" the classification algorithm from learning the correct training data distribution, therefore if poisoning points were not outlier, their effect would be negligible. They counter therefore the poisoning problem with outlier removal techniques [34, 64]. Namely, they try to identify and remove the training points that are far from the average of the point belonging to each class distribution. Those approaches are therefore vulnerable to stealthy poisoning point attacks where the attacker inject poisoning point that are not so different from the original training samples points. Another line of defense tries to decrease the influence of each single point on the classifier decision. In [13] bagging, a well-known ensemble construction method where each classifier in the ensemble is trained on a different bootstrap replicate of the training set is shown to reduce the outlier influence. In [18], instead, during the learning it is assumed that the label of each training sample can be independently flipped with the same probability. Despite having demonstrated empirically to increase the classifier security, being heuristic method they do not provide guarantee of optimality.

2.3 Limitation and Open Issues

Even though the research in this field has been carried on from almost ten years, there is still a huge number of open problems due to the variability of the machine learning system characteristics and of the attacks that can threaten them.

The recent increasing interest about this field with the consequent enlargement of the machine learning community is producing a significant boost on the number of countermeasures (mainly devised to counter Neural Network vulnerabilities) that is published daily. Among those recently published works there is, however, a lack of methodology to evaluating the efficacy of the proposed countermeasures whose security is therefore barely comparable. Apart from the used methodology shortcoming, the instruments that could aid this evaluation are still missing for some machine learning systems. The first step to evaluate machine learning system security is

to understand which attacks can threaten the system. A framework that enable one to envision different attacks scenarios against a machine learning system were proposed in previous works. However, in the state of the art framework does not comprise multiple classifiers poisoning. Once the the attack scenarios are clear, the security of the system under each attack scenario has to be evaluated. The studies about formal techniques that allow one to quantify the security of machine learning systems are still ongoing, therefore as we will explain later on in this thesis, the security evaluation of a machine learning system requires, to date, to simulate strong attacks against it. Observing the behavior of the system under an attack scenario help, infact, understanding on which extent it is vulnerable to that specific attack scenario. Attacks are therefore a prerequisite to the security evaluation. While is already well-known how to exploit evasion attacks against the majority of the machine learning systems, only a subset of them can be poisoned using state-of-art techniques.

Machine learning techniques are nowadays often applied in devices with limited hardware resources. Such systems exploit often sparsity to be more efficient. To date, however, has not been investigated if sparsity increases to some extent system vulnerabilities.

Another limitation to the state of art is related to the countermeasures that have been proposed so far. The majority of them is computational expensive. Moreover, a full comprehension of the conditions under which those defenses can be effective is still missing.

Chapter 3

Contributions of this thesis

We give below a brief overview of the contributions that are provided through this thesis to the Adversarial Machine Learning field.

The first contribution is to provide tools that can aid to evaluate the security of machine learning systems. The first step to create secure machine learning systems is understanding how attackers can threaten them. A framework that allow one to envision the attacks that can be perpetrated against binary classifiers was proposed in previous works. As the multiclass classifier case open new possible goal for an attacker, in this thesis, we extend it to consider possible attacks against multiclass classifier. We then describe a clear methodology that can be used to compare the security of different machine learning systems. As formal verification techniques are still far from being applicable to real system, strong attacks are needed to carry on the system security evaluation. While evasion attacks have been devised against the majority of machine learning systems, the state of art poisoning strategy is exploitable only against a subset of binary machine learning systems. It is moreover really computational expensive. We provide an efficient poisoning attack strategy applicable to a broader set of machine learning algorithms, including multiclass classifiers that can be trained with gradient-based procedures, like deep neural networks. The second main contribution is an analysis of the relationship between sparsity and security. Sparsity is a desirable property for a machine learning system as it reduces the computational complexity allowing algorithms to be executed on hardware with limited capability and while providing more interpretable decisions. However, its impacts on the security of machine learning systems has been, to the best of our knowledge, never questioned before.

Finally, exploiting the tools that we provide to evaluate machine learning security we analyze different systems understanding the causes of their vulnerabilities and providing well-founded and efficient defenses. Those defenses are shown to be applicable to improve the security of existent systems with limited hardware resources on two case studies regarding Android malware detection and robot vision.

This thesis is based on the following publications to which I contributed during my Ph.D: [41], [40], [37], [94], [80], [74]

Moreover, during my Ph.D, I collaborated to these publications that are however unrelated to this thesis: [39], [38]

Chapter 4

Adversarial Attack Framework

How to evaluate and compare the security of multiclass classifiers is at the state of the art still an open problem. In this section, how this goal can be reached is explained. The first step to perform a security evaluation is to understand which are the possible attacks that could threaten the system. A framework which enables one to envision different attack scenarios against learning algorithms and to craft the corresponding attack samples is here proposed. Remarkably, this includes attacks at training and at test time, usually referred to as poisoning and evasion attacks [52, 17, 12, 19, 108, 73] or, more recently, as adversarial (training and test) examples (when crafted against deep learning algorithms) [102, 84, 83]. This framework is based on the ones originally proposed in [7, 6, 52] and subsequently extended in [17]. It characterizes the attacker according to her goal, knowledge of the targeted system, and capability of manipulating the input data. Based on these assumptions, it allows one to define an optimal attack strategy as an optimization problem whose solution amounts to the construction of the attack samples. This framework, originally developed for binary classification problems, is here extended to multiclass classification. How to exploit this framework to perform the system security evaluation if finally explained.

4.1 Attacker's Goal

If the attacker goal is to induce some error the attack can be classified depending on *Attack Specificity* and in multiclass classifier also depending on *Error Specificity*. Moreover, the attacker's goal may cause different security violations. These characteristics are below explained in detail.

Attack Specificity. This characteristic ranges from *targeted* to *indiscriminate*, respectively, if the attack aims to cause misclassification of a specific set of samples (to target a given system user or protected service), or of any sample (to target any system user or protected service).

Error Specificity. We introduce here this characteristic to disambiguate the notion of misclassification in multiclass problems. The error specificity can thus be: *specific*,

if the attacker aims to have a sample misclassified as a specific class; or *generic*, if the attacker aims to have a sample misclassified as any of the classes different from the true class.[1]

4.1.1 Security Violation

This characteristic defines the high-level security violation caused by the attacker, as it is normally done in security.

Privacy Violation. The attacker may be interested in inferring confidential information about the system or its resources. Although the given framework encloses also this kind of attacks they will not be debated through this thesis. The security analysis of attacks in which the attacker aims to discover sensitive information about one or a group of training samples is covered by the differential privacy field whose aim is to understand how to reveal distributional information about a private data set, without revealing too much about any single individual in the dataset [44].

Integrity Violation. When machine learning is used for security or safety-related applications the goal of the attacker is often to make the classifier misclassifying some samples. e.g. The aim of an identity theft is being recognized as her victim. She may try therefore to modify the system in order to reach her goal, violating its integrity.

Availability Violation. The attack is configured as an availability violation if the error induced by an attacker is so consistent that the system is no more able to provide a sufficient number of correct results to be useful. e.g. A spammer would be satisfied if it is able to modify the spam recognition system like that it is not able to recognize any spam email.

4.2 Attacker's Knowledge

The attacker can have different levels of knowledge of the targeted system, including: $(k.i)$ the training data $\mathcal{D}_{tr}$; $(k.ii)$ the feature extraction/selection algorithm ϕ; $(k.iii)$ the information related to the classifier namely the learning algorithm M, along with the objective function L minimized during training and the decision function f ; and, possibly, $(k.iv)$ its (trained) parameters $\boldsymbol{w}$. The attacker's knowledge can thus be characterized in terms of a vector in a space Θ that encodes the aforementioned assumptions $(k.i)$-$(k.iv)$ as $\boldsymbol{\theta} = (\mathcal{D}, \phi, M, L, f, \boldsymbol{w})\}$. Depending on the assumptions made on each of these components, one can envisage different attacker knowledge scenarios. Typically, two main settings are considered, namely *Perfect-knowledge* and *Limited-knowledge*.

[1] In [84], the authors defined *targeted* and *indiscriminate* attacks (at test time) depending on whether the attacker aims to cause *specific* or *generic* errors. Here we do not follow their naming convention, as it can cause confusion with the interpretation of *targeted* and *indiscriminate* attacks introduced in previous work [7, 6, 52, 17, 108, 20, 13, 21].

4.2.1 Perfect-Knowledge (PK) Attack

In this case, the attacker is assumed to know everything about the targeted system. Although this setting, that is called also white-box scenario, may be not always representative of practical cases, it enables the defender to perform a worst-case evaluation of the security of learning algorithms under attack, highlighting the upper bounds on the performance degradation that may be incurred by the system under attack. In this case, the attacker knowledge can be represented by a vector $\boldsymbol{\theta}_{\text{PK}} = (\mathcal{D}, \phi, M, L, f, \boldsymbol{w})$.

4.2.2 Limited-Knowledge (LK) Attack

When some of the information related to the target system is missing the attack is called Limited-Knowledge Attack. Sometimes it is also referred as black-box or gray box attack. Although this admits a wide range of possibilities, the attacker is typically assumed to know the feature representation ϕ, the learning algorithm M, the training loss function L and the decision function f but not the training data (for which surrogate data from similar sources can be collected). This case is here named as LK attacks with Surrogate Data (LK-SD), and it is denoted with $\boldsymbol{\theta}_{\text{LK-SD}} = (\hat{\mathcal{D}}, \phi, M, L, f, \hat{\boldsymbol{w}})$ (where the *hat* symbol is used to denote limited knowledge of a given component).Notably, in this case, as the attacker is only given a surrogate data set $\hat{\mathcal{D}}$, also the learner's parameters have to be estimated by the attacker, e.g., by optimizing L on $\hat{\mathcal{D}}$.

Similarly, the case in which the attacker knows the training data (e.g., if the learning algorithm is trained on publicly-available data), but not the learning algorithm (for which a surrogate learner can be trained on the available data) is here named as LK attacks with Surrogate Learners (LK-SL). This scenario can be denoted with $\boldsymbol{\theta}_{\text{LK-SL}} = (\mathcal{D}, \phi, \hat{M}, \hat{L}, \hat{f}, \hat{\boldsymbol{w}})$, even though the parameter vector $\hat{\boldsymbol{w}}$ may belong to a different vector space than that of the targeted learner. Note that LK-SL attacks also includes the case in which the attacker knows the learning algorithm, but she is not able to derive an optimal attack strategy against it (e.g., if the corresponding optimization problem is not tractable or difficult to solve), and thus uses a surrogate learning model to this end. Experiments on the *transferability* of attacks among learning algorithms firstly demonstrated in [12] and then in subsequent work on deep learners [83], fall under this category of attacks.

4.3 Attacker Capability

An attacker may have different capabilities depending on her skills and the structure of the targeted machine learning system that usually consists of different modules e.g. feature extractor, classifier and so on. If the system components are accessible from users, an attacker may potentially threat each of them. However, the majority

of these attack is rarely practically applicable as usually user does not have access to the machine learning system components. Machine Learning systems get often data from the users, therefore a malicious ones have the chance to manipulate them in order to threaten the system. As Data are the most common attack vector, therefore, in this thesis only this kind of attack will be exploited.

In the next subsections we explain how attacks are subdivided based on attack capabilities. Moreover, how and why data manipulations might be constrained is clarified.

Attack Influence. In supervised learning, the attacker may influence both training and test data. The attack is therefore called causative if the attacker can influence also the training data, or exploratory if the attacker can only manipulate test data. These settings are more commonly referred to as *poisoning* and *evasion* attacks [7, 52, 17, 12, 19, 108, 73].

Data Manipulation Constraint. The manipulation of the input data might be subjected to some constraints which are however strongly dependent on the given practical scenario. For example, if the attacker aims to evade a malware classification system, she should manipulate the exploitation code embedded in the malware sample without compromising its intrusive functionality. This may imply that she can not change some features at all. Moreover, a clever attacker would keep the input data perturbation under a maximum amount. This may be important to craft attack samples which are more difficult to detect with data pre-filtering or outlier detection techniques. Typically, these constraints can be nevertheless accounted for in the definition of the optimal attack strategy. In particular, we characterize them by assuming that an initial set of attack samples $\mathcal{D}_a$ is given, and that it is modified according to a space of possible modifications $\Psi(\mathcal{D}_a)$ (e.g., constraining the norm of the input perturbation on each poisoning sample).

4.4 Attack Strategy

Once the adversary's goal, knowledge, and capability are defined, the optimal attack strategy can be formulated as an objective function. Maximizing this function, the attacker will be able to exploit the attack threatening the system.

More formally, given the attacker's knowledge $\boldsymbol{\theta} \in \Theta$ and a set of manipulated attack samples ${\mathcal{D}_a}' \in \Psi(\mathcal{D}_a)$, the attacker's goal can be characterized in terms of an objective function $A({\mathcal{D}_a}', \boldsymbol{\theta}) \in \mathbb{R}$ which evaluates how effective the attacks ${\mathcal{D}_a}'$ are. The optimal attack strategy can be thus given as:

$$ {\mathcal{D}_a}^* \in \underset{{\mathcal{D}_a}' \in \Psi(\mathcal{D}_a)}{\arg\max} A({\mathcal{D}_a}', \boldsymbol{\theta}) \tag{4.1} $$

Notably, this high-level formulation encompasses both evasion and poisoning attacks. A more detailed discussion about them is presented in the next chapters.

In the subsequent notation, the *hat* symbols denote all the elements that depends on the attacker knowledge $\boldsymbol{\theta}$ to remind that they can be not known and therefore they are surrogate versions of the originals ones.

4.5 Security Evaluation Methodology

A possible way to provide security guarantees for a machine learning system would be to apply formal verification techniques. Unfortunately, despite the effort made by researchers in order to provide those methods [89, 53, 56], they are still not applicable as they heavily rely on different assumptions (about either the attacker or the learning algorithm). The security evaluation of a machine learning systems has therefore to be done testing it under adversarial attacks [2]. However, for being informative these tests have to be really carefully designed. This aspect seems underestimated in the recent literature. In order to test the vulnerability of computer-vision system to evasion attack, adversarial examples based on minimum-distance perturbations are created [102, 47, 77, 84] and the system is declared as vulnerable if they are not recognizable from humans. Despite this method seem nowadays standard-de-facto is not clear how the human recognizability can be measured and moreover it is classifier dependent. This method might be misleading in some case if used to compare the security of different classifiers. In order to produce comparable results, classifier security can be tested instead under attacks performed with increasing attack power. In the evasion case the attack power can be measured as the amount of perturbation applied by the attacker on a sample under a chosen norm. In poisoning case it can instead be measured as the number of poisoning points (eventually created with a maximum perturbation constraint) inserted by the attacker. Notably, the proposed framework allows one to exploit this constrained attacks against multiclass classifiers providing the instrument needed to compare the security of two different classifiers.

Chapter 5

Test-time Evasion Attacks against Machine Learning

Machine learning is nowadays widely used in security-sensitive settings like spam and malware detection, despite its vulnerability to *adversarial* attacks, i.e., the *deliberate* manipulation of training or test data, to subvert the system functionalities; e.g., spam emails can be manipulated (at test time) to evade a trained anti-spam classifier [36, 66, 67, 59, 82, 7, 15, 12, 17, 16, 52, 113]. To overcome this limitation, adversary-aware learning algorithms have been developed, exploiting robust optimization and game-theoretical models to incorporate knowledge of potential adversarial data manipulations into the learning algorithm. Despite these techniques have been shown to be effective in some adversarial learning tasks, their adoption in practice is hindered by different factors, including the difficulty of meeting specific theoretical requirements, the complexity of implementation, and scalability issues, in terms of computational time and space required during training.

In this chapter we show that leveraging theoretical results it is possible to create computationally efficient countermeasures.

Firstly, the evasion attack scenario derived from the adversarial framework proposed in the previous chapter is formally defined in Section 5.1. A gradient-based algorithm that can be exploited to compute the attack sample is discussed in Section 5.2. Exploiting that algorithm an attacker can be able to create sparse or dense attack samples (as it is required from the targeted application). The difference between them is highlighted in Section 5.3. As linear classifiers are still one of the widely used classifier thanks to their computational efficiency and to the interpretability of their decisions. Their security under both kind of attacks is analyzed in Section 5.4. The sparsity of the linear classifier solution is often forced in application with strict efficiency requirements. We show that there is a trade-off between sparsity and security to sparse attacks. As sparsity is a desirable property a countermeasure that allows one to obtain a trade-off between these two properties, is proposed. In Section 5.5 a countermeasure designed for multiclass classifiers

inspired from open set recognition techniques is finally presented.

5.1 Evasion Attack Scenario

As it is said before, an attack is called evasion when the attacker has the ability to modify only test samples. Below we present the evasion attack scenarios for multiclass classifiers. The approach presented below is based on extending the work in [12] for evasion of binary classifiers to the multiclass case. To this end, the formulation for the two possible evasion settings i.e., ways of creating adversarial examples, which further differentiate our technique from previous work on the creation of minimally-perturbed adversarial examples [102, 47, 77, 84], namely *error-generic* and *error-specific* evasion is derived. In the *error-generic* scenario, the attacker is interested in misleading the classification process, regardless the output class predicted by the classifier for the adversarial examples; e.g., for a known terrorist the goal may be to evade detection by a video surveillance system, regardless of the identity erroneously associated to his/her face. Conversely, in the *error-specific* setting, the attacker still aims to mislead classification, but requiring the adversarial examples to be misclassified as a specific, target class; e.g., imagine an attacker aiming to impersonate a specific user. The two settings can be formalized in terms of two distinct optimization problems, though using the same formulation for the objective function $A(\boldsymbol{x})$:

$$A(\boldsymbol{x}) = \hat{f}_k(\boldsymbol{x}) - \max_{l \neq k} \hat{f}_l(\boldsymbol{x})\,. \tag{5.1}$$

This function essentially represents a difference between a preselected discriminant function (associated to class k) and the competing one, i.e., the one exhibiting the highest value at $\boldsymbol{x}$ among the remaining $\mathsf{n_c} - 1$ classes (i.e., all classes $\{1, \ldots, \mathsf{n_c}\}$ except k). Below, we discuss how class k is chosen in the two considered settings.

5.1.1 Error-generic Evasion

In this case, the optimization problem can be formulated as:

$$\min_{\boldsymbol{x}'} \quad A(\boldsymbol{x}')\,, \tag{5.2}$$

$$\text{s.t.} \quad d(\boldsymbol{x}, \boldsymbol{x}') \leq d_{max}\,, \tag{5.3}$$

$$\boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub}\,, \tag{5.4}$$

where $f_k(\boldsymbol{x})$ in the objective function $A(\boldsymbol{x})$ (Eq. 5.1) denotes the probability estimated by the classifier of $\boldsymbol{x}$ belonging to his true class. This objective function allows the attacker to increase the probability to have her attack sample misclassified as the class at which the classifier assigns the highest score after the true one, minimizing the score difference between them. $d(\boldsymbol{x}, \boldsymbol{x}') \leq d_{max}$ represents a

constraint on the maximum input perturbation d_{max} between $\boldsymbol{x}$ (i.e., the original adversarial sample) and the optimized ones $\boldsymbol{x}'$, given in terms of distance in the input space. In case the samples are images, normally, the ℓ_2 distance between pixel values is used as function $d(\cdot, \cdot)$, but other metrics can be also adopted. The box constraint $\boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub}$ is optional and can be used to bound the input values $\boldsymbol{x}$ of the adversarial examples; e.g., each pixel value in images is bounded between 0 and 255. Nevertheless, the box constraint can be also used to manipulate only some feature values in the attack sample. This is of crucial importance for creating real-world adversarial examples, as it allows one to avoid manipulating, in the optimal attack sample computed by the algorithm, features which can not be manipulated by the attacker. For example, in malware detection some features if manipulated may compromise the malware functionalities as we will explain in the case of study presented in Chapter 8. In image recognition application instead, it can be used to avoid manipulating pixels which do not belong to the object of interest. This may enable one to create an "unusual" sticker that can be attached to an *adversarial* object as we show in the case of study presented in Chapter 9 similarly to the idea exploited in [98] for the creation of wearable objects used to fool face recognition systems. In this case, the adversary can set in $\boldsymbol{x}_{lb}$ and $\boldsymbol{x}_{ub}$ the values of pixels that can not be manipulated equal to those of $\boldsymbol{x}$.

5.1.2 Error-specific Evasion

The problem of error-specific evasion is formulated as:

$$\max_{\boldsymbol{x}'} \quad A(\boldsymbol{x}')\,, \tag{5.5}$$

$$\text{s.t.} \quad d(\boldsymbol{x}, \boldsymbol{x}') \leq d_{max}\,, \tag{5.6}$$

$$\boldsymbol{x}_{lb} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{ub}\,, \tag{5.7}$$

where $f_k(\boldsymbol{x})$ in the objective function $A(\boldsymbol{x})$ (Eq. 5.1) denotes the probability estimated by the classifier of $\boldsymbol{x}$ belonging to the targeted class, i.e., the class which the adversarial example should be assigned to. This objective function allows the attacker to increase the probability to have her sample misclassified as belonging to the target class, maximizing the difference between the score of the target class and the one at which the classifier assign a higher score.

5.2 Gradient-Based Evasion Attack Algorithm

The evasion problem can be solved, considering the original classifier or a differentiable surrogate, using a gradient-based approach using the Algorithm 1. The basic idea is to update the adversarial example by following the steepest descent (or ascent) direction (depending on whether we are considering error-generic or error-specific evasion), and use a projection operator Π to keep the updated point within

Algorithm 1 Computation of Adversarial Examples

Input: $\boldsymbol{x}_0$: the initial adversarial sample; η: the step size; $r \in \{-1, +1\}$: variable set to -1 ($+1$) for error-generic (error-specific) evasion; $\varepsilon > 0$: a small number.
Output: $\boldsymbol{x}'$: the adversarial example.

1: $\boldsymbol{x}' \leftarrow \boldsymbol{x}_0$
2: **repeat**
3: $\quad \boldsymbol{x} \leftarrow \boldsymbol{x}'$, and $\boldsymbol{x}' \leftarrow \Pi\left(\boldsymbol{x} + r\eta\nabla A(\boldsymbol{x})\right)$
4: **until** $|A(\boldsymbol{x}') - A(\boldsymbol{x})| \leq \varepsilon$
5: **return** $\boldsymbol{x}'$

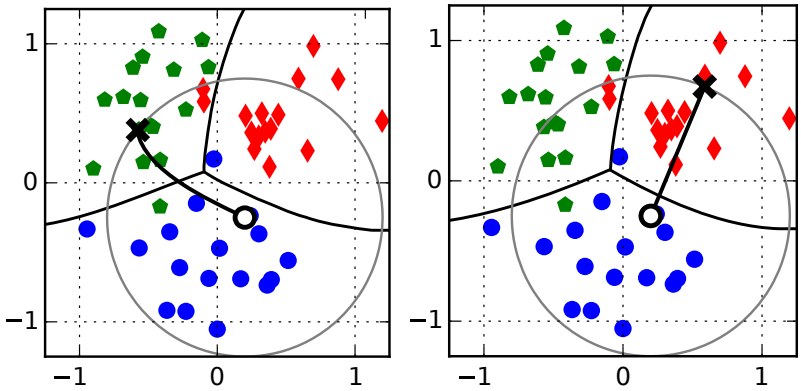


Figure 5.1: Error-specific (*left*) and error-generic (*right*) evasion of a multiclass SVM with the Radial Basis Function (RBF) kernel. Decision boundaries among the three classes (blue, red and green points) are shown as black solid lines. In the error-specific case, the initial (blue) sample is shifted towards the green class (selected as the target one). In the error-generic case, instead, it is shifted towards the red class, as it is the closest class to the initial sample. The ℓ_2 distance constraint is also shown as a gray circle.

the feasible domain (given by the intersection of the box and the ℓ_2 constraint). An example of the different behavior exhibited by the two attacks is given in Fig. 5.1.

5.3 Sparse and Dense Attacks

As we explained in Chapter 4, the attacker can have different data manipulation capabilities. In security related applications for example, the attacker has typically a cost depending on the number of modified features. This can be configured as a *sparse* attack. In order to find her optimal attack sample, she would constrain the original sample perturbation with a ℓ_1 norm constraint. When the attack sample is an image, the attacker goal may be instead to avoid that it becomes easily recognizable from either humans or machines. In this case the attacker would probably constraint the perturbation with an ℓ_2 norm perturbation (obtaining a *dense* at-

tack sample), as it only produces a slightly-blurred effect on the image, while sparse attacks create more evident artifacts. The aforementioned attack, computed with Algorithm 1 is shown on a two dimensional gaussian dataset in Fig. 5.2 and in an handwritten recognition problem, where the samples are images in Fig. 5.3.

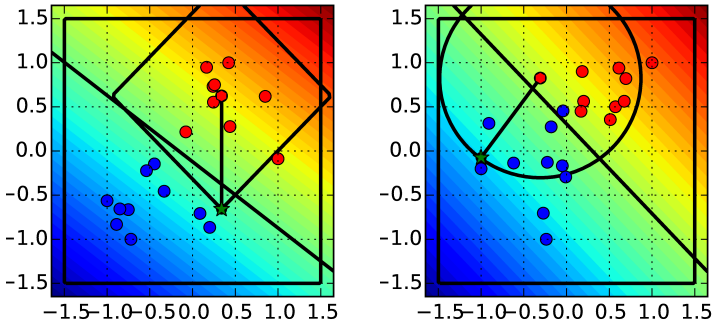


Figure 5.2: Evasion attacks against linear SVM, trained on blue (legitimate) and red (malicious) samples. The shown attacks are respectively sparse (first plot) and dense (second plot). The initial malicious point $\boldsymbol{x}$ is found at the center of the distance constraint, while the evasion sample $\boldsymbol{x}^\star$ is denoted with a green star. For each classifier, $g(\boldsymbol{x})$ values are shown in colors, and the black line denotes the decision boundary.

5.4 Trading sparsity for security: Octagonal Regularization

Linear classifiers have been increasingly used in embedded systems and mobile devices for their low processing time and memory requirements. Nonetheless they are also a preferred choice as they provide easier-to-interpret decisions (with respect to nonlinear classification methods). For instance, the widely-used SpamAssassin anti-spam filter exploits a linear classifier [15, 82].[1] Work in the adversarial machine learning literature has already investigated the security of linear classifiers to evasion attacks [59, 15], suggesting the use of more evenly-distributed feature weights as a mean to improve their security. Such a solution is however based on heuristic criteria, and a clear understanding of the conditions under which it can be effective, or even optimal, is still lacking. Moreover, in mobile and embedded systems, *sparse* weights are more desirable than evenly-distributed ones, in terms of processing time, memory requirements, and interpretability of decisions.

In this section recent findings on the relationship between regularization and robustness properties of learning algorithm are firstly summarized as they help us

[1] See also http://spamassassin.apache.org.

Figure 5.3: Initial digit "9" (left) and its versions modified to be misclassified as "8". *sparse*(center) and *dense*(right) evasion attack samples. Note also how the blurring effect induced by dense attacks is more difficult to spot for humans than the salt-and-pepper noise induced by sparse attacks.

to shed light on the role of the regularization on the linear classifier security.

We show that the maximum variation that an attacker can induce on classifier's discriminant function can be bounded. This result highlight that the security of a linear classifier can be improved by selecting a proper regularizer, depending on the kind of evasion attack.

A novel octagonal regularizer that allows one to achieve a proper trade-off between sparsity of feature weights, which is desirable for reducing processing cost and the security of linear classifiers is finally proposed.

5.4.1 Robustness and Regularization

The goal of this section is to clarify the connection between regularization and input data uncertainty, leveraging the recent findings in [110, 100, 65] as it will be exploited later in this thesis to link regularization and security of linear classifiers, depending on the type of attack. In particular, Xu et al. [110] have considered the following *robust* optimization problem:

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{u}_1,\ldots,\boldsymbol{u}_n \in \mathcal{U}} \sum_{i=1}^{n} \left(1 - y_i(\boldsymbol{w}^\top(\boldsymbol{x}_i - \boldsymbol{u}_i) + b)\right)_+ , \tag{5.8}$$

where $(z)_+$ is equal to $z \in \mathbb{R}$ if $z > 0$ and 0 otherwise, $\boldsymbol{u}_1, ..., \boldsymbol{u}_n \in \mathcal{U}$ define a set of bounded perturbations of the training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n \in \mathbb{R}^n \times \{-1, +1\}^n$, and the so-called *uncertainty set* $\mathcal{U}$ is defined as $\mathcal{U} := \{(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) | \sum_{i=1}^n \|\boldsymbol{u}_i\|^* \leq c\}$, being $\|\cdot\|^*$ the dual norm of $\|\cdot\|$. Typical examples of uncertainty set according to the above definition include ℓ_1 and ℓ_2 balls [110, 100].

Problem (5.8) basically corresponds to minimizing the hinge loss for a two-class classification problem under worst-case, bounded perturbations of the training samples $\boldsymbol{x}_i$, i.e., a typical setting in robust optimization [110, 100, 65]. Under some mild assumptions easily verified in practice (including non-separability of the training data), the authors have shown that the above problem is equivalent to the following

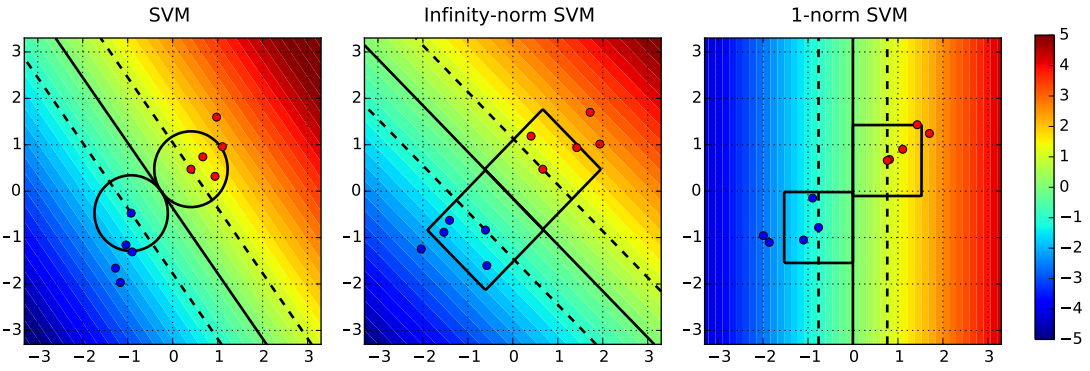


Figure 5.4: Discriminant function $f(\boldsymbol{x})$ for SVM, Infinity-norm SVM, and 1-norm SVM (in colors). The decision boundary ($g(\boldsymbol{x}) = 0$) and margins ($g(\boldsymbol{x}) = \pm 1$) are respectively shown with black solid and dashed lines. Uncertainty sets are drawn over the support vectors to show how they determine the orientation of the decision boundary.

non-robust, regularized optimization problem (*cf.* Th. 3 in [110]):

$$\min_{\boldsymbol{w},b} \; c\|\boldsymbol{w}\| + \sum_{i=1}^{n} \left(1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\right)_+ . \tag{5.9}$$

Where c is the same constant as the previously defined problem. This means that, if the ℓ_2 norm is chosen as the dual norm characterizing the uncertainty set $\mathcal{U}$, then $\boldsymbol{w}$ is regularized with the ℓ_2 norm, and the above problem is equivalent to a standard Support Vector Machine (SVM) [33]. If input data uncertainty is modeled with the ℓ_1 norm, instead, the optimal regularizer would be the ℓ_∞ regularizer, and vice-versa.[2] This notion is clarified in Fig. 5.4, where we consider different norms to model input data uncertainty against the corresponding SVMs; i.e., the standard SVM [33], the Infinity-norm SVM [11] and the 1-norm SVM [117] against ℓ_2, ℓ_1 and ℓ_∞-norm uncertainty models, respectively. Is worth mention that the cited work focus is on SVM but remarkably the presented equivalence proofs are applicable also to different loss functions.

These results are relevant for us as the perturbation added from the attacker to the attack sample can be seen as uncertainty on that sample.

5.4.2 Classifier Security Analysis

Here we show that the maximum variation of a linear classifier's discriminant function under an evasion attack can be bounded, highlighting the factors that may harm classifier security, and discussing how to limit their impact. This will give us a

[2]Note that the ℓ_1 norm is the dual norm of the ℓ_∞ norm, and vice-versa, while the ℓ_2 norm is the dual norm of itself.

set of guidelines to help to design more secure learning algorithms against evasion. It is also worth remarking that, very interestingly, some of the results arising from our analysis corroborate the findings discussed in the previous section for linear classifiers.

We start by analyzing the worst-case variation of the discriminant function of a linear classifier under evasion. The discriminant function of a linear classifier is simply given as $g(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$. In security problems the benign samples are usually identified as belonging to the negative class (-1) and the malicious one as belonging to the positive (1) class. The attacker goal is to decrease the discriminant function value to have her malicious sample misclassified as a benign one. Assuming that $\boldsymbol{x}$ is an initial malicious sample, and $\boldsymbol{x}'$ the corresponding manipulated evasion sample, the score difference between the original and the modified attack sample is:

$$\Delta g = g(\boldsymbol{x}) - g(\boldsymbol{x}') = \boldsymbol{w}^\top (\boldsymbol{x} - \boldsymbol{x}') \,. \tag{5.10}$$

Note that, from the attacker's perspective, this variation has to be maximized to increase chances of successfully evade the targeted classifier..

Sparse Attacks. Under sparse evasion attacks, it is not difficult to see that Δg (from Eq. 5.10) is upper bounded by the following quantity:

$$\Delta g \leq \|\boldsymbol{w}\|_\infty \times \|\boldsymbol{x} - \boldsymbol{x}'\|_1 \,, \tag{5.11}$$

where we remind the reader that $\|\boldsymbol{w}\|_\infty = \max_{j=1,\ldots,\mathsf{d}} |w_j|$. In fact, for sparse attacks, the solution $\boldsymbol{x}'$ is found by modifying the features that have been assigned the highest absolute weight values (see, e.g., Fig. 5.2, left plot). In the worst case, the maximum Δg is attained by modifying the most relevant feature of a quantity equal to $\|\boldsymbol{x} - \boldsymbol{x}'\|_1$.

Dense Attacks. Under dense evasion attacks, instead, the worst-case increase of Δg corresponds to a linear shift of $\boldsymbol{x}$ towards the decision boundary (along the opposite direction to the hyperplane normal $\boldsymbol{w}$), i.e., $\boldsymbol{x}' = \boldsymbol{x} - \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$ (see Fig. 5.2, right plot), which implies that:

$$\Delta g \leq \frac{\boldsymbol{w}^\top \boldsymbol{w}}{\|\boldsymbol{w}\|_2} \times \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \|\boldsymbol{w}\|_2 \times \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \,. \tag{5.12}$$

5.4.3 Countering Sparse and Dense Attacks

The analysis of the worst-case Δg values for linear classifiers highlights two interesting facts. The former is that the feature values should be bounded, to bound the maximum variation of the relevant features. This is normally not a problem, if feature normalization is used, as normalization techniques often map the input samples onto a compact domain. The latter fact is that the security of linear classifier can be improved choosing the more appropriate regularizer according to the expected attack type. This novel result in the context of adversarial learning also

confirms the findings by Xu et al. [110] related to the relationship between robustness and regularization of learning algorithms. We consider the optimal classifier against respectively dense and sparse attack. Despite below we consider the hinge loss remarkably, the regularizer choice is still valid for different losses.

Countering Dense Attacks

As the bound of the discriminant function variation in case of sparse attack is dependent from the ℓ_2 norm of $\boldsymbol{w}$, the best regolarizer choice would be an ℓ_2 regularizer.

2-norm SVM (SVM). This is the standard SVM learning algorithm [33]. It finds $\boldsymbol{w}$ and b by solving the following quadratic programming problem:

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{\mathsf{m}} (1 - y_i g(\boldsymbol{x}_i))_+ \ , \tag{5.13}$$

where $g(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ denotes the SVM's linear discriminant function. Note that ℓ_2 regularization does not induce sparsity on $\boldsymbol{w}$.

Countering Sparse Attack

As the bound of the discriminant function variation in case of sparse attack is dependent from the ℓ_∞ norm of $\boldsymbol{w}$, the more secure classifier against this kind of attack will exploit the ℓ_∞ norm regularize in its objective function. We thus consider the SVM formulation, but changing the regularization term:

Infinity-norm SVM (∞-norm). In this case, the ℓ_∞ regularizer bounds the weights' maximum absolute value as $\|\boldsymbol{w}\|_\infty = \max_{j=1,\ldots,\mathsf{d}} |w_j|$ [24]:

$$\min_{\boldsymbol{w},b} \quad \|\boldsymbol{w}\|_\infty + C\sum_{i=1}^{\mathsf{m}} (1 - y_i g(\boldsymbol{x}_i))_+ \ . \tag{5.14}$$

As the standard SVM, this classifier is not sparse on $\boldsymbol{w}$; however, the above learning problem can be solved using a simple linear programming approach.

Sec-SVM. In some applications is needed to apply different upper and lower bounds to different feature sets, depending on how their values can be manipulated. An application example is malware detection that we will analyze as case study in Chapter 8 where removing some feature the attacker may compromise the malware functionalities.

As an alternative to considering an additional term to the learner's objective function $\mathcal{L}$, one can still control the ℓ_∞-norm of $\boldsymbol{w}$ by adding a box constraint on it. This is a well-known property of convex optimization [26]. Following this approach allow to apply different upper and lower bounds to different feature sets. Moreover, it preserves *convexity* of the objective function minimized by the learning algorithm.

This gives us the possibility of deriving computationally-efficient training algorithms with (potentially strong) convergence guarantees.

We define our Secure SVM learning algorithm (Sec-SVM) as:

$$\min_{\boldsymbol{w},b} \quad \tfrac{1}{2}\boldsymbol{w}^\top\boldsymbol{w} + C\sum_{i=1}^{\mathsf{n}} \max\left(0, 1 - y_i f(\boldsymbol{x}_i)\right) , \tag{5.15}$$

$$\text{s.t.} \quad w_k^{\text{lb}} \leq w_k \leq w_k^{\text{ub}} \, , \; k = 1, \ldots, \mathsf{d} \, . \tag{5.16}$$

Note that this optimization problem is identical to the standard SVM, except for the presence of a box constraint on $\boldsymbol{w}$. The lower and upper bounds on $\boldsymbol{w}$ are defined by the vectors $\boldsymbol{w}^{\text{lb}} = (w_1^{\text{lb}}, \ldots, w_{\mathsf{d}}^{\text{lb}})$ and $\boldsymbol{w}^{\text{ub}} = (w_1^{\text{ub}}, \ldots, w_{\mathsf{d}}^{\text{ub}})$, which should be selected with a suitable procedure (see Sect. 8.3.4). For notational convenience, in the sequel we will also denote the constraint given by Eq. (5.16) compactly as $\boldsymbol{w} \in \mathcal{W} \subseteq \mathbb{R}^{\mathsf{d}}$.

The corresponding learning algorithm is given as Algorithm 2. It is a constrained variant of Stochastic Gradient Descent (SGD) that also considers a simple line-search procedure to tune the gradient step size during the optimization. SGD is a lightweight gradient-based algorithm for efficient learning on very large-scale datasets, based on approximating the subgradients of the objective function using a single sample or a small subset of the training data, randomly chosen at each iteration [114, 25]. In our case, the subgradients of the objective function (Eq. 5.15) are given as:

$$\nabla_{\boldsymbol{w}}\mathcal{L} \;\approxeq\; \boldsymbol{w} + C\sum_{i\in\mathcal{S}} \nabla_\ell^i \, \boldsymbol{x}_i \, , \tag{5.17}$$

$$\nabla_b\mathcal{L} \;\approxeq\; C\sum_{i\in\mathcal{S}} \nabla_\ell^i \, , \tag{5.18}$$

where $\mathcal{S}$ denotes the subset of the training samples used to compute the approximation, and ∇_ℓ^i is the gradient of the hinge loss with respect to $f(\boldsymbol{x}_i)$, which equals $-y_i$, if $y_i f(\boldsymbol{x}_i) < 1$, and 0 otherwise.

One crucial issue to ensure quick convergence of SGD is the choice of the initial gradient step size $\eta^{(0)}$, and of a proper *decaying function* $s(t)$, i.e., a function used to gradually reduce the gradient step size during the optimization process. As suggested in [114, 25], these parameters should be chosen based on preliminary experiments on a subset of the training data. Common choices for the function $s(t)$ include linear and exponential decaying functions.

Is worth mention that Sec-SVM formulation is quite general; one may indeed select different combinations of loss and regularization functions to train different, secure variants of other linear classification algorithms. Our Sec-SVM learning algorithm is only an instance that considers the hinge loss and ℓ_2 regularization, as the standard SVM [33, 106].

Exploiting Cost-sensitive Learning

The over-mentioned work by Xu et al. [110] only considers uncertainty sets of the same size, i.e., the same perturbation is applied on both the legitimate and the

Algorithm 2 Sec-SVM Learning Algorithm

Input: $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{\mathsf{n}}$, the training data; C, the regularization parameter; $\boldsymbol{w}^{\text{lb}}, \boldsymbol{w}^{\text{ub}}$, the lower and upper bounds on $\boldsymbol{w}$; $|\mathcal{S}|$, the size of the sample subset used to approximate the subgradients; $\eta^{(0)}$, the initial gradient step size; $s(t)$, a decaying function of t; and $\varepsilon > 0$, a small constant.

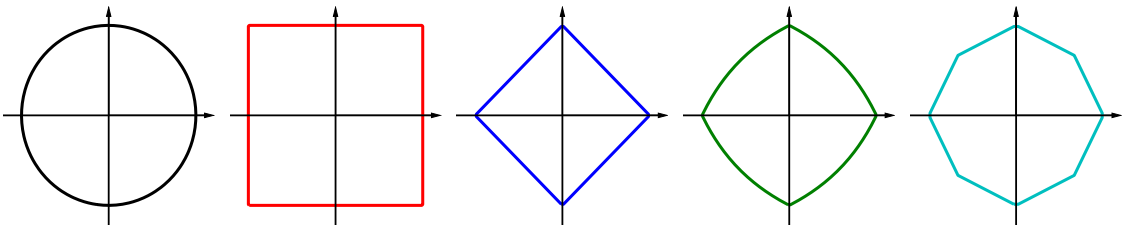
Output: $\boldsymbol{w}, b$, the trained classifier's parameters.

Set iteration count $t \leftarrow 0$.
Randomly initialize $\boldsymbol{v}^{(t)} = (\boldsymbol{w}^{(t)}, b^{(t)}) \in \mathcal{W} \times \mathbb{R}$.
Compute the objective function $\mathcal{L}(\boldsymbol{v}^{(t)})$ using Eq. (5.15).
repeat
    Compute $(\nabla_{\boldsymbol{w}}\mathcal{L}, \nabla_b\mathcal{L})$ using Eqs. (5.17)-(5.18).
    Increase the iteration count $t \leftarrow t + 1$.
    Set $\eta^{(t)} \leftarrow \gamma\, \eta^{(0)} s(t)$, performing a line search on γ.
    Set $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta^{(t)} \nabla_w \mathcal{L}$.
    Project $\boldsymbol{w}^{(t)}$ onto the feasible (box) domain $\mathcal{W}$.
    Set $b^{(t)} \leftarrow b^{(t-1)} - \eta^{(t)} \nabla_b \mathcal{L}$.
    Set $\boldsymbol{v}^{(t)} = (\boldsymbol{w}^{(t)}, b^{(t)})$.
    Compute the objective function $\mathcal{L}(\boldsymbol{v}^{(t)})$ using Eq. (5.15).
until $|\mathcal{L}(\boldsymbol{v}^{(t)}) - \mathcal{L}(\boldsymbol{v}^{(t-1)})| < \varepsilon$
return: $\boldsymbol{w} = \boldsymbol{w}^{(t)}$, and $b = b^{(t)}$.

malicious class. However, it is clear that, under evasion, the malicious samples are potentially affected by a stronger worst-case perturbation than legitimate data as the attacker will modify it in order to evade the classifier. Interestingly, in their recent work, Katsumata and Takeda [55] have shown that different uncertainty sets can be accounted for on each sample (and thus, on each class too), by simply modifying the cost of each classification error. This means that it suffices to penalize differently errors in different classes to consider uncertainty sets of different sizes. In the SVM learning algorithm, this can be simply accounted for by setting a different C value for legitimate and malicious samples. However, as we have shown in [94] it only introduces a slight improvement in terms of security.

5.4.4 Octagonal regularizer

As we previously explained in this section, if one considers an ℓ_1 (sparse) attacker, facing a higher cost when modifying more features, it turns out that the optimal regularizer is given by the ℓ_∞ norm of $\boldsymbol{w}$, which tends to yield more uniform weights. In particular, the solution provided by ℓ_∞ regularization (in the presence of a strongly-regularized classifier) tends to yield weights which, in absolute value, are all equal to a (small) maximum value. This also implies that ℓ_∞ regularization does not provide a *sparse* solution. As it is said before, sparsity is a desirable property in many ap-



(a) ℓ_2 (b) ℓ_∞ (c) ℓ_1 (d) elastic net (e) octagonal

Figure 5.5: Unit balls for different norms.

plications. For this reason we propose a novel *octagonal* (8gon) regularizer,[3] given as a linear (convex) combination of ℓ_1 and ℓ_∞ regularization is here proposed:

$$\|\boldsymbol{w}\|_{8\text{gon}} = (1-\rho)\|\boldsymbol{w}\|_1 + \rho\|\boldsymbol{w}\|_\infty \tag{5.19}$$

where $\rho \in (0,1)$ can be increased to trade sparsity for security. In Fig. 5.5 the shape of different regularizers is compared with the presented one. Notably, the proposed regularizer allow one to obtain the maximum level of sparsity maintaining the security level required for the specific application.

Considering the hinge as classifier loss we can define the Octagonal-norm SVM (8gon).

Octagonal-norm SVM (8gon). This novel SVM is based on our octagonal-norm regularizer, combined with the hinge loss:

$$\min_{\boldsymbol{w},b} \quad (1-\rho)\|\boldsymbol{w}\|_1 + \rho\|\boldsymbol{w}\|_\infty + C\sum_{i=1}^{\mathsf{m}} (1 - y_i g(\boldsymbol{x}_i))_+ \,. \tag{5.20}$$

The above optimization problem is linear, and can be solved using state-of-the-art solvers. The sparsity of $\boldsymbol{w}$ can be increased by decreasing the trade-off parameter $\rho \in (0,1)$, at the expense of classifier security.

5.5 Securing Multiclass Classifier with Distance-based Rejection

If the evasion algorithm drives the adversarial examples deeply into regions populated by known training classes (as shown in Fig. 5.1), there is no much one can do to correctly identify them from the rest of the data by only re-training or modifying the classifier, i.e., modifying the shape of the decision boundaries in the feature space.

[3] Note that octagonal regularization has been previously proposed also in [24]. However, differently from our work, the authors have used a pairwise version of the infinity norm, for the purpose of selecting (correlated) groups of features.

If the feature vector of an adversarial example becomes *indistinguishable* from those of the training samples of a different class, it can only be detected by using a different feature representation. This is infact an intrinsic *vulnerability* of the *feature representation*. However, sometimes classifiers may be easily evaded with adversarial sample that belongs to low support region. These samples are called *blind-spot* adversarial samples. If a machine learning system can be evaded with such kind of samples is a *classifier vulnerability* as the classifier is not able to understand that it does not have sufficient evidence to decide.

In many applications, as in a robot-vision case study that we report in Chapter 9, the feature space is given and, due to application constraints, it can not be changed. In those cases, the only possibility to improve the system security is to reduce the classifier vulnerability. Here we propose a countermeasure inspired by the general Open Set Recognition principles that are below briefly summarized.

5.5.1 Open Set Recognition

Open Set Recognition [95, 9] is the field of study that aims to learn a classifier when not all the classes encountered during testing are known during the training phase. This is a recurrent problem in object recognition as also the biggest dataset can not contain examples for each class of existent objects. In order to avoid misclassifying an unknown object as a known one, the classifier should be able to discriminate the objects that do not belong to one of the known classes. Scheirer et al. in [95] defined therefore a new formal model that they called *compact abating probability* (CAP). The peculiarity of CAP model is that the probability of a sample belonging to a class abate as it moves farther from training data.

5.5.2 Distance-based rejection

We show here a countermeasure that allows one to improve multiclass classifier security against blind spot adversarial examples.

Different approaches have been proposed based on modifying the classifier, ranging from 1.5-class classification (based on the combination of anomaly detectors and two-class classifiers) [14] to open-set recognition techniques [95, 9]. We propose here a more direct approach, based on the same idea underlying the notion of classification with a reject option, and leveraging some concepts from open-set recognition. In particular, we propose a one-versus-all classifier composed by binary SVMs with RBF kernels. Then, by applying a simple rejection mechanism on its discriminant function, we can identify samples which are far enough from the rest of the training data, i.e., blind-spot adversarial examples. Our idea is thus to modify the one-versus-all classifier decision rule as:

$$c^\star = \arg\max_{k=1,\ldots,\mathsf{c}} f_k(\boldsymbol{x})\,, \text{ only if } f_{c^\star}(\boldsymbol{x}) > 0\,, \tag{5.21}$$

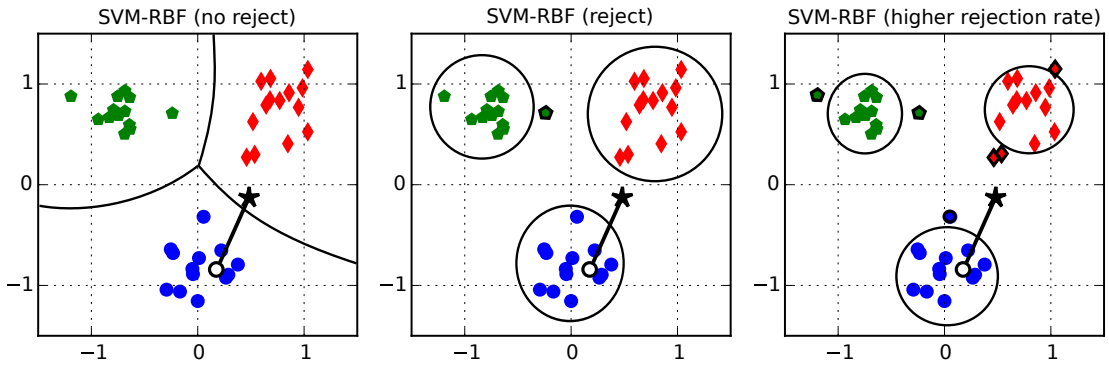


Figure 5.6: Conceptual representation of the proposed classifier security to adversarial examples, using multiclass SVMs with RBF kernels (SVM-RBF), without reject option (no defense, *left*), with reject option (*middle*), and with modified thresholds to increase the rejection rate (*right*). Rejected samples are highlighted with black contours. The adversarial example (black star) is misclassified as a red sample by SVM-RBF (left plot), while SVM-RBF with reject option correctly identifies it as an adversarial example (middle plot). Rejection thresholds can be modified to increase classifier security (right plot), though at the expense of misclassifying more legitimate (i.e., non-manipulated) samples.

otherwise classify $\boldsymbol{x}$ as an adversarial example (i.e., a novel class). In practice, this means that, if no classifier assigns the sample to an existing class (i.e., no value of f is positive), then we simply categorize it as an adversarial example. In this case the system may eventually ask for a human feedback depending on the specific application.

The threshold of each discriminant function (i.e., the biases of the one-versus-all SVMs) can be adjusted to tune the trade-off between the rejection rate of adversarial examples and the fraction of incorrectly-rejected samples (which are not adversarially manipulated), as shown in Fig. 5.6.

Notably, this classifier belongs to the CAP model where for us the unknown class is the adversarial example class.

Chapter 6

Training-time Poisoning Attacks against Machine Learning

A number of online services nowadays rely upon machine learning to extract valuable information from data collected in the wild. This exposes learning algorithms to the threat of data poisoning, i.e., a coordinated attack in which a fraction of the training data is controlled by the attacker and manipulated to subvert the learning process.

The poisoning attack scenario has been already formalized only for two-class learning algorithms. We overcome this limitation presenting in Section 6.1 a Poisoning Attack Scenario, derived from the adversarial framework proposed in Chapter 4, that enables one to envision multiclass classifier poisoning. . A gradient-based algorithm that the attacker can exploit to compute poisoning point is illustrated in Section 6.2 along with the state of art technique for the computation of the gradients required from the algorithm. To date poisoning has been devised only against a limited class of binary learning algorithms. In Section 6.3 we exploit a recent technique called *back-gradient optimization*, originally proposed for hyper-parameter optimization [10, 43, 69, 87], to implement a computationally-efficient poisoning attack. It allows poisoning all the classifiers that can be learned with gradient-based techniques, like neural networks. Despite the increasing use of neural networks, SVM is still one of the most used classifiers. In Section 6.4 we analyze the vulnerabilities of SVM against a specific kind of poisoning attack and we provide a countermeasure to enhance their security. Notably, the proposed approach helps us to shed light between sparsity in feature space and security against poisoning attacks.

6.1 Poisoning Attack Scenarios

When the attacker is able to manipulate the training data the attack is called poisoning. The attacker goal may be to induce an error in some specific class (*Error-Specific Poisoning*) e.g. made a system that recognizes traffic signal, misclassify all stop signal as go signal or made the system not useful at all making it commits

many errors as possible (*Error-Generic Poisoning*).

Here the formulation of this two poisoning attack scenarios is derived. Notably, this formulation includes both, *target* and *indiscriminate* poisoning attack scenarios as the only difference between them is the initial attack sample subset $\mathcal{D}_a$ chosen from the attacker according to her purpose.

The main technical difficulty in devising a poisoning attack is the computation of the poisoning samples, also recently referred to as *adversarial training examples* [58]. This requires solving a bilevel optimization problem in which the outer optimization amounts to maximizing the classification error on an untainted validation set, while the inner optimization corresponds to training the learning algorithm on the poisoned data [73].

Since solving this problem with black-box optimization is too computationally demanding, previous works have exploited gradient-based optimization, along with the idea of *implicit differentiation*. The latter consists of replacing the inner optimization problem with its stationarity (Karush-Kuhn-Tucker, KKT) conditions to derive an implicit equation for the gradient [19, 108, 73, 58].

This approach however can only be used against a limited class of learning algorithms, excluding neural networks and deep learning architectures, due to the inherent complexity of the procedure used to compute the required gradient.

The Gradient-based poisoning attack and the methodology exploited in previous work to compute the needed gradients are briefly reviewed at the end of this section along with their limitations.

6.1.1 Error-Generic Poisoning Attacks

The most common scenario considered in previous work [19, 108, 73] considers poisoning two-class learning algorithms to maximize the classification error. In the multiclass case, it is thus natural to extend this scenario assuming that the attacker is not aiming to cause specific errors, but only *generic* misclassifications. As in [19, 108, 73], this poisoning attack (as any other poisoning attack) requires solving a bilevel optimization, where the inner problem is the learning problem. This can be made explicit by rewriting Eq. (4.1) as:

$$\mathcal{D}_a{}^* \in \arg\max_{\mathcal{D}_a{}' \in \Psi(\mathcal{D}_a)} \quad A(\mathcal{D}_a{}', \boldsymbol{\theta}) = \mathcal{L}(\hat{\mathcal{D}_{val}}, \hat{\boldsymbol{w}})\,, \tag{6.1}$$

$$\text{s.t.} \quad \hat{\boldsymbol{w}} \in \arg\min_{\boldsymbol{w}' \in \mathcal{W}} L(\hat{\mathcal{D}_{tr}} \cup \mathcal{D}_a{}', \boldsymbol{w}')\,, \tag{6.2}$$

where the surrogate data $\hat{\mathcal{D}}$ available to the attacker is divided into two disjoint sets $\hat{\mathcal{D}_{tr}}$ and $\hat{\mathcal{D}_{val}}$. The former, along with the poisoning points $\mathcal{D}_a{}'$ is used to learn the surrogate model, while the latter is used to evaluate the impact of the poisoning samples on untainted data, through the function $A(\mathcal{D}_a{}', \boldsymbol{\theta})$. In this case, the function

$A(\mathcal{D}_a', \boldsymbol{\theta})$ is simply defined in terms of a loss function $\mathcal{L}(\hat{\mathcal{D}_{val}}, \hat{\boldsymbol{w}})$ that evaluates the performance of the (poisoned) surrogate model on $\hat{\mathcal{D}_{val}}$. The dependency of A on $\mathcal{D}_a'$ is thus indirectly encoded through the parameters $\hat{\boldsymbol{w}}$ of the (poisoned) surrogate model. Note that, since the learning algorithm (even if convex) may not exhibit a unique solution in the feasible set $\mathcal{W}$, the outer problem has to be evaluated using the exact solution $\hat{\boldsymbol{w}}$ found by the inner optimization. Worth remarking, this formulation encompasses all previously-proposed poisoning attacks against binary learners [19, 108, 73], provided that the loss function $\mathcal{L}$ is selected accordingly (e.g., using the hinge loss against SVMs [19]). In the multiclass case, one can use a multiclass loss function like the log-loss with softmax activation.

6.1.2 Error-Specific Poisoning Attacks

Here, we assume that the attacker's goal is to cause specific misclassifications (a plausible scenario only for multiclass problems). The poisoning problem remains that given by Eqs. (6.1)-(6.2), though the objective is defined as:

$$A(\mathcal{D}_a', \boldsymbol{\theta}) = -\mathcal{L}(\hat{\mathcal{D}_{val}}', \hat{\boldsymbol{w}})\,, \tag{6.3}$$

where $\hat{\mathcal{D}_{val}}'$ is a set that contains the same data as $\hat{\mathcal{D}_{val}}$, though with different labels, chosen by the attacker. These labels correspond to the desired misclassifications, and this is why there is a minus sign in front of $\mathcal{L}$, i.e., the attacker effectively aims at *minimizing* the loss on her desired set of labels.

6.2 Gradient-Based Poisoning Attack

In this section, the general gradient-based poisoning algorithm is firstly depicted, then how the required gradients have been computed in previous work is discussed.

For some classes of loss functions $\mathcal{L}$ and learning objective functions L, the poisoning problem can be solved through *gradient ascent* if duly modified to reduce his complexity. This is created making the same assumptions made in previous work [19, 108, 73] to reduce the complexity of Problem (6.1)-(6.2): (i) we consider the optimization of one poisoning point $\boldsymbol{x}$ at a time; and (ii) we assume that its label y is initially chosen by the attacker, and kept fixed during the optimization. The poisoning problem can be thus simplified as:

$$\boldsymbol{x}^* \in \underset{\boldsymbol{x}' \in \Psi(\{\boldsymbol{x}, y\})}{\arg\max} \quad A(\{\boldsymbol{x}', y\}, \boldsymbol{\theta}) = \mathcal{L}(\hat{\mathcal{D}_{val}}, \hat{\boldsymbol{w}})\,, \tag{6.4}$$

$$\text{s.t.} \quad \hat{\boldsymbol{w}} \in \underset{\boldsymbol{w}' \in \mathcal{W}}{\arg\min}\ L(\boldsymbol{x}', \boldsymbol{w}')\,. \tag{6.5}$$

The function Ψ imposes constraints on the manipulation of $\boldsymbol{x}$, e.g., upper and lower bounds on its manipulated values. These may also depend on y, e.g., to ensure that

Algorithm 3 Poisoning Attack Algorithm

Input: $\hat{\mathcal{D}_{tr}}$, $\hat{\mathcal{D}_{val}}$, L, $\mathcal{L}$, the initial poisoning point $\boldsymbol{x}^{(0)}$, its label y, the learning rate η, a small positive constant ε.

1: $i \leftarrow 0$ (iteration counter)
2: **repeat**
3: $\quad \hat{\boldsymbol{w}} \in \arg\min_{\boldsymbol{w}'} \; L(\boldsymbol{x}^{(i)}, \boldsymbol{w}')$ (train learning algorithm [1])
4: $\quad \boldsymbol{x}^{(i+1)} \leftarrow \Pi_{\Psi}\left(\boldsymbol{x}^{(i)} + \eta \nabla_{\boldsymbol{x}} A(\{\boldsymbol{x}^{(i)}, y\})\right)$ (Π maps $\boldsymbol{x}$ into the feasible domain)
5: $\quad i \leftarrow i + 1$
6: **until** $A(\{\boldsymbol{x}^{(i)}, y\}) - A(\{\boldsymbol{x}^{(i-1)}, y\}) < \varepsilon$

Output: the final poisoning point $\boldsymbol{x} \leftarrow \boldsymbol{x}^{(i)}$

the poisoning sample is labeled as desired when updating the targeted classifier. Note also that, for notational simplicity, we only report $\boldsymbol{x}'$ as the first argument of L instead of $\hat{\mathcal{D}_{tr}} \cup \{\boldsymbol{x}', y\}$. Thanks to this simplifications, a gradient-based algorithm can be derived and it is here depicted in Algorithm 3. Fig. 6.1 shows an example of poisoning using the depicted algorithm in a multiclass setting that highlights the difference between error-generic and error-specific poisoning attacks scenarios. Notably Algorithm 3 can be exploited to optimize multiple poisoning points too. As in [108], the idea is to perform several passes over the set of poisoning samples, using Algorithm 3 to optimize each poisoning point at a time, while keeping the other points fixed. Line searches can also be exploited to reduce complexity.

Gradient Computations. Provided that the attacker loss function is differentiable w.r.t. $\hat{\boldsymbol{w}}$ and $\boldsymbol{x}$, we can compute the gradient $\nabla_{\boldsymbol{x}} A$ using the chain rule:

$$\nabla_{\boldsymbol{x}} A = \nabla_{\boldsymbol{x}} \mathcal{L} + \frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}}^{\top} \nabla_{\boldsymbol{w}} \mathcal{L} \,, \tag{6.6}$$

where $\mathcal{L}(\hat{\mathcal{D}_{val}}, \hat{\boldsymbol{w}})$ is evaluated on the parameters $\hat{\boldsymbol{w}}$ learned after training (including the poisoning point). The main difficulty here is computing $\frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}}$, i.e., understanding how the solution of the learning algorithm varies w.r.t. the poisoning point.

Under some regularity conditions, this can be done by replacing the inner learning problem with its stationarity (KKT) conditions. For example, this holds if the learning problem L is convex, which implies that all stationary points are global minima [87]. In fact, poisoning attacks have been developed so far only against learning algorithms with convex objectives [19, 108, 73, 58]. The trick here is to

[1] In the case of error-specific poisoning attacks (Sect. 6.1.2), the outer objective in Problem (6.4)-(6.5) is $-\mathcal{L}(\hat{\mathcal{D}_{val}}', \hat{\boldsymbol{w}})$. This can be regarded as a minimization problem, and it thus suffices to modify line 4 in Algorithm 3 to update the poisoning point along the opposite direction.

replace the inner optimization with the implicit function $\nabla_{\boldsymbol{w}} L(\hat{\mathcal{D}}_{tr} \cup \{\boldsymbol{x}, y\}, \hat{\boldsymbol{w}}) = \boldsymbol{0}$, corresponding to its KKT conditions. Then, assuming that it is differentiable w.r.t. $\boldsymbol{x}$, one yields the linear system $\nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{w}} L + \frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}}^{\top} \nabla^2_{\boldsymbol{w}} L = \boldsymbol{0}$. If $\nabla^2_{\boldsymbol{w}} L$ is not singular, we can solve this system w.r.t. $\frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}}$, and substitute its expression in Eq. (6.6), yielding:

$$\nabla_{\boldsymbol{x}} A = \nabla_{\boldsymbol{x}} \mathcal{L} - (\nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{w}} L)(\nabla^2_{\boldsymbol{w}} L)^{-1} \nabla_{\boldsymbol{w}} \mathcal{L} \,. \tag{6.7}$$

This gradient is then iteratively used to update the poisoning point through gradient ascent, as shown in Algorithm 3.[2] Recall that the projection operator Π_{Ψ} is used to map the current poisoning point onto the feasible set Ψ (cf. Eqs. 6.4-6.5).

This is the state-of-the-art approach used to implement current poisoning attacks [19, 108, 73, 58]. The problem here is that computing and inverting $\nabla^2_{\boldsymbol{w}} L$ scales in time as $\mathcal{O}(p^3)$ and in memory as $\mathcal{O}(p^2)$, being p the cardinality of $\boldsymbol{w}$. Moreover, Eq. (6.7) requires solving one linear system per parameter. These aspects make it prohibitive to assess the effectiveness of poisoning attacks in a variety of practical settings.

6.3 Poisoning Neural Networks with Back-gradient

In the previous section the gradient-based algorithm (Algorithm 3) that can be exploited to compute poisoning points has been illustrated. However, using state of the art poisoning strategies to compute the gradient required by the algorithm it can be applied only to a subset of machine learning system namely binary learning algorithms with the attacker loss function $\mathcal{L}$ differentiable and the learning loss L convex. We propose here a new algorithm to compute the gradient of interest based on the idea of back-gradient optimization. It computes the gradient through automatic differentiation, while also reversing the learning procedure to drastically reduce the complexity. Compared to current poisoning strategies, this approach is able to target a wider class of learning algorithms, trained with gradient-based procedures, including neural networks and deep learning architectures.

As we said in the previous section the main difficulty to face in order to apply Algorithm 3 for computing the poisoning points is the computation of the gradient $\frac{\partial \hat{\boldsymbol{w}}}{\partial \boldsymbol{x}}$ i.e., to understand how the solution of the learning algorithm varies w.r.t. the poisoning point.

In previously proposed work it has been done by replacing the inner learning problem with its stationarity (KKT) conditions [19, 108, 73, 58]. However, this approach is computationally costly and memory expensive.

[2] Note that Algorithm 3 can be exploited to optimize multiple poisoning points too. As in [108], the idea is to perform several passes over the set of poisoning samples, using Algorithm 3 to optimize each poisoning point at a time, while keeping the other points fixed.$\mathcal{L}$ine searches can also be exploited to reduce complexity.

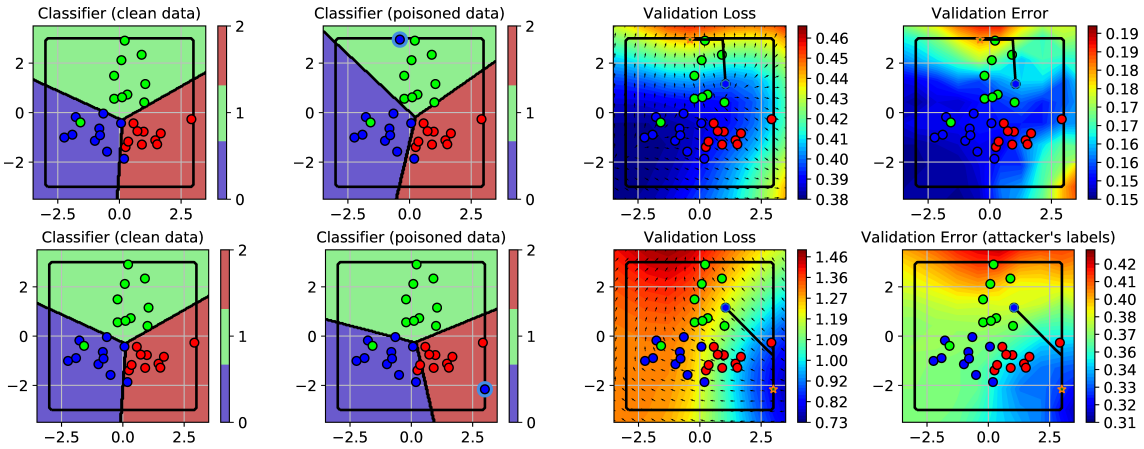


Figure 6.1: Error-generic (top row) and error-specific (bottom row) poisoning attacks on a three-class synthetic dataset, against a multiclass logistic classifier. In the error-specific case, the attacker aims to have red points misclassified as blue, while preserving the labels of the other points. We report the decision regions on the clean (first column) and on the poisoned (second column) data, in which we only add a poisoning point labelled as blue (highlighted with a blue circle). The validation loss $\mathcal{L}(\hat{\mathcal{D}_{val}}, \hat{\boldsymbol{w}})$ and $\mathcal{L}(\hat{\mathcal{D}_{val}}, \hat{\boldsymbol{w}})$, respectively maximized in error-generic and minimized in error-specific attacks, is shown in colors, as a function of the attack point $\boldsymbol{x}$ (third column), along with the corresponding back-gradients (shown as arrows), and the path followed while optimizing $\boldsymbol{x}$. To show that the logistic loss used to estimate the classifier loss on validation data provides a good approximation of the true error, we also report the validation error measured with the zero-one loss on the same data (fourth column).

Algorithm 4 Gradient Descent

Input: initial parameters $\boldsymbol{w}_0$, learning rate η, $\hat{\mathcal{D}}_{\text{tr}}$, L.

1: **for** $t = 0, \ldots, T-1$ **do**
2: $\quad \mathbf{g}_t = \nabla_{\boldsymbol{w}} L(\hat{\mathcal{D}}_{\text{tr}}, \boldsymbol{w}_t)$
3: $\quad \boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta\, \mathbf{g}_t$
4: **end for**

Output: trained parameters $\boldsymbol{w}_T$

To mitigate these issues, as suggested in [42, 43, 69, 58], one can apply conjugate gradient descent to solve a simpler linear system, obtained by a trivial reorganization of the terms in the second part of Eq. (6.7). In particular, one can set $(\nabla^2_{\boldsymbol{w}} L)\ \mathbf{v} = \nabla_{\boldsymbol{w}} \mathcal{L}$, and compute $\nabla_{\boldsymbol{x}} A = \nabla_{\boldsymbol{x}} \mathcal{L} - \nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{w}} L\ \mathbf{v}$. The computation of the matrices $\nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{w}} L$ and $\nabla^2_{\boldsymbol{w}} L$ can also be avoided using Hessian-vector products [86]:

$$(\nabla_{\boldsymbol{x}} \nabla_{\boldsymbol{w}} L)\ \mathbf{z} = \lim_{h \to 0} \frac{1}{h} \left(\nabla_{\boldsymbol{x}} L\left(\boldsymbol{x}', \hat{\boldsymbol{w}} + h\mathbf{z}\right) - \nabla_{\boldsymbol{x}} L\left(\boldsymbol{x}', \hat{\boldsymbol{w}}\right) \right) ,$$
$$(\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}} L)\ \mathbf{z} = \lim_{h \to 0} \frac{1}{h} \left(\nabla_{\boldsymbol{w}} L\left(\boldsymbol{x}', \hat{\boldsymbol{w}} + h\mathbf{z}\right) - \nabla_{\boldsymbol{w}} L\left(\boldsymbol{x}', \hat{\boldsymbol{w}}\right) \right) .$$

Although this approach allows poisoning learning algorithms more efficiently w.r.t. to those propose in state of art work [19, 108, 73], it still requires the inner learning problem to be solved exactly. From a practical perspective, this means that the KKT conditions have to be met with satisfying numerical accuracy. However, as these problems are always solved to a finite accuracy, it may happen that the gradient $\nabla_{\boldsymbol{x}} A$ is not sufficiently precise, especially if convergence thresholds are too loose [43, 69].

It is thus clear that such an approach can not be used, in practice, to poison learning algorithms like neural networks and deep learning architectures, as it may not only be difficult to derive proper stationarity conditions involving all parameters, but also as it may be too computationally demanding to train such learning algorithms with sufficient precision to correctly compute the gradient $\nabla_{\boldsymbol{x}} A$.

Back-gradient Poisoning. We overcome this limitation by exploiting *back-gradient optimization* [43, 69]. This technique has been first exploited in the context of energy-based models and hyperparameter optimization, to solve bilevel optimization problems similar to the poisoning problem discussed before. The underlying idea of this approach is to replace the inner optimization with a set of iterations performed by the learning algorithm to update the parameters $\boldsymbol{w}$, provided that such updates are *smooth*, as in the case of gradient-based learning algorithms. According to [43], this technique allows to computing the desired gradients in the outer problem using the parameters $\boldsymbol{w}_T$ obtained from an incomplete optimization of the

Algorithm 5 Back-gradient Descent

Input: trained parameters $\boldsymbol{w}_T$, learning rate η, $\hat{\mathcal{D}_{tr}}$, $\hat{\mathcal{D}_{val}}$, poisoning point $\boldsymbol{x}'$, y, attacker loss function $\mathcal{L}$, learner's objective L.
initialize $d\boldsymbol{x} \leftarrow \mathbf{0}$, $d\boldsymbol{w} \leftarrow \nabla_{\boldsymbol{w}}\mathcal{L}(\hat{\mathcal{D}_{val}}, \boldsymbol{w}_T)$

1: **for** $t = T, \ldots, 1$ **do**
2: $\quad d\boldsymbol{x} \leftarrow d\boldsymbol{x}' - \eta\, d\boldsymbol{w}\nabla_{\boldsymbol{x}}\nabla_{\boldsymbol{w}}L(\boldsymbol{x}', \boldsymbol{w}_t)$
3: $\quad d\boldsymbol{w} \leftarrow d\boldsymbol{w} - \eta\, d\boldsymbol{w}\nabla_{\boldsymbol{w}}\nabla_{\boldsymbol{w}}L(\boldsymbol{x}', \boldsymbol{w}_t)$
4: $\quad \mathbf{g}_{t-1} = \nabla_{\boldsymbol{w}_t}L(\boldsymbol{x}', \boldsymbol{w}_t)$
5: $\quad \boldsymbol{w}_{t-1} = \boldsymbol{w}_t + \alpha\mathbf{g}_{t-1}$
6: **end for**

Output: $\nabla_{\boldsymbol{x}}A = \nabla_{\boldsymbol{x}}\mathcal{L} + d\boldsymbol{x}$

inner problem (after T iterations). This represents a significant computational improvement compared to traditional gradient-based approaches, since it only requires a reduced number of training iterations for the learning algorithm. This is especially important in large neural networks and deep learning algorithms, where the computational cost per iteration can be high. Then, assuming that the inner optimization runs for T iterations, the idea is to exploit reverse-mode differentiation, or *back-propagation*, to compute the gradient of the outer objective. However, using back-propagation in a naïve manner would not work for this class of problems, as it requires storing the whole set of parameter updates $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T$ performed during training, along with the forward derivatives. These are indeed the elements required to compute the gradient of the outer objective with a *backward* pass (in [69] more details about this point are given). This process can be extremely memory-demanding if the learning algorithm runs for a large number of iterations T, and especially if the number of parameters $\boldsymbol{w}$ is large (as in deep networks). Therefore, to avoid storing the whole training trajectory $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_T$ and the required forward derivatives, [43] and [69] proposed to compute them directly during the backward pass, by *reversing* the steps followed by the learning algorithm to update them. Computing $\boldsymbol{w}_T, \ldots, \boldsymbol{w}_1$ in reverse order w.r.t. the forward step is clearly feasible only if the learning procedure can be exactly traced backwards. Nevertheless, this happens to be feasible for a large variety of gradient-based procedures, including gradient descent with fixed step size, and stochastic gradient descent with momentum.

We leverage back-gradient descent to compute $\nabla_{\boldsymbol{x}}A$ (Algorithm 5) by reversing a standard gradient-descent procedure with fixed step size that runs for a truncated training of the learning algorithm to T iterations (Algorithm 4). Notably, lines 2-3 in Algorithm 5 can be efficiently computed with Hessian-vector products, as discussed before. We exploit this algorithm to compute the gradient $\nabla_{\boldsymbol{x}}A$ in line 4 of our

poisoning attack algorithm (Algorithm 3). In this case, line 3 of Algorithm 3 is replaced with the incomplete optimization of the learning algorithm, truncated to T iterations. Note finally that, as in [43, 69], the time complexity of our back-gradient descent is $\mathcal{O}(T)$. This drastically reduces the complexity of the computation of the outer gradient, making it feasible to evaluate the effectiveness of poisoning attacks also against large neural networks and deep learning algorithms. Moreover, this outer gradient can be accurately estimated from a truncated optimization of the inner problem with a reduced number of iterations. This allows for a tractable computation of the poisoning points in Algorithm 3, since training the learning algorithm at each iteration can be prohibitive, especially for deep networks.

6.4 Securing Kernel-based Classifiers from Poisoning Attacks

In many applications, after the classification stage, users are asked to supply the correct label to the system (e.g. in the spam classification). The labels collected from them are used to retrain the system improving its performance. A malicious user can therefore, on purpose provide a wrong label in order to poison the classifier training dataset. This specific poisoning scenario where users can provide only the label is called *label flip* when the target classifier is binary or more in general *adversarial label noise*. The training dataset of this kind of system contains typically hundreds of thousands of samples and an attacker can be able to poison only a little fraction of them. Therefore, label flip can be seen as *sparse* attacks in terms of the training points that are manipulated or injected from the attacker. Support Vector Machines (SVMs) are a well-known and widely-used learning algorithm. They make their decisions based on a subset of training samples, known as support vectors. In application where the labels of a subset of the training samples can be manipulated this behavior poses risks to system security. To the best of our knowledge, the only specific countermeasure against label-flip attacks for this kind of classifier is the one proposed in [18]: it is an approach called Label Noise Robust SVM (LN-SVM) that heuristically try to enforces the classifier to increase the number of support vector, to enhance the stability of the decision function with respect to changes of training labels. Another SVM variant that decrease the impact of each single point compared to the original one is the Least-Square SVM [101] (LS-SVM) that uses a quadratic loss function instead of hinge loss, considering all the point instead of just some support vector in its decision function. We propose below a more theoretically-sound countermeasure rooted in recent finding about the relationship between regularization and robust optimization.

6.4.1 Dual Infinity-norm Support Vector Machines

In Section 5.4 the connection between regularization and robustness has been clarified. Label-flip attacks can be seen as *sparse* attacks in terms of the influenced training points since, typically, only the labels of few training samples can be manipulated by the attacker. The idea behind the proposed method is thus to exploit ℓ_∞ regularization to enforce more evenly-distributed α weights on the *training data*, similarly to the intuition in [18]. The reason is that this would decrease the impact of every single point during learning of the decision function. This effect is here obtained by training a (linear) Infinity-norm SVM (that we call Dual Infinity-norm SVM) directly in the kernel space, i.e., using the kernel matrix as the input training data, to learn a discriminant function of the form $f(\boldsymbol{x}) = \sum_{i=1}^{\mathsf{n}} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b$, where $K(\cdot, \cdot)$ is the kernel function, and $\{\boldsymbol{x}_i\}_{i=1}^{\mathsf{n}}$ and $\{\alpha_i\}_{i=1}^{\mathsf{n}}$ are respectively the training samples and their α weights. Under this setting, the α values and the bias b are obtained by solving the following *linear programming* problem:

$$\min_{\boldsymbol{\alpha}, b} \quad \|\boldsymbol{\alpha}\|_\infty + C \sum_{i=1}^{\mathsf{n}} (1 - y_i f(\boldsymbol{x}_i))_+ \ . \tag{6.8}$$

Notably, this approach can be used also with kernels that are not necessarily positive semi-definite (i.e., indefinite kernels).

The difference between the presented Dual Infinity-norm SVM, the standard SVM and the previously proposed classifier is shown in a two-dimensional example in Fig. 6.2.
The considered dataset is Gaussian with mean $[y, 0]$ (for class y) and diagonal covariance matrix equal to $\mathrm{diag}([0.5, 0.5])$. 60 samples have been included in the training set and 40 in the testing set. The adversarial label-flip attack has been used to flip 18 training labels. We set $C = 1$ for SVM and LN-SVM, and $C = 0.01$ for LS-SVM and Dual Infinity-Norm SVM. From the figure one can appreciate how the Dual Infinity-norm SVM is effectively able to spreads in a more uniform manner the (absolute) weight values α over the training samples. Note indeed that the decision hyperplane obtained by Dual Infinity-norm SVM under attack, and the corresponding test error are less affected by the attack.

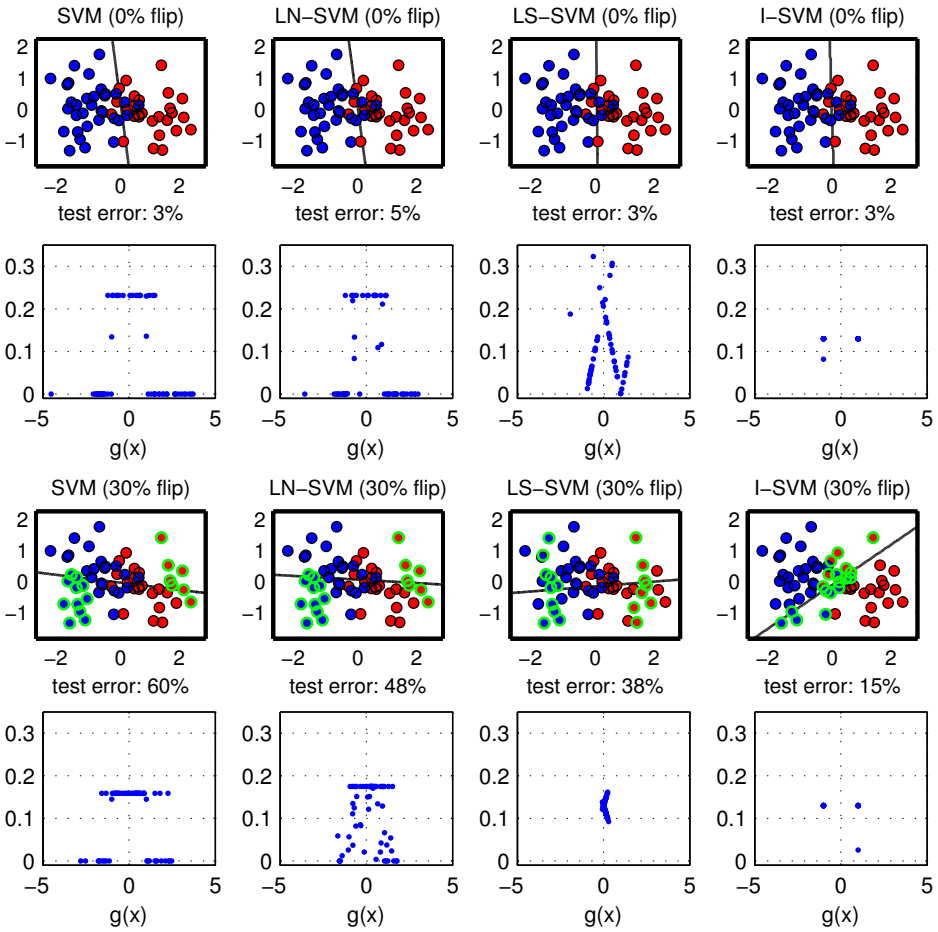


Figure 6.2: Decision boundaries for the SVM, the LN-Robust SVM [18] (with S=0.1), the LS-SVM [101], and the Dual Infinity-norm SVM, respectively trained on untainted (first row) and tainted (third row) data. Adversarial label flips are highlight with green circles. For each SVM, we also report the α values assigned to the training samples against the corresponding $g(\boldsymbol{x})$ values.

Chapter 7

Experimental Evaluation

In this Chapter, we validate some of the methods that are proposed through this thesis using the methodology explained in Section 4.5.

In Section 7.1 we report the results related to the study of the relationship between sparsity and security to evasion attacks. The experiments aimed to assess the effectiveness of the poisoning attack methodology against multiclass classifier and those aimed to validate a countermeasure for Kernel-based classifier against label flip that we are proposing are then illustrated in Section 7.2.

7.1 Evasion

In this section the security of the countermeasure against linear classifier evasion namely octagonal regularizer that we present in Section 5.4 is experimentally assessed.

7.1.1 Trading sparsity for security: octagonal regularization

In Section 5.4 we discuss the trade-off between sparsity of linear classifier weight and their security. The aforementioned trade-off is here assessed on different security-related dataset illustrated below to evaluate whether and to what extent the choice of a proper regularizer may have a significant impact on practice. We further analyze the weight distribution of SVMs using different regularizers, included the proposed Octagonal Regularizer to provide a better understanding of how sparsity is related to classifier security under the considered evasion attacks.

Dataset

Here we describe the security-related dataset that we use in the presented experiments. We propose some experiments also on a Handwritten Recognition dataset as being composed by images it is useful to get more insights about the created attack samples that is described below as well.

PDF Malware Detection. Nowadays PDF is the most used document type due to the fact that it presents documents in an independent manner from the operative systems. A PDF document can host not only text and images but also JavaScript and Flash scripts. This makes it one of the most exploited vectors for conveying malware (i.e., malicious software).

The used Pdf Malware detection dataset is made up of about 5500 legitimate and 6000 malicious PDF files. We represent every file using the 114 features that are described in [72]. They consist of the number of occurrences of a predefined set of keywords, where every keyword represents an action performed by one of the objects that are contained in the PDF file (e.g., opening another document that is stored inside the file).

An attacker cannot trivially remove keywords from a PDF file without corrupting its functionality. Conversely, she can easily add new keywords by inserting new object's operations. For this reason, we simulate this attack by only considering feature increments (decrementing a feature value is not allowed). Accordingly, the most convenient strategy to mislead a malware detector (classifier) is thus to insert as many occurrences of a *given* keyword as possible, which is a sparse attack.

Spam Filtering. This is a well-known application subject to adversarial attacks. Most spam filters include an automatic text classifier that analyzes the email's body text. In the simplest case, Boolean features are used, each representing the presence or absence of a given term, as in the dataset used in the reported experiments. Feature addition and removal are both feasible operations and are equally costly for an attacker. In the presented experiment the TREC 2007 dataset is used. It consists of about 25000 legitimate and 50000 spam emails [31]. We extract a dictionary of terms (features) from the first 5000 emails (in chronological order) using the same parsing mechanism of SpamAssassin, and then select the 200 most discriminant features according to the information gain criterion [96].

Handwritten Digit Classification. The Handwritten Digit Classification problem involves 10 classes (each corresponding to a digit, from 0 to 9). We used the MNIST handwritten digit dataset [61]. In that dataset, each digit image consists of $28 \times 28 = 784$ pixels, ranging from 0 to 255 (images are in gray-scale). We divide each pixel value by 255 and use it as a feature.

Classifiers

We consider different versions of the SVM classifier obtained by combining the hinge loss with the different regularizers shown in Fig. 5.5.

Sparsity and Security Measures

We evaluate the degree of sparsity S of a given linear classifier as the fraction of its weights that are equal to zero:

$$S = \frac{1}{\mathsf{d}} |\{w_j | w_j = 0, j = 1, \ldots, \mathsf{d}\}| \,, \tag{7.1}$$

being $|\cdot|$ the cardinality of the set of null weights.

To evaluate security of linear classifiers, we define a measure E of *weight evenness*, similarly to [59, 15], based on the ratio of the ℓ_1 and ℓ_∞ norm:

$$E = \frac{\|\boldsymbol{w}\|_1}{\mathsf{d}\|\boldsymbol{w}\|_\infty} \,, \tag{7.2}$$

where dividing by the number of features d ensures that $E \in \left[\frac{1}{\mathsf{d}}, 1\right]$, with higher values denoting more evenly-distributed feature weights. In particular, if only a weight is not zero, then $E = \frac{1}{\mathsf{d}}$; conversely, when all weights are equal to the maximum (in absolute value), $E = 1$.

Experimental Setup

We randomly select 500 legitimate and 500 malicious samples from each dataset, and equally subdivide them to create a training and a test set. We optimize the regularization parameter C of each SVM (along with λ and ρ for the Elastic-net and Octagonal SVMs, respectively) through 5-fold cross-validation, maximizing the following objective on the training data:

$$\mathrm{AUC} + \alpha E + \beta S \tag{7.3}$$

where AUC is the area under the ROC curve, and α and β are parameters defining the trade-off between security and sparsity. We set $\alpha = \beta = 0.1$ for the PDF and digit data, and $\alpha = 0.2$ and $\beta = 0.1$ for the spam data, to promote more secure solutions in the latter case. These parameters allow us to accept a marginal decrease in classifier security only if it corresponds to much sparser feature weights. After classifier training, we perform evasion attacks on all malicious test samples and evaluate the corresponding performance as a function of the number of features modified by the attacker. We repeat this procedure five times and report the average results of the original and modified test data.

Experimental Results

We consider first PDF malware and spam detection. In these applications, as mentioned before, only sparse evasion attacks make sense, as the attacker aims to minimize the number of modified features. In Fig. 7.1, we report the AUC at 10% false

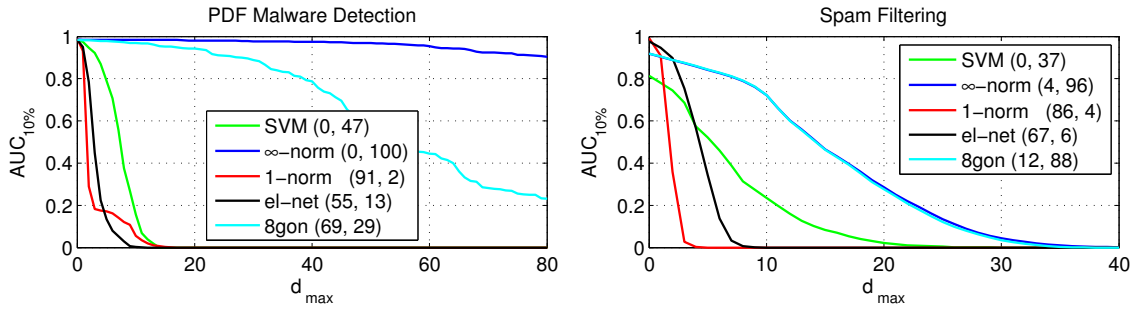


Figure 7.1: Classifier performance under attack for PDF malware and spam data, measured in terms of $\text{AUC}_{10\%}$ against an increasing number $d_{\max}$ of modified features. For each classifier, we also report (S, E) percentage values (Eqs. 7.1-7.2) in the legend.

positive rate for the considered classifiers, against an increasing number of words/keywords changed by the attacker. This experiment shows that the most secure classifier under sparse evasion attacks is the Infinity-norm SVM, since its performance degrades more gracefully under attack. This is an expected result, given that, in this case, infinity-norm regularization corresponds to the dual norm of the attacker's cost/distance function. Notably, the Octagonal SVM yields reasonable security levels while achieving much sparser solutions, as expected (*cf.* the sparsity values S in the legend of Fig. 7.1). This experiment really clarifies how much the choice of a proper regularizer can be crucial in real-world adversarial applications.

By looking at the values reported in Fig. 7.1, it may seem that the security measure E does not properly characterize classifier security under attack; e.g., note how Octagonal SVM exhibits lower values of E despite being more secure than SVM on the PDF data. The underlying reason is that the attack implemented in the PDF data only considers feature increments, while E generically considers any kind of manipulation. Accordingly, one should define alternative security measures depending on specific kinds of data manipulation. However, the security measure E allows us to properly tune the trade-off between security and sparsity also in this case and, thus, this issue may be considered negligible.

Finally, to visually demonstrate the effect of sparse and dense evasion attacks, we report some results on the MNIST handwritten digits. In Fig. 7.2, we show the "9" digit image modified by the attacker to have it misclassified by the classifier as an "8". These modified digits are obtained by solving Problem (5.1) through a simple projected gradient-descent algorithm, as in [12]. Note how dense attacks only produce a slightly-blurred effect on the image, while sparse attacks create more evident visual artifacts. By comparing the values of $g(\boldsymbol{x})$ reported in Fig. 7.2, one may also note that this simple example confirms again that Infinity-norm and Octagonal SVM are more secure against sparse attacks, while SVM and Elastic-net SVM are more secure against dense attacks.

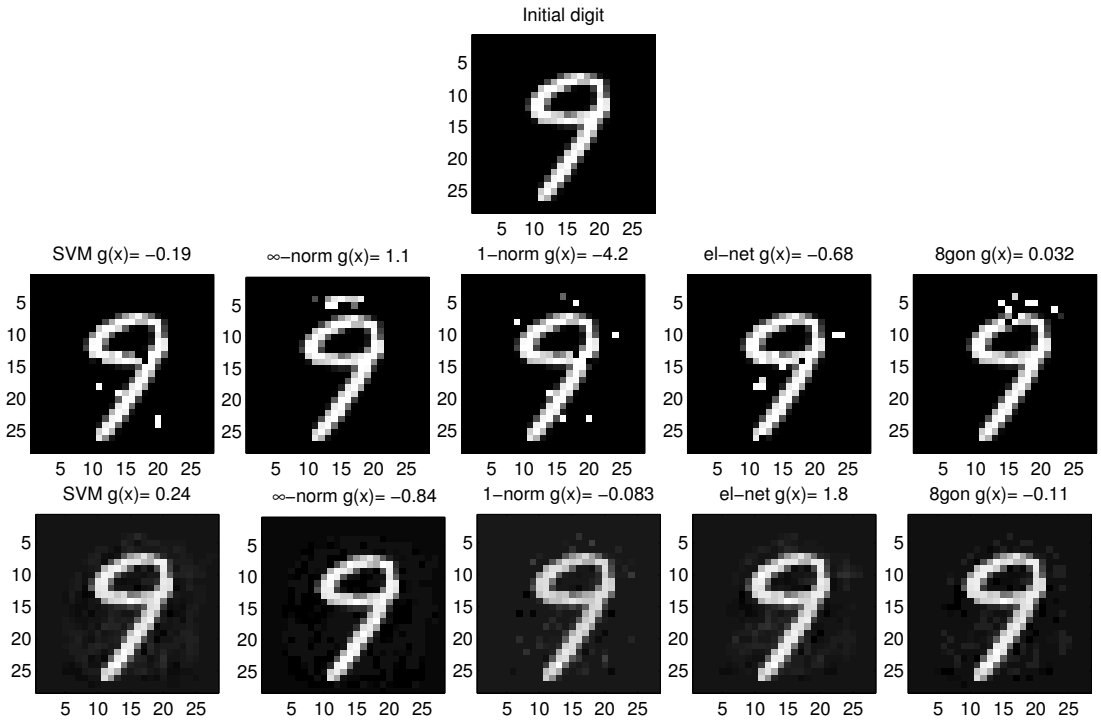


Figure 7.2: Initial digit "9" (*first row*) and its versions modified to be misclassified as "8" (*second and third row*). Each column corresponds to a different classifier (from *left* to *right* in the second and third row): SVM, Infinity-norm SVM, 1-norm SVM, Elastic-net SVM, Octagonal SVM. *Second row*: sparse attacks (ℓ_1), with $d_{\max} = 2000$. *Third row*: dense attacks (ℓ_2), with $d_{\max} = 250$. Values of $g(\boldsymbol{x}) < 0$ denote a successful classifier evasion (i.e., more vulnerable classifiers).

Discussion

We have shown on real-world adversarial applications that the choice of a proper regularizer is crucial. In fact, in the presence of sparse attacks, Infinity-norm SVMs can drastically outperform the security of standard SVMs. We believe that this is an important result, as (standard) SVMs are widely used in security-related tasks without taking the risk of adversarial attacks too much in consideration. Moreover, we have shown that the new octagonal regularizer that we propose really enables trading sparsity for a marginal loss of security under sparse evasion attacks. This is extremely useful in applications where sparsity and computational efficiency at test time are crucial. When dense attacks are instead deemed more likely, the standard SVM may be retained a good compromise. In that case, if sparsity is required, one may trade some level of security for sparsity using the Elastic-net SVM.

7.2 Poisoning

We open this section evaluating the effectiveness of the poisoning attack that we propose in Section 6.3 that for the first time allows to poisoning all multiclass classifiers that can be trained with a gradient-based algorithm. Finally, we assess the security of the countermeasure to label flip attack against Kernel-based classifiers that is proposed in Section 6.4.

7.2.1 Poisoning Neural Network with Back-gradient

Using the state of the art techniques only binary classifiers with easily-derivable KKT conditions are poisonable. In Section 6.3 we propose an approach that allows one to poison each classifier that can be trained with gradient-based learning techniques. The datasets and the experimental setup that are used in the presented experiments are here firstly presented. The effectiveness of the proposed back-gradient poisoning attack is then evaluated. Whether poisoning samples can be *transferred* across different learning algorithms is investigated and the impact of error-generic and error-specific poisoning attacks is assessed. Finally, we report the first proof-of-concept adversarial training examples computed by poisoning a convolutional neural network in an *end-to-end* manner (i.e., not just using a surrogate model trained on the deep features, as in [58]).

Dataset

We empirically evaluate the effectiveness of the poisoning attack that we propose in several applications, including spam filtering, malware detection (that are described below) and the MNIST datasets that is described in the previous section.

Spam Filtering.
In the presented experiments we used the `Spambase` dataset [23]. It consists of a collection of $4,601$ emails, including $1,813$ spam emails. Each email is encoded as a feature vector consisting of 54 binary features.

Ransomware. Ransomware is a very recent kind of malware which encrypts the data of infected machine and requires the victim to pay a ransom to obtain the decryption key. The `Ransomware` data [97] consists of 530 ransomware samples and 549 benign applications. This dataset has 400 binary features accounting for different sets of actions, API invocations, and modifications in the file system and registry keys during the execution of the software.

Experimental Setup

We consider the following leaning algorithms: (*i*) Multi-Layer Perceptrons (MLPs) with one hidden layer consisting of 10 neurons; (*ii*) Logistic Regression (LR); and

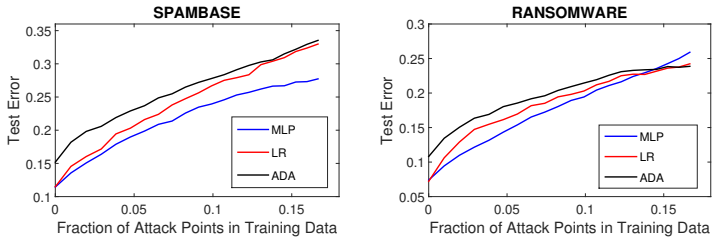


Figure 7.3: Results of PK poisoning attacks.

(*iii*) Adaline (ADA). For MLPs, we have used hyperbolic tangent activation functions for the neurons in the hidden layer, and softmax activations in the output layer. Moreover, for MLPs and LR, we use the cross-entropy (or log-loss) as the loss function, while we use the mean squared error for ADA.

Effectiveness and Transferability of Poisoning Samples

Here the effectiveness and the transferability of the samples created with the presented poisoning attack is investigated assuming that the attacker aims to cause a denial of service, and thus runs a poisoning *availability* attack whose goal is simply to maximize the classification error.

Accordingly, we exploit the Algorithm 3 to injecting up to 20 poisoning points in the training data. The poisoning points are initialized by cloning training points and flipping their labels. We set the number of iterations T for obtaining stable back-gradients to 200, 100, and 80, respectively for MLPs, LR and ADA. We consider two distinct settings: PK attacks, in which the attacker is assumed to have full knowledge of the attacked system (for a worst-case performance assessment); and LK-SL attacks, in which she knows everything except for the learning algorithm, and thus she uses a surrogate learner $\hat{\mathcal{M}}$. This scenario, as we discuss in Sect. 4.2, is useful to assess the *transferability* property of the attack samples. To the best of our knowledge, this has been demonstrated in [12, 83] for evasion attacks (i.e., adversarial *test* examples) but never for poisoning attacks (i.e., adversarial *training* examples). To this end, we optimize the poisoning samples using alternatively MLPs, LR or ADA as the surrogate learner, and then we evaluate the impact of the corresponding attacks against the other two algorithms.

The experimental results, that are shown in Figs. 7.3-7.4, are averaged on 10 independent random data splits. In each split, 100 samples are used for training and 400 for validation, i.e., to respectively construct $\mathcal{D}_{\rm tr}$ and $\mathcal{D}_{\rm val}$. Recall indeed that in both PK and LK-SL settings, the attacker has perfect knowledge of the training set used to learn the true (attacked) model, i.e., $\hat{\mathcal{D}}_{\rm tr} = \mathcal{D}_{\rm tr}$. The remaining samples are used for testing, i.e., to assess the classification error under poisoning.[1]

[1]Note indeed that the validation error only provides a biased estimate of the true classification error, as it is used by the attacker to optimize the poisoning points [19].

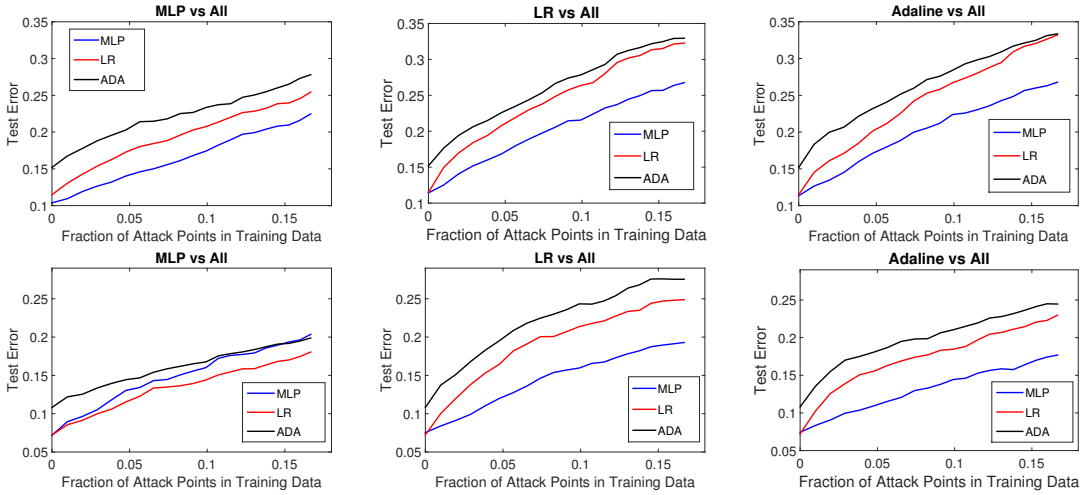


Figure 7.4: Results of LK-SL poisoning attacks (transferability of poisoning samples) on Spambase (top row) and Ransomware (bottom row).

From Fig. 7.3 is observable that PK poisoning attacks can significantly compromise the performance of all the considered classifiers. In particular, on `Spambase`, they cause the classification error of ADA and LR to increase up to 30% even if the attacker only controls 15% of the training data. Although the MLP is more resilient to poisoning than these linear classifiers, its classification error also increases significantly, up to 25%, which is not tolerable in several practical settings. The results of PK attacks on `Ransomware` are similar, although the MLP seems as vulnerable as ADA and LR in this case.

Regarding LK-SL poisoning attacks, is visible from Fig. 7.4 that the attack points generated using a linear classifier (either ADA or LR) as the surrogate model have a very similar impact on the other linear classifier. In contrast, the poisoning points crafted with these linear algorithms have a lower impact on the MLP, although its performance is still noticeably affected. When the MLP is used as the surrogate model, instead, the performance degradation of the other algorithms is similar. However, the impact of these attacks is much lower. To summarize, the presented results show that the attack points can be effectively transferred across linear algorithms and also have a noticeable impact on (nonlinear) neural networks. In contrast, transferring poisoning samples from nonlinear to linear models seems to be less effective.

Error-generic and Error-specific Attack

Here error-generic and error-specific poisoning strategies are evaluated against a multiclass LR classifier using softmax activation and the log-loss as loss function. The MNIST dataset is used to study the error variation on different dataset classes.

In case of Error-generic Attack, the attacker aims to maximize the classification

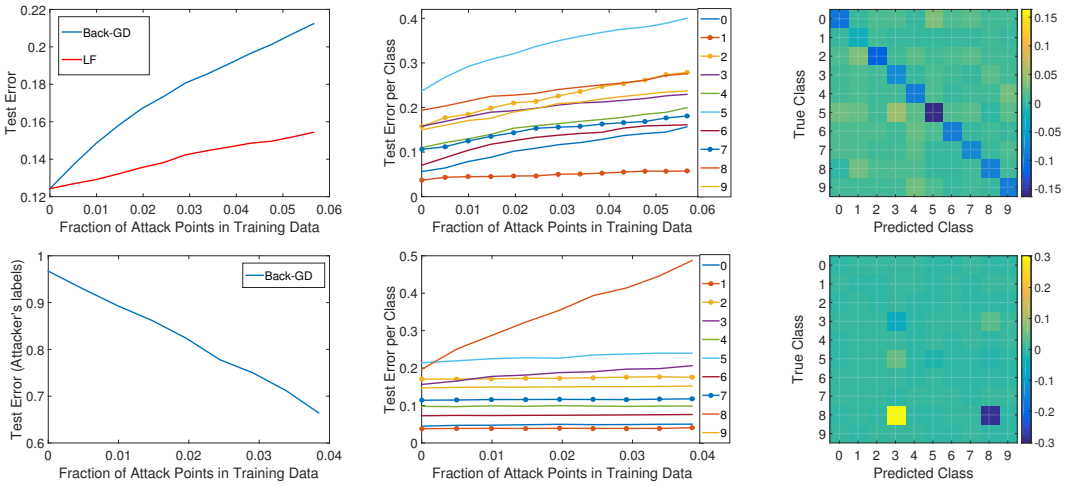


Figure 7.5: Error-generic (top row) and error-specific (bottom row) poisoning against multiclass LR on the MNIST data. In the first column, the test error (which, for error-specific poisoning attacks is computed using the attacker's labels instead of the true labels, and so it decreases while approaching the attacker's goal is reported). In the second column, the error per class, i.e., the probability of misclassifying a digit given that it belongs to the class reported in the legend is shown. In the third column, the difference between the confusion matrix obtained under poisoning (after injecting the maximum number of poisoning samples) and that obtained in the absence of attack, to highlight how the errors affect each class is reported.

error regardless of the resulting kind of errors, as described in Sect. 6.1. This is thus an *availability* attack, aimed to cause a denial of service. We generate 10 independent random splits using 1000 samples for training, 1000 for validation, and 8000 for testing. To compute the back-gradients $\nabla_{\boldsymbol{x}_c}\mathcal{A}$ required by our poisoning attack, we use $T = 60$ iterations. We initialize the poisoning points by cloning randomly-chosen training points and changing their label at random. In addition, the presented poisoning attack strategy is compared against a label-flip attack in which the attack points are drawn from the validation set and their labels are flipped at random. In both cases, up to 60 attack points are injected into the training set. The results are shown in Fig. 7.5 (top row). Note first that the proposed error-generic poisoning attack almost doubles the classification error in the absence of poisoning, with less than 6% of poisoning points. It is much more effective than random label flips and, as expected, it causes a similar increase of the classification error over all classes (although some classes are easier to poison, like digit 5). This is even more evident from the difference between the confusion matrix obtained under 6% poisoning and the one obtained in the absence of attacks.

In order to test the Error-specific case we assume that the attacker aims to misclassify 8s as 3s, while not having any preference regarding the classification of

the other digits. This can be thus regarded as an *availability* attack, targeted to cause the misclassification of a specific set of samples. 10 independent random splits with 1000 training samples are generated with 4000 samples for validation, and 5000 samples for testing. Recall that the goal of the attacker in this scenario is described by Eq. (6.3). In particular, she aims at minimizing $L(\hat{\mathcal{D}}'_{val}, \hat{\boldsymbol{w}})$, where the samples in the validation set $\hat{\mathcal{D}}'_{val}$ are re-labeled according to the attacker's goal. Here, the validation set thus only consists of digits of class 8 labeled as 3. We set T equal to 60 to compute the back-gradient used in our poisoning attack, and inject up to 40 poisoning points into the training set. The poisoning points are initialized by cloning randomly-chosen samples from the classes 3 and 8 in the training set, and flipping their label from 3 to 8, or vice-versa. Only these two classes are here considered as they are the only two actively involved in the attack.

The results are shown in Fig. 7.5 (bottom row). We can observe that only the classification error rate for digit 8 is significantly affected, as expected. In particular, it is clear from the difference of the confusion matrix obtained under poisoning and the one obtained in the absence of attack that most of the 8s are misclassified as 3s. After adding less than 4% of poisoning points, in fact, the error rate for digit 8 increases approximately from 20% to 50%. Note that, as a side effect, the error rate of digit 3 also slightly increases, though not to a significant extent.

Poisoning Deep Neural Networks

Finally, a proof-of-concept experiment to show the applicability of our attack algorithm to poison a deep network is reported. Notably, it allows one to poisoning the network in an *end-to-end* manner, i.e., accounting for all weight updates in each layer (instead of using a surrogate model trained on a frozen deep feature representation [58]). To this end, the convolutional neural network (CNN) that was proposed in [61] is considered for the classification of the MNIST digit data, which requires optimizing more than $450,000$ parameters.[2] In this proof-of-concept attack, 10 poisoning points are injected into the training data, and the experiment is repeated on 5 independent data splits, considering $1,000$ samples for training, and $2,000$ for validation and testing. For simplicity, only the classes of digits 1, 5, and 6 are considered in this case. Algorithm 3 is used to craft each single poisoning point, but, similarly to [108], they are optimized iteratively, making 2 passes over the whole set of poisoning samples. The line search exploited in [108] is also used, instead of a fixed gradient step size, to reduce the attack complexity (i.e., the number of training updates to the deep network).

Under this setting, however, the generated attack points only slightly increase the classification error, though not significantly, while random label flips do not have any substantial effect. For comparison, also multiclass LR classifier is attacked

[2] the implementation that is used is the one available at https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_mnist.py.

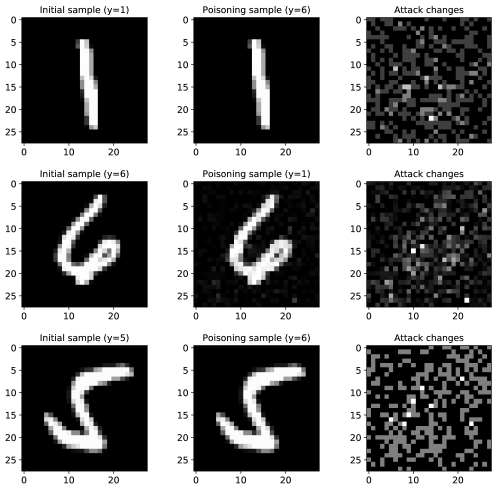


Figure 7.6: Poisoning samples targeting the CNN.

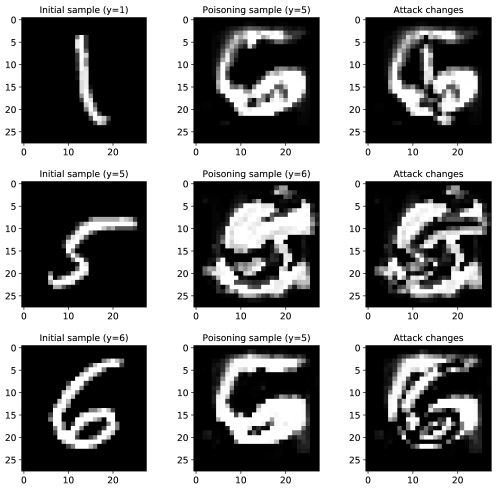


Figure 7.7: Poisoning samples targeting the LR.

under the same setting, yielding an increase of the error rate from 2% to 4.3% with poisoning attacks, and to only 2.1% with random label flips. This shows that, at least in this simple case, deep networks seem to be more resilient against (a very small fraction of) poisoning attacks (i.e., less than 1%). Some of the poisoning samples crafted against the CNN and the LR are shown in Figs. 7.6 and 7.7. In the figure the initial digit (and its true label y), its poisoned version (and its label y_c), and the difference between the two images in absolute value (rescaled to visually appreciate the modified pixels) are shown. Notably, similarly to adversarial test examples, also poisoning samples against deep networks are visually indistinguishable from the initial image (as in [58]), while this is not the case when targeting the LR classifier. This might be due to the specific shape of the decision function learned by the deep network in the input space, as explained in the case of adversarial test examples [102, 47]. This aspect needs to further investigation and it is here left for future work along with a more systematic security evaluation of deep networks against poisoning attacks. We finally execute a simple *transferability* experiment, in which the poisoning samples crafted against the LR classifier are used to attack the CNN, and vice-versa. In the former case, the attack is totally ineffective (as the minimal modifications to the CNN-poisoning digits are clearly irrelevant for the LR classifier), while in the latter case it has a similar effect to that of random label flips.

Discussion

The empirical evaluation on spam filtering and malware detection shows that neural networks can be significantly compromised even if the attacker only controls a small fraction of training points. It also empirically shows that poisoning samples designed against one learning algorithm can be rather effective also in poisoning another

algorithm, highlighting an interesting *transferability* property, as that shown for evasion attacks (a.k.a. adversarial test examples) [12, 107, 83].

The main limitation of the presented experiments is that we do not provide an extensive evaluation of poisoning attacks against deep networks to thoroughly assess their security. The preliminary experiments that we report seem to show that they can be more resilient against this threat than other learning algorithms. This may be due to their higher capacity and number of parameters, which may allow the network to memorize the poisoning samples without affecting what has been correctly learned elsewhere. However, the attack that we tested is designed to maximize the overall classification error, thus attacks with less ambitious goals (such as targeted attacks aimed at misclassifying only a small subset of samples, as in [58]), may still be more effective also against this kind of classifiers. Therefore, a more complete and systematic analysis remains to be performed.

7.2.2 Securing Kernel-based Classifier from Poisoning Attacks

When some training set labels are acquired from users a malicious one can provide wrong labels in order to intelligently threat the system obtaining the effect that she desires. This attack is called label flip in case it is exploited on two class datasets. In Section 6.4 we propose a countermeasure for SVM against label flip. We here empirically validate it on a Pdf filtering dataset.

Dataset

In the reported experiment we use the Pdf Filtering dataset called Lux0r [32]. This dataset contains around $12,000$ malicious PDFs and about $5,000$ benign samples. Every PDF is represented by 736 features, each representing the number of occurrences of a specific Javascript function (API call) into the PDF. Each API call corresponds to an action performed by one of the objects that belong to the PDF.

Experimental Setup

Firstly, we normalize data in $[-1, 1]$ using min-max normalization. Then we randomly split the data into 5 distinct training and test set pairs, consisting of 60% and 40% of the data.

Label Flip Attacks

We consider two different kinds of label flip attacks. In both cases we set a constraint to the fraction of labels that the adversary can change, to reflects a likely limitation in real-world scenarios.

Random label flip. is a baseline attack, which consists of flipping the labels of a randomly-chosen fraction of training samples, without exploiting any knowledge of

the targeted classifier.

Adversarial Label-Flip Attack (ALFA-Tilt). is a different attack proposed in [18, 109]. It assumes a skilled attacker whose aim is to maximize the classifier error on untainted (testing) data.

Since finding the subset of samples whose label flip maximizes the testing error is a non-trivial problem, the authors devised a heuristic approach that maximizes a surrogate measure of the testing error, namely, the angle between the decision hyperplane found by the untainted classifier and the one under attack. We consider the worst case namely perfect knowledge, although in real scenarios the attacker is likely to have only a limited knowledge of the system.

Results

The averaged results on the Spam dataset, for different C values under random and ALFA-tilt attacks, are respectively reported in Fig. 7.8.

Notably, the LS-SVM and Infinity-norm SVM attain the best performance for low C values, as the effect of regularization is stronger. These classifiers are nevertheless the most secure under both the random and the ALFA-tilt attack. The performance with the best C for each classifier are dataset-dependent, however Infinity-norm SVM is clearly able to achieve good performance and it has the highest accuracy under the ALFA-tilt attack.

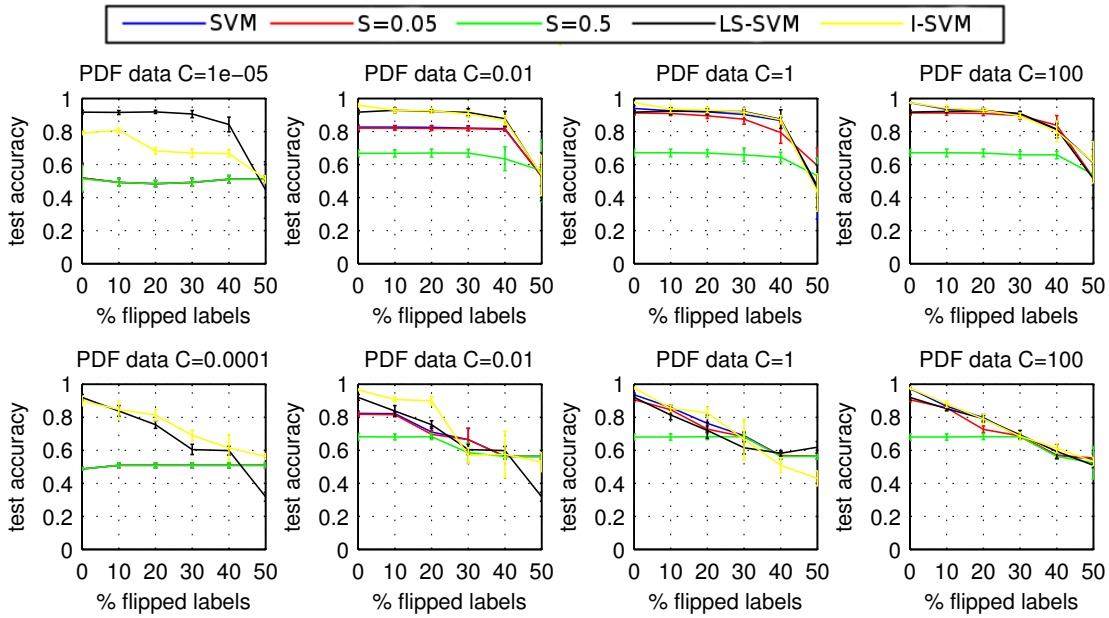


Figure 7.8: Random (first row) and label tilt attack (second row) against SVM, LN-Robust SVM (with S=0.05 e S=0.5), Least-Square SVM, Dual Infinity-norm SVM for different C values on PDF malware detection dataset.

Discussion

With the reported experiment, we have investigated the security of different SVMs under the label flip attack. We have shown that the sparsity of the SVM α values may be considered a threat to its security in the presence of training data contamination. We have moreover shown that forcing the SVM to learn more evenly-distributed α, as we do with the proposed Dual Infinity-norm SVM, can help to mitigate this threat.

Chapter 8

Securing Android Malware Detectors against Evasion Attacks

In Chapter 5.4 we show that relying on robustness and regularization theory is possible design machine learning algorithms that are more robust under evasion attacks. In this Chapter we show how the proposed countermeasure enable one to leverage on the specific application peculiarities to further enhance the security of the considered application with a case study on Android malware detection. The relevance of this task is witnessed by the fact that Android has become the most popular mobile operating system, with more than a billion users around the world, while the number of malicious applications targeting them has also grown simultaneously: anti-virus vendors detect thousands of new malware samples daily, and there is still no end in sight [116, 63]. Notably, here we do not consider attacks that can completely defeat static analysis [79], like those based on packer-based encryption [111] and advanced code obfuscation [30, 90, 50, 51, 91]. The main reason is that such techniques may leave detectable traces, suggesting the use of a more appropriate system for classification; e.g., the presence of system routines that perform dynamic loading of libraries or classes, potentially hiding embedded malware, demands for the use of dynamic analysis for a more reliable classification. We aim, instead to improve the security of a previously proposed Android malware detection called Drebin against *stealthier* attacks, i.e., carefully-crafted malware samples that evade detection without exhibiting significant evidence of manipulation.

To perform a well-crafted security analysis of Drebin and, more generally, of Android malware detection tools against such attacks, we exploit the adversarial framework that we propose in Chapter 4. We test the system under different *attack scenarios* in which the attacker exhibits an increasing capability of manipulating the input data, and level of knowledge about the targeted system. To simulate evasion attacks in which the attacker does not exploit any knowledge of the targeted system, we consider some obfuscation techniques that are not specifically targeted against Drebin, by running an analysis similar to that reported in [71]. To this end, we

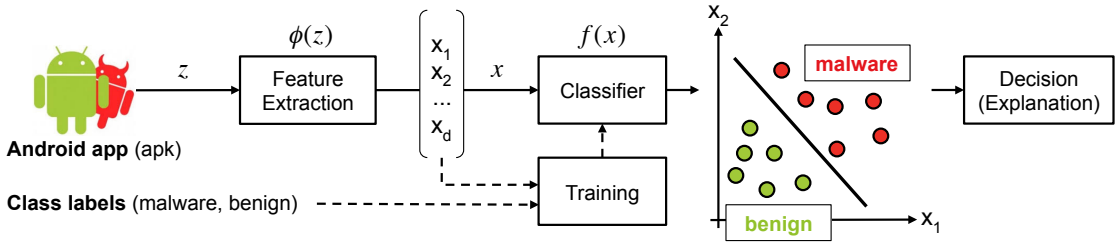


Figure 8.1: A schematic representation of the architecture of Drebin. First, applications are represented as vectors in a d-dimensional feature space. A linear classifier is then trained on an available set of labeled application, to discriminate between malware and benign applications. During classification, unseen applications are evaluated by the classifier. If its output $f(\boldsymbol{x}) \geq 0$, they are classified as malware, and as benign otherwise. Drebin also provides an interpretation of its decision, by highlighting the most suspicious (or benign) features that contributed to the decision [4].

make use of the commercial obfuscation tool `DexGuard`,[1] which has been originally designed to make reverse-engineering of benign applications more difficult. Note that, even if considering obfuscation attacks is out of our score, `DexGuard` only partially obfuscates the content of Android applications. For this reason, the goal of this analysis is simply to empirically assess whether the static analysis performed by Drebin remains effective when Android applications are not thoroughly obfuscated, or when obfuscation is not targeted.

We propose to reduce the vulnerability of Drebin to evasion attack using the Sec-SVM algorithm that we propose in Section 5.4. With respect to previous techniques for *secure learning* [28, 46, 59, 15], Sec-SVM is able to retain computational efficiency and scalability on large datasets (as it exploits a linear classification function), while also being well-motivated from a theoretical perspective.

We show that our method outperforms state-of-the-art classification algorithms, including *secure* ones, without losing significant accuracy in the absence of well-crafted attacks, and can even guarantee some degree of robustness against `DexGuard`-based obfuscations.

8.1 Android Background

Android is the most used mobile operating system. Android applications are in the `apk` format, i.e., a zipped archive containing two files: the Android `manifest` and classes.dex. Additional `xml` and resource files are respectively used to define the application layout, and to provide additional functionalities or multimedia content. As Drebin analyzes the Android `manifest` and classes.dex files, below we provide a

[1] https://www.guardsquare.com/dexguard

brief description of their characteristics.

Android Manifest. The `manifest` file holds information about the application structure. Such structure is organized in *application components*, i.e., parts of code that perform specific actions; e.g., one component might be associated to a screen visualized by the user (*activity*) or to the execution of audio in the background (*services*). The actions of each component are further specified through *filtered intents*; e.g., when a component sends data to other applications, or is invoked by a browser. Special types of components are *entry points*, i.e., activities, services and receivers that are loaded when requested by a specific filtered intent (e.g., an activity is loaded when an application is launched, and a service is activated when the device is turned on). The `manifest` also contains the list of *hardware components* and *permissions* requested by the application to work (e.g., Internet access).

Dalvik Bytecode (`dexcode`). The classes.dex file contains the compiled source code of an application. It contains all the user-implemented methods and classes. Classes.dex might contain specific API calls that can access sensitive resources such as personal contacts (*suspicious calls*). Moreover, it contains all system-related, *restricted API calls* whose functionality require *permissions* (e.g., using the Internet). Finally, this file can contain references to *network addresses* that might be contacted by the application.

8.2 Drebin

Drebin conducts multiple steps and can be executed directly on the mobile device, as it performs a lightweight static analysis of Android applications. The extracted features are used to embed applications into a high-dimensional vector space and train a classifier on a set of labeled data. An overview of the system architecture is given in Fig. 8.1. In the following, we describe the single steps in more detail.

Feature Extraction. Initially, Drebin performs a static analysis of a set of available Android applications,[2] to construct a suitable feature space. All features extracted by Drebin are presented as *strings* and organized in 8 different feature sets, as listed in Table 8.1. Android applications are then mapped onto the feature space as follows. Let us assume that an Android application (i.e., an `apk` file) is represented as an object $\boldsymbol{z} \in \mathcal{Z}$, being $\mathcal{Z}$ the abstract space of all `apk` files. We then denote with $\Phi : \mathcal{Z} \mapsto \mathcal{X}$ a function that maps an `apk` file $\boldsymbol{z}$ to a `d`-dimensional feature vector $\boldsymbol{x} = (x^1, \ldots, x^{\mathtt{d}})^\top \in \mathcal{X} = \{0,1\}^{\mathtt{d}}$, where each feature is set to 1 (0) if the corresponding *string* is present (absent) in the `apk` file $\boldsymbol{z}$. An application encoded

[2] We use here a modified version of Drebin that performs a static analysis based on the `Androguard` tool, available at:
https://github.com/androguard/androguard.

Table 8.1: Overview of feature sets.

Feature sets		
`manifest`	S_1	Hardware components
	S_2	Requested permissions
	S_3	Application components
	S_4	Filtered intents
`dexcode`	S_5	Restricted API calls
	S_6	Used permission
	S_7	Suspicious API calls
	S_8	Network addresses

in feature space may thus look like the following:

$$\boldsymbol{x} = \Phi(\boldsymbol{z}) \mapsto \begin{pmatrix} \cdots & \cdots \\ 0 & \texttt{permission::SEND\_SMS} \\ 1 & \texttt{permission::READ\_SMS} \\ \cdots & \cdots \\ 1 & \texttt{api\_call::getDeviceId} \\ 0 & \texttt{api\_call::getSubscriberId} \\ \cdots & \cdots \end{pmatrix} \begin{matrix} \\ \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} S_2 \\ \\ \left.\begin{matrix} \\ \\ \end{matrix}\right\} S_5 \\ \\ \end{matrix}$$

Limitations and Open Issues.

Although Drebin has shown to be capable of detecting malware with high accuracy, it exhibits intrinsic vulnerabilities that might be exploited by an attacker to evade detection. Since Drebin is designed to run directly on the mobile device, its most obvious limitation is the lack of a dynamic analysis. Unfortunately, static analysis has clear limitations, as it is not possible to analyze malicious code that is downloaded or decrypted at runtime, or code that is thoroughly obfuscated [79, 30, 111, 90, 50, 51, 91]. For this reason, considering such attacks would be irrelevant as it is not doable to create secure systems based on static analysis against them. Our focus is rather to understand and to improve the security properties of learning algorithms against specifically-targeted attacks, in which the amount of manipulations performed by the attacker is limited. The rationale is that the manipulated malware samples should not only evade detection, but should also be difficult to detect traces of their adversarial manipulation. Although these limitations have been also discussed in [4], the effect of carefully-targeted attacks against Drebin has never been studied before. For this reason, in the following, we firstly discuss the evasion attack scenarios that we consider in our evaluation. Then, we present a systematic evaluation of these attacks on Drebin and on a novel learning algorithm (Sec-SVM) that we propose to alleviate their effects.

8.3 Drebin Evasion

In this section the evasion attacks that we considered in our experiment are illustrated.

Firstly, as Malware detection is a binary problem we can simplify the evasion objective function given in Chapter 5 as:

$$\boldsymbol{z}^\star = \arg\min_{\boldsymbol{z}' \in \Psi(\boldsymbol{z})} \hat{f}(\Phi(\boldsymbol{z}')) = \arg\min_{\boldsymbol{z}' \in \Psi(\boldsymbol{z})} \hat{\boldsymbol{w}}^\top \boldsymbol{x}', \tag{8.1}$$

where $\boldsymbol{x}' = \Phi(\boldsymbol{z}')$ is the feature vector associated to the modified attack sample $\boldsymbol{z}'$, and $\hat{\boldsymbol{w}}$ is the weight vector estimated by the attacker (e.g., from the surrogate classifier $\hat{f}$). The above equation essentially tells the attacker which features should be modified to maximally decrease the value of the classification function, i.e., to maximize the probability of evading detection [17, 12].

In the following, we explain the application specific data manipulation constraints and the evasion scenario that we considered in our experiments. We then describe the evasion attack algorithm that we use in the proposed experiments. Finally, we give more detail about the obfuscation mechanism that we use in one of the tested evasion scenario.

8.3.1 Malware Data Manipulation

As we explained before, an attacker may have constraints on data manipulation. Those constraints are application dependent. We consider two different data manipulation constraints in our experiment, that are illustrated below.

Feature Addition. Within this setting, the attacker can independently inject (i.e., set to 1) every feature.

Feature Addition and Removal. This scenario simulates a more powerful attacker that can inject every feature, and also remove (i.e., set to 0) features from the `dexcode`.

These settings are motivated by the fact that malware has to be manipulated to evade detection, but its semantics and intrusive functionality must be preserved. In this respect, *feature addition* is generally a safe operation, in particular, when injecting `manifest` features (e.g., adding permissions does not influence any existing application functionality). With respect to the `dexcode`, one may also safely introduce information that is not actively executed, by adding code after `return` instructions (*dead code*) or with methods that are never called by any `invoke` type instructions. Listing 8.1 shows an example where a URL feature is introduced by adding a method that is never invoked in the code.

```
.method public addUrlFeature()V
.locals 2
const-string v1, "http://www.example.com"
invoke-direct {v0, v1},
Ljava/net/URL;-><init>(Ljava/lang/String;)V
return-void
.end method
```

Listing 8.1: Smali code to add a URL feature.

However, this only applies when such information is not directly executed by the application and could be stopped at the parsing level by analyzing only the methods belonging to the application *call graph*. In this case, the attacker would be enforced to change the executed code, and this requires considering additional and stricter constraints. For example, if she wants to add a suspicious API call to a `dexcode` method that is executed by the application, she should adopt virtual machine registers that have not been used before by the application. Moreover, the attacker should pay attention to possible artifacts or undesired functionalities that are brought by the injected calls, which may influence the semantics of the original program. Accordingly, injecting a large number of features may not always be feasible.

Feature removal is even a more complicated operation. Removing permissions from the `manifest` is not possible, as this would limit the application functionality. The same holds for intent filters. Some application component names can be changed but, as we explain more in detail in Sect. 8.3.4, this operation is not easy to be automatically performed: the attacker must ensure that the application component names in the `dexcode` are changed accordingly, and must not modify any of the entry points. Furthermore, the feasible changes may only slightly affect the whole `manifest` structure (as shown in our experiments with automated obfuscation tools). With respect to the `dexcode`, multiple ways can be exploited to remove its features; e.g., it is possible to hide IP addresses (if they are stored as strings) by encrypting them with the introduction of additional functions, and decrypting them at runtime. Of course, this should be done by avoiding the addition of features that are already used by the system (e.g., function calls that are present in the training data).

With respect to suspicious and restricted API calls, the attacker should encrypt the method or the class invoking them. However, this could introduce other calls that might increase the suspiciousness of the application. Moreover, one mistake at removing such API references might completely destroy the application functionality. The reason is that Android uses a *verification* system to check the integrity of an application during execution (e.g., it will close the application, if a register passed as a parameter to an API call contains a wrong type), and chances of compromising this behavior increases if features are deleted carelessly.

For the aforementioned reasons, performing a fine-grained evasion attack that

changes a lot of features may be very difficult in practice, without compromising the malicious application functionality. In addition, another problem for the attacker is getting to know precisely which features should be added or removed, which makes the construction of evasion attack samples even more complicated.

8.3.2 Evasion Scenarios

In the following, we consider different evasion scenarios, according to the framework discussed in Chapter 4 where the attacker owns an increasing knowledge of the system. In particular, we consider in our experiment five distinct attack scenarios. Here we depict three scenarios, sorted for increasing attacker knowledge, that does not require to the attacker knowledge of the system. Note that, when the attacker knows more details of the targeted system, her estimate of the classification function becomes more reliable, thus facilitating the evasion task (in the sense of requiring fewer manipulations to the malware samples).

Zero-effort Attacks. This is the standard scenario in which malware data is neither obfuscated nor modified at all. From the viewpoint of the attacker's knowledge, this scenario is characterized does not require any knowledge.

DexGuard-based Obfuscation Attacks. As another attack scenario in which the attacker does not exploit any knowledge of the attacked system, we consider a setting similar to that reported in [71]. In particular, we assume that the attacker attempts to evade detection by performing invasive code transformations on the `classes.dex` file, using the commercial Android obfuscation tool `DexGuard`. Note that this tool is designed to ensure protection against disassembling/decompiling attempts in benign applications, and not to obfuscate the presence of malicious code; thus, despite the introduction of many changes in the executable code, it is not clear whether and to what extent the obfuscations implemented by this tool may be effective against a learning-based malware detector like Drebin, i.e., how they will affect the corresponding feature values and classification output. The obfuscations implemented by `DexGuard` are described more in detail later on in this section.

Mimicry Attacks. Under this scenario, the attacker is assumed to be able to collect a surrogate dataset including malware and benign samples and to know the feature space. Accordingly, $\boldsymbol{\theta} = (\hat{\mathcal{D}}, \mathcal{X})$. In this case, the attack strategy amounts to manipulating malware samples to make them as close as possible to the benign data (in terms of conditional probability distributions or, alternatively, distance in feature space). To this end, in the case of Drebin (which uses binary feature values), we can assume that the attacker still aims to minimize Eq. (8.1), but estimates each component of $\hat{\boldsymbol{w}}$ independently for each feature as $\hat{w}_k = p(\hat{x}_k = 1|y = +1) - p(\hat{x}_k = 1|y = -1)$, $k = 1, \ldots, \mathtt{d}$. This will indeed induce the attacker to add (remove) first features which are more frequently present (absent) in benign files, making the probability distribution of malware samples closer to that of the benign data. It is worth finally remarking that this is a more sophisticated mimicry attack than

those commonly used in practice, in which an attacker is usually assumed to merge a malware application with a benign one [107, 116].

8.3.3 Evasion attack algorithm

We depict here the algorithm used to implement our evasion attacks. For linear classifiers with binary features, the solution to Problem (8.1) can be found as follows. First, the estimated weights $\hat{\boldsymbol{w}}$ have to be sorted in descending order of their absolute values, along with the feature values $\boldsymbol{x}$ of the initial malicious sample. This means that, if the sorted weights and features are denoted respectively with $\hat{w}_{(1)}, \ldots, \hat{w}_{(\mathtt{d})}$ and $x_{(1)}, \ldots, x_{(\mathtt{d})}$, then $|\hat{w}_{(1)}| \geq \ldots \geq |\hat{w}_{(\mathtt{d})}|$. Then, for $k = 1, \ldots, \mathtt{d}$:

- *if* $x_{(k)} = 1$ and $\hat{w}_{(k)} > 0$ (and the feature is not in the `manifest` sets S1-S4), *then* $x_{(k)}$ is set to zero;
- *if* $x_{(k)} = 0$ and $\hat{w}_{(k)} < 0$, *then* $x_{(k)}$ is set to one;
- *else* $x_{(k)}$ is left unmodified.

If the maximum number of modified features has been reached, the for loop is clearly stopped in advance.

8.3.4 DexGuard-based Obfuscation Attacks

Although commercial obfuscators are designed to protect benign applications against reverse-engineering attempts, it has been recently shown that they can also be used to evade anti-malware detection systems [71]. We thus use `DexGuard`, a popular obfuscator for Android, to simulate attacks in which no specific knowledge of the targeted system is exploited, as discussed in Sect. 8.3.2. Recall that, although considering obfuscation attacks is out of the scope of this work, the obfuscation techniques implemented by `DexGuard` do not completely obfuscate the code. For this reason, we aim to understand whether this may make static analysis totally ineffective, and how it affects our strategy to improve classifier security. A brief description of the `DexGuard`-based obfuscation attacks is given below.

Trivial obfuscation. This strategy changes the names of *implemented* application packages, classes, methods, and fields, by replacing them with random characters. Trivial obfuscation also performs negligible modifications to some `manifest` features by renaming some application components that are *not* entry points (see Sect. 8.1). As the application functionality must be preserved, Trivial obfuscation does not rename any system API or method imported from native libraries. Given that Drebin mainly extracts information from system APIs, we expect that its detection capability will be only barely affected by this obfuscation.

String Encryption. This strategy encrypts strings defined in the `dexcode` with the instruction `const-string`. Such strings can be visualized during the application

execution, or may be used as variables. Thus, even if they are retrieved through an identifier, their value must be preserved during the program execution. For this reason, an additional method is added to decrypt them at runtime, when required. This obfuscation tends to remove URL features (S8) that are stored as strings in the `dexcode`. Features corresponding to the decryption routines extracted by Drebin (S7) are instead not affected, as the decryption routines added by `DexGuard` do not belong to the system APIs.

Reflection. This obfuscation technique uses the Java Reflection API to replace `invoke-type` instructions with calls that belong to the `Java.lang.Reflect` class. The main effect of this action is destroying the application call graph. However, this technique does not affect the system API names, as they do not get encrypted during the process. It is thus reasonable to expect that most of the features extracted by Drebin will remain unaffected.

Class Encryption. This is the most invasive obfuscation strategy, as it encrypts all the application classes, except entry-point ones (as they are required to load the application externally). The encrypted classes are decrypted at runtime by routines that are added during the obfuscation phase. Worth noting, the class encryption performed by `DexGuard` does not completely encrypt the application. For example, classes belonging to the API components contained in the `manifest` are not encrypted, as this would most likely compromise the application functionality. For the same reason, the `manifest` itself is preserved. Accordingly, it still possible to extract static features using Drebin, and analyze the application. Although out of the scope of our study, it is still worth remarking here that using packers (e.g., [111]) to perform full dynamic loading of the application classes might completely evade static analysis.

Combined Obfuscations. The aforementioned strategies can also be combined to produce additional obfuscation techniques. As in [71], we will consider three additional techniques in our experiments, by respectively combining (*i*) trivial and string encryption, (*ii*) adding reflection to them, and (*iii*) adding class encryption to the former three.

Parameter Selection.

To tune the parameters of our classifiers, as suggested in [17, 113], one should not only optimize accuracy on a set of collected data, using traditional performance evaluation techniques like cross validation or bootstrapping. More properly, one should optimize a trade-off between accuracy and *security*, by accounting for the presence of potential, unseen attacks during the validation procedure. Here we optimize this trade-off, denoted with $r(f_{\boldsymbol{\mu}}, \mathcal{D})$, as:

$$\boldsymbol{\mu}^{\star} = \arg\max_{\boldsymbol{\mu}} r(f_{\boldsymbol{\mu}}, \mathcal{D}) = A(f_{\boldsymbol{\mu}}, \mathcal{D}) + \lambda S(f_{\boldsymbol{\mu}}, \mathcal{D})\,, \tag{8.2}$$

where we denote with $f_{\boldsymbol{\mu}}$ the classifier learned with parameters $\boldsymbol{\mu}$ (e.g., for our Sec-SVM, $\boldsymbol{\mu} = \{C, \boldsymbol{w}^{\text{lb}}, \boldsymbol{w}^{\text{ub}}\}$), with A a measure of classification accuracy in the absence

of attack (estimated on $\mathcal{D}$), with S an estimate of the classifier security under attack (estimated by simulating attacks on $\mathcal{D}$), and with λ a given trade-off parameter.

Classifier security can be evaluated by considering distinct attack settings, or a different amount of modifications to the attack samples. In our experiments, we will optimize security in a worst-case scenario, i.e., by simulating a PK evasion attack with both feature addition and removal. We will then average the performance under attack over an increasing number of modified features $m \in [1, \mathsf{M}]$. More specifically, we will measure security as:

$$S = \frac{1}{\mathsf{M}} \sum_{m=1}^{\mathsf{M}} A(f_{\boldsymbol{\mu}}, \mathcal{D}'_k)\,, \tag{8.3}$$

where $\mathcal{D}'_k$ is obtained by modifying a maximum of m features in each malicious sample in the *validation set*,[3] as suggested by the PK evasion attack strategy.

8.4 Experimental Analysis

In this section, we report an experimental evaluation of our proposed secure learning algorithm (Sec-SVM) by testing it under different evasion scenarios (see Sect. 8.3).

8.4.1 Experimental Setup

We here describe the experimental setup that we have used to produce the reported experiments.

Classifiers. We compare our Sec-SVM approach with the standard Drebin implementation (denoted with SVM), and with a previously-proposed technique that improves the security of linear classifiers by using a Multiple Classifier System (MCS) architecture to obtain a linear classifier with more evenly-distributed feature weights [59, 15]. To this end, multiple linear classifiers are learned by sampling uniformly from the training set (a technique known as *bagging* [27]) and by randomly subsampling the feature set, as suggested by the *random subspace method* [49]. The classifiers are then combined by averaging their outputs, which is equivalent to using a linear classifier whose weights and bias are the average of the weights and biases of the base classifiers, respectively. With this simple trick, the computational complexity at test time remains thus equal to that of a single linear classifier [15]. As we use linear SVMs as the base classifiers, we denote this approach with MCS-SVM. We finally consider a version of our Sec-SVM trained using only `manifest` features that we call Sec-SVM (M). The reason is to verify whether considering only features, which can not be removed, limits closely mimicking benign data and thereby yields a more secure system.

[3] Note that, as in standard performance evaluation techniques, data is split into distinct training-validation pairs, and then performance is averaged on the distinct validation sets. As we are considering *evasion attacks*, training data is not affected during the attack simulation, and only malicious samples in the validation set are thus modified.

Datasets. In our experiments, we use two distinct datasets. The first (referred to as *Drebin*) includes the data used in [4], and consists of 121,329 benign applications and 5,615 malicious samples, labeled using the VirusTotal service. A sample is labeled as malicious if it is detected by at least five anti-virus scanners, whereas it is labeled as benign if no scanner flagged it as malware. The second (referred to as *Contagio*) includes the data used in [71], and consists of about 1,500 malware samples, obtained from the *MalGenome*[4] and the *Contagio Mobile Minidump*[5] datasets. Such samples have been obfuscated with the seven obfuscation techniques described in Sect. 8.3.4, yielding a total of about 10,500 samples.

Training-test splits. We average our results on 10 independent runs. In each repetition, we randomly select 60,000 applications from the *Drebin* dataset, and split them into two equal sets of 30,000 samples each, respectively used as the training set and the surrogate set (as required by the LK and mimicry attacks discussed in Sect. 8.3). As for the test set, we use all the remaining samples from *Drebin*. In some attack settings (detailed below), we replace the malware data from *Drebin* in each test set with the malware samples from *Contagio*. This enables us to evaluate the extent to which a classifier (trained on some data) preserves its performance in detecting malware from *different* sources.[6]

Feature selection. When running Drebin on the given datasets, more than one million features are found. For computational efficiency, we retain the most discriminant d' features, for which $|p(x_k = 1|y = +1) - p(x_k = 1|y = -1)|$, $k = 1, \ldots, \mathsf{d}$, exhibits the highest values (estimated on training data). In our case, using only $\mathsf{d}' = 10,000$ features does not significantly affect the accuracy of Drebin. This is consistent with the recent findings in [92], as it is shown that only a very small fraction of features is significantly discriminant and usually assigned a non-zero weight by Drebin (i.e., by the SVM learning algorithm). For the same reason, the sets of selected features turned out to be the same in each run. Their sizes are reported in Table 8.2.

Parameter setting. We run some preliminary experiments on a subset of the training set and noted that changing C did not have a significant impact on classification accuracy for all the SVM-based classifiers (except for higher values, which cause overfitting). Thus, also for the sake of a fair comparison among different SVM-based learners, we set $C = 1$ for all classifiers and repetitions. For the MCS-SVM classifier, we train 50 base linear SVMs on random subsets of 80% of the training samples and 50% of the features, as this ensures a sufficient diversification of the base classifiers, providing more evenly-distributed feature weights. The bounds of the Sec-SVM are selected through a 5-fold cross-validation, following the procedure explained in Sect. 8.3.4. In particular, we set each element of $\boldsymbol{w}^{\mathrm{ub}}$ ($\boldsymbol{w}^{\mathrm{lb}}$) as w^{ub} (w^{lb}),

[4] http://www.malgenomeproject.org/

[5] http://contagiominidump.blogspot.com/

[6] Note however that a number of malware samples in *Contagio* are also included in the *Drebin* dataset.

Table 8.2: Number of features in each set for SVM, Sec-SVM, and MCS-SVM. Feature set sizes for the Sec-SVM (M) using only `manifest` features are reported in brackets. For all classifiers, the total number of selected features is $\mathsf{d}' = 10,000$.

Feature set sizes					
`manifest`	S_1	13 (21)	`dexcode`	S_5	147 (0)
	S_2	152 (243)		S_6	37 (0)
	S_3	2,542 (8,904)		S_7	3,029 (0)
	S_4	303 (832)		S_8	3,777 (0)

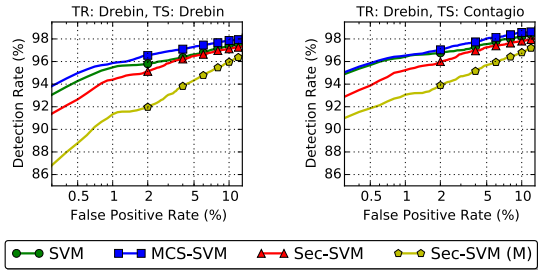


Figure 8.2: Mean ROC curves on *Drebin* (*left*) and *Contagio* (*right*) data, for classifiers trained on *Drebin* data.

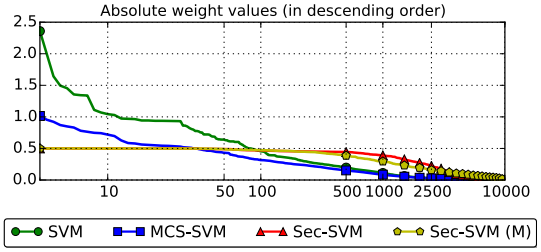


Figure 8.3: Absolute weight values in descending order (i.e., $|w_{(1)}| \geq \ldots \geq |w_{(\mathsf{d})}|$), for each classifier (averaged on 10 runs). Flatter curves correspond to more evenly-distributed weights, i.e., more secure classifiers.

and optimize the two scalar values $(w^{\mathrm{ub}}, w^{\mathrm{lb}}) \in \{0.1, 0.5, 1\} \times \{-1, -0.5, -0.1\}$. As for the performance measure $A(f_{\boldsymbol{\mu}}, \mathcal{D})$ (Eq. 8.2), we consider the Detection Rate (DR) at 1% False Positive Rate (FPR), while the security measure $S(f_{\boldsymbol{\mu}}, \mathcal{D})$ is simply given by Eq. (8.3). We set $\lambda = 10^{-2}$ in Eq. (8.2) to avoid worsening the detection of both benign and malware samples in the absence of attack to an unnecessary extent. Finally, as explained in Sect. 5.4, the parameters of Algorithm 2 are set by running it on a subset of the training data, to ensure quick convergence, as $\eta^{(0)} = 0.5$, $\gamma \in \{10, 20, \ldots, 70\}$ and $s(t) = 2^{-0.01t}/\sqrt{\mathsf{n}}$.

8.4.2 Experimental Results

We present our results by reporting the performance of the given classifiers against (*i*) zero-effort attacks, (*ii*) obfuscation attacks, and (*iii*) advanced evasion attacks, including PK, LK and mimicry attacks, with both feature addition, and feature addition and removal (see Sects. 8.3).

Zero-effort attacks. Results for the given classifiers in the absence of attack are reported in the ROC curves of Fig. 8.2. They report the Detection Rate (DR, i.e., the fraction of correctly-classified malware samples) as a function of the False Positive

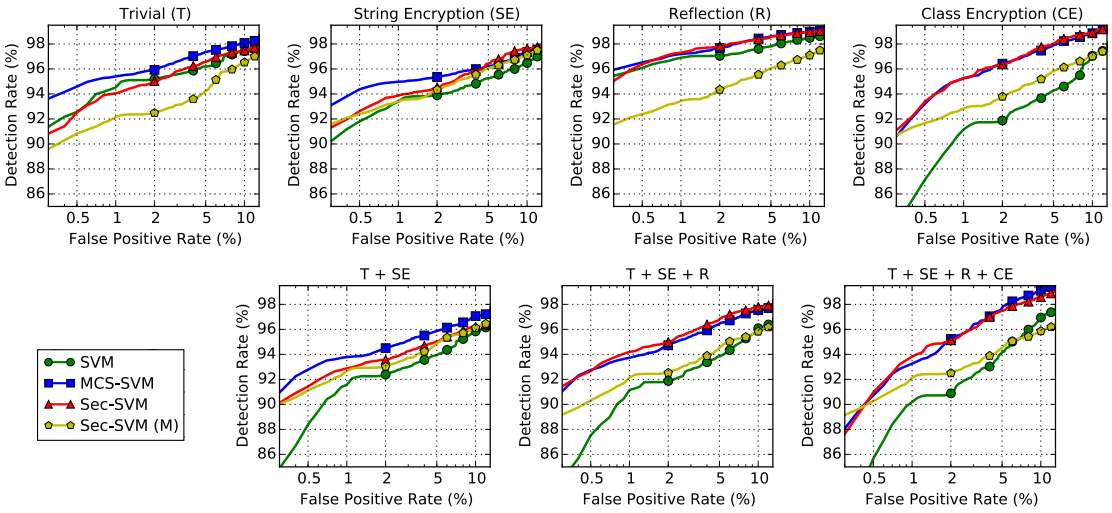


Figure 8.4: Mean ROC curves for all classifiers against different obfuscation techniques, computed on the *Contagio* data.

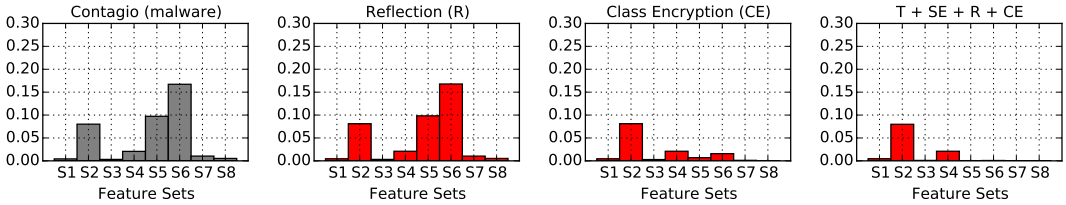


Figure 8.5: Fraction of features equal to one in each set (averaged on 10 runs), for non-obfuscated (*leftmost* plot) and obfuscated malware in *Contagio*, with different obfuscation techniques. While obfuscation deletes `dexcode` features (S5-S8), the `manifest` (S1-S4) remains mostly intact.

Rate (FPR, i.e., the fraction of misclassified benign samples) for each classifier. We consider two different cases: (*i*) using both training and test samples from *Drebin* (*left* plot); and (*ii*) training on *Drebin* and testing on *Contagio* (*right* plot), as previously discussed. Notably, MCS-SVM achieves the highest DR (higher than 96% at 1% FPR) in both settings, followed by SVM and Sec-SVM, which only slightly worsen the DR. Sec-SVM (M) performs instead significantly worse. In Fig. 8.3, we also report the absolute weight values (sorted in descending order) of each classifier, to show that Sec-SVM classifiers yield more evenly-distributed weights, also with respect to MCS-SVM.

DexGuard-based obfuscation attacks. The ROC curves reported in Fig. 8.4 show the performance of the given classifiers, trained on *Drebin*, against the `DexGuard`-based obfuscation attacks (see Sect. 8.3.2 and Sect. 8.3.4) on the *Contagio* malware. Here, Sec-SVM performs similarly to MCS-SVM, while SVM and Sec-SVM (M) typically exhibit lower detection rates. Nevertheless, as these obfuscation attacks do not completely obfuscate the malware code, and the feature changes induced by them

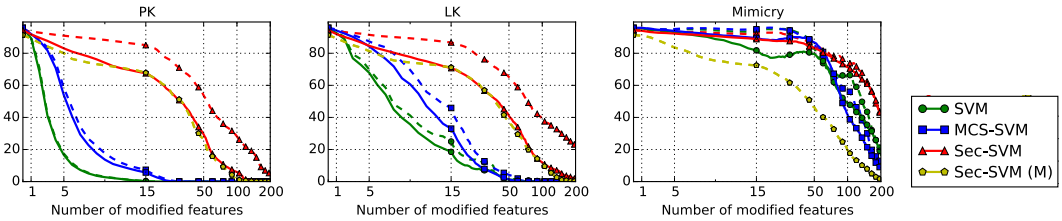


Figure 8.6: Detection Rate (DR) at 1% False Positive Rate (FPR) for each classifier under the *Perfect-Knowledge* (left), *Limited-Knowledge* (middle), and *Mimicry* (right) attack scenarios, against an increasing number of modified features. Solid (dashed) lines are obtained by simulating attacks with feature addition (feature addition and removal).

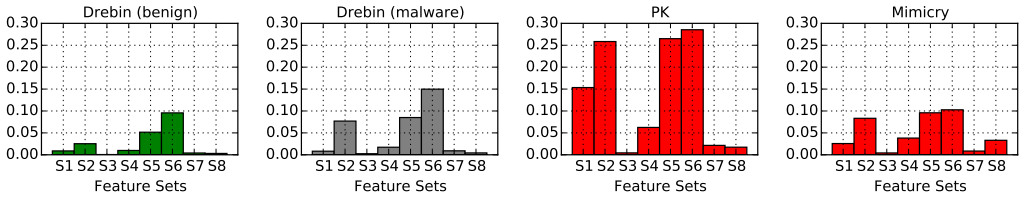


Figure 8.7: Fraction of features equal to one in each set (averaged on 10 runs) for benign (*first* plot), non-obfuscated (*second* plot) and DexGuard-based obfuscated malware in *Drebin*, using PK (*third* plot) and mimicry (*fourth* plot) attacks. It is clear that the mimicry attack produces malware samples which are more similar to the benign data than those obtained with the PK attack.

are not specifically targeted against any of the given classifiers, the classification performances are not significantly affected. In fact, the DR at 1% FPR is never lower than 90%. As expected (see Sect. 8.3.4), strategies such as Trivial, String Encryption and Reflection do not affect the system performances significantly, as Drebin only considers system-based API calls, which are not changed by the aforementioned obfuscations. Among these attacks, Class Encryption is the most effective strategy, as it is the only one that more significantly modifies the S5 and S7 feature sets (in particular, the first one), as it can be seen in Fig. 8.5. Nevertheless, even in this case, as `manifest`-related features are not affected by `DexGuard`-based obfuscations, Drebin still exhibits good detection performances.

Advanced evasion. We finally report results of the PK, LK, and mimicry attacks in Fig. 8.6, considering both feature addition, and feature addition and removal. As we are not removing `manifest`-related features, Sec-SVM (M) is clearly tested only against feature-addition attacks. Worth noting, Sec-SVM can drastically improve security compared to the other classifiers, as its performance decreases more gracefully against an increasing number of modified features, especially in the PK and LK attack scenarios. In the PK case, while the DR of Drebin (SVM) drops to 60% after modifying only *two* features, the DR of the Sec-SVM decreases to the same amount only when *fifteen* feature values are changed. This means that our

Table 8.3: Top 5 modified features by the PK evasion attack with feature addition (A) and removal (R), for SVM, MCS-SVM, and Sec-SVM (highlighted in bold). In the first column, the feature family is reported. The probability of a feature being equal to one in malware data is denoted with p. For each classifier and each feature, we then report two values (averaged on 10 runs): (i) the probability q' that the feature is modified by the attack (*left*), and (ii) its relevance (*right*), measured as its absolute weight divided by $\|\boldsymbol{w}\|_1$. If the feature is not modified within the first 200 changes, we report that the corresponding values are only lower than the minimum ones observed. In the last column, we also report whether the feature has been added (↑) or removed (↓) by the attack.

Set	Feature Name	p	SVM		MCS-SVM		Sec-SVM		A/R
S6	susp_calls::android/telephony/gsm/SmsMessage;→getDisplayMessageBody	2.40%	**89.60%**	0.25%	3.99%	0.05%	<0.03%	<0.02%	↑
S1	req_perm::android.permission.USE_CREDENTIALS	0.05%	**65.77%**	0.18%	**67.57%**	0.13%	<0.03%	<0.02%	↑
S1	req_perm::android.permission.WRITE_OWNER_DATA	0.52%	**64.76%**	0.16%	**49.23%**	0.11%	<0.03%	<0.02%	↑
S0	features::android.hardware.touchscreen	0.60%	**64.01%**	0.14%	41.75%	0.09%	<0.03%	<0.02%	↑
S6	susp_calls::android/telephony/gsm/SmsMessage;→getMessageBody	3.50%	**60.13%**	0.13%	17.30%	0.05%	<0.03%	<0.02%	↑
S3	intent_filters::android.intent.action.SENDTO	0.73%	55.70%	0.13%	**60.38%**	0.11%	<0.03%	<0.02%	↑
S6	susp_calls::android/telephony/CellLocation;→requestLocationUpdate	0.05%	50.87%	0.12%	**48.37%**	0.08%	<0.03%	<0.02%	↑
S6	susp_calls::android/net/ConnectivityManager;→getBackgroundDataSetting	0.51%	28.86%	0.07%	**43.59%**	0.09%	<0.03%	<0.02%	↑
S6	susp_calls::android/telephony/TelephonyManager;→getNetworkOperator	46.41%	36.08%	0.17%	43.12%	0.19%	**39.88%**	0.04%	↓
S6	susp_calls::android/net/NetworkInfo;→getExtraInfo	24.81%	19.25%	0.17%	13.42%	0.10%	**10.25%**	0.03%	↓
S6	susp_calls::getSystemService	93.44%	<3.53%	<0.02%	<0.12%	<0.05%	**11.02%**	0.02%	↓
S7	urls::www.searchmobileonline.com	9.42%	4.59%	0.11%	6.91%	0.13%	**4.83%**	0.03%	↓
S6	services::com.apperhand.device.android.AndroidSDKProvider	10.83%	6.78%	0.13%	4.19%	0.09%	**5.14%**	0.03%	↓

Sec-SVM approach can improve classifier security of about *ten* times, in terms of the number of modifications required to create a malware sample that evades detection. The underlying reason is that Sec-SVM provides more evenly-distributed feature weights, as shown in Fig. 8.3. Note that Sec-SVM and Sec-SVM (M) exhibit a maximum absolute weight value of 0.5 (on average). This means that, in the worst case, modifying a single feature yields an average decrease of the classification function equal to 0.5, while for MCS-SVM and SVM this decrease is approximately 1 and 2.5, respectively. It is thus clear that, to achieve a comparable decrease of the classification function (i.e., a comparable probability of evading detection), more features should be modified in the former cases. Finally, it is also worth noting that mimicry attacks are less effective, as expected, as they exploit an inferior level of knowledge of the targeted system. Despite this, an interesting insight on the behavior of such attacks is reported in Fig. 8.7. After modifying a large number of features, the mimicry attack tends to produce a distribution that is very close to that of the benign data (even without removing any `manifest`-related feature). This means that, in terms of their feature vectors, benign and malware samples become very similar. Under these circumstances, no machine-learning technique can separate benign and malware data with satisfying accuracy. The vulnerability of the system may be thus regarded as intrinsic in the choice of the feature representation, rather than in how the classification function is learned. This clearly confirms the importance of designing features that are more difficult to manipulate for an attacker.

Feature manipulation. To provide some additional insights, in Table 8.3 we report the top 5 modified features by the PK attack with feature addition and removal for

SVM, MCS-SVM, and Sec-SVM. For each classifier, we select the top 5 features by ranking them in descending order of the probability of modification q'. This value is computed as follows. First, the probability q of modifying the k^{th} feature in a malware sample, regardless of the maximum number of admissible modifications, is computed as:

$$q = \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x}|y=+1)}\{x_k \neq x'_k\} = p^{\nu}(1-p)^{1-\nu}\,, \tag{8.4}$$

where $\mathbb{E}$ denotes the expectation operator, $p(\boldsymbol{x}|y=+1)$ the distribution of malware samples, x_k and x'_k are the k^{th} feature values before and after manipulation, and p is the probability of observing $x_k = 1$ in malware. Note that $\nu = 1$ if $x_k = 1$, x_k does not belong to the `manifest` sets S1-S4, and the associated weight $\hat{w}_k > 0$, while $\nu = 0$ if $\hat{w}_k < 0$ (otherwise the probability of modification q is zero). This formula denotes compactly that, if a feature can be modified, then it will be changed with probability p (in the case of deletion) or $1-p$ (in the case of insertion). Then, to consider that features associated to the highest absolute weight values are modified more frequently by the attack, with respect to an increasing maximum number m of modifiable features, we compute $q' = \mathbb{E}_m\{q\}$. Considering $m = 1, \ldots, \mathsf{d}$, with uniform probability, each feature will be modified with probability $q' = q\,(\mathsf{d} - r)/\mathsf{d}$, with $r = 0$ for the feature $x_{(1)}$ assigned to the highest absolute weight value, $r = 1$ for the second ranked feature $x_{(2)}$, etc. In general, for the k^{th}-ranked feature $x_{(k)}$, $r = k - 1$, for $k = 1, \ldots, \mathsf{d}$. Thus, q' decreases depending on the feature ranking, which in turn depends on the feature weights and the probability p of the feature being present in malware. Regarding Table 8.3, note first how the probability of modifying the top features, along with their *relevance* (i.e., their absolute weight value with respect to $\|\boldsymbol{w}\|_1$), decreases from SVM to MCS-SVM, and from MCS-SVM to Sec-SVM. These two observations are clearly connected. The fact that the attack modifies features with a lower probability depends on the fact that weights are more evenly distributed. To better understand this phenomenon, imagine the limit case in which all features are assigned the same absolute weight value. It is clear that, in this case, the attacker could randomly modify any subset of features and obtain the same effect on the classification output; thus, on average, each feature will have the same probability of being modified.

The probability of modifying a feature, however, does not only depend on the weight assigned by the classifier, but also on the probability of being present in malware data, as mentioned before. For instance, if a (non-manifest) feature is present in all malware samples, and it has been assigned a very high positive weight, it will be always removed; conversely, if it only rarely occurs in malware, then it will be deleted only from few samples. This behavior is clearly exhibited by the top features modified by Sec-SVM. In fact, since this classifier basically assigns the same absolute weight value to almost all features, the top modified ones are simply those appearing more frequently in malware. More precisely, in our experiments this classifier, as a result of our parameter optimization procedure, assigns a higher (absolute) weight to features present in malware, and a lower (absolute) weight to

features present in benign data (i.e., $|w_k^{\text{ub}}| > |w_k^{\text{lb}}|$, $k = 1, \ldots, \mathsf{d}$). This is why, conversely to SVM and MCS-SVM, the attack against Sec-SVM tends to remove features, rather than injecting them. To conclude, it is nevertheless worth pointing out that, in general, the most frequently-modified features clearly depend on the data distribution (i.e., on class imbalance, feature correlations, etc.), and not only on the probability of being more frequent in malware. In our analysis, this dependency is intrinsically captured by the dependency of q' on the feature weights learned by the classifier.

8.5 Discussion

We have here considered a specific case study involving Drebin, an Android malware detection tool, and shown that its performances can be significantly downgraded in presence of skilled attackers that can carefully manipulate malware samples to evade classifier detection. We have proposed our Sec-SVM to improve the security of Drebin under evasion attack. Despite the very promising results achieved by our Sec-SVM, it is clear that such an approach exhibits some intrinsic limitations. First, as Drebin performs a static code analysis, it is clear that also Sec-SVM can be defeated by more sophisticated encryption and obfuscation attacks. However, it is also worth remarking that this is not a vulnerability of the learning algorithm itself, but rather of the chosen feature representation, and for this reason, we have not considered these attacks in this study. A similar behavior is observed when a large number of features is modified by our evasion attacks, and especially in the case of mimicry attacks (see Sect. 8.4), in which the manipulated malware samples almost exactly replicate benign data (in terms of their feature vectors). This is again possible due to an intrinsic vulnerability of the feature representation, and no learning algorithm can clearly separate such data with satisfying accuracy. Nevertheless, this problem only occurs when malware samples are significantly modified and, as we pointed out in Sect. 8.3.1, it might be very difficult for the attacker to do that without compromising their intrusive functionality, or without leaving significant traces of adversarial manipulation. For example, the introduction of changes such as reflective calls requires a careful manipulation of the Dalvik registers (e.g., verifying that old ones are correctly re-used and those new ones can be safely employed). A single mistake in the process can lead to verification errors and the application might not be usable anymore (we refer the reader to [50, 51] for further details). Another limitation of our approach may be its unsatisfying performance under PK and LK attacks, but this can be clearly mitigated with simple countermeasures to prevent that the attacker gains sufficient knowledge of the attacked system, such as frequent system re-training and diversification of training data collection [16]. To summarize, although our approach is clearly not bulletproof, we believe that it significantly improves the security of the baseline Drebin system (and of the standard SVM algorithm).

Chapter 9

Securing CNN-based Robot-vision Systems

After decades of research spent in exploring different approaches, ranging from search algorithms, expert and rule-based systems to more modern machine-learning algorithms, several problems involving the use of an artificial intelligence have been finally tackled through the introduction of a novel paradigm shift based on *data-driven* artificial intelligence technologies. In fact, due to the increasing popularity and use of the modern Internet, along with the powerful computing resources available nowadays, it has been possible to extract meaningful knowledge from the huge amount of data collected online, from images to videos, text and speech data [35]. Deep learning algorithms have provided an important resource in this respect. Their flexibility to deal with different kinds of input data, along with their learning capacity have made them a powerful instrument to successfully tackle challenging applications, reporting impressive performance on several tasks in computer vision, speech recognition and human-robot interactions [48, 85].

Despite their undiscussed success in several real-world applications, several open problems remain to be addressed. Research work has been investigating how to interpret decisions taken by deep learning algorithms, unveiling the patterns learned by deep networks at each layer [112, 70]. Although significant progress has been made in this direction, and it is now clear that such networks gradually learn more abstract concepts (e.g., from detecting elementary shapes in images to more abstract notions of objects or animals), a relevant effort is still required to gain deeper insights. This is also important to understand why such algorithms can be *vulnerable*, at least in principle, to the presence of *adversarial examples*, i.e., input data that are slightly modified to mislead classification by the addition of an almost-imperceptible adversarial noise [68, 78]. The presence of adversarial examples has been shown on a variety of tasks, including object recognition in images, handwritten digit recognition, and face recognition [99, 102, 47, 78, 84].

We are the first to show that robot-vision systems based on deep learning al-

gorithms are also vulnerable to such potential threat. A peculiarity of humanoid robots is that they have to learn in an online fashion, from the stimuli received during their exploration of the surrounding environment. For this reason, a crucial requirement for them is to embody completely the acquired knowledge, and a reliable and efficient learning paradigm. As discussed in previous work [85], this is a conflicting goal with the current state of deep learning algorithms, which are too computationally and power demanding to be fully embodied by a humanoid robot. For this reason, the authors in [85] have proposed to use a pre-trained deep network for object recognition to perform feature extraction (essentially considering as the feature vector for the detected object one of the last convolutional layers in the deep network), and then train a multiclass classifier on such feature representation.

Here we show the vulnerability of these kinds of robot-vision system to adversarial examples. To this end, we exploit the attack framework proposed in Chapter 4 that conversely to previous work dealing with the generation of adversarial examples based on minimum-distance perturbations [102, 47, 78, 84], enables creating such examples under a maximum input perturbation, which in turns allows one to assess classifier security more thoroughly, by evaluating the probability of evading detection as a function of the maximum input perturbation. Notably, it also allows manipulating only a region of interest in the input image, such that creating real-world adversarial examples becomes easier; e.g., one may only modify some image pixels corresponding to a sticker that can be subsequently applied to the object of interest.

Furthermore, we evaluate to which extent the computationally-efficient countermeasure, inspired by works on classification with the reject option and open-set recognition, that we propose in Section 5.5 allows mitigating the threat posed by adversarial examples. Its underlying idea is to detect and reject the so-called blind-spot evasion points, i.e., samples which are sufficiently far from known training data. This countermeasure is particularly suited to our case study, as it requires modifying only the learning algorithm applied on top of the deep feature representation, i.e., only the output layer. In particular, although it does not completely address the vulnerability of such system to adversarial examples, it requires one to significantly increase the amount of perturbation on the input images to reach a comparable probability of misleading a correct object recognition. To better understand the reason behind this phenomenon, we provide a further, simple and intuitive empirical analysis, showing that the mapping learned by the deep network used for deep feature extraction essentially violates the smoothness assumption of learning techniques in the input space. This means that, in practice, for a sufficiently high amount of perturbation, the proposed algorithm creates adversarial examples that are mapped onto a region of the deep feature space which is densely populated by training examples of a different class. Accordingly, only modifying the classification algorithm on top of the pre-trained deep features (without re-training the underlying deep network) may not be sufficient in this case.

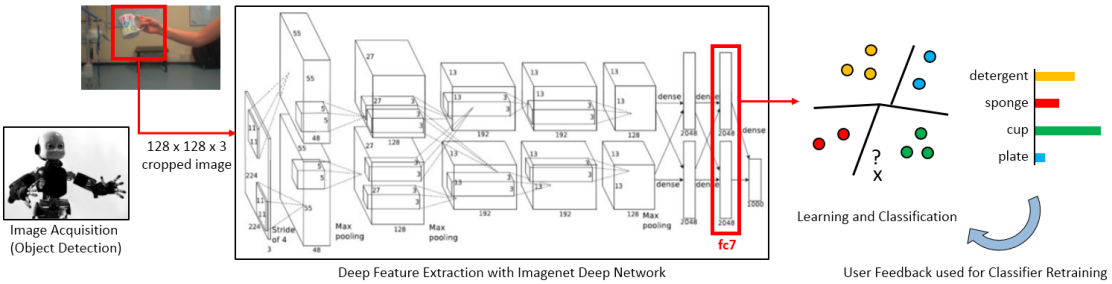


Figure 9.1: Architecture of the iCub robot-vision system [85]. After image acquisition, a region of interest containing the object is cropped and processed by the ImageNet deep network [60]. The deep features extracted from the penultimate layer of such network (fc7) are then used as input to the classification algorithm to perform the recognition task, in which the probabilities that the object belongs to each (known) class are reported. A human annotator can then validate or correct decisions, and the classification algorithm can be updated accordingly; for instance, to learn that an object belongs to a never-before-seen class.

9.1 The iCub Humanoid

Our case study focuses on the iCub humanoid, as it provides a cognitive humanoid robotic platform well suited to our task [75, 85]. In particular, the visual recognition system of this humanoid relies on deep learning technologies to interact with the surrounding environment, enabling it to detect and to recognize *known* objects, i.e., objects that have been verbally annotated in a previous session by a human teacher. Furthermore, iCub is capable of performing *online* learning, i.e., after classification, it asks the human teacher whether the corresponding decision is correct. If the decision is wrong (e.g., in the case of an object belonging to a never-before-seen class), the human teacher can provide feedback to the robot, which in turn updates its classification model through online or incremental learning techniques (e.g., by expanding the set of known object classes). This a clear example of how a robot can learn from experience to improve its capabilities, i.e., a key aspect of why embodying knowledge within robots is of crucial relevance for these tasks [85]. However, given the limited hardware and power resources of the humanoid, it is clear that retraining the whole deep learning infrastructure becomes too computationally demanding. For this reason, the visual system of iCub exploits the pre-trained ImageNet deep network [60] only for extracting a set of deep features (from one of the highest convolutional layers) and uses this feature vector to represent the object detected by iCub in the input image. As described in Fig. 9.1, this deep feature vector is then classified using a separate classifier, which can be retrained online in an efficient manner when feedback from the human annotator is received. In particular, in [85] this classifier is implemented using a one-versus-all scheme to combine a set of c linear classifiers, being c the number of known classes. The linear classifiers used for

this purpose include Support Vector Machines (SVMs) and Recursive Least Square (RLS) classifiers, as both can be efficiently updated online [85]. Notably, previous work has shown that replacing the softmax layer in deep networks with a multiclass SVM can be effective also in different applications [104].

9.2 iCub Evasion

In order to empirically assess the security of the original iCub system and of the proposed countermeasure aimed to increase its security to evasion attack we rely on the adversarial framework that we propose in Chapter 4 and on the evasion Algorithm that we propose in Chapter 5. We discuss below how the gradient $\nabla A(\boldsymbol{x})$ needed in Algorithm 1 can be computed in our case study.

Gradient computation. One key issue of the aforementioned algorithm is the computation of the gradient of $A(\boldsymbol{x})$, which involves the gradients of the discriminant function $f_i(\boldsymbol{x})$ for $i \in 1, \ldots, \mathsf{c}$. It is not difficult to see that this can be computed using the chain rule to decouple the gradient of the discriminant function of the classifier trained on the deep feature space and the gradient of the deep network used for feature extraction, as $\nabla f_i(\boldsymbol{x}) = \frac{\partial f_i(\boldsymbol{z})}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$, being $\boldsymbol{z} \in \mathbb{R}^{\mathsf{m}}$ the set of deep features. In our case study, these are the $\mathsf{m} = 4,096$ values extracted from layer fc7 (see Fig. 9.1). Notably, the gradient of the deep network $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$ is readily available through automatic differentiation, as also highlighted in previous work [102, 47, 78, 84], whereas the availability of the gradient $\frac{\partial f_i(\boldsymbol{z})}{\partial \boldsymbol{z}}$ depends on whether the chosen classifier is differentiable or not. Several of the most used classifiers are differentiable, including, e.g., SVMs with differentiable kernels (we refer the reader to [12] for further details). Nevertheless, if the classifier is not differentiable (e.g., like in the case of decision trees), one may use a surrogate differentiable classifier to approximate it, as also suggested in [12, 41, 94].

9.3 Experimental Analysis

In this section, we report the results of the security evaluation performed on the iCub system (see Sect. 9.1) along with few adversarial examples to show how the proposed evasion algorithm can be exploited to create real-world attack samples. We then provide a conceptual representation and an empirical analysis to explain why neural networks are easily fooled and how our defense mechanism can improve their security in this context.

9.3.1 Experimental Setup

Our analysis has been performed using the *iCubWorld28* dataset [85], consisting of 28 different classes which include 7 different objects (cup, plate, etc..) of 4 different

Figure 9.2: Example images (one per class) from the *iCubWorld28* dataset, and subset of classes used in the *iCubWorld7* dataset (highlighted in red).

kinds each (e.g., cup1, cup2, etc..), as shown in Fig. 9.2. Each object was shown to iCub which automatically detected it and cropped the corresponding object image. Four acquisition sessions were performed on four different days, ending up with approximately $20,000$ images for training and test sets. As shown in [85], it is very difficult for iCub to be able to distinguish such slight category distinctions, like different kinds of cups. For this reason, we also consider here a reduced dataset, *iCubWorld7*, consisting only of 7 different objects, each of a different kind. The selected objects are highlighted in red in Fig. 9.2.

We implement the classification algorithm using three different multiclass SVM versions, all based on a one-versus-all scheme: a linear SVM (denoted with SVM in the following); an SVM with the RBF kernel (SVM-RBF); and an SVM with the RBF kernel implementing our defense mechanism based on rejection of adversarial examples (SVM-adv, Sect. 5.5). The regularization parameter $C \in \{10^{-3}, \ldots, 10^{3}\}$ and the RBF kernel parameter $\gamma \in \{10^{-6}, \ldots, 10^{-2}\}$ have been set equal for all one-versus-all SVMs in each multiclass classifier, by maximizing recognition accuracy through 3-fold cross validation.

9.3.2 Experimental Results

We report below the security evaluation results on the original system and

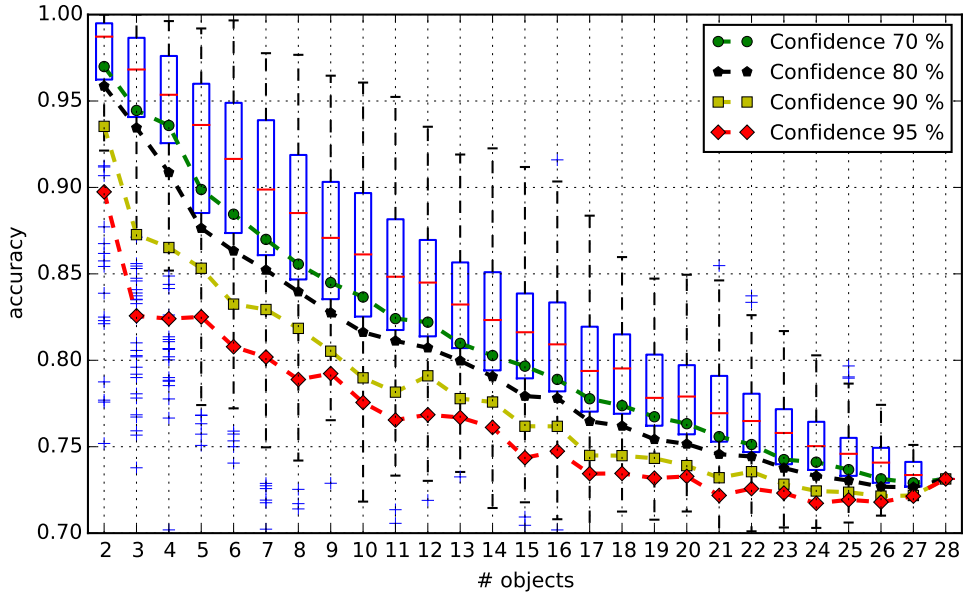


Figure 9.3: Box plots of the recognition accuracies measured for linear SVM predictors trained on random subsets from 2 to 28 objects (whiskers with maximum 1.5 interquartile range). Dotted super-imposed curves represent the minimum accuracy guaranteed within a fixed confidence level.

Baseline Performance. In Fig. 9.3 we report a box plot showing the empirical probability distributions of the accuracy achieved by the SVM classifier on increasingly larger object identification tasks, as suggested in [85]. To this end, we randomly select 300 subsets of increasing size from the *iCubWorld28* dataset (day4 acquisitions), and then train and test the classifier on each subset. The achieved accuracy is considered an observation for estimating the empirical distributions. The minimum accuracy value for which the fraction of observations in the estimated distribution was higher than a specific confidence threshold is indicated by a dotted line. Notably the reported performances for the linear SVM are almost identical to those reported in [85], where a different algorithm is used. Similar performances (omitted for brevity) are obtained using SVM-RBF.

Security Evaluation against Adversarial Examples. We now investigate the security of iCub in the presence of adversarial examples. In this experiment, we consider the first 100 examples per class for both the *iCubWorld28* and *iCubWorld7* datasets, ending up with training and test sets consisting of $2,800$ and 700 samples, respectively. The recognition accuracy against an increasing maximum admissible ℓ_2 perturbation (i.e., $d_{\max}$ value) is reported in Fig. 9.4 for both error-specific (top row plots) and error-generic (bottom row plots) attack scenarios. For error-specific evasion, we average our results not only on different training-test set splits, but also by considering a different target class in each repetition. While SVM and SVM-RBF show a comparable decrease in accuracy at increasing $d_{\max}$, SVM-adv is able

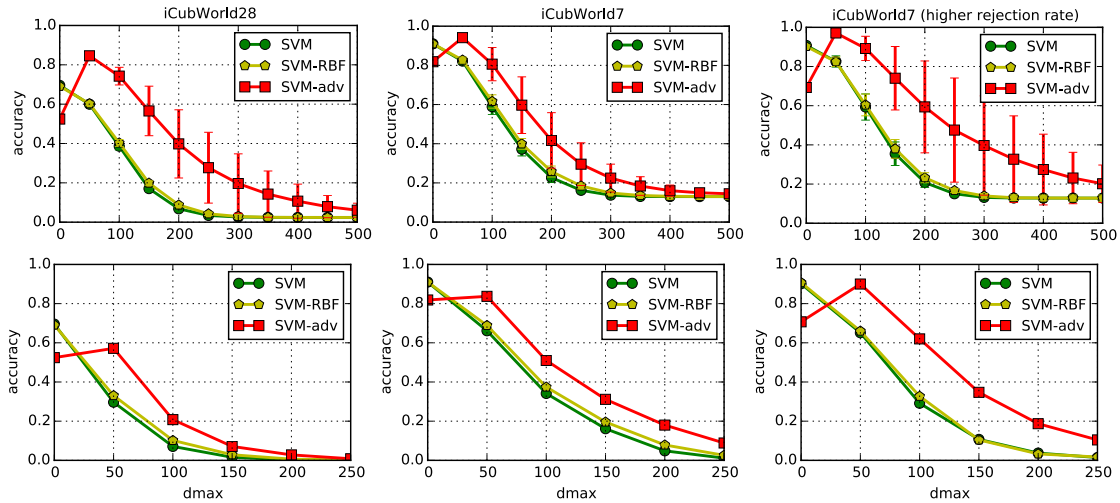


Figure 9.4: Recognition accuracy of iCub (using the three different classifiers SVM, SVM-RBF, and SVM-adv) against an increasing maximum admissible ℓ_2 input perturbation $d_{\max}$, for *iCubWorld28* (*left column*) and *iCubWorld7* (*middle* and *right* columns), using error-specific (*top row*), and error-generic (*bottom row*) adversarial examples. Baseline accuracies (in the absence of perturbation) are reported at $d_{\max} = 0$.

to strongly improve the security in most of the cases (as the corresponding curve decreases more gracefully). Notably, the performance of SVM-adv even increases for low values of $d_{\max}$. A plausible reason is that, even if all testing images are only slightly modified in input space, they immediately become blind-spot adversarial examples, ending up in a region which is far from the rest of the data. As the input perturbation increases, such samples are gradually drifted inside a different class, becoming *indistinguishable* from the samples of such class.

To further improve the security of iCub to adversarial examples, we set the rejection threshold of SVM-adv to a more conservative value, increasing the false negative rate for each base classifier of 5% (estimated on a validation set). This results in a significant security improvement, as shown in the rightmost plots in Fig. 9.4. However, as expected, this comes at the expense of misclassifying more legitimate (i.e. non-manipulated) samples.

Real-world Adversarial Examples. In Fig. 9.5 we report few adversarial examples generated using an error-specific evasion attack on the *iCubWorld28* data. Notably, the adversarial perturbation required to evade the system can be barely perceived by human eyes. As an important real-world application of the proposed attack algorithm, in the bottom right plots of Fig. 9.5, we report an adversarial example generated by manipulating only a subset of the image pixels, corresponding to the label of the detergent. In this case, the perturbation becomes easier to spot for a human, but localizing the noise in a region of interest allows the attacker to construct a practical, real-world adversarial object, by simply attaching an "adversarial"

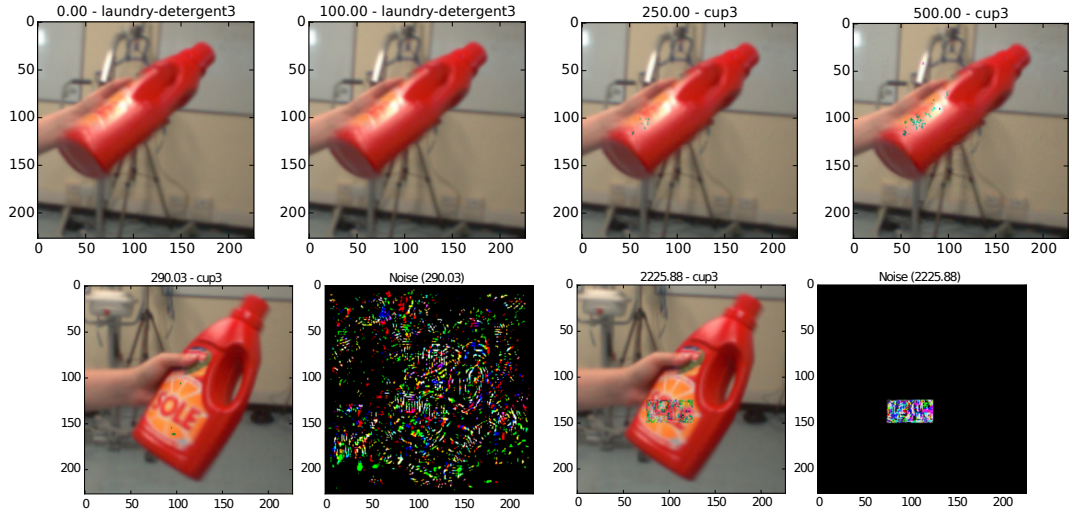


Figure 9.5: Plots in the top row show an adversarial example from class *laundry-detergent3*, modified to be misclassified as *cup3*, using an error-specific evasion attack, for increasing levels of input perturbation (reported in the title of the plots). Plots in the bottom row show the minimally-perturbed adversarial example that evades detection (i.e., the sample that evades detection with minimum $d_{\max}$), along with the corresponding noise applied to the initial image (amplified to make it clearly visible), for the case in which all pixels can be manipulated (first and second plot), and for the case in which modifications are constrained to the label of the detergent (i.e., simulating a sticker that can be applied to the real-world *adversarial* object).

sticker to the original object before showing it to the iCub humanoid robot.

Why are Deep Nets Fooled? Our analysis shows that also the iCub vision system can be fooled by adversarial examples, even by only adding a slightly-noticeable noise to the input image. To better understand the root causes of this phenomenon, we now provide an empirical analysis of the sensitivity of the feature mapping induced by the ImageNet deep network used by iCub, by comparing the ℓ_2 distance corresponding to random and adversarial perturbations in the input space, with the one measured in the deep feature space. To this end, we randomly perturb each training image such that the ℓ_2 distance between the initial and the perturbed image in the input space equals 10. We then measure the ℓ_2 distance between the deep feature vectors corresponding to the same images. For randomly-perturbed images, the average distance in deep space (along with its standard deviation) is 0.022 ± 0.002, while for the adversarially-perturbed images, it is 2.386 ± 0.386. This means that random perturbations in the input space only result in a very small shift in the deep space, while even light alterations of an image along the adversarial direction cause a large shift in deep space, which in turn highlights a significant *instability* of the deep feature space mapping induced by the ImageNet network. In other words, this means that images in the input space are very close to the decision

boundary along the adversarial (gradient) direction, as conceptually represented in Fig. 9.6. Note that this is a general issue for deep networks, not only specific to ImageNet [103, 45, 102, 47, 78, 84].

It should be thus clear that even a well-crafted modification of the last layers of the network, as in our proposed defense mechanism SVM-adv, can only mitigate this vulnerability. Indeed, it remains intimately related to the stability of the deep feature space mapping, which can be only addressed by imposing specific constraints while training the deep neural network; e.g., by imposing that small shifts in the input space correspond to small changes in the deep space, as recently proposed in [115]. Another possible countermeasure to improve stability of such mapping is to enforce classification of samples within a minimum *margin*, by modifying the neurons' activation functions and, potentially, considering a different regularizer for the objective function optimized by the deep network. In this respect, it would be interesting to investigate more in detail the intimate connections between robustness to adversarial input noise and regularization, as highlighted in [110, 94].

9.4 Discussion

Deep learning has shown groundbreaking performance in several real-world application domains, encompassing areas like computer vision, speech recognition and language processing, among others. Despite its impressive performances, recent work has shown how deep neural networks can be fooled by well-crafted adversarial examples affected by a barely-perceivable adversarial noise.

We have investigated the security of the robot-vision system of the iCub humanoid. Even if we do not restrict ourselves to the manipulation of pixels belonging to the object of interest in the image (which could lead one to more easily generate the corresponding real-world adversarial object, e.g., by mean of the application of specific stickers to objects), we have shown how our evasion algorithm enables this additional possibility. We have demonstrated and quantified the vulnerability of iCub to the presence of adversarial manipulations of input images, and suggested a simple countermeasure to mitigate the threat posed by such an issue. We have additionally shown that, while blind-spot adversarial examples can be detected using our defense mechanism, to further improve the security of iCub against *indistinguishable* adversarial examples, re-training the classification algorithm on top of a pre-trained deep neural network is not sufficient. To this end, different strategies to enforce the deep network to learn a more stable deep feature representation (in which small perturbations to the input data correspond to small perturbations in the deep feature space) should also be adopted, like the one proposed in [115].

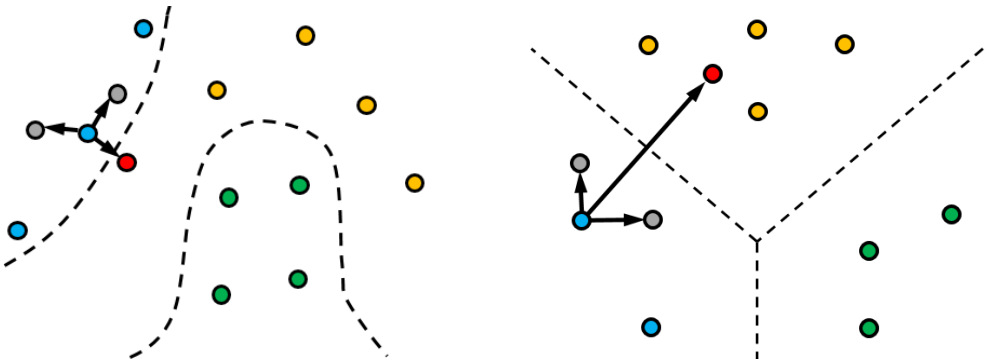


Figure 9.6: Conceptual representation of the vulnerability of the deep feature space mapping. The left and right plots respectively represent images in input space and the corresponding deep feature vectors. Randomly-perturbed versions of an input image are shown as gray points, while the adversarially-perturbed image is shown as a red point. Despite these points are at the same distance from the input image in the input space, the adversarial example is much farther in the deep feature space. This also means that images are very close to the decision boundary in input space, although in an adversarial direction that is difficult to guess at random due to the high dimensionality of the input space.

Chapter 10

Contributions and Limitations of this Doctoral Dissertation

Adversarial machine learning is a research field that was born almost ten years ago to evaluate the security of machine-learning systems used in security-related applications as spam and malware detection. This research field, has recently gained popularity in the computer vision community, since when, trying to interpret neural network decisions in classification tasks, it has been found that these extremely accurate systems can be fooled by adversarial images. In particular, these adversarial images are carefully obfuscated with a visually-indistinguishable noise that allows them to be misclassified as a different class, potentially chosen by the attacker, with high confidence. It is even possible to have completely noisy images misclassified as desired by applying the same well-crafted input perturbation.
However, computer-vision problems like object recognition in images are not necessarily adversarial in nature. When machine learning is used in security-related applications, it is often a natural target for attackers, for instance, cybercriminals create malware that has to bypass anti-malware technologies based on machine learning, or fabricate face mask to impersonate a victim user aiming to mislead face-recognition technologies. The underlying reason is that, in these cases, cybercriminals have a clear economical incentive to trick the system. In computer-vision applications, instead, it is really unlikely nowadays that an attacker has such an economic incentive. The application that is often considered as case of study by the computer vision community is the recognition or road signs. Machine learning algorithms are indeed used in self-driving cars and autonomous vehicles to this end. However, we are really far from having a mature technology to let cars drive without any human supervision on public streets. Even Tesla, which has state-of-the-art technology for self-driving cars, specifies that the Autopilot system is not supposed to replace a human driver, who has thus always to keep his hand on the steering wheel and his attention on the street; and Waymo, the new autonomous vehicles by Google is so far isolated in a 100-square-mile area with an employee on board.

From the next 2 of April they will be free to drive outside from this area. Nevertheless, they will be remotely controlled by Google employees. The accuracy of machine-learning algorithms in this case is in fact still not sufficient to make them suitable for safety-critical applications and even if it were, they could still commit, as humans, some errors causing mortal incidents. Substantially, even if machine learning becomes as accurate as humans, there may always be the need of a human supervision (e.g. think of the driver of the Tesla's car, and the Google employees that control the Waymo remotely). This may be required, for instance, to clearly assign the responsibility of potential errors committed by the automatic system, as it would be difficult to imagine that the technology manufacturers will take on a similar ethical and penal risk. Although there aren't practical computer vision applications of adversarial machine learning, studying the security of those systems is anyway useful. It allows, infact, to gain a further understanding about how they work highlighting the conditions in which they fail. This further knowledge can, of course, help the community to construct more accurate systems.
This study has lead therefore to a misconception about the security evaluation of machine learning systems. To evaluate the security of computer vision systems different samples are minimally perturbed using fast and approximate algorithms generating the so called "visually indistinguishable samples" and submitted to a classifier. The classifier is considered vulnerable if it recognizes those samples as belonging to a class different from the original ones. This evaluation raises different problems. Firstly, because so far there is no way to measure how much an image could resemble to one or another class for the complex human vision system. Secondly, because it is an optimistic evaluation of classifier security as an attacker would not apply a minimal perturbation to a samples but it would modify it as much as she can to increase the probability to fool the system. This claim is substantiated from the fact that there is a large number of defenses, published in top-tier machine-learning venues, which were broken immediately after they had been presented to the community [5]. To overcome these issues, we have provided a methodology that allows one to evaluate classifier security thoroughly, and not in an optimistic manner. The main drawback of the proposed methodology is that it may be computationally expensive. Moreover, as it exploits a gradient descent strategy, if the classifier is not differentiable, it requires the effort to find a differentiable surrogate that provides a good approximation of the original classifier. However, as formal verification techniques and certifiable defenses are still in their early stages, to the best of our knowledge, our methodology, is the only one that allows comparing the security of different classifiers under the same threat model (assumptions on the attacker's goal, knowledge and capability, and attack scenario). We hope therefore that it will be adopted to provide a fairer security evaluation of the proposed defenses by the community.
As we explained, the proposed security evaluation needs strong attack algorithms to be carried on. We have presented a novel poisoning algorithm based on back-gradient optimization that can be applied to a wider class of learners compared to the state

of art ones. It is applicable to all the learning algorithms trained through gradient-based procedures, including large neural networks and deep learning architectures. The empirical evaluation that we performed on spam filtering and malware detection dataset, shows that shallow neural networks can be significantly compromised even if the attacker only controls a small fraction of training points. Interestingly, it shows also that poisoning samples designed against one learning algorithm can be rather effective also against a different algorithm, highlighting an interesting transferability property, similar to that exhibited by evasion attacks.
Another relevant problem at the state of the art of adversarial machine learning is that efficient defenses are lacking, besides effective ones. This is a critical point to make them applicable in real systems with limited hardware resources. Moreover, it is not always clear under which conditions such defenses can be retained optimal, i.e., which attacks are they expected to successfully counter (and which ones are not going to be detected, instead). Leveraging recent results on the relationship between robustness and regularization we have analyzed the security of linear classifiers explaining which is the optimal regularizer choice against different evasion attacks. Sparse evasion attacks are one of the most exploited attacks against security-related systems as the attacker has often a constraint on the number of modified features. As we have shown one can drastically improve the security against those attacks using a classifier with an infinity-norm regularizer. We have discussed that this is equivalent to bound the weights updates during training, and we have developed a learning algorithm based on this strategy. The corresponding classifier allows applying different upper and lower bounds to different set of features, permitting to exploit application-specific constraints on the attacker capability and improving the system security (as shown in our case study on Android malware detection). We have further proposed a new octagonal regularizer that is a convex combination between a ℓ_1-norm and ℓ_∞-norm regularizer. When feature sparsity is crucial, this regularizer allows improving classifier security while retaining a good level of sparsity. We have then highlighted that poisoning can be seen as a sparse attack where sparsity is referred to the number of training samples that an attacker may have the possibility to inject into the training set. We have shown that this can be particularly dangerous for classifiers that take decisions based only on few influential training samples. We have then shown that the infinity-norm regularizer can be used in kernel space to enhance the security of those classifiers against label-flip attacks. We think that those result do not only help securing the considered classifiers but they can provide a useful starting point to increase the security of different systems, including also deep neural networks.
We have analyzed and quantified the vulnerability of a robot-vision system based on deep neural network to the presence of adversarial manipulations of input images and suggested a simple countermeasure inspired by open-set recognition techniques to mitigate the threat posed by this issue. We have shown that, while this defense mechanism enables detecting blind-spot adversarial examples, the system is still

vulnerable to *indistinguishable* adversarial examples (in terms of their deep-feature representation with respect to samples belonging to different classes). This vulnerability shows that employing pretreated neural networks as feature extractors may be a risk. The reason is that the mapping learned by these networks is unstable, i.e. there are directions in which a minimal perturbation in input space causes a large shift in the deep feature space.
An interesting research direction is thus to find proper regularizers that increase the margin in input space and consequently classifier security against specific types of evasion attack (e.g. sparse and dense), similarly to what we have shown for linear classifiers. Even though the most appropriate regularizer can, of course, increase classifier security, it could not avoid having samples misclassified as belonging to a totally different class. To reach this goal there is probably the need to inject some further knowledge into the algorithm. This knowledge could be, for instance, related to the similarity between objects of different classes or to the object structure itself.
Another interesting direction is related to poisoning against deep neural network classifiers. We have proposed a poisoning algorithm applicable to deep neural network. However, it is still to be evaluated if it is really effective against them. This could not be the case for different reasons. The first one is that neural networks are almost always trained on a huge number of samples. The number of samples that an attacker needs to change, to modify significantly the decision function of the classifier could be therefore so large to make the attack impracticable. The second one is that, due to their large capacity, deep neural networks may be able to learn the poisoning samples without affecting their decisions on legitimate, non-adversarial data points. In this case the poisoning points could not be able to sufficiently change the decision region to make the classifier misclassify different samples. We believe that this analysis is a very interesting direction for future work.

To sum up, different challenges for the adversarial machine learning community are still far from being efficiently solved. Throughout this thesis, we have provided theoretically sound methodologies and tools to increase classifier security. We have highlighted that, to be informative, the evaluation of machine learning systems should be carefully designed. To this end, we have provided some tools that help to performing a comparative, proactive and fairer security evaluation. We have shown that leveraging theory is possible to get a better understanding of machine-learning systems security, to create efficient and effective countermeasures, and to highlight potential trade-offs, such as that between sparsity and security.

Bibliography

[1] Yousra Aafer, Wenliang Du, and Heng Yin. DroidAPIMiner: Mining API-level features for robust malware detection in android. In *Proc. of International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2013.

[2] Ian Goodfellow an Nicholas Papernot. The challenge of verification and testing of machine learning, 2017.

[3] I. Arce. The weakest link revisited [information security]. *IEEE Security & Privacy*, 1(2):72–76, Mar 2003.

[4] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. Drebin: Efficient and explainable detection of android malware in your pocket. In *Proc. 21st Annual Network & Distributed System Security Symposium (NDSS)*. The Internet Society, 2014.

[5] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. 2018.

[6] Marco Barreno, Blaine Nelson, Anthony Joseph, and J. Tygar. The security of machine learning. *Machine Learning*, 81:121–148, 2010.

[7] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proc. ACM Symp. Information, Computer and Comm. Sec.*, ASIACCS '06, pages 16–25, New York, NY, USA, 2006. ACM.

[8] David Barrera, H. Güneş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010.

[9] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1563–1572, 2016.

[10] Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

[11] Kristin P. Bennett and Erin J. Bredensteiner. Duality and geometry in svm classifiers. In *Proc. 17th Int'l Conf. Mach. Learn.*, ICML '00, pages 57–64, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[12] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný, editors, *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III*, volume 8190 of *LNCS*, pages 387–402. Springer Berlin Heidelberg, 2013.

[13] Battista Biggio, Samuel Rota Bulò, Ignazio Pillai, Michele Mura, Eyasu Zemene Mequanint, Marcello Pelillo, and Fabio Roli. Poisoning complete-linkage hierarchical clustering. In P. Franti, G. Brown, M. Loog, F. Escolano, and M. Pelillo, editors, *Joint IAPR Int'l Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, volume 8621 of *Lecture Notes in Computer Science*, pages 42–52, Joensuu, Finland, 2014. Springer Berlin Heidelberg.

[14] Battista Biggio, Igino Corona, Zhi-Min He, Patrick P. K. Chan, Giorgio Giacinto, Daniel S. Yeung, and Fabio Roli. One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time. In Friedhelm Schwenker, Fabio Roli, and Josef Kittler, editors, *Multiple Classifier Systems*, volume 9132 of *Lecture Notes in Computer Science*, pages 168–180. Springer International Publishing, 2015.

[15] Battista Biggio, Giorgio Fumera, and Fabio Roli. Multiple classifier systems for robust classifier design in adversarial environments. *Int'l J. Mach. Learn. and Cybernetics*, 1(1):27–41, 2010.

[16] Battista Biggio, Giorgio Fumera, and Fabio Roli. Pattern recognition systems under attack: Design issues and research challenges. *Int'l J. Patt. Recogn. Artif. Intell.*, 28(7):1460002, 2014.

[17] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):984–996, April 2014.

[18] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *Journal of Machine Learning Research - Proc. 3rd Asian Conf. Machine Learning*, volume 20, pages 97–112, November 2011.

[19] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In John Langford and Joelle Pineau, editors, *29th Int'l Conf. on Machine Learning*, pages 1807–1814. Omnipress, 2012.

[20] Battista Biggio, Ignazio Pillai, Samuel Rota Bulò, Davide Ariu, Marcello Pelillo, and Fabio Roli. Is data clustering in adversarial settings secure? In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 87–98, New York, NY, USA, 2013. ACM.

[21] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. Poisoning behavioral malware clustering. In *2014 Workshop on Artificial Intelligent and Security*, AISec '14, pages 27–36, New York, NY, USA, 2014. ACM.

[22] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.

[23] C. Blake and C.J. Merz. UCI Repository of machine learning databases. *http://www. ics. uci. edu/˜ mlearn/MLRepository. html*, 1998.

[24] Bondell and Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR. 2008.

[25] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT'2010*, pages 177–186. Springer, 2010.

[26] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[27] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[28] Michael Brückner, Christian Kanzow, and Tobias Scheffer. Static prediction games for adversarial learning problems. *J. Mach. Learn. Res.*, 13:2617–2654, September 2012.

[29] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2:121–167, June 1998.

[30] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland, July 1997.

[31] Gordon V. Cormack. Trec 2007 spam track overview. In Ellen M. Voorhees and Lori P. Buckland, editors, *TREC*, volume Special Publication 500-274. National Institute of Standards and Technology (NIST), 2007.

[32] Igino Corona, Davide Maiorca, Davide Ariu, and Giorgio Giacinto. Lux0r: Detection of malicious pdf-embedded javascript code through discriminant analysis of API references. In *Proc. 2014 Workshop on Artificial Intelligent and Security Workshop*, AISec '14, pages 47–57, New York, NY, USA, 2014. ACM.

[33] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.

[34] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, and Angelos D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *IEEE Symposium on Security and Privacy*, pages 81–95. IEEE Computer Society, 2008.

[35] Nello Cristianini. Intelligence reinvented. *New Scientist*, 232(3097):37–41, 2016.

[36] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 99–108, Seattle, 2004.

[37] Ambra Demontis, Battista Biggio, Giorgio Fumera, Giorgio Giacinto, and Fabio Roli. Infinity-norm support vector machines against adversarial label contamination. In Alessandro Armando, Roberto Baldoni, and Riccardo Focardi, editors, *First Italian Conference on Cybersecurity (ITASEC17)*, number 1816 in CEUR Workshop Proceedings, pages 106–115, Aachen, 2017.

[38] Ambra Demontis, Battista Biggio, Giorgio Fumera, and Fabio Roli. Super-sparse regression for fast age estimation from faces at test time. In *Image Analysis and Processing—ICIAP 2015*, pages 551–562. Springer, 2015.

[39] Ambra Demontis, Marco Melis, Battista Biggio, Giorgio Fumera, and Fabio Roli. Super-sparse learning in similarity spaces. *IEEE Computational Intelligence Magazine*, 11(4):36–45, Nov 2016.

[40] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Trans. Dependable and Secure Computing*, In press.

[41] Ambra Demontis, Paolo Russu, Battista Biggio, Giorgio Fumera, and Fabio Roli. On security and sparsity of linear classifiers for adversarial settings. In Antonio Robles-Kelly, Marco Loog, Battista Biggio, Francisco Escolano, and Richard Wilson, editors, *Joint IAPR Int'l Workshop on Structural, Syntactic,*

and Statistical Pattern Recognition, volume 10029 of *LNCS*, pages 322–332, Cham, 2016. Springer International Publishing.

[42] C. Do, C.S. Foo, and A.Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *Advances in Neural Information Processing Systems*, pages 377–384, 2008.

[43] Justin Domke. Generic methods for optimization-based modeling. In Neil D. Lawrence and Mark Girolami, editors, *15th Int'l Conf. Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 318–326, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[44] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[45] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[46] Amir Globerson and Sam T. Roweis. Nightmare at test time: robust learning by feature deletion. In William W. Cohen and Andrew Moore, editors, *Proceedings of the 23rd International Conference on Machine Learning*, volume 148, pages 353–360. ACM, 2006.

[47] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[48] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[49] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[50] Johannes Hoffmann, Teemu Rytilahti, Davide Maiorca, Marcel Winandy, Giorgio Giacinto, and Thorsten Holz. Evaluating analysis tools for android apps: Status quo and robustness against obfuscation. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy, CODASPY 2016, New Orleans, LA, USA, March 9-11, 2016*, pages 139–141, 2016.

[51] Johannes Hoffmann, Teemu Rytilahti, Davide Maiorca, Marcel Winandy, Giorgio Giacinto, and Thorsten Holz. Evaluating analysis tools for android apps: Status quo and robustness against obfuscation. In *Technical Report TR-HGI-2016-003, Horst Görtz Institute for IT Security*, 2016.

[52] L. Huang, A. D. Joseph, B. Nelson, B. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *4th ACM Workshop on Artificial Intelligence and Security (AISec 2011)*, pages 43–57, Chicago, IL, USA, 2011.

[53] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. *CoRR*, abs/1610.06940, 2016.

[54] Anthony D. Joseph, Pavel Laskov, Fabio Roli, J. Doug Tygar, and Blaine Nelson. Machine Learning Methods for Computer Security (Dagstuhl Perspectives Workshop 12371). *Dagstuhl Manifestos*, 3(1):1–30, 2013.

[55] Shuichi Katsumata and Akiko Takeda. Robust cost sensitive support vector machine. In G. Lebanon and S.V.N. Vishwanathan, editors, *18th Int'l Conf. on Artificial Intelligence and Statistics (AISTATS)*, volume 38 of *JMLR Workshop and Conference Proceedings*, pages 434–443. JMLR.org, 2015.

[56] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.

[57] Marius Kloft and Pavel Laskov. Security analysis of online centroid anomaly detection. *Journal of Machine Learning Research*, 13:3647–3690, 2012.

[58] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning (ICML)*, 2017.

[59] Aleksander Kolcz and Choon Hui Teo. Feature weighting for improved classifier robustness. In *Sixth Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2009.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[62] Martina Lindorfer, Matthias Neugschwandtner, and Christian Platzer. Marvin: Efficient and Comprehensive Mobile App Classification Through Static

and Dynamic Analysis. In *Proceedings of the 39th Annual International Computers, Software & Applications Conference (COMPSAC)*, 2015.

[63] Martina Lindorfer, Matthias Neugschwandtner, Lukas Weichselbaum, Yanick Fratantonio, Victor van der Veen, and Christian Platzer. Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors. In *Proceedings of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2014.

[64] Chang Liu, Bo Li, Yevgeniy Vorobeychik, and Alina Oprea. Robust linear regression against training data poisoning. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, pages 91–102, 2017.

[65] Roi Livni, Koby Crammer, Amir Globerson, Elsc-icnc Edmond, and Lily Safra. A simple geometric interpretation of SVM using stochastic adversaries. In *JMLR W&CP - Proc.*, volume 22 of *AISTATS '12*, pages 722–730, 2012.

[66] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proc. 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 641–647, Chicago, IL, USA, 2005. ACM Press.

[67] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *Second Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2005.

[68] Yan Luo, Xavier Boix, Gemma Roig, Tomaso Poggio, and Qi Zhao. Foveation-based mechanisms alleviate adversarial examples. *arXiv preprint arXiv:1511.06292*, 2015.

[69] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2113–2122. JMLR.org, 2015.

[70] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015.

[71] Davide Maiorca, Davide Ariu, Igino Corona, Marco Aresu, and Giorgio Giacinto. Stealth attacks. *Comput. Secur.*, 51(C):16–31, June 2015.

[72] Davide Maiorca, Giorgio Giacinto, and Igino Corona. A pattern recognition system for malicious pdf files detection. In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, volume 7376 of *Lecture Notes in Computer Science*, pages 510–524. Springer Berlin Heidelberg, 2012.

[73] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *29th AAAI Conf. Artificial Intelligence (AAAI '15)*, 2015.

[74] Marco Melis, Ambra Demontis, Battista Biggio, Gavin Brown, Giorgio Fumera, and Fabio Roli. Is deep learning safe for robot vision? adversarial examples against the icub humanoid. In *ICCV 2017 Workshop on Vision in Practice on Autonomous Robots (ViPAR)*, In Press.

[75] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM, 2008.

[76] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *Proceedings of 5th International Conference on Learning Representations (ICLR)*, 2017.

[77] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[78] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[79] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Proc. of Annual Computer Security Applications Conference (ACSAC)*, 2007.

[80] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *10th ACM Workshop on Artificial Intelligence and Security*, In Press.

[81] B. Nelson, M. Barreno, F.J. Chi, A.D. Joseph, B.I.P. Rubinstein, U. Saini, C.A. Sutton, J.D. Tygar, and K. Xia. Exploiting Machine Learning to Subvert your Spam Filter. *LEET*, 8:1–9, 2008.

[82] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

[83] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '17, pages 506–519, New York, NY, USA, 2017. ACM.

[84] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proc. 1st IEEE European Symposium on Security and Privacy*, pages 372–387. IEEE, 2016.

[85] Giulia Pasquale, Carlo Ciliberto, Francesca Odone, Lorenzo Rosasco, Lorenzo Natale, and Ingegneria dei Sistemi. Teaching icub to recognize objects using deep convolutional neural networks. In *MLIS@ ICML*, pages 21–25, 2015.

[86] B.A. Pearlmutter. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–160, 1994.

[87] F. Pedregosa. Hyperparameter optimization with approximate gradient. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 737–746, New York, New York, USA, 20–22 Jun 2016. PMLR.

[88] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[89] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *Proceedings of the 22Nd International Conference on Computer Aided Verification*, CAV'10, pages 243–257, Berlin, Heidelberg, 2010. Springer-Verlag.

[90] Vaibhav Rastogi, Zhengyang Qu, Jedidiah McClurg, Yinzhi Cao, and Yan Chen. *Uranine: Real-time Privacy Leakage Monitoring without System Modification for Android*, pages 256–276. Springer International Publishing, Cham, 2015.

[91] Bradley Reaves, Jasmine Bowers, Sigmund Albert Gorski III, Olabode Anise, Rahul Bobhate, Raymond Cho, Hiranava Das, Sharique Hussain, Hamza Karachiwala, Nolen Scaife, Byron Wright, Kevin Butler, William Enck, and Patrick Traynor. *droid: Assessment and evaluation of android application analysis tools. *ACM Comput. Surv.*, 49(3):55:1–55:30, oct. 2016.

[92] Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara. Experimental study with real-world data for android app security analysis using machine learning. In *Proceedings of the 31th Annual Computer Security Applications Conference*, In press.

[93] Benjamin I.P. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J. D. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference*, IMC '09, pages 1–14, New York, NY, USA, 2009. ACM.

[94] Paolo Russu, Ambra Demontis, Battista Biggio, Giorgio Fumera, and Fabio Roli. Secure kernel machines against evasion attacks. In *9th ACM Workshop on Artificial Intelligence and Security*, AISec '16, pages 59–69, New York, NY, USA, 2016. ACM.

[95] W.J. Scheirer, L.P. Jain, and T.E. Boult. Probability models for open set recognition. *IEEE Trans. Patt. An. Mach. Intell.*, 36(11):2317–2324, 2014.

[96] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34:1–47, March 2002.

[97] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E.C. Lupu. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection. *arXiv preprint arXiv:1609.03020*, 2016.

[98] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016.

[99] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540. ACM, 2016.

[100] Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright. *Optimization for Machine Learning*. The MIT Press, 2011.

[101] J.A.K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[102] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[103] Thomas Tanay and Lewis Griffin. A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint arXiv:1608.07690*, 2016.

[104] Yichuan Tang. Deep learning using support vector machines. In *ICML Workshop on Representational Learning*, volume arXiv:1306.0239, Atlanta, USA, 2013.

[105] Choon Hui Teo, Amir Globerson, Sam Roweis, and Alex Smola. Convex learning with invariances. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1489–1496. MIT Press, Cambridge, MA, 2008.

[106] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[107] Nedim Šrndic and Pavel Laskov. Practical evasion of a learning-based classifier: A case study. In *Proc. 2014 IEEE Symp. Security and Privacy*, SP '14, pages 197–211, Washington, DC, USA, 2014. IEEE CS.

[108] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In Francis Bach and David Blei, editors, *JMLR W&CP - Proc. 32nd Int'l Conf. Mach. Learning (ICML)*, volume 37, pages 1689–1698, 2015.

[109] Huang Xiao, Battista Biggio, Blaine Nelson, Han Xiao, Claudia Eckert, and Fabio Roli. Support vector machines under adversarial label contamination. *Neurocomputing, Special Issue on Advances in Learning with Label Noise*, 160(0):53 – 62, 2015.

[110] Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *Journal of Machine Learning Research*, 10:1485–1510, July 2009.

[111] Wenbo Yang, Yuanyuan Zhang, Juanru Li, Junliang Shu, Bodong Li, Wenjun Hu, and Dawu Gu. *AppSpear: Bytecode Decrypting and DEX Reassembling for Packed Android Malware*, pages 359–381. Springer International Publishing, Cham, 2015.

[112] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[113] F. Zhang, P.P.K. Chan, B. Biggio, D.S. Yeung, and F. Roli. Adversarial feature selection against evasion attacks. *IEEE Transactions on Cybernetics*, 46(3):766–777, 2016.

[114] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *21st Int'l Conf. Machine Learning*, ICML '04, pages 116–123, New York, NY, USA, 2004. ACM.

[115] Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4480–4488, 2016.

[116] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, 2012.

[117] Ji Zhu, Saharon Rosset, Robert Tibshirani, and Trevor J. Hastie. 1-norm support vector machines. In S. Thrun, L.K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, pages 49–56. MIT Press, 2004.

[118] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67(2):301–320, 2005.